Dynamic ORB_SLAM*

Justin Bi, Yicheng Tao, Yuliang Zhu, Li Chen, Pradeep Suresh
{bijustin, yichengt, yuliangz, ilnehc, spradeep}@umich.edu

***Abstract*—For autonomous robotic systems, Simultaneous Localization and Mapping (SLAM) is often a necessity. Many state-of-the-art SLAM systems over the past decade have shown impressive performance, but they oftentimes make the assumption that the environment is static. This tends to be a naïve assumption, as this is not the case with robots that must operate with humans in their environment such as autonomous vehicles. In this work, we present a visual SLAM system built on ORB_SLAM2 that is more robust to dynamic environments while operating in real time, one with the use of deep learning and one without the usage of deep learning, YOLO Dynamic ORB_SLAM and FAST Dynamic ORB_SLAM respectively. For the YOLO version, we use the YOLOv3 to segment out objects that may be dynamic. For the FAST version, a combination of optical flow and epipolar constraints are used to prevent dynamic objects from impacting the SLAM system, without the need of special hardware such as a GPU. We show that the our systems are capable of improving the accuracy of ORB_SLAM on the TUM RGB-D dataset.**

I. Introduction

Simultaneous Localization and Mapping (SLAM) is the process by which a robotic system constructs a map of the environment using different kinds of sensors while estimating its own position in the environment simultaneously [1]. Among the different sensor modalities, RGB-D cameras provide rich information of the environment that allows for robust and accurate place recognition [1], [2]. Therefore, visual SLAM (Vision-based SLAM, VSLAM) solutions, in which the main sensor is a camera, are of major interest [3].

VSLAM is divided into direct method and indirect methods according to whether the visual odometer extracts features [4]. The indirect method is also called feature point method, which extracts feature points from the obtained image, then performs feature matching to obtain matching pairs of corresponding points. Numerous indirect methods have been proposed, such as MonoSLAM, Parallel Tracking and Mapping (PTAM), Stereo Odometry Feature Tracking (SOFT-SLAM) [5], [6]. Raúl Mur-Artal *et al.* [3], [7] proposed feature-based monocular ORB_SLAM system which can achieve real time operation on large environments, real time loop closing and real time camera relocalization, and extended it to RGB-D and stereo cameras.

However, typical algorithms or datasets assume a static environment and don't consider the potential consequences of accidentally adding dynamic objects to the 3D map. This shortcoming limits the applicability of VSLAM in many real-world cases such as long-term mapping. Several methods for dynamic environments SLAM were proposed during the past decades. For example, Zou *et al.* [8] proposed using multiple cameras collaboratively to do 3D construction. Tan *et al.* [9] detect changes that take place in the scene by projecting the map features into the current frame for appearance and structure validation. Bescos *et al.* [10] use Mask R-CNN to segment the *a priori* dynamic objects (*e.g.*, people and cars) and use feature-based sparse tracking (not reconstructing moving objects at all).

In this study, we first introduce a method - Moving Object Detection - to detect and eliminate moving objects, given two consecutive frames. The three main steps of this method are: 1) Finding correspondences between ORB-Features of the two input frames. We compare two methods for this step. One provides accuracy while the other provides speed. 2) Robustly solving for the 'Fundamental Matrix' F between the two input frames using inlier correspondences. Given a point in first frame, multiplying it by F tells us which epipolar line to search along for the corresponding point in the second frame. 3) Extracting the features belonging to a dynamic region. This is done by first finding the epipolar lines along which the features should lie in the second frame if they are static. Next, we find the actual location of the corresponding feature from frame 1, in frame 2 and calculate the distance - d between this location and its supposed epipolar line. We then eliminate the features for which d lies within some threshold values.

We then demonstrate the use of this method by extending ORB_SLAM to FAST Dynamic ORB_SLAM. We also compare this with our modification to DynaSLAM [10] called YOLO Dynamic_ORB SLAM in which we replace Mask R-CNN used in [10] with the YOLOv3 method to rapidly extract an approximate mask for dynamic objects.

The report is organized as follows: Section I introduces the Visual SLAM problem with its various implementations and gives a brief overview of our method.

*Demo video: https://www.youtube.com/watch?v=sazkFVVjITQ
*FAST Dynamic ORB_SLAM: https://github.com/bijustin/Fast-Dynamic-ORB-SLAM
*YOLO Dynamic ORB_SLAM: https://github.com/bijustin/YOLO-DynaSLAM

Section II discusses the relevant work and background in the area of Visual SLAM, Optical Flow and Real-time object detection. Sections III, IV and V describe the methods we investigated to detect and eliminate feature points belonging to dynamic regions of the incoming camera frame. Section VI discusses the results of our investigation.

II. Related Background

A. ORB_SLAM & ORB_SLAM2

ORB_SLAM2 for stereo and RGB-D cameras is built on the monocular feature-based ORB_SLAM [3]. A general overview of the system is shown in Fig. 1. The structure contains four threads that work in parallel, as shown in Fig. 1a: 1) the Tracking uses each frame to track the localization of the camera by finding the feature matches and minimizing the reprojection error. It also decides which frame should be used as a keyframe. 2) the Local Mapping uses keyframes to reconstruct the surroundings of the camera pose. Inserted keyframes will be culled if they are found to be redundant or similar. 3) the Loop Closing detects large loops and corrects accumulated drift. 4) the Full Bundle Adjustment (BA) is performed to achieve an optimized and consistent reconstruction of the environment for the entire map. Main components of ORB_SLAM are basically the same with ORB_SLAM2, only without the last module Full BA. Fig. 1b shows other pre-possessing for stereo and RGB-D images extensions to build ORB_SLAM2.

ORB_SLAM (representing ORB_SLAM2 below, if not specified) uses the same ORB (Orientated FAST and Rotated BRIEF) features for tracking, mapping and place relocalization tasks [3], [11]. These features are robust to rotation and scale and present a good invariance to camera auto-gain and auto-exposure, and illumination changes. Meanwhile, they are extremely fast to compute and match. ORB_SLAM constructs an undirected weighted pose graph, with each keyframe being a node. An edge between two keyframes exists if they share observations of the same map points. Then a pose graph optimization will be performed to distribute the loop closing error along the graph [7].

B. Optical Flow

Estimating per pixel motion from one image to the subsequent image is the problem of calculating optical flow between two images. A pair of two images is the minimum amount for us to calculate the motion. The optical flow is a two dimensional vector which gives the information of the relative displacement of each pixel compared to the previous frame. Optical flow has broad applications, such as motion based segmentation, global motion compensation and video compression. To calculate optical flow, we generally use the brightness constancy assumption [12]. We have the following three-dimensional equation:

$$f(x, y, t) = f(x + dx, y + dy, t + dt) \tag{1}$$

Using Taylor expansion, we get:

$$f(x, y, t) = f(x, y, t) + \frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial y}dy + \frac{\partial f}{\partial t}dt \tag{2}$$

This can also be written as:

$$f_x u + f_y v + f_t = 0 \tag{3}$$

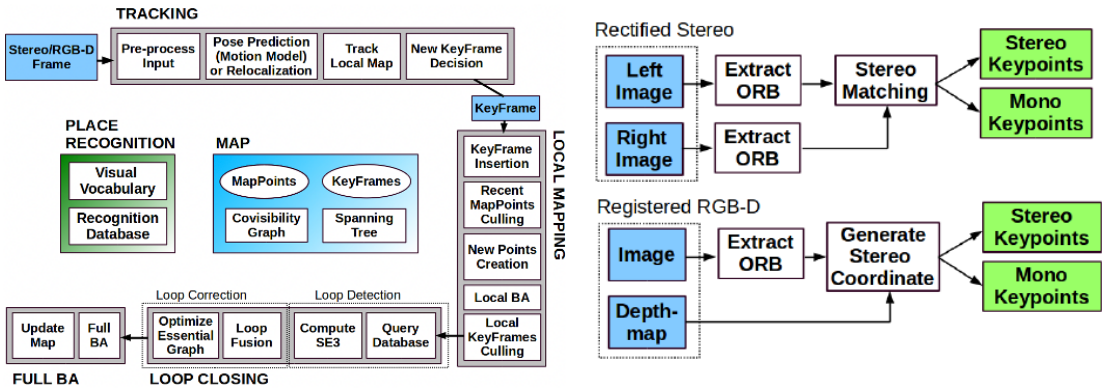
where u is the unit displacement in the direction of x axis, v is the unit displacement in the direction of y axis. We can also treat and solve the optical flow problem as an optimization problem as suggested in [13]. However, Horn-schunck and Lucas-Kanade optical methods work only for small motion. If the objects move fast and the brightness changes rapidly, small masks (e.g. size of 2*2 or 3*3) will fail to estimate spatiotemporal derivations. In that case, pyramids can be useful for computing large optical flow vectors. In out project, we utilize the optical flow algorithm to get the corresponding feature points on the second frame based on the points on the first frame.

C. DynaSLAM & YOLOv3

1) DynaSLAM: One more robust approach in dynamic scene than conventional SLAM methods is DynaSLAM. Fig. 2 shows the overview of its system [10]. For monocular and stereo cameras, the images pass through a Mask R-CNN for computing the pixel-wise semantic segmentation of the *a priori* dynamic content, *e.g.*, the people or vehicles, before being used for a mapping and tracking. In RGB-D case (black dashed line), a second approach based on multi-view geometry is added for a more accurate motion segmentation, for which a simpler and computationally lighter version of the tracking module in ORB_SLAM2 is proposed. The Low-Cost Tracking projects the map features in the image frame, searches for the correspondences in the static areas of the image, and minimizes the reprojection error to optimize the camera pose.

Note that DynaSLAM is not optimized for real-time operation. In the TUM Dynamic Objects dataset, DynaSLAM can achieve best RGB-D SLAM solution with average computational time over 500 ms per frame on offline mode.

2) YOLOv3: You only look once (YOLO) is a state-of-the-art, real-time object detection system [14], [15]. The system runs a single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes, then thresholds the resulting detections by the model's confidence. YOLO provides



(a) ORB_SLAM2 system overview, system threads and modules (b) input prepossessing

Fig. 1: Algorithm structure of ORB_SLAM2

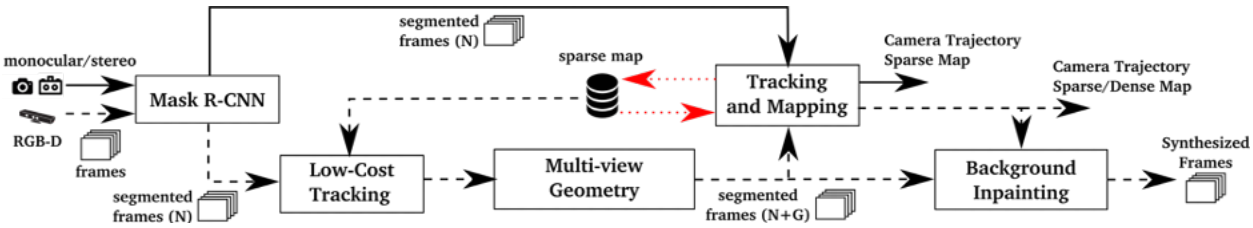


Fig. 2: DynaSLAM system overview

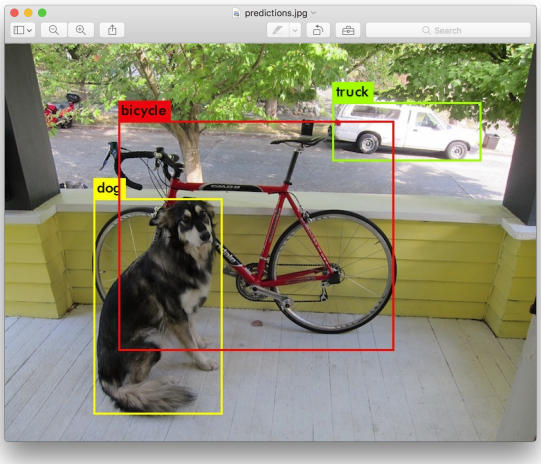


Fig. 3: Pretrained YOLOv3 example

weights trained on COCO dataset. It includes 80 types of objects, such as person, bicycle, car, motorcycle, airplane, bus, train, truck, boat, bird, cat, dog, etc. We consider that for most environments, the dynamic objects likely to appear are included within it. One example of YOLOv3 is shown in Fig. 3.

The biggest advantage of YOLO is its speed. As in certain experiments in [14], it is better than ResNet-101 and 1.5× faster and has similar performance to ResNet-152 and 2× faster. Hence, it is a possibility to replace Mask R-CNN with YOLOv3 to achieve real-time operation.

III. Moving Object Detection

To make SLAM results more accurate, we try to separate moving objects from the static background on monocular cameras and RGB-D cameras. The pipeline of moving object detection is straightforward. Firstly, ORB feature matches between two close frames are found. Secondly, the fundamental matrix is solved using the Random Sampling Consensus (RANSAC) algorithm. The third step is calculating the epipolar lines (corresponding to the outlier matches) in the second frame, and the distance between the feature points and the epipolar lines. If the distance is greater than some threshold, then we consider its corresponding feature point to be part of the moving object.

A. Feature Match Generation

In the first step, we examined two different methods to generate the feature matches. The first one is to extract ORB features from two consecutive frames and then using brute force method to do the matching. In practice, we find that applying hamming loss will get better matches. However this method will still produce many mismatches (Fig. 4), which are not useful at all in the

later RANSAC process. To eliminate these mismatches, we sorted these matches in the ascending order by their hamming-loss distance, and then select only the good part of (*e.g.*, first one third) the matches for calculating the fundamental matrix.

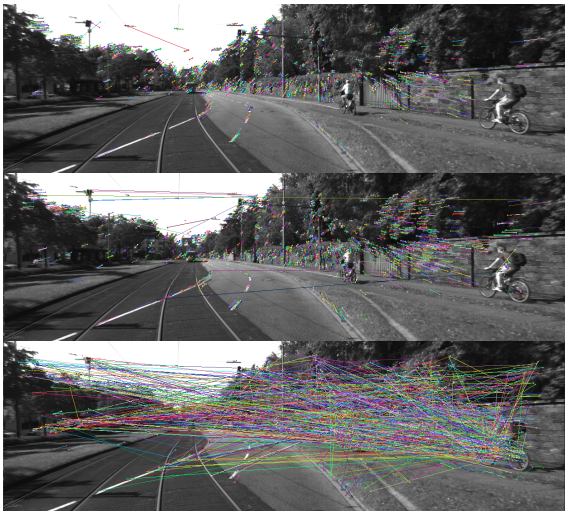


Fig. 4: Three equally-sized subsets of the sorted feature matches in the ascending order

The second method is first to extract ORB features from the first frame, and then feed them into the optical flow pyramid algorithm to get the corresponding feature points in the second frame. This method is more straightforward and create almost zero mismatches (Fig. 5).

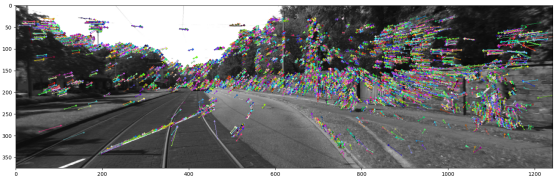


Fig. 5: Matches obtained by optical flow

B. Fundamental Matrix

The second step is to solve the fundamental matrix using the RANSAC algorithm, which can extract inlier matches from all the matches from step one. We assume all the inlier matches belong to the static objects according to the fundamental matrix model, while the outliers are matches either belonging to the moving objects or other noisy matches. In practice, the result we get using ORB feature extracting and pairing method is more convincing after doing the RANSAC. In Fig. 6, for the brute force method, there are no optical flows in the bicycle area of the first image, while there are some points on the moving object for the optical flow method.

However, for the whole system, using brute force to get all the matches is time-consuming. There is a trade-off between the accuracy and running speed. In our project, we go with the brute force method, because the accuracy is our first priority, and even with the reduction in speed, our program is near-real time.

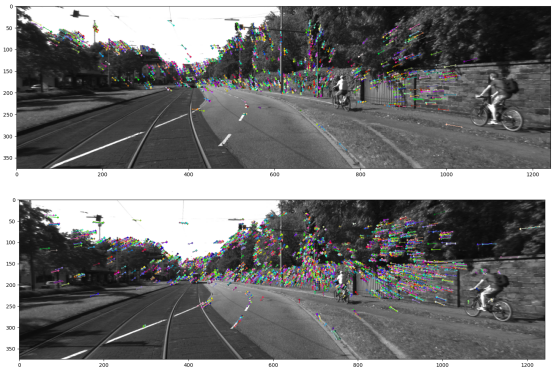


Fig. 6: The first image shows the results after RANSAC using brute force matches, the second image shows the results after RANSAC using optical flow

C. Dynamic Feature Points Segmentation

The third step is to get the points corresponding to moving objects among the outliers (In Fig. 7) after applying RANSAC. According to the fundamental matrix model (in Fig. 8). Similar to human eyes, the fundamental matrix model is using more than one camera to get the depth information of the objects, which is also known as stereo vision. Given two consecutive frames, the fundamental matrix is capable of transforming one point in the first frame to a corresponding epiline in the second frame. p_1, p_2 are the matched points in the first frame and the second frame respectively, and p_1e_1, p_2e_2 are the epipolar lines in the respective local frames. p_2e_2 can be calculated by F (fundamental matrix) and $p_2e_2 = Fp_1$ [16] [17]. Denote

$$p_1 = [x_1, y_1, 1]^T \tag{4}$$

$$p_2 = [x_2, y_2, 1]^T \tag{5}$$

$$p_2e_2 = \begin{bmatrix} A \\ B \\ C \end{bmatrix} = Fp_1 = F \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \tag{6}$$

where A, B, C represents the line vector. The distance between the feature points and the epipolar lines can be calculated using the following formula:

$$d = \frac{p_2^T F p_1}{\sqrt{A^2 + B^2 + C^2}} \tag{7}$$

After getting the distance, we set two threshold values to extract moving points from the outliers points set: if

$$threshold_1 < Distance < threshold_2 \quad (8)$$

then we consider this point to be part of a moving object. The dynamic object detection result is shown in Fig. 7, where there are many points on the two moving bicycles. However, there are still some result points (in the red rectangle) belong to the static environment, and their optical flow direction is similar to those among the inliers.

Fig. 7: The outlier optical flows

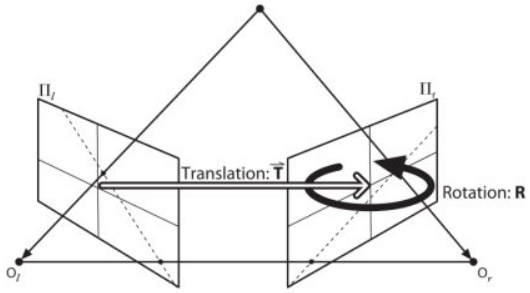


Fig. 8: The fundamental matrix model

To address this problem and get more accurate results, we have developed an optimization algorithm. In Fig. 9, the left image shows that the optical flows on the bicycle is different from the ones in the surrounding environment. For the moving object, its moving pattern is different from the surrounding because the static objects' moving pattern is decided by the camera movement. In the second part we assume all the inliers are static objects. From the right image, there are few optical flows in the scope of the bicycle. For each optical flow we get from the third step, we compare its direction with the direction of the optical flow of its nearest inlier. If the direction difference is large than some threshold, we consider this feature point to be part of a moving object.

After processing the optimization algorithm, we get a very promising result in Fig. 10. Note that unfortunately due to time constraints we were not able to implement our preprocessing methods for the stereo variant of ORB-SLAM2, so the KITTI dataset that the above figures originate from is not used for evaluation.

IV. Fast Dynamic ORB_SLAM

In order to extract dynamic points from the ORB features generated by ORB_SLAM, a significant amount of

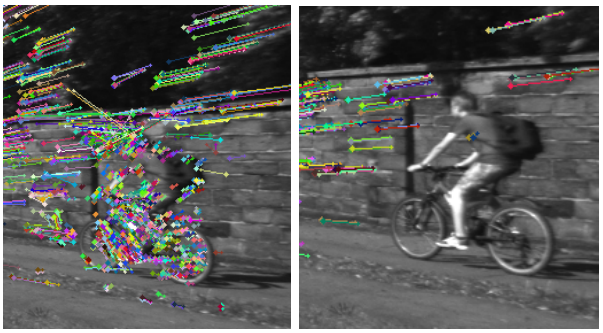


Fig. 9: The left image shows the optical flows in the bicycle area, the right image shows the inlier optical flows in the bicycle area.

Fig. 10: the image shows the final result of the moving object extraction

effort was invested into determining how to do this. The simplest solution that was found was to first calculate the pixel locations of the ORB features, filter them using our method as described above, and then pipe them back into the rest of the algorithm to calculate their descriptors. ORB_SLAM2, while being very impressive, is rather hard to navigate and understand, which made integration of our dynamic feature point segmentation algorithm quite difficult.

The overall algorithm used to segment the dynamic points is very similar to what was described above, but with an additional processing step: A statistical method was used to determine outliers in the optical flow magnitude. When indoors, all of the points that are in the background tend to have a similar optical flow when the camera is moved, as they all move as the camera moves. However, dynamic objects will often have a different magnitude of flow, as they are moving independently of the background. Note that this method works particularly well for the TUM RGB-D dataset, as it was captured indoors, resulting in a relatively shallow depth of view. If this were implemented on an outdoors dataset such as the KITTI dataset, this method would likely be less successful.

Once optical flow of the ORB feature points is calculated, we calculate the magnitude of the flows, and then find the mean and standard deviation of the magnitude. All flows outside of the 2-sigma boundary are removed, as they are likely to not be representative of the background. Additionally, if the standard deviation

is below a threshold, the requirement for becoming an outlier is lowered to a simple difference from the mean. In other words. given a low standard deviation, any flows that are more than a certain amount in magnitude off of the mean magnitude are considered outliers.

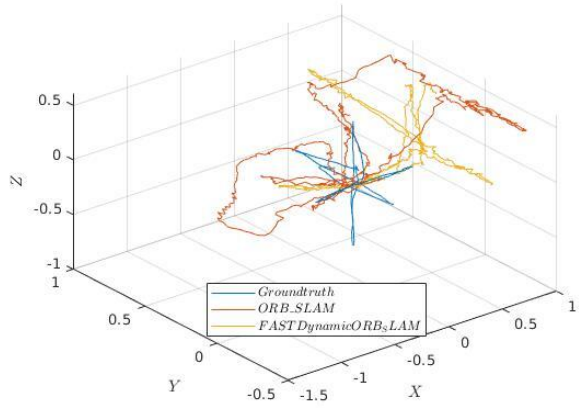


Fig. 11: FAST Dynamic ORB_SLAM trajectory comparison on TUM Dynamic Objects dataset `rgbd_dataset_freiburg3_walking_xyz`

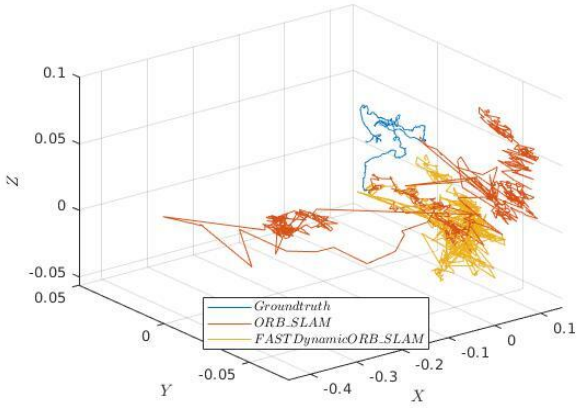


Fig. 12: FAST Dynamic ORB_SLAM trajectory comparison on TUM Dynamic Objects dataset `rgbd_dataset_freiburg3_walking_static`

Fig. 11 and 12 show results of FAST Dynamic ORB_SLAM on the TUM RGB-D dataset. While both are not extremely close to the ground truth trajectory, it is still a reasonable improvement over the original ORB_SLAM. For `freiburg3_walking_xyz`, the absolute trajectory root mean squared error (ATRMSE) for FAST dynamic ORB_SLAM is 0.418m compared to ORB_SLAM's 0.659m. For `freiburg3_walking_static`, the ATRMSE error for FAST dynamic ORB_SLAM is 0.081m compared to ORB_SLAM's 0.391m. In both cases, the improvement is significant, but also less than desired.

FAST dynamic ORB_SLAM runs at approximately 15 Hz on the laptop i5-7300HQ processor, compared to approximately 30 Hz for ORB_SLAM2. While FAST dynamic ORB_SLAM is half as fast as the original ORB_SLAM2 due to preprocessing, with further optimizations to the code as well as a stronger CPU, real-time execution speeds can be achieved. Additionally, this does not require a GPU, which is a notable distinction from other methods.

V. YOLO Dynamic ORB_ SLAM

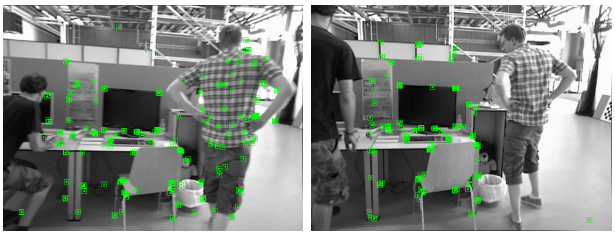
We replaced Mask R-CNN segmentation with YOLOv3 for faster processing speed and lighter computational cost. YOLO detects objects with a bounding box and we simply considered the overall box as a mask. As a result, it may lose some information inside the bounding box compared with DynaSLAM, which did pixel-wise segmentation so it can precisely construct a contour for the object. But the lightness and accuracy are always worth exploring their balance for this complex SLAM task.

Fig. 13 shows the result of YOLO Dynamic ORB_SLAM. In Fig. 13a, great amount of ORB feature points are extracted on the body of the moving person. The left and right movement of the dynamic objects results in the errors when reconstructing the map and relocalization. As shown in Fig. 13c, the trajectory in horizontal direction is deviated to left of the middle vertical line. With YOLO object detection and masks, Fig. 13b shows a good result of moving all the feature points on the moving person. Meanwhile, the camera trajectory shown in Fig. 13d will not be affected by the horizontally moving dynamic object. The final trajectories of ground truth, ORB_SLAM and YOLO Dynamic ORB_SLAM are shown in Fig. 14. Actually, the result is not as perfect as DynaSLAM, but still better than ORB_SLAM without dynamic object removal. Due to the computation resources limit, our experiment is run on personal computer CPU, Intel® Core™ i7-7700 Processor, getting speed about 0.5s per frame, which is approximate to the overall computation time of DynaSLAM. It can be positively assumed that the YOLO method can reach below 0.01s performance with certain GPU.

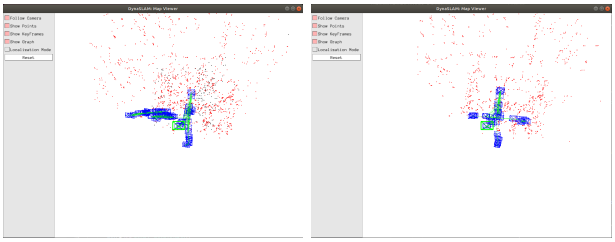
VI. Discussion

A. Moving Object Detection

Although the moving object detection works decently on some data, it still has some limitations. The first one is that in moving object detection, we employ the stereo vision model to process two consecutive frames to get



(a) ORB detection without masks (b) ORB detection with masks



(c) ORB_SLAM with only geometrical approach (d) ORB_SLAM with YOLOv3 dynamic object detection

Fig. 13: Comparison of ORB_SLAM and YOLO Dynamic ORB_SLAM

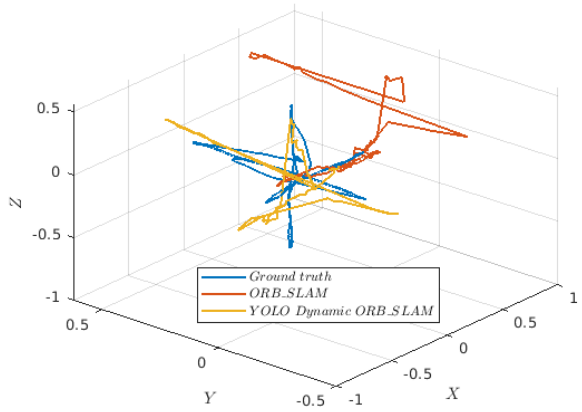


Fig. 14: YOLO Dynamic ORB_SLAM trajectory comparison on TUM Dynamic Objects dataset `rgbd_dataset_freiburg3_walking_xyz`

the points on the dynamic objects. This approximation may actually violate the geometric constraints of the stereo vision system, because the two frames are not the photos that are taken from the same object at the same time and from two fixed angles. But within some tolerance, this model works pretty well when we want to estimate the relative information between two frames with close time stamps. The second limitation is that all the possible detected dynamic points have to be among the ORB feature points we detect in the first place. ORB feature points are quite accurate in capturing the critical descriptive points of the objects in the image, but for real-time implementation we will have to limit the number of the feature points for the ORB solver. This will cause the loss of some important information in the image. In other words, we may not have enough feature points to fully describe a moving object or we could miss the chance of detecting a whole subject that is dynamic! In addition, there are limitations on the dataset we are to detect. If the ORB feature points are too clustered, the optical flow would be unpredictable and thus the method to differentiate from the angular difference would not work.

B. FAST Dynamic ORB_SLAM

FAST Dynamic ORB_SLAM does improve upon ORB_SLAM's performance when concerning scenes with dynamic objects, but overall it does not have the ability to match the performance of methods that use some form of deep learning. This is because without semantic segmentation, there is no way to know if there are potentially dynamic objects in the camera frame that are currently static. If such objects are static, the feature points belonging to them will become added to the internal map of ORB_SLAM, resulting in erroneous map points that could result in poor feature matching later on. Even with theoretical perfect performance of the FAST Dynamic ORB_SLAM algorithm, there is no way of overcoming this problem without some form of deep neural network.

Additionally, our FAST Dynamic ORB_SLAM suffers in a few cases: 1) If there is a significant amount of large and fast of rotations, both the performance of fundamental matrix estimation and calculation of epipolar constraints suffer significantly, often resulting in tracking loss for such sequences. 2) Very slow moving dynamic objects often do not differ significantly enough to be considered outliers, but still impact the performance of the tracking rather significantly. 3) The estimation of the fundamental matrix operates under the assumption that the correct fundamental matrix will have the greatest number of features associated with it. However, if a well-featured dynamic object takes up a majority of the image, the fundamental matrix will be calculated to fit the dynamic object instead of the overall scene.

Case 1 is potentially solvable with a more powerful algorithm, but making robust fundamental matrix calculation with deliberate outliers taking up a reasonable amount of the feature points. Case 2 is most likely difficult to solve, as all algorithms have some minimum threshold to be considered an inlier *v.s.* an outlier, and having such low threshold that very slow moving objects are seen as outliers is likely to cause degradation in performance, as many inliers would be removed as well. Case 3 is more or less unsolvable without a neural network, because the only way to solve it is to have *a*

priori knowledge that a majority of feature points belong to a dynamic object.

C. YOLO Dynamic ORB_SLAM

YOLO Dynamic ORB_SLAM speeds up the original DynaSLAM with relatively good accuracy, but there still exists a few main drawbacks: 1) It is not applicable for monocular or stereo cameras. When it comes to the cases in which there are few dynamic objects, the monocular or stereo cameras Dynamic ORB_SLAM could perform even a little worse than ORB_SLAM, such as on some KITTI scenes. 2) Applying YOLOv3 instead of Mask R-CNN is the tradeoff between accuracy and speed. YOLO Dynamic ORB_SLAM is promised to run in real-time, while the pixel-wise segmentation of DynaSLAM assures a better life-long map built offline. 3) YOLO Dynamic ORB_SLAM is based on a 53 convolutional layer CNN, hence it is still computationally expensive and needs a GPU to support the algorithm in real time.

VII. Conclusions

Overall, we conclude that while a robust non-deep learning based solution for solving the issue of dynamic objects in SLAM is appealing, it is most likely not feasible, given the fact that there is no way to properly segment features belonging to dynamic objects that are currently static without *a priori* knowledge, which is generally only obtainable using a deep learning solution. However, there are cases where a non-deep learning solution can help improve the overall tracked trajectory, although the performance will likely never match that of methods using deep learning. Additionally, using YOLOv3 in place of Mask R-CNN is a potential solution to running DynaSLAM in real time, the accuracy tradeoff required is likely not acceptable. Given the development speed of deep learning, most likely there will be a semantic segmentation algorithm that runs in real time with similar accuracy to Mask R-CNN in the near future.

Acknowledgment

We would like to thank Prof. Maani, the GSIs and the graders for a great learning experience despite the difficult circumstances presented to us all.

References

[1] C. Debeunne and D. Vivet, "A review of visual-lidar fusion based simultaneous localization and mapping," *Sensors*, vol. 20, no. 7, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/7/2068

[2] R. Mur-Artal and J. D. Tardós, "Visual-inertial monocular slam with map reuse," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 796–803, 2017.

[3] ——, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[4] A. Li, X. Ruan, J. Huang, X. Zhu, and F. Wang, "Review of vision-based simultaneous localization and mapping," in *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, 2019, pp. 117–123.

[5] M. Filipenko and I. Afanasyev, "Comparison of various slam systems for mobile robot in an indoor environment," in *2018 International Conference on Intelligent Systems (IS)*, 2018, pp. 400–407.

[6] A. Singandhupe and H. M. La, "A review of slam techniques and security in autonomous driving," in *2019 Third IEEE International Conference on Robotic Computing (IRC)*, 2019, pp. 602–607.

[7] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "Orb-slam: A versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[8] D. Zou and P. Tan, "Coslam: Collaborative visual slam in dynamic environments," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 2, pp. 354–366, 2013.

[9] Wei Tan, Haomin Liu, Z. Dong, G. Zhang, and H. Bao, "Robust monocular slam in dynamic environments," in *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2013, pp. 209–218.

[10] B. Bescos, J. M. Fácil, J. Civera, and J. Neira, "Dynaslam: Tracking, mapping, and inpainting in dynamic scenes," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 4076–4083, 2018.

[11] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International Conference on Computer Vision*, 2011, pp. 2564–2571.

[12] E. Meinhardt-Llopis, J. S. Pérez, and D. Kondermann, "Horn-schunck optical flow with a multi-scale strategy," *Image Processing on line*, vol. 2013, pp. 151–172, 2013.

[13] J.-Y. Bouguet *et al.*, "Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm," *Intel Corporation*, vol. 5, no. 1-10, p. 4, 2001.

[14] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[15] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2015.

[16] C. Yu, Z. Liu, X.-J. Liu, F. Xie, Y. Yang, Q. Wei, and Q. Fei, "Ds-slam: A semantic visual slam towards dynamic environments," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1168–1174.

[17] D. Nguyen, C. Hughes, and J. Horgan, "Optical flow-based moving-static separation in driving assistance systems," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, 2015, pp. 1644–1651.