Monocular SLAM for Real-Time Applications on Mobile Platforms

Mohit Shridhar
mohits@stanford.edu

Kai-Yuan Neo
kneo@stanford.edu

Jun 7, 2015

Abstract

The current state-of-the-art in monocular visual SLAM comprises of 2 systems: Large-Scale Direct Monocular SLAM (LSD-SLAM), and Oriented FAST and Rotated BRIEF SLAM (ORB-SLAM). This paper experiments with an adapted version of ORB-SLAM, and is also inspired by other groundbreaking SLAM implementations including LSD-SLAM. It outlines a lean pipeline for development of a monocular SLAM system on a mobile device, and highlights the key insights and results of developing such a system.

1 Introduction

Simultaneous Localization and Mapping (SLAM) is a hot field of research in computer vision due to its significant applications to next-generation technologies in robotics and augmented reality. Robots need to know where they are in relation to their environments to mobilize effectively, and AR devices need perform SLAM in order to augment existing scenes with virtual objects.

The publication of Parallel Tracking and Mapping (PTAM)[7] in 2007 was groundbreaking in visual SLAM (vSLAM) research, outlining a means for SLAM to become generalizable to ordinary environments and tasks with a single handheld camera. Since then, researchers have worked extensively on applications of similar technology, culminating in the pinnacle of visual SLAM (vSLAM) to date with LSD-SLAM[14] and ORB-SLAM[16], both published within 2014-15.

Empirical studies show that LSD-SLAM, which relies on depth filters, performs consistently more powerfully than ORB-SLAM, which relies on ORB-feature mapping. Implementing LSD-SLAM would be infeasible for the timeline of this project, so we opt to implement ORB-SLAM which still affords many valuable insights. This project seeks to clarify and substantiate some of the possibilities and limitations involved with developing a monocular SLAM system based on ORB-SLAM. Through development of this project, we have built a minimal solution of the ORB-SLAM concept on a mobile device. It is currently capable of estimating camera pose and triangulating point clouds within indoor scenes in real time on a tablet device, but has yet to match the accuracy of more robust implementations, primarily due to its lack of bundle adjustment or loop closure techniques.

2 Previous Work

Parallel Tracking and Mapping for Small AR Workspaces (PTAM)[7] proposes a simple, efficient, and low-entry-level solution to the SLAM problem. Rather than relying on color descriptors, markers placed at specific locations within the environment, and existing CAD models of the environment, PTAM proposes using FAST corners that are quick to detect and relatively cheap to process. PTAM also proposes detecting camera pose and mapping out the environment in parallel, reducing what used to be expensive batch operations into real-time operation. The strengths of PTAM are also its limitations: It is a relatively simple system, so it doesn't extract structural meaning from the reconstructed environment other than the points detected by the FAST corners. It is also limited by the need to perform stereo initialization in a very specific way, requiring the user to be trained.

SVO: Fast Semi-Direct Monocular Visual Odometry (SVO)[15] presents a direct color-intensity-based scheme to map the motion of an object in an environment. It focuses on providing robust performance for aerial vehicles in outdoor environments, and uses probabilistic models to build semi-dense depth maps. The advantages of this method are that it saves a lot of time by not computing features to perform motion estimation, and it is relatively robust to the shape of the environment. The disadvantages of this method are that it requires a large dataset to determine parameters for its probabilistic model, does not consider a global map, and is highly sensitive to lighting situations.

LSD-SLAM[14] proposes depth-based method similar to SVO for performing vSLAM. In particular, it converts frames to depth maps, from which it can overlap depth maps corresponding to consecutive frames in order to accurately map out the environment. The main advantage of LSD-SLAM over feature-based methods is that it is able to generate a more complete reconstruction of the scene that contains textures that may not have been detected by feature-based methods, for example smooth surfaces. These reconstructions can then be used to build interactive meshes for AR applications. Its limitations include the relative complexity of constructing depth-based probabilistic models, and the non-convexity of direct image alignment. As a result of its fully-featured SLAM implementation, it also runs slower than SVO.

ORB-SLAM[16] is a very recent paper in SLAM and is yet to be published. ORB features have been used extensively since their inception in 2011[11] and are relatively efficient to extract and match. Combining ORB features with the PTAM skeleton, feature-based vSLAM becomes much more robust and relatively simple to implement. Combining this with other developments in SLAM techniques since 2007, such as flexible planar or non-planar stereo initialization and more advanced bag-of-words loop-closure, ORB-SLAM is one of the most successful feature-based SLAM methods to date.

2.1 Contributions

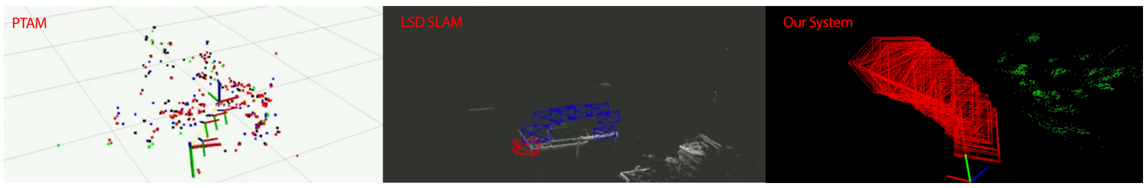

Figure 1: Monocular SLAM systems.

We built a functional SLAM system based on existing methodologies in state-of-the-art solutions. In the process, we took creative liberties that resulted in deviations such as our improved stereo initialization. Our contributions comprise monocular SLAM results on a next-generation mobile platform, and key insights and observations from constructing a SLAM system from scratch. See Github[1] source code for full implementation.

3 Pipeline

Our implementation is a lean version of the pipeline presented in Parallel Tracking and Mapping (PTAM) [7] and Oriented FAST and Rotated BRIEF (ORB) SLAM [16]. Tracking and mapping are executed sequentially. Stereo initialization provides a baseline camera pose and a set of points with 2D-3D correspondences to kick-start tracking. The system then saves the first post-initialization frame as a keyframe. The tracker uses feature matching to estimate the new camera pose with respect to triangulated points in the previous keyframe. After each pose estimation, the system checks the quality of tracking and decides whether to insert a new keyframe or to continue tracking. If the quality of tracking is lower than a threshold, the system triangulates a new set of points and records a new keyframe. Due to time constraints, we did not implement advanced techniques shown to improve accuracy such as relocalization, loop closure, bundle adjustment, and keyframe management. However, we designed the system to be fully scalable for future improvements.

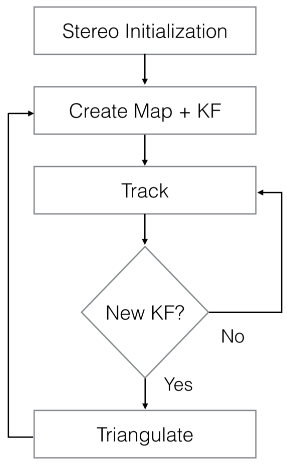

Figure 2: Serial Pipeline

3.1 Feature Matching

We extract ORB features at 8 Gaussian pyramid levels with a 1.2 scale factor between levels. We use these features for triangulating new points and estimating camera poses. We prefer ORB [11] over other popular solutions such as SIFT and SURF because of its robustness and efficiency, making it ideal for real-time mobile applications. Feature matching is performed with brute-force k-Nearest-Neighbors descriptor matching and the ratio test with k = 2.

3.2 Stereo Initialization

Tracking and mapping are inter-dependent processes. A camera pose facilitates triangulation of 3D point coordinates; 3D point coordinates facilitate estimation of a camera pose. Within feature-based methods, we have two approaches for computing the initial camera pose. The first is with a homography H_{cr}, used with planar scenes, i.e. scenes that only involve a planar surface or objects at a large distance away. The second is with a fundamental matrix F_{cr}, which we calculate with a stereo pair of frames, and does not rely on planar scenes. For each match x_c from the current frame to the reference match x_r in the reference frame:

[1] `https://github.com/MohitShridhar/shield_slam`

$$x_c = H_{cr} x_r \tag{1a}$$

$$x_c = F_{cr} x_r \tag{1b}$$

Using the pinhole projection model, we can compute the essential matrix $E_{cr} = K^T F_{cr} K$ where K is the intrinsic camera matrix. Most often, we don't know if the scene is planar, and this ambiguity can lead to poor initialization. If the objects in the scene have limited motion parallax, then F_{cr} produces erroneous transformations. The same is true for H_{cr} if objects have significant motion parallax. Traditionally, systems like PTAM rely on a fixed model (either F_{cr} or H_{cr}) for this step. They also require the user to move the camera in a specific pattern before computing E_{cr} using the 5-point algorithm [5]. R. Mur-Artal et al. [16] propose an elegant solution, which automatically chooses the right transformation model. This involves computing both H_{cr} and F_{cr} simultaneously and using an error-checking mechanism to find the best fitting model. For each matched key point between the current frame and the reference frame, we compute a score s_M:

$$s_M = \rho_M(d^2_{cr,M}(x_c, x_r)) + \rho_M(d^2_{rc,M}(x_c, x_r))$$
$$\rho_M(d^2) = \begin{cases} \Gamma - d^2 & d^2 < T_M \\ 0 & d^2 \geq T_M \end{cases}$$

d^2_{cr} and d^2_{rc} are the symmetric error transfers, T_M is the outlier rejection threshold, and Γ is the error score constant. Based on statistical analysis, the optimal parameter values are $\Gamma = 5.99$ for homography and $\Gamma = 3.84$ for fundamental matrix [16]. Individual keypoint error scores sum to $S_{M\{H,F\}} = \sum_{i=1}^{n} s^i_{M\{H,F\}}$. We then normalize the two model scores to R_H:

$$R_H = \frac{S_H}{S_H + S_F} \tag{2}$$

If $R_H > 0.45$ we use the homography to perform the next pose estimation. Otherwise, we use the fundamental matrix. H_{cr} is given a slight preference over F_{cr} as ORB is more confident about feature matching with planar scenes[11]. Once we have chosen the appropriate model we need to resolve pose ambiguities. A homography transformation generates 8 motion [2] hypotheses and fundamental matrix generates 4 [4]. To disambiguate, we calculate a reprojection inlier score R_S using n 2D-3D correspondences:

$$R_S = \sum_{i=1}^{n} \epsilon(x^i_c, x^i_r)$$
$$\epsilon(x_c, x_r) = \begin{cases} 1 & d(x_c, \hat{x_c})^2 + d(x_r, \hat{x_r})^2 < T_{Proj} \\ 0 & d(x_c, \hat{x_c})^2 + d(x_r, \hat{x_r})^2 \geq T_{Proj} \end{cases}$$

where $d(x_c, \hat{x_c})$ and $d(x_r, \hat{x_r})$ are reprojection errors computed using the corresponding projection matrices. From empirical analysis, the optimal parameter value of $T_{proj} = 5.991$. Finally, we normalize the solution with the highest reprojection score $R_{S,max}$:

$$R_{S,norm} = \frac{R_{S,max}}{\sum_{i=1}^{4,8} R_{S,i}}$$

If and only if $R_{S,norm} > 0.7$, i.e. one of the 4 or 8 solutions contains 70% of the total inliers found, we record a new keyframe and proceed to tracking.

3.3 Keyframes

Keyframes are the fundamental data structures of our SLAM system. The latest keyframe is used as the primary reference for tracking and triangulation. Each keyframe K_i stores:

- 2D image points with corresponding 3D object points
- ORB feature descriptors extracted from the keyframe image
- Estimated world camera pose $P_{cw} = [R_{cw}|t_{cw}]$

The system also maintains a complementary global map of 2D-3D correspondences through aggregated keyframe data for visualization. This data is also useful for bundle adjustment to make correspondences more mutually consistent, but we have yet to implement this feature.

3.4 Tracking

Once we have a set of known 2D-3D correspondences we can use the Perspective-n-Point (PnP) algorithm [13] to estimate the new camera pose. For each input frame, we compute matches to the reference keyframe. These matched image points are then paired with the corresponding 3D points from the reference keyframe to compute a pose using Levenberg-Marquardt's iterative non-linear optimization:

$$f(p + \delta_P) \approx f(p) + J\delta_P$$

J denotes the Jacobian matrix $\frac{\partial f(p)}{\partial p}$. PnP inadvertently relies on the quality of the feature matching to reliably estimate the camera pose, hence it is crucial that the reference keyframe contains most of the keypoints in the current frame. In our implementation, we experimented with multiple variants of the PnP algorithm: EPnP [8], supposedly more efficient and scale consistent, and P3P [6], a non-iterative pose estimation method. Based on early results, iterative Levenberg-Marquardt produces the most accurate results, though it takes longer to compute poses than EPnP or P3P.

3.5 KF Insertion

Fewer and fewer triangulated points are visible in the camera frustrum as it moves through space. To maintain a sufficient number of triangulated points in the camera's field of view and sustain the tracking-mapping cycle, the system needs to actively triangulate new 2D-3D correspondences with updated world camera poses P_{cw}. New keyframes are inserted if the following conditions are met:

1. 20 frames have passed since the previous keyframe insertion
2. Fewer than 50 keypoints tracked in the current frame
3. Fewer than 70% of the KF keypoints tracked in the current frame

3.6 Triangulation

Given the relative translation and rotation between two camera poses, we can use epipolar geometry to estimate the 3D coordinates of two corresponding 2D image points in a stereo image-pair. Let $U = w(u, v, 1)^{\mathcal{T}}$ denote an observed image point in homogeneous coordinate space, where w is an unknown scale factor. Let $X = (x, y, z, 1)^{\mathcal{T}}$ denote the corresponding object point and $p_i^{\mathcal{T}}$ be the i^{th} row of the transformation matrix $[R_{1,2}|t_{1,2}]$. From [3] we derive 2 equalities:

$$up_3^{\mathcal{T}}X = p_1^{\mathcal{T}}X$$
$$vp_3^{\mathcal{T}}X = p_2^{\mathcal{T}}X$$

Now we obtain 4 linear equations in the coordinates of X, and express it in the form $AX = 0$, where A is a 4×4 matrix. Then we solve X by computing the singular value decomposition [1]. Each triangulated point undergoes a reprojection inlier test similar to the one carried out during stereo initialization.

4 Results

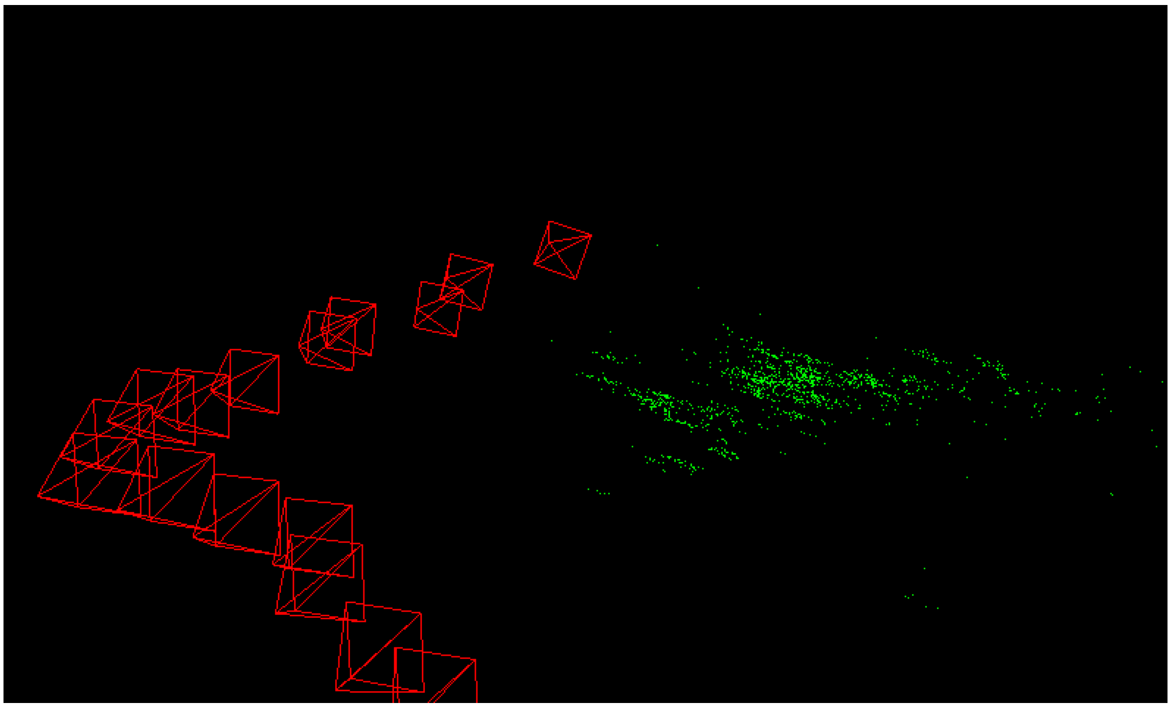

Figure 3: TUM Household Dataset: PCL Visualization

We tested our system on TUM datasets [12] to perform quantitative analysis on tracking accuracy. Although initial trajectory estimates are somewhat similar to that of the scaled ground truths, the tracker quickly deviates with discernible error propagation (see Figure 4). Figure 5 shows the

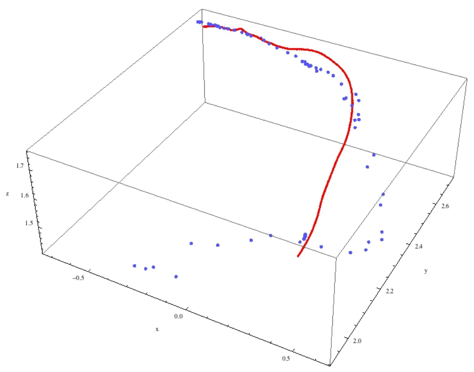
Figure 4: Ground Truth vs Estimated Trajectory : TUM Dataset

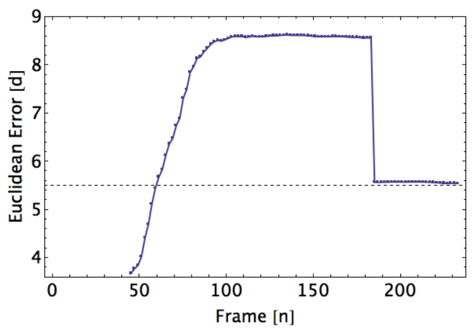

Figure 5: Euclidean Distance between Ground Truth and Estimated Trajectory

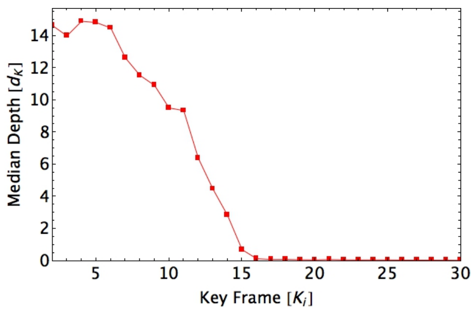

Figure 6: Median Depth of keyframes

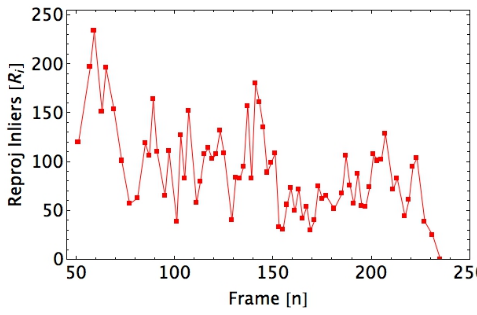

Figure 7: Number of tracked Reprojection Inliers

increase in Euclidean distance between the estimated camera position and the ground truth. Due to time constraints, we did not manage to resolve this issue. However, in the next section we discuss potential reasons and solutions.

We used the Point Cloud Library (PCL) for runtime visualization and debugging. Figure 8 is a snapshot of the estimated frustrum trajectory with a global map of triangulated points. See video clip[2] for the full system in action and our NVIDIA Tegra Shield Android implementation.

5 Experiments

The following experiments were conducted to gain a deeper understanding of the issues mediated through the pipeline.

5.1 Scale Decay

Figure 6 shows an apparent decay in the scale of the triangulated points during keyframe insertion. Just after 15 insertions, the median depth of the local point cloud has decayed down to 0. This causes the triangulated points in the keyframe to cluster thus rendering the reprojection inlier test useless. Figure 7 indicates that a sufficient number of 2D-3D correspondences are being tracked even though most of the 3D points have zero depth. This could potentially be a bug in the implementation. But moreover, it highlights one of the inherent flaws of the pipeline: tight coupling of tracking and

[2]`https://youtu.be/5oBg3Giad6A`

mapping. A small error in the pose estimate leads to inaccurate triangulation, and small errors in the triangulation lead to incorrect pose estimates. These errors then compounded over time to produce the erroneous trajectories seen in Figure 4. Furthermore, our pipeline doesn't enforce a consistent scale between frames to account for scale drift [9].

5.2 Performance

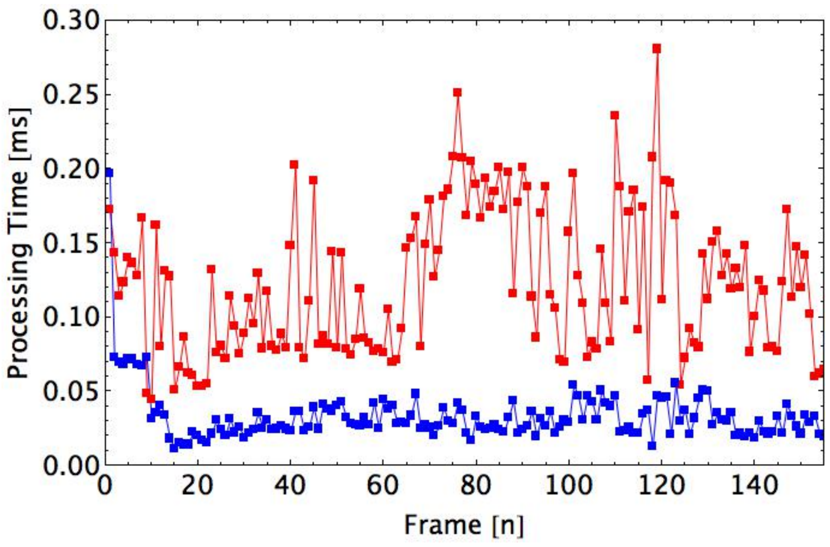

Figure 8: Time taken to process each frame.
Android NVIDIA Tegra Shield vs. PC Intel Core i7 2.6GHz

Our system performs reasonably well on the NVIDIA Tegra Shield. Even with a mostly CPU-based implementation, the application runs at an average rate of 8.16 FPS on mobile and 30.2 FPS on PC. This might be unacceptable for AR use-cases, but definitely applicable for low-speed robotics. Since the Shield platform supports CUDA, a quick performance hack could be to re-implement the feature extraction, feature matching, PnP pose estimation etc. on the GPU. Furthermore, ARM NEON optimizations could also help boost the performance to acceptable standards for augmented reality.

5.3 Analysis

During testing, we encountered several problems that our system has yet to address. These include problems with pure rotation, where it is impossible for the system to triangulate new object points from frames that deviate only by an angle from a fixed camera position[10]. Other problems include keyframe management, where we observed that our memory usage gradually increases with each additional keyframe without any limitations, which is ultimately unscalable.

We have much future work to accomplish for this project, but our primary goal is to produce a baseline implementation that is as simple and efficient as possible, and is portable across all mobile platforms. As such, we need the following list of improvements to make this a practical and fully-functionally SLAM system:

1. **Loop closure** will allow us to correct for pose and scale drift, and prune recurring keyframes by detecting when the camera has reached a previously-reached position in the environment. When such an event occurs, we can "close the loop" by modifying the measured camera pose and feature point drift to be consistent with that obtained from the start of the loop. After loop closure, we will also no longer need to store keyframes that the system recorded in between the start and end of the loop, which may significantly reduce our memory usage in common cases where the camera consistently patrols the same area.

   One simple method for implementing this would be to keep a bag of words for each keyframe, and at registration of each new keyframe, compare the bag of words of the new keyframe to that of each previous keyframe. If a sufficient similarity threshold is reached, we could consider this a loop closure, and recompute the camera position to minimize drift.

2. **Relocalization** is a technique that will help our system recover from a lost state, something that our system currently has no method of dealing with. The tracker is deemed to be lost when the system finds insufficient point correspondences between the previous keyframe and the current frame, or the current frame doesn't contain enough features to generate a new keyframe. To mitigate this problem, we could implement a bag of words method that does the following: Generate and store bag of words features for each keyframe, and compare each frame during the lost phase to all previous keyframes. If a similarity threshold is reached, close the loop and restart tracking at the matched keyframe.

3. **Keyframe management** is a significant part of practical SLAM implementations because robots and vehicles may need to traverse massive amounts of space, and hence will need to be memory efficient in order to map typically large environments. One way of accomplishing this is to perform keyframe management to curb the memory demands imposed by the need to store keyframes. One solution could be to discard keyframes after a certain number of frames have passed, eliminating the possibility of loop closure, but also making sure that memory usage remains finite. Another optimization would be to remove keyframes that are deemed above a certain similarity threshold over their bag of words representations.

Should we decide to augment this product's functionality further, we could try implementing popular optimization techniques such as bundle adjustment, which simultaneously refines the triangulated point cloud and camera pose estimates. Another direction would be to convert this system from feature-based tracking to depth-filter-based tracking, which could potentially decrease the time spent on feature-matching computation and produce more dense reconstructions for better accuracy.

6 Conclusion

Based on the foundations laid by PTAM and ORB-SLAM, we have presented a lean vSLAM implementation that performs basic 6DOF tracking on a mobile device with a single camera. This simple implementation highlights the core necessities of a SLAM system, which include stereo initialization, tracking, keyframe generation, and relocalization. Based on our tests with the TUM ground-truth datasets, our system achieves a reasonable amount of accuracy for the first few frames. Although it is not deployable as of yet, it still demonstrates viability in terms of capability and system architecture. SLAM systems need to cater to a wide range of environments and conditions, thus the complexity of the optimization problem is massive. Designing a universal system that is useful for most applications remains an important and unsolved problem.

References

[1] Gene H Golub and Christian Reinsch. “Singular value decomposition and least squares solutions”. In: *Numerische Mathematik* 14.5 (1970), pp. 403–420.

[2] Olivier D Faugeras and Francis Lustman. “Motion and structure from motion in a piecewise planar environment”. In: *International Journal of Pattern Recognition and Artificial Intelligence* 2.03 (1988), pp. 485–508.

[3] Richard I Hartley and Peter Sturm. “Triangulation”. In: *Computer vision and image understanding* 68.2 (1997), pp. 146–157.

[4] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision.* 2nd ed. New York, NY, USA: Cambridge University Press, 2003. ISBN: 0521540518.

[5] Hongdong Li and Richard Hartley. “Five-point motion estimation made easy”. In: *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on.* Vol. 1. IEEE. 2006, pp. 630–633.

[6] Yingming Hao et al. “Robust analysis of P3P pose estimation”. In: *Robotics and Biomimetics, 2007. ROBIO 2007. IEEE International Conference on.* Dec. 2007, pp. 222–226. DOI: `10.1109/ROBIO.2007.4522164`.

[7] Georg Klein and David Murray. “Parallel Tracking and Mapping for Small AR Workspaces”. In: *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR’07).* Nara, Japan, Nov. 2007.

[8] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. “Epnp: An accurate o (n) solution to the pnp problem”. In: *International journal of computer vision* 81.2 (2009), pp. 155–166.

[9] Hauke Strasdat, JMM Montiel, and Andrew J Davison. “Scale Drift-Aware Large Scale Monocular SLAM.” In: *Robotics: Science and Systems.* Vol. 2. 3. 2010, p. 5.

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

[11] Ethan Rublee et al. “ORB: an efficient alternative to SIFT or SURF”. In: *Computer Vision (ICCV), 2011 IEEE International Conference on.* IEEE. 2011, pp. 2564–2571.

[12] J. Sturm et al. “A Benchmark for the Evaluation of RGB-D SLAM Systems”. In: *Proc. of the International Conference on Intelligent Robot Systems (IROS).* Oct. 2012.

[13] Yinqiang Zheng et al. “Revisiting the pnp problem: A fast, general and optimal solution”. In: *Computer Vision (ICCV), 2013 IEEE International Conference on.* IEEE. 2013, pp. 2344–2351.

[14] Jakob Engel, Thomas Schöps, and Daniel Cremers. “LSD-SLAM: Large-scale direct monocular SLAM”. In: *Computer Vision–ECCV 2014.* Springer, 2014, pp. 834–849.

[15] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. “SVO: Fast semi-direct monocular visual odometry”. In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on.* IEEE. 2014, pp. 15–22.

[16] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. “ORB-SLAM: a Versatile and Accurate Monocular SLAM System”. In: *CoRR* abs/1502.00956 (2015). URL: `http://arxiv.org/abs/1502.00956`.