TECHNICAL UNIVERSITY OF MUNICH
DEPARTMENT OF INFORMATICS

GUIDED RESEARCH PROJECT

LOCAL AND GLOBAL MAPPING FOR DIRECT SLAM

Erkam Uyanik

Advisor: Nikolaus Demmel
Examiner: Prof. Daniel Cremers

06.11.2020

Abstract

In this guided research project, we present an extension of Direct Sparse Mapping (DSM) [1] to a direct SLAM system with global mapping and loop closure by introducing loop closure detection and pose graph optimization. We modify the point selection procedure which does not take repeatability into account and introduce some repeatable corner features. This helps us to detect loop closure candidates using a feature based bag of words (BoW) technique without deteriorating the performance of DSM. We verify loop closure candidates and estimate Sim(3) relative pose constraints by minimizing geometric errors. We apply a pose graph optimization using relative pose constraints from verified loop closures and covisibility graph. On the EuRoC dataset, we show that the modified point selection procedure preserves performance and loop closure corrections for sequences with detected loops give an 8.86% improvement compared to DSM.

Introduction

Localization of an agent given visual sensor inputs has been an important problem for computer vision and robotics with proposed solutions for visual odometry (VO) and simultaneous localization and mapping (SLAM) [2]. Methods can be categorized as direct and indirect based on whether they use raw sensor measurements or not. Indirect methods use extracted features from images to optimize a geometric error where usually correspondences of distinctive corner points are utilized. Direct methods, on the other hand, use raw pixel values and thus optimize a photometric error. While indirect methods were more popular in the past owing to their speed and robustness to distortions, they suffer in scenes without rich texture and enough distinctive corners. Direct methods do not depend on characteristic features and thus are more robust in such scenes. Direct methods have gained popularity recently partly because of increased computational power and better visual sensors which provide well calibrated data with less geometric distortion. Not losing information due to a feature extraction step also provides room for direct methods to utilize underlying information better.

Direct Sparse Odometry (DSO) [3] is a direct visual odometry system that jointly optimizes motion and structure using photometric bundle adjustment (PBA). DSO combines advantages of direct approaches with sparse data. By preferring to use sparse data, DSO eliminates the computational infeasibility that would be introduced by a geometry prior in a dense approach. It also benefits from a more precise sensor model with photometric calibration. DSO was one of the early direct applications that showed direct methods can be used in

real-time. As a visual odometry method, DSO only predicts the trajectory incrementally in a sliding window. By marginalizing frames and points that leave the local window, it forgets previously visited areas and cannot use the rich information from revisited scenes. This results in accumulated drift in global translation, rotation, and scale which prevents successful applications to long trajectories.

Detecting revisited scenes and updating the trajectory to better reflect correspondences between frames helps reduce accumulated drift in VO systems. Direct Sparse Odometry with Loop Closure (LDSO) [4] extends DSO to a visual SLAM system with loop closure detection and pose graph optimization as a correction technique. While preserving robustness of DSO, LDSO achieves better performance by significantly reducing accumulated drift. This is partly achieved by increasing repeatability of map points and computing distinctive descriptors for a small subset of map points. Loop closure detection and verification, relative pose computation, as well as pose graph optimization steps in LDSO are the same as in a typical indirect SLAM system.

Although LDSO detects and corrects loop closures in predicted trajectory, existing map points are not reused to reflect reobservations and thus reobservations are ignored in the sliding window odometry optimization. Instead of keeping a temporary map and marginalizing old keyframes, retaining keyframes and maintaining a global map allows to detect reobservations and also to reuse existing map points, and thus prevents duplication of points. Direct Sparse Mapping (DSM) [1] is a visual SLAM system with a persistent map based on photometric bundle adjustment. DSM proposes a local map covisibility window criteria to select active keyframes from both temporally close frames and frames that are covisible with the temporal keyframes. While DSM uses main ideas of indirect visual SLAM techniques, it lacks loop closure correction. It is also stated in [1] that there is room for improvement regarding computation speed of DSM by better parallelizing operations and utilizing processor level instructions.

In this guided research project, our goal is to propose a direct SLAM system with global mapping and loop closure. In previous related works, a direct VO, a direct SLAM system with loop closure, and a direct SLAM system with global mapping are proposed. We aim to benefit from these and combine advantages of global mapping and loop closure features and expect to see an increase of accuracy with the addition of loop closure technique compared to DSM. We also aim to achieve a system with real-time or close to real-time capabilities.

Direct Sparse Mapping

In the project, we take DSM as a baseline implementation for a direct SLAM system with global mapping and extend it by adding loop closure. Therefore, it is necessary to explain how DSM works before explaining our main work.

DSM is a visual SLAM system with a persistent map based on local photometric bundle adjustment. The persistent map enables retention of all keyframes and reuse of information in PBA, but since there is no explicit compensation of the drift for larger loops, keyframe reuse is effectively limited to local mapping and smaller loops.

DSM consists of a tracking frontend and an optimization backend. The tracking frontend tracks frames and points and also handles coarse initialization for the optimization. The optimization backend determines which keyframes should be included in the local window and jointly optimizes all active keyframes and map point parameters. In the implementation there are two threads for tracking and mapping. The tracking thread obtains a camera pose for each frame and decides when to add a new keyframe. The mapping thread processes new frames to track points from active keyframes and handles new keyframe creation by recalculating the local window, activating new points, and invoking PBA. The mapping thread also maintains the map locally consistent by removing outliers, detecting occlusions, and avoiding duplicating points.

DSM uses a local map covisibility window (LMCW) which is a combination of temporal and covisibility criteria with respect to the latest keyframe. The temporal part is selected in a similar approach as in DSO where most recent keyframes are chosen. This part is crucial during exploration as new points are initialized and it helps to maintain odometry accuracy. In the covisible window, keyframes that are covisible with those in the temporal part are selected. For keyframes in the covisible part, map points that project into otherwise depleted areas (with respect to active map) in the currently tracked frame are favored to get a more representative active map. LMCW does not take photo-consistency into consideration, it only considers geometry.

In addition, DSM uses a coarse-to-fine optimization scheme which increases the convergence radius of PBA and a robust influence function together with an outlier management strategy based on the t-distribution.

In DSM, feature points are hosted in keyframes. In the following, we describe how new points are first initialized, then become ready for activation when their depth has been refined, and

finally are activated to be optimized in the local PBA. When a frame becomes a keyframe, candidate points are extracted for this frame. With each new keyframe, candidates of active keyframes are tracked in this new keyframe to initialize or refine their depth estimate. Then, "good candidates" (which are initialized successfully, can be projected into the last keyframe, etc.) of active temporal keyframes are refined further by optimization based on all active keyframes. When a candidate is initialized, it has an initial depth estimate, but this depth estimate only becomes reliable after it is successfully optimized. Note that neither can every candidate point be initialized successfully nor every initialized candidate be optimized successfully. After candidates are refined, eligible optimized candidates of active temporal keyframes are activated. When a candidate becomes active, its depth is not updated again. So, after a point is optimized, there is no difference whether it is a candidate point or an active point in terms of depth (before local bundle adjustments). If an active point later becomes an outlier, it is deleted.

Feature point selection

For correcting loops, we first need to detect loop candidates. Visual bag of words methods are commonly used for detecting loops. For this, we need repeatable feature points so that we can get similar frames by comparing those feature points between frames. For indirect methods, this is not a problem since they already extract feature points and compute their descriptors even without a loop closure step. For our case, DSM extracts points based on image gradients without taking repeatability into consideration and these points are not usually distinguishable, repeatable points, e.g. points from weakly textured regions or edges. So, we cannot just compute descriptors of these points to use for detecting loops. Instead, we need to introduce extraction of feature points such as corners. While we cannot use non-feature points for loop closure detection, we can use both feature and non-feature points for the odometry, as the feature point selection mechanism also favors pixels with high gradient. So, what we do is extract feature points as a small percentage of the all extracted points. All points are used by DSM as usual but in addition we compute descriptors for the feature points and use them for detecting loops.

DSM uses a grid approach for selecting the desired number of points in each keyframe. That is, a frame is first divided into cells, then for each cell it tries to select the best possible point. Depending on the number of total selected points compared to the desired number, the window size is heuristically adjusted, i.e. if we get more points than desired the window size

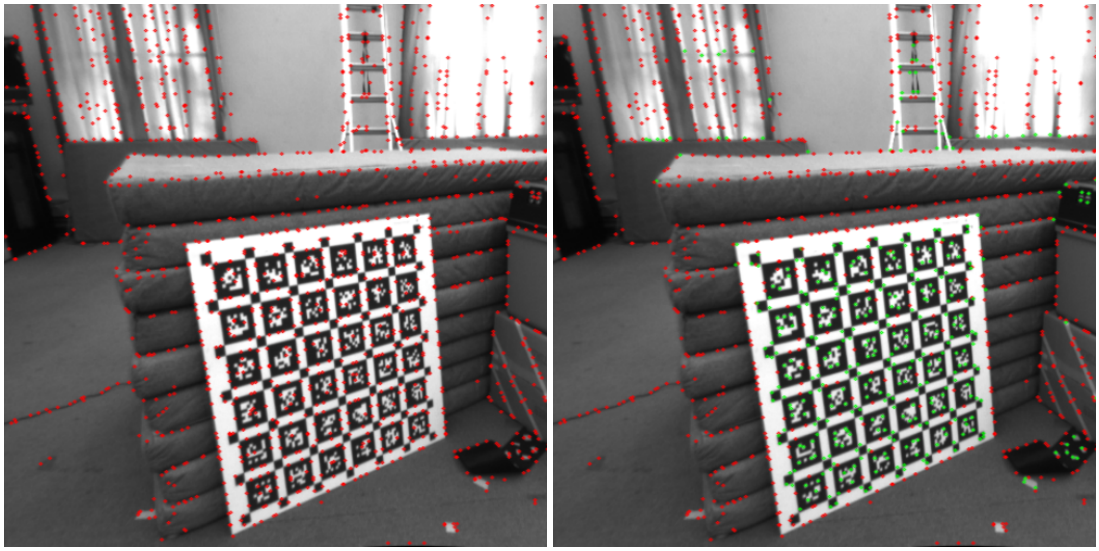

Figure 1: An example of the original (left) and updated (right) point selection procedure. Green points show feature points. By selecting corners, we make sure that we will select mostly the same points for this board in different frames.

is increased so a single point is selected for a larger cell and vice versa. While we keep this logic to select non-feature points, we first detect repeatable feature points using a similar grid approach. Since the window size is adjusted to get the desired number of selected points, we still get the same number of points in total.

For feature point selection, we use the FAST corner detector [5]. The main reason for selecting this method is its computational efficiency, such that this extra step does not affect the runtime of the point selection in DSM considerably. We utilize the OpenCV implementation of FAST. Since what we call a corner changes depending on the resolution of an image, we run FAST on all levels of the image pyramid of the frame and get corners for each level. From the found corners, we keep one corner per cell for each level. Since we use the same window size parameter, adjusted window size affects the selected corners the same way as other points.

Since not all candidate points are used later, in the point selection step DSM selects a high number of redundant points. As a result, only a small subset of these candidate points gets activated. For our needs, we want as many of the corners to be activated as possible to get their estimated depth to be optimized. Although for some candidate points we also get optimized depth, all active points have an optimized depth, which is also optimized in local bundle adjustment. To increase probability of corners getting activated and also to prevent selection of duplicate points, we ensure that no new non-feature points are being selected in

the cells where we already have corners. This way, we don't get adjacent points that are very similar, which can hinder the corner from getting activated.

After selecting all points, we also compute feature orientation, which we need later for descriptor computation. For this, we use an intensity centroid approach as is usually done for ORB descriptors [6]. We compute moments within a circular region using pixel intensities. The vector from the corner's center to the centroid gives the orientation of the corner. We then compute ORB descriptors, which are rotation invariant binary descriptors. As a binary descriptor, ORB is computationally very efficient compared to alternative methods such as SIFT [7].

Loop Closure

Loop closure detection is run for each keyframe. This happens with a delay after the keyframe is created so that we get as many active points as possible for the keyframe with more accurate depth estimations after a number of local bundle adjustments. We start this step after a keyframe leaves the temporal window, but this is a heuristic parameter that can be adapted.

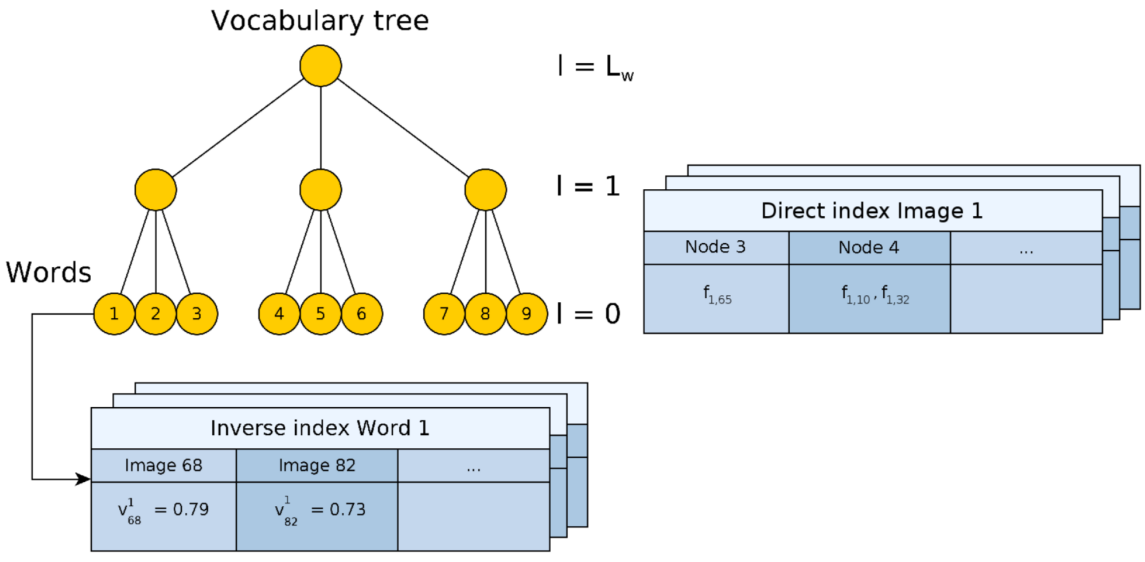

Figure 2: An example of vocabulary tree and direct and inverse indices that compose the database. (Source [8])

DBoW3 [9] is an open source library which converts images into bag of words vectors and

implements a database which can be queried to get similar images for a given image representation. We first compute the bag of words representation for the keyframe using DBoW3. Then we query the database for keyframes that are similar to current keyframe. After the query, the keyframe is added to the image database. At this point, we apply a sanity check for the query results. We wouldn't want to get keyframes that are close to the current keyframe in time and place, i.e. if they are in the same segment of the trajectory it is not considered a loop. This is usually done by using temporal information, e.g. excluding active keyframes and temporally adjacent keyframes from the results. Since we also keep track of covisibility information, we exclude covisible keyframes of the current keyframe from the results. Covisible frames are already handled in local bundle adjustment and some of them also happen to be active. This way we apply a broader filter than just using the temporal window (active keyframes in the temporal window are not yet included in the database anyway). So, results do not contain any covisible keyframe with respect to the reference keyframe, and in particular no active keyframe.

We get a score associated with each loop candidate. This score is based on the similarity of the candidate keyframe and the query keyframe which in turn is based on the similarity of their feature points. Ideally, we should consider all the candidates whose score is above an adaptive threshold, but for now we just use the candidate keyframe with the highest score for the following steps. Loop closure verification consists of many steps and at each step we stop if we do not meet necessary conditions and thresholds. This is supposed to ensure that no false positive loop closure is inserted in the global map.

We find ORB matches between the candidate frame and the query frame with the help of DBoW3, which is approximate but faster than brute-force matching. It builds a vocabulary tree that discretizes the descriptor space and we match features that correspond to the same word in the tree. From all feature pairs that correspond to the same word, we pick the one that has the minimum hamming distance if it is substantially better than the second best pick.

If we get enough matches, then we try to find the relative pose between the two frames using PnP with RANSAC [10]. For this, we use 3D points from the candidate keyframe and 2D points from the reference keyframe. We use active points or optimized candidate points as 3D points in the PnP problem. We solve the PnP RANSAC problem using OpenCV. RANSAC helps us to use only inliers for finding the relative pose since we might get wrong matches in the previous step. Additionally, we also apply PnP the other way round, where 3D points are from the reference keyframe. This is needed for computing the relative scale (we cannot get a scale

only using 3D-2D correspondences in one way). This relative scale is computed simply by taking the ratio of translation norms. If we do not get enough inliers for both cases, we stop here. In the next steps we refine the relative pose we find here, so relative pose from PnP serves just as an initialization of the subsequent optimization. We represent the relative pose with scale as a Sim(3) transformation.

Now that we have an initial relative pose between the keyframes we try to find additional matches, which were not detected initially (or discarded as non-unique). Unlike the previous set of matches for which we only use descriptor similarity without considering spatial positions, we now match 3D points by relying mainly on their spatial position as a constraint. We project points that are active or optimized from the candidate keyframe to the reference keyframe. Then, for each projected point, we get nearby points of the reference frame that are active or optimized and select the best match from these nearby points. Here, we filter points that do not have similar corner angles and then compare their descriptors to find the best match.

Using these additional matches we now refine the relative pose between the two keyframes. We have matched points which almost coincide when one point projected to the other frame. Assuming that most of these matches actually point to the same 3D point in the world (we applied many filters until this point which support this assumption), ideally with a perfect relative pose (and assuming perfect points) depths they need to coincide exactly. That is, we can optimize the relative pose by checking the correspondence of these matched points. For the optimization problem we use Ceres [11].

Let $Q = \{q_i\}$ be the 3D points from the candidate frame and $P = \{p_i\}$ be the matched 3D points from the current frame.

The Sim(3) transformation from candidate frame to current frame $\mathbf{S}$ is optimized by minimizing the following:

$$E_{LOOP} = \sum_{q_i \in Q} (||p_i - \mathbf{S} * q_i||_{\gamma_1} + ||\Pi(p_i) - \Pi(\mathbf{S} * q_i)||_{\gamma_2}) \tag{1}$$

where $\Pi(\cdot)$ is the projection function from 3D world coordinates to 2D image plane and $||\cdot||_{\gamma_1}$, $||\cdot||_{\gamma_2}$ are the Huber norm functions.

After the optimization, we check residuals to get inliers using fixed thresholds for 3D and 2D Euclidean errors, then remove the outlier matches from the problem. We solve the

optimization problem again with only using the inliers. At this point, we check the average cost per point for the optimized problem to determine whether we have a good case for loop closure. If the normalized cost is small enough, we save the optimized relative pose for these keyframes and consider the loop closure verified.

After getting the relative poses, we run pose graph optimization with all keyframes to reduce the accumulated drift. Here, for each keyframe pose we have one Sim(3) pose to optimize. To not interfere with local bundle adjustment, we fix parameters for active keyframes. We also fix the pose of the current keyframe. Each relative pose from loop closures as well as the covisibility graph adds a constraint to the optimization problem.

We use a threshold for number of shared points between keyframes to select connected keyframes from covisibility graph. The relative pose constraints for connected keyframes from covisibility graph are computed using current SE(3) frame poses where the poses are first converted to Sim(3) with a scale of 1.

The relative pose $\mathbf{S}_{ij}$ from keyframe K_j to keyframe K_i is computed using Sim(3) frame poses $\mathbf{S}_{wi}, \mathbf{S}_{wj}$ from respective frame to world as:

$$\mathbf{S}_{ij} = \mathbf{S}_{wi}^{-1} * \mathbf{S}_{wj} \tag{2}$$

Relative pose constraints from covisibility graph are recomputed for each PGO since keyframe poses are optimized in PBA whereas relative poses from verified loop closures are not recomputed.

The non-fixed frame poses $\mathbf{S}_{wi}$ are optimized by minimizing the following:

$$E_{PGO} = \sum_{i,j \in C} \log_{Sim(3)}(\mathbf{S}_{ij} * \mathbf{S}_{wj}^{-1} * \mathbf{S}_{wi}) \tag{3}$$

where C is the set of connected keyframe pairs that are derived from loop closures or covisibility graph and $\log_{Sim(3)}$ transforms error to $\mathbb{R}^7$.

After the pose graph optimization, we update the frame poses and their points' depths (2D positions of points are always fixed, only the depth changes when frame pose changes). We absorb the optimized scale s_i into the points and drop it from the transformation. This way,

we get the transformation by simply using rotation matrix $\mathbf{R}_i$ and translation t_i of $\mathbf{S}_{wi}$:

$$\begin{aligned} \mathbf{S}_{wi} &:= \begin{bmatrix} s_i * \mathbf{R}_i & t_i \\ o & 1 \end{bmatrix} \\ \mathbf{T}_{wi} &\leftarrow \begin{bmatrix} \mathbf{R}_i & t_i \\ o & 1 \end{bmatrix} \end{aligned} \tag{4}$$

where SE(3) pose $\mathbf{T}_{wi}$ and optimized Sim(3) pose $\mathbf{S}_{wi}$ are represented with 4×4 transformation matrices and o represents a 3-column vector of zeros.

For every point j hosted on keyframe i with optimized scale factor s_i, inverse depth d_j is updated as:

$$d_j \leftarrow \frac{d_j}{s_i} \tag{5}$$

Although we solve pose graph optimization after computing the relative pose for the loop candidate and reference frame, pose graph optimization is a separate step than loop detection. It can be run after finding multiple such relative poses between frames. In fact, for each pose graph optimization we use all previously found relative poses as problem constraints.

Results

For the experiments we use the EuRoC MAV [12] dataset. The EuRoC dataset contains 11 stereo sequences in 3 different indoor environments. In [1], DSM is only evaluated on the EuRoC dataset.

We compare our proposed system with loop closure, namely "DSM-loop", with DSM. We also evaluate an extension of DSM with the corner selection approach, namely "DSM-corner", in order to verify that modified point selection procedure does not deteriorate the accuracy.

We run each method on all the EuRoC sequences forwards and backwards, 5 times for both left and right camera images, for 220 runs in total. We do not enforce real-time execution, this way the only performance difference is due to corner selection and loop closure steps. We

Sequence	DSM	DSM-corner	DSM-loop	%	# loops
MH_01_l	0.0446	0.0417	**0.0407**	2.34	**1.3**
MH_02_l	0.0406	0.0388	**0.0385**	0.90	0.3
MH_03_l	0.0558	0.0566	**0.0548**	1.78	0.5
MH_04_l	0.0821 (2)	**0.0703 (1)**	0.0777	-10.56	0.1
MH_05_l	0.0867	0.0777	**0.0768**	1.06	0.1
V1_01_l	0.0998	0.0997 (2)	**0.0950 (1)**	4.75	**3.0**
V1_02_l	**0.0632**	0.0647	0.0635	-0.43	0.7
V1_03_l	0.0741 (5)	0.0974 (3)	**0.0731 (6)**	1.38	0.4
V2_01_l	0.0742	**0.0643**	0.0655	-1.90	**4.1**
V2_02_l	0.0642 (2)	0.0647	**0.0635 (1)**	1.12	**2.3**
MH_01_r	0.0449	**0.0408**	0.0435	-6.70	**2.6**
MH_02_r	0.0363	0.0370	**0.0356**	2.02	0.0
MH_03_r	**0.0474**	0.0589	0.0581	-22.63	0.7
MH_04_r	0.0845 (1)	0.0860	**0.0827**	2.09	0.1
MH_05_r	**0.0701**	0.0738 (1)	0.0706	-0.74	0.3
V1_01_r	0.0320 (1)	0.0552	**0.0282**	11.88	**3.3**
V1_02_r	**0.0433**	0.0450	0.0448	-3.38	0.7
V1_03_r	0.0582 (3)	**0.0466 (5)**	0.0786 (1)	-34.99	0.9
V2_01_r	0.0469	0.0509	**0.0468**	0.11	**3.5**
V2_02_r	0.0696	**0.0577**	0.0581 (1)	-0.68	**2.2**

Table 1: RMS ATE (m) for left (top half) and right (bottom half) camera images. V2_03 is omitted since all methods failed on it. Bold cells indicate the best performing method for the row. Numbers in parentheses gives the number of failed runs (out of 10). % column shows the percentage difference between DSM-loop method and the best performing method from the rest (positive values in favor of DSM-loop). *# loops* column gives the number of detected loop candidates in the sequence. Sequences with more than 1 loop on average is shown bold.

use the same parameter settings for all methods. You can see qualitative results from DSM and DSM-loop for V2_02 sequence at Figure 4.

We report RMS absolute trajectory error (ATE) after aligning estimated trajectories with groundtruths in Sim(3). We provide normalized cumulative error plots (Figure 3a) which show the percentage of successful runs with an error below a certain threshold. This way we visualize both accuracy and robustness of the methods at the same time. With higher thresholds we see the number of sequences the methods do not fail and with lower thresholds we see the number of sequences the methods perform well. Table 1 shows median errors for each sequence in the dataset. We also provide heatmaps full evaluation results for DSM-loop and DSM in Figure 3b.

Before evaluating effect of loop closure correction, we must ensure that the modified point

selection procedure does not deteriorate the performance of DSM. From figure 3a we can see that DSM-corner and DSM methods perform very similarly according to the area under the curve of the normalized cumulative plot, which is 50.78% and 50.58%, respectively.

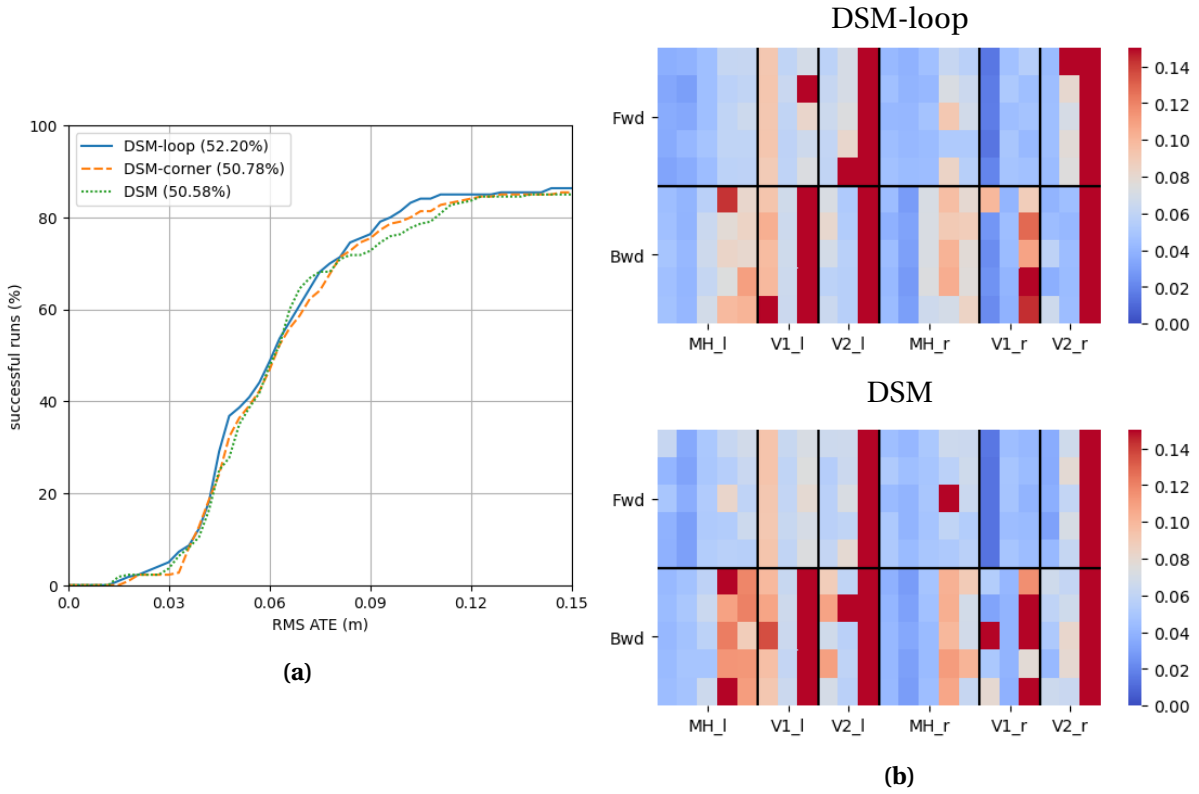

Figure 3: a) Normalized cumulative errors. Numbers in the legend gives the area under curve. b) Full evaluation results. Columns represent sequences and rows give forwards and backwards iterations.

While evaluating effect of loop closure correction, one should take number of corrected loops into consideration. If there is no loop to correct or the method is not able to detect any loops, then we simply compare the results of the method. In table 1 we see that our method detects more than 1 loop candidates on average for the sequences MH_01, V1_01, V2_01, and V2_02 (for both left and right cameras). For these sequences, DSM-loop has an average ATE of 0.0545 while DSM has 0.0598, which indicates to an **8.86% improvement**.

Conclusion

In this guided research project, we propose a direct SLAM system with global mapping and loop closuere. Benefiting from the advancements of previous systems DSO, LDSO, and DSM, we build our direct SLAM system as an extension of DSM, by changing the point selection

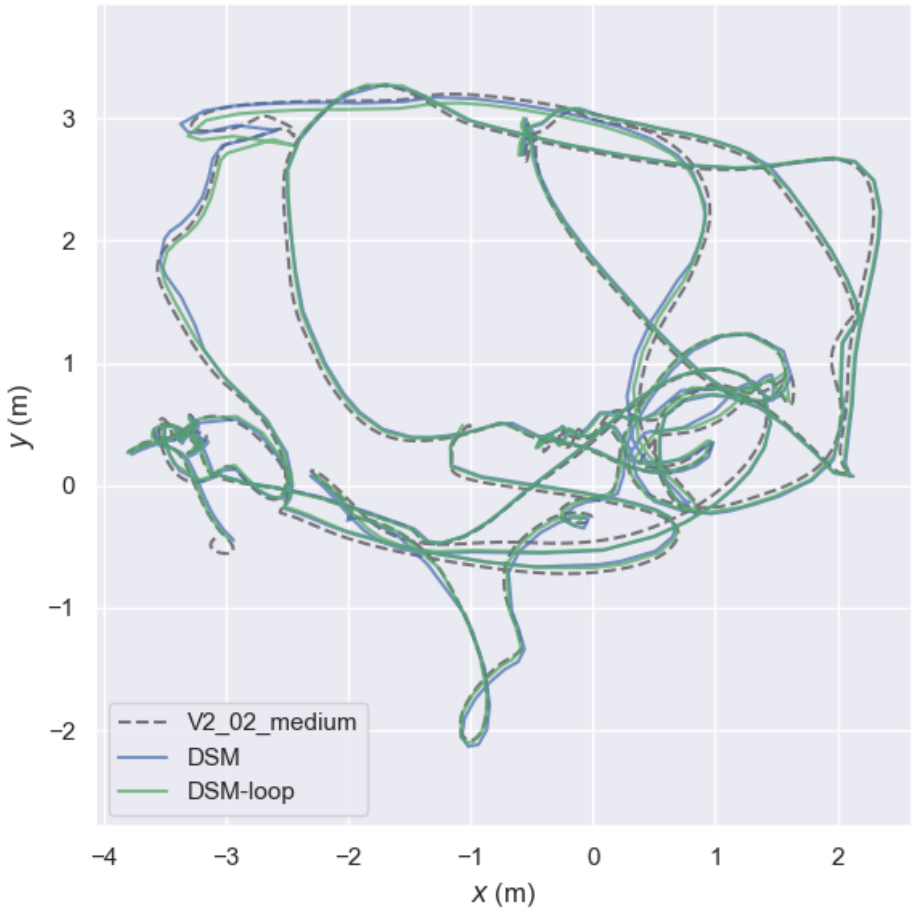

Figure 4: Trajectory in V2_02 sequence ([13])

procedure and introducing loop closure detection and pose graph optimization.

From the experiments, we see that changed point selection procedure which now includes a small percentage of repeatable feature points does not negatively affect the performance of the direct SLAM system.

With loop closure correction, our method on average gives 8.86% better results than DSM for the sequences with detected loops in EuRoC dataset.

As a future work, we need to fine tune the loop closure detection and verification parts to get higher number of successful loop closure candidates while still avoiding false positives. Currently these parts are very conservative and avoid false positives but we don't get as many detected loops for sequences as we expect. We also need to evaluate the methods on other datasets as well, since on EuRoC DSM is already quite accurate so it is hard to see

improvements with loop closure.

Bibliography

[1] J. Zubizarreta, I. Aguinaga, and J. M. M. Montiel, "Direct sparse mapping," *IEEE Transactions on Robotics*, 2020.

[2] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.

[3] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 3, pp. 611–625, 2017.

[4] X. Gao, R. Wang, N. Demmel, and D. Cremers, "Ldso: Direct sparse odometry with loop closure," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2198–2204, IEEE, 2018.

[5] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *European conference on computer vision*, pp. 430–443, Springer, 2006.

[6] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International conference on computer vision*, pp. 2564–2571, Ieee, 2011.

[7] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the seventh IEEE international conference on computer vision*, vol. 2, pp. 1150–1157, Ieee, 1999.

[8] D. Gálvez-López and J. D. Tardos, "Bags of binary words for fast place recognition in image sequences," *IEEE Transactions on Robotics*, vol. 28, no. 5, pp. 1188–1197, 2012.

[9] R. Muñoz-Salinas, "DBoW3." https://github.com/rmsalinas/DBoW3, 2017.

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

[11] S. Agarwal, K. Mierle, and Others, "Ceres solver." http://ceres-solver.org.

[12] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, vol. 35, no. 10, pp. 1157–1163, 2016.

[13] M. Grupp, "evo: Python package for the evaluation of odometry and slam.." https://github.com/MichaelGrupp/evo, 2017.