Evaluation of the Modern Visual SLAM Methods

Arthur Huletski, Dmitriy Kartashov
The Academic University
Saint-Petersburg, Russia
{hatless.fox, dmakart}@gmail.com

Kirill Krinkin
St. Petersburg State Electrotechnical University "LETI"
Saint-Petersburg, Russia
kirill.krinkin@fruct.org

***Abstract*—Simultaneous Localization and Mapping (SLAM) is a challenging task in robotics. Researchers work hard on it, so several novel SLAM algorithms as well as enhancements for the known ones are published every year. We have selected recent (2013-mid. 2015) approaches that in theory can be run on mobile robot and evaluated it. This paper gives brief intuitive description of ORB-SLAM, LSD-SLAM, L-SLAM and OpenRatSLAM algorithms, then compares the algorithms theoretically (based on given description) and evaluates them with TUM RGB-D benchmark.**

I. Introduction

Simultaneous Localization and Mapping is a promising area of research in robotics. The area focuses on methods that allow robots to estimate their position (localization part) simultaneously with gathering information related to environment (mapping part). SLAM problem is hard due to its "chicken-egg" nature: robot should use constructed map to localize itself simultaneously with pose usage to update the map. The problem becomes even harder since robot's sensors and actuators are subject to noises (e.g. robot have to use a map for localization since odometry error grows very fast). Various probabilistic methods are used to deal with uncertainty introduced by noise: Kalman Filter, Extended Kalman Filter, Particle Filter, etc. In a few words all these methods are based on Bayesian inference provide a technique for random value estimation [1]. The main idea of probabilistic methods is to continuously incorporate new observation to the existent one in order to refine the estimate. Thrun et al. provide good introduction to these methods [1].

Many proposed SLAM algorithms are based on described idea. It is useful to answer the following questions to have the very first impression of algorithm's uniqueness and its features:

- Which one and how values are estimated?
- How environment map is estimated?
- Which sensors and actuators are used?
- How the algorithms differs from its base method?
- What assumptions and approximation were made to derive the algorithm?

We believe that the last question is the most important since its answer implicitly determines algorithm's limits and quality. The answer is also important since probabilistic robotics can be thought as "the art of approximation" because there is a simple algorithm (based on Bayes theorem) that produce correct result but the algorithm is not useful since it is computationally expensive [1].

Comparison versus the base method (the last but one question) is also worth to be considered since it refines the algorithm's contribution. As an example consider DP-SLAM algorithm [2], that is based on Rao-Blackwellized Particle Filter [3]. It proposes a new effective way for map organization as well as hierarchical organization of itself to be able to process larger maps [4].

Several algorithms that use only monocular camera (and sometimes odometry) have been presented recently. These approaches try to extract information about environment and (sometimes) robot motion from camera images and use it to solve SLAM problem. Such algorithms are known as visual SLAM algorithms. It is worth to compare these algorithms since many of them require only monocular camera as a sensor. This makes a robot more affordable since monocular camera is an inexpensive sensor (comparing, for example, with a LIDAR).

This paper aims at comparing recent visual SLAM algorithms that have open source implementation. The remainder of the paper is organized as follows: Section II describes criteria that were used for comparison. We list algorithms chosen for analysis and provide brief description for each of them in Section III. Section IV provides comparison in two forms: "theoretical" (table with algorithms' summary) and "practical" (evaluation of performance on various datasets). A brief conclusion is given in section V.

II. SLAM algorithms characteristics

SLAM methods can be classified at least by used sensors and output map type and sometimes they have common underlying math methods (e.g. Kalman filter or bundle adjustment). Such "theoretical" features of the algorithms are described in Section IV-a. In this section we focus on the measurable characteristics.

There are several metrics in the SLAM area that don't depend on sensor or output map types. Though it's clear that the actual output of the SLAM algorithm often depends on sensor quality or algorithm implementation, these metrics allow to estimate performance and quality of an algorithms and decide whether an algorithm is suitable for the concrete task.

- Localization accuracy is the main property of the SLAM algorithm. Usually it's a root-mean-square error (RMSE) between robot's positions predicted by a SLAM algorithm and the ground truth data.
- Dataset processing time or data chunk processing

ISBN 978-952-68397-0-7 (paperback), ISBN 978-952-68397-1-4 (PDF)

time or mean CPU usage reflects the computational effectiveness of an algorithm.

- Peak memory consumption allows to estimate memory requirements of the one algorithm relative to other algorithms on the same dataset. This is often important for mobile robots which have the limited hardware resources.

Also there are metrics that are specific for particular SLAM type, for example:

- Camera frame processing time or FPS (frames per second) is specific for visual SLAM. Actually, it's desirable to process camera video stream in real time to timely react to the environment changes. It's especially important for the flying robots.
- Map quality, i.e. differences in SLAM output map and ground truth map, is specific for SLAMs which output map is in form of occupancy grid (e.g. GMapping [5]). It can't be measured for most graph-based SLAM as there's no actual environment map.

Besides the metrics mentioned above, some important but hard to measure SLAM algorithm characteristics exists. These include:

- Robustness can be defined as the ability of the algorithm to not degrade the localization accuracy over the long time or the ability to work in any environment (e. g. indoor and outdoor). From the other point of view, robustness can be thought as algorithm's ability to produce similar results for multiple runs on the same input.
- Convergence can be defined as the time that is necessary for the algorithm to minimize the localization error after the last relocalization procedure.

III. Description of recently proposed algorithms

We looked through a set of algorithms that were proposed recently (2013-2015), have open implementation and can be (at least in theory) used by common autonomous ground/aerial mobile robot.

The following algorithms were picked for evaluation:

- ORB-SLAM;
- OpenRatSLAM;
- LSD-SLAM;
- L-SLAM.

The algorithms are briefly described below.

A. ORB-SLAM

ORB-SLAM [6] is the visual SLAM method that utilizes ORB-features [7]. As opposed to some other visual SLAM algorithms it doesn't use any external odometry. This algorithm has several features. During robot exploration the place recognition database is constructed. This database contains bag of words representation of the current camera image that is bound to the specific position in the map. This database allows to perform queries with the set of currently observed ORB descriptors to recognize current place. Details on the usage of such database are described in [8]. Another feature of this SLAM is the *covisibility graph* in which vertices are keyframes and an edge connects two vertices if they share enough common features. Such graph is useful for finding several frames with the images of the same object from different view angles.

The SLAM system consists of 3 modules that work in parallel threads:

- *tracking* thread localizes camera in every new frame and decides when a new keyframe insertion is needed;
- *local mapping* processes new keyframes and performs local bundle adjustment to achieve an optimal space reconstruction in the surroundings of the camera pose;
- *loop closure* searches for large loops when the new keyframe is available.

Camera *tracking* begins with ORB feature extraction from the new camera frame. If the tracking was successful in the last frame then the first estimation of the camera pose in the new frame is the camera pose in the last frame. For each ORB feature from the previous frame, which has a map point associated, a match in the current frame is searched. This gives a set of 3D to 2D correspondences, so the camera pose can be computed by solving a PnP problem. If the tracking is lost then the frame is converted into the bag of words and the recognition database is queried for relocalisation candidates. For each candidate the ORB correspondences between the current frame and the ORB features associated to map points are computed. This gives a set of 2D to 3D correspondences so the PnP problem can be solved for each candidate keyframe. Best solution is considered as the camera pose estimation.

Previous steps produce an estimation of the camera pose and an initial set of feature matches. This allows to project the map into the frame and search for extra map point correspondences. Two sets of keyframes are selected to achieve this: the set K_1 consists of keyframes that share map points with the current frame, and the set K_2 consists of neighbors to the keyframes K_1 in the covisibility graph. Then each map point that is seen in K_1 and K_2 is searched in the current frame.

Finally, the camera pose is optimized using the initial estimation and all correspondences found between ORB features in the frame and local map points. The optimization minimizes the reprojection error using the Levenberg-Marquadt algorithm robustified with the Huber cost function.

The *local mapping* thread processes new keyframes, and updates and optimizes their local neighborhood. The *loop closure* thread detects loops in the map and if the loop is found it performs global optimizations that maintain the global consistency of the map. Both these threads process each new keyframe. The detailed description of these modules can be found in [6].

B. OpenRatSLAM

OpenRatSLAM [9] is the open-source implementation of the RatSLAM algorithm, that is appearance-based visual SLAM system originally proposed in [10]. The main feature of this SLAM algorithm is that it models navigational processes in the hippocampus (a part of the mammalian brain). It uses monocular camera as the main source of information about the environment and also able to visually estimate odometry (though it may use any other odometry source). As the robot explores the environment RatSLAM builds robot's trajectory map in form of graph in which vertices corresponds to some unique visual experience and edges contain odometry information. SLAM system consists of three major modules: pose cells, local view cells and experience map. Each of these components is described in details below.

Pose Cells Network (PCN) is a three-dimensional continuous attractor network. In fact, it's a 3D array of cells (or graph with vertices arranged in rectangular prism) and each cell is connected with other by excitatory and inhibitory links which wrap across the boundaries of the PCN. The dimensions of this array correspond to the x, y and θ coordinates of the ground-based robot. Each cell has associated activity (energy) level. The centroid of the cluster of currently active cells is the local estimation of the robot's current pose. During the localization process the internal activity in the PCN is recomputed as follows:

- active cells propagate their energy through excitatory links to other cells with respect to the link weight;
- energy in each cell is reduced by specified constant (global inhibition);
- activity in the whole network is normalized so that the total energy in the system is equals to 1.

After internal activity update the odometry information is used to shift activity in PCN. Activity shift is a simple displacement of the current activity state in the direction that is specified by odometry.

Local View Cells (LVC) is an array of units, each of which represents a distinct visual scene in the environment. Local view cell consist of raw pixel data (visual template) extracted from camera image and the weighted excitatory link that connects cell and the current pose estimation cell in the PCN. Each LVC has activity level that determines the similarity between the current view and the visual template. The visual template is produced from camera image by detecting visually interesting region in the image, converting to grayscale format and downsampling to the fixed size.

On each step of the SLAM process the visual template for the current camera image is matched with each previously gained visual template. A similarity measure is based on the sum of absolute differences (SAD): the comparison process finds the minimum SAD while shifting the stored templates along to the current template in the horizontal direction. If the smallest difference between the current and stored templates is less than a threshold, then the corresponding template is selected as active LVC. Otherwise, the new local view cell is created for the current visual template. All active LVCs inject activity into the associated pose cells through excitatory links and thereby affects the activity state of the PCN.

Experience Map is a graphical map that estimates a global robot pose by combining information from the PCN and LVC. Each node in the experience map contains current activity levels in PCN and LVC and global position of the node. A new experience is created when the current activity state in PCN and LVC is not closely matched by the state associated with any existing experiences. As the robot moves between experiences, a link that contains odometry information is formed between the previously active experience and the new experience. Finally, the map graph is updated by relaxation algorithm that distributes odometric error throughout the graph, providing a human-readable map of the robot's environment.

OpenRatSLAM implementation also includes *visual odometry* module that determines camera motion by comparing successive images. The module makes implicit assumption that the maximum camera motion is limited and the robot travels at a relatively constant speed. It allows for separate regions in the image to be specified for determining forward translational and rotational speeds using scanline profile that is the sum of the image columns. The rotational velocity is estimated by determining what relative horizontal offset of two consecutive scanline profiles minimizes the mean of absolute differences between the two profiles. The translational velocity is estimated by multiplying the minimum difference by a scaling factor and limited to a maximum value. Such visual odometry estimation is very rough and can be applied only for car-like robots, but the authors claim that it's sufficient to yield a topological map that is representative of the true environment.

So in brief, on each step of SLAM process PCN updates internal activity using excitatory and inhibitory links. In the same time the robot attempts to match current visual scene with known scenes in LVC. If the current scene can't be matched, i.e. visual scene is unique, this scene is sampled and stored in LVC with binding to the currently active cell in PCN. Otherwise the energy is injected in the PCN cell that is bound to the matched view. For most of local view cells a node in the experience map is created. As the new node is added to the graph, a relaxation operation that corrects the odometric error is performed over the map graph.

C. LSD-SLAM

Large-scale direct monocular SLAM (LSD-SLAM) [11] is an algorithm that uses only RGB images from a monocular camera as information about environment and sequentially builds topological map (graph-based map). The most distinctive properties of LSD-SLAM are its ability to reconstruct 3D environments and direct nature (i. e. entire image is used for localization and mapping instead of some set of features, such as angles or lines). The algorithm is actually an extension of the method that allows to estimate semi-dense inverse depth map with monocular camera. The method was introduced in [12].

The semi-dense inverse depth map is a probabilistic partial estimation of inverse (i.e. d^{-1}, d - actual distance) distance to point in environment that corresponds to a pixel on the image. The inverse depth map is represented by 2D array of random

values. Each r.v. *approximates* inverse depth with Gaussian, so it is described by mean and variance values. Pixel to inverse depth correspondence is partial, since inverse depth map can be estimated for "good" pixels on the image. The "goodness" is mainly defined by intensity gradient direction (i.e. direction in which intensity grows) of a pixel. In a few words, the depth map is estimated by disparity estimation for each pixel on current image and some image that has been captured before. After that estimated values (i.e. new observation) are merged with current inverse depth map (i.e. previous observation) with Gaussian merging (similar to Kalman filter). So, the method allows to continuously estimate current inverse depth map.

The knowledge about the environment is stored as topological (graph) map. Each node consists of an image and probabilistic inverse depth map. Nodes are connected with edges that hold information about relative scale-aware alignment (estimated) and corresponding covariance.

We believe the following intuition helps to understand the algorithm: the entire history of observations (camera images) is approximated by a set of *keyframes* (that directly corresponds to nodes), so entire map is divided into clusters and keyframe stores cluster-specific data. The main part of this data is inverse depth map, that acts as a "feature" (i.e. something that makes keyframe unique). This role of inverse depth maps makes clear its intention of being scale-aware: since actual inverse depth map's values depends on scale (e. g. an obstacle proximity), it is normalized to present uniform knowledge about environment's "profile". The information about scale is stores in an edge that connects two keyframes as well as estimated transformation.

The update loop of LSD-SLAM has the following steps:

1) capture *new frame* (image) and estimate transformation with respect to *current keyframe* (reference image). The estimation is performed by photometric error minimization with Gauss-Newton method;
2) compute "distance" between *new frame* and *current keyframe*;
3) if "distance" is below threshold – estimate inverse depth map of the *new frame* and merge it with *current keyframe*'s map. Then go to step 1;
4) create *new keyframe* from *new frame*: create initial depth map estimate by projecting inverse depth map of *current keyframe* with the estimate transformation, then remove outliers in it;
5) add *current keyframe* to the environment map and perform loop closure: find keyframe candidates (10 the most closest (by distance) keyframes and keyframes proposed by appearance-based mapping [13]), then try to insert edges between *current keyframe* and candidates: the edge is inserted when estimated independently back and forth transformations are statistically similar.

There is also a background thread that performs optimization with g2o framework [14].

D. L-SLAM

6-DoF Low Dimensionality SLAM or L-SLAM [15] is an improvement of FastSLAM 2.0 algorithm [16]. Both algorithms are probabilistic, feature-based and use Rao-Blackwellized particle filter to track posteriors. Position of each feature (i.e. points on the environment map) is treated as random variable as well as robot's pose. L-SLAM uses Particle Filter to estimate *robot orientation* and Kalman Filter to estimate robot and feature positions. This scheme differs from the one used by FastSLAM algorithm: Particle Filter is used to estimate entire robot pose (orientation and location), *Extended* Kalman Filter is used for pose estimation of each feature. L-SLAM also performs Kalman Smoothing to propagate the newest measurement backwards to already estimated robot poses (i. e. trajectory) and features (i. e. map).

L-SLAM algorithm uses odometry to sample new robot orientation with Particle Filter. Posteriors for robot's (as well as features') poses are estimated with Kalman Filter, since derived approximation of kinematic model of 6 DoF vehicle is linear.

The observation is defined as a set of features, extracted by an abstract sensor. Generally speaking, features of any kind can be used, but model derived in the paper uses 3D laser scanning sensor as an input. The only (soft) requirement for feature detector is low noise (since such assumption was made in process of derivation).

The particle update loop of L-SLAM has the following steps (for each particle):

1) *sample orientation*: update particle with odometry;
2) *estimate feature correspondence*: estimate features' importance factor (Mahalanobis distance between observation and estimation), then pick feature with maximum factor;
3) *update observed features*: if maximum factor of selected feature is less than the threshold – add a new feature; otherwise – update position of selected feature with *Kalman Filter*;
4) *update trajectory*: use *Kalman Smoothing* to update previously estimated poses and feature positions that are not observed by current measurements;
5) *update particle weight* by multiplying it by selected feature weight.

When every particle is updated, loop resampling is performed as part of Particle Filter algorithm. The map is represented as cloud of features.

Speed of L-SLAM outperforms FastSLAM 2.0 by factor of 3 according to [15]. This fact can be explained by changing Extended Kalman Filters to (linear) Kalman Filters for features' position estimation. L-SLAM also uses Particle Filter for the space of much less size (orientation vs pose) which in theory makes it more robust. The accuracy of both algorithms are nearly the same as estimated in [15].

L-SLAM algorithm is presented as an algorithm that uses odometry and laser scanner as a sensor, but it can be thought as back-end for a visual feature detector, that returns reasonably small number of features and has low noise. So the original algorithm can be altered to be monocular by incorporation of visual odometry module and visual feature detector.

TABLE I. SLAM ALGORITHMS OVERVIEW

Algorithm	Core estimated values	Map type	Used information	Main contribution	Assumptions
ORB-SLAM	• camera transformation (by feature matching error minimization); • 3D feature position;	• cloud of ORB features; • keyframe graph;	RGB image	• usage of ORB as environment features;	• robot movement between 2 consecutive frames is relatively small.
RatSLAM	• robot position and orientation (by Pose Cell Network); • visual odometry (optionally);	*Graph:* • *vertex* – position of the unique visual experience; • *edge* – distance and travel time between vertices;	• RGB image; • odometry;	• first biological-inspired SLAM;	• excitatory links weight matrix has Gaussian distribution.
LSD-SLAM	• camera transformation (by photometric error minimization); • inverse depth map (by pixelwise Kalman Filter);	*Graph:* • *vertex* – keyframe (image + depth map); • *edge* – constraint (scale-aware transformation);	RGB image	• tracking structure of environment; • monocular camera is the only sensor;	• inverse depth is Gaussian; • noises are Gaussian;
L-SLAM	• orientation (by Particle Filter); • robot and feature position (by Kalman Filter);	cloud of features	• odometry; • extracted features;	• one-dimension particle filter; • derivation of linear model; • Kalman Smoothing to refine previous estimations;	• sensor noise is low; • noises are Gaussian;

IV. EVALUATION

We found useful to compare algorithms in two steps. The first one recaps algorithms' description and gives answers to the set of questions mentioned in the first section of the paper. The second part of evaluation compares criteria described in the section II. Actual values for each algorithm are obtained by running it on various datasets.

A. Theoretical Comparison

We compare the selected SLAM algorithms using the following criteria:

- *Estimated values* are parameters that are used in the specific algorithm in order to compute robot's position and construct the environment map.
- *Map type*. Different SLAM methods provide different map types (occupancy grid, feature cloud, trajectory graph, etc.). Each applied task may require specific map type: for example, the graph maps are more suitable for self-driving cars and street mapping while occupancy grid map may be used for tasks where the environment is static.
- *Used information* defines the set of sensors that is required for running the algorithm. Most of the modern visual SLAM methods require only monocular camera and estimate odometry using the camera video stream, but some 2D SLAM methods (e.g. GMapping [5]) require external odometry as well as range-scan data.
- *Contributions* of the algorithm in the solving of the whole SLAM problem.
- *Assumptions*. As it was mentioned above, the assumptions, on which the SLAM algorithm is based, implicitly determines algorithm's limits and quality.

The algorithm comparison results are aggregated in the Table I.

B. Practical Comparison

The practical comparison of the selected SLAM methods is performed on the Amazon EC2 virtual server with 2.4 GHz Intel Xeon CPU and 8 Gb RAM. The algorithms are tested using the subset of the TUM RGB-D benchmark datasets [17] by running each algorithm 15 times on each dataset. We made the following tweaks that allowed to achieve better output quality:

- ORB-SLAM: camera settings were updated with the settings from dataset;
- RAT-SLAM: datasets were played at 0.5 speed.

We excluded L-SLAM from the testing because the version described in corresponding paper is not a visual SLAM algorithm. The testing results for RMSE are shown in the Table II and also visualized in Fig. 2. Also a sample of the SLAM output trajectory for fr3_long_office_household dataset is shown in the Fig. 1.

Value that corresponds to a median has two parts: RMSE mean and information related to algorithm robustness in brackets (optional). Robustness has the following values:

- "-" – none of trials was executed successfully (e. g. tracking was lost (LSD-SLAM)). RMSE result is calculated for partial trajectory;
- "number" – percents of trials that were executed

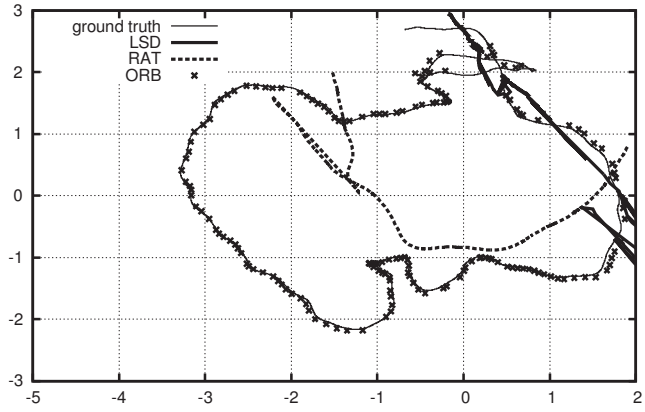

Fig. 1. SLAM trajectories for fr3_long_office_household test

TABLE II. RMSE COMPARISON IN THE TUM RGB-D BENCHMARK [17]

Dataset	Median (m)				Mean (m)				Standard Deviation (m)			
	LSD	ORB	ORB Sc.	Rat	LSD	ORB	ORB Sc.	Rat	LSD	ORB	ORB Sc.	Rat
fr1_desk	0.75 (7%)	0.05	0.05	1.21	0.75	0.05	0.04	1.12	X	0.02	0.01	0.22
fr1_room	0.05 (-)	0.19 (87%)	0.09	0.97 (93%)	0.05	0.18	0.1	0.95	0	0.12	0.07	0.06
fr1_xyz	0.17 (93%)	0.04	0.013	0.22	0.26	0.05	0.018	0.22	0.18	0.03	0.01	0.01
fr2_desk	0.43 (7%)	0.74	0.16	2.56 (93%)	0.43	0.76	0.19	2.56	X	0.09	0.14	0.01
fr2_pioneer_slam2	0.55 (7%)	0.24	0.05	1.62	0.55	0.62	0.13	1.62	X	0.62	0.12	0.006
fr3_large_cabinet	0.80 (20%)	1.75 (40%)	0.53	1.88	0.71	1.75	0.56	1.88	0.14	0.07	0.11	0.001
fr3_long_office_household	0.54 (-)	1.11	0.04	1.38	0.54	1.10	0.04	1.26	0.22	0.02	0.008	0.25

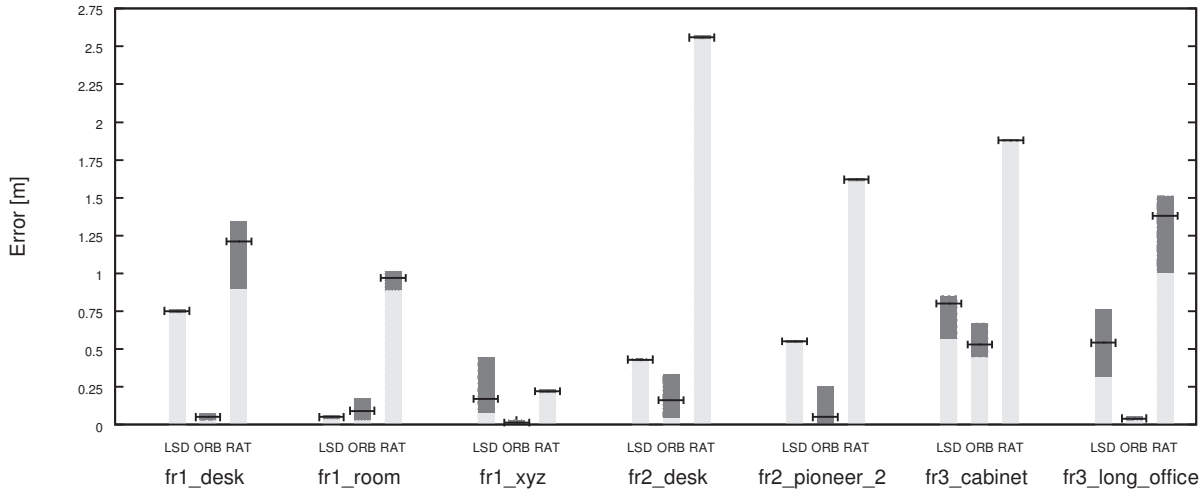

Fig. 2. RMSE comparison

successfully. RMSE was calculated using only these results.

The absence value in brackets means that all trials were executed successfully. The "X" value in standard deviation column indicates that we haven't enough data to compute it (e.g. only one trial was successful).

Note that ORB-SLAM has extra "Sc." measurements. We haven't changed OotB settings (except camera) and the algorithm's output we get doesn't have proper trajectory scale, so we had to tune the scale *manually* to obtain the smallest error. Note that "scaled" measurements has the same rate of success runs as the original ones. Since LSD-SLAM and RatSLAM are supposed to be scale-aware, scale adjustment wasn't performed for them.

The following observations and conclusions can be made basing on our experience and data after the testing:

- We couldn't obtain stable tracking and low error of trajectory on the dataset for any of described visual algorithms. For example, RMSE is big enough and rate of success runs is low on some real-world tests (e.g. in case of fr3_large_cabinet).
- We couldn't get robust results for any of the algorithms when run their out-of-the-box implementations, because: RatSLAM is not accurate and fast enough (datastream was explicitly slowed down by factor of 2, otherwise the algorithm wasn't able to finish successfully at all); ORB-SLAM requires postprocessing and LSD-SLAM requires tweaking and very non-deterministic.
- LSD-SLAM showed very non-deterministic behavior (rate of success runs) on the dataset. In addition, tracking was lost in completely different frames for some tests (e.g. fr3_long_office household). We can conclude that the algorithm require more sophisticated tuning that we were able to do. FoV of camera that captured the dataset can be thought of another possible reason of low tracking robustness.
- ORB-SLAM requires manual adjusting of the trajectory scale, which can be seen by comparing data in "ORB" and "ORB Sc." columns. On the one hand, adjusted trajectory is very accurate (see Fig. 1), but on the other hand, posterior scale adjustment is unacceptable for robots that work in real-life environments.
- RatSLAM is the most slow and inaccurate of all. The last is probably caused by inaccuracy of its visual odometry model.

V. CONCLUSION

Visual SLAM algorithms occupies notable place in taxonomy of SLAM methods. Such algorithms require monocular camera, which makes a robot affordable. Recently proposed visual SLAM methods (LSD-SLAM, ORB-SLAM and Open-RatSLAM) were analyzed and evaluated at RGB-D dataset from TUM. Unfortunately, our runs of examined methods either showed significant error or requirement of manual postprocessing to make the error small.

Estimated RMSE values lead us to equirement for a new visual SLAM algorithm that can be fairly (i. e. without postprocessing) used by mobile robot:

- method that estimates camera transformation should have at least medium level on noise (RatSLAM drawback);
- an algorithm should be scale-aware (ORB-SLAM drawback);
- in case of tracking loss an algorithm should use some fallback strategy instead of giving up (LSD-SLAM drawback).

We believe the most promising areas for future efforts in monocular SLAM includes methods of robust and accurate camera transformation estimation and approaches that automatically adjust trajectory scale, since robust monocular odometry is a necessary condition for a monocular SLAM as it was shown by tests.

ACKNOWLEDGMENT

Authors would like to thank JetBrains company for provided support and materials for working on this research. The paper has been prepared within the scope of project part of the state plan of the Board of Education of Russia (task # 2.136.2014/K).

REFERENCES

[1] S. Thrun, W. Burgard, D. Fox, "Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)". Cambridge: The MIT Press, 2005.

[2] A. Eliazar, R. Parr, "DP-SLAM: Fast, Robust Simultaneous Localization and Mapping Without Predetermined Landmarks", *in Proc. of the 18th International Joint Conference on Artificial Intelligence*, pp. 1135-1142, 2003.

[3] A. Doucet, N. de Freitas, K. Murphy, S. Russell, "Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks", *in Proc. of the 16th Conference on Uncertainty in Artificial Intelligence*, pp. 176-183, 2000.

[4] A. Eliazar, R. Parr, "Hierarchical Linear/Constant Time SLAM Using Particle Filters for Dense Maps", *in Proc. of NIPS*, pp. 339-346, 2005.

[5] G. Grisetti, C. Stachniss, W. Burgard, "Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters", *IEEE Transactions on Robotics*, pp. 34-46, 2007.

[6] R. Mur-Artal, J. M. M. Montiel, J. D. Tardos, "ORB-SLAM: A Versatile and Accurate Monocular SLAM System", *IEEE Transactions on Robotics*, 2015.

[7] E. Rublee, V. Rabaud, K. Konolige, G. Bradski, "ORB: an efficient alternative to SIFT or SURF", *ICCV*, 2011.

[8] R. Mur-Artal, J. D. Tardos, "Fast relocalisation and loop closing in keyframe-based SLAM", *in Proc. of IEEE International Conference on Robotics and Automation (ICRA)*, June 2014.

[9] D. Ball, S. Heath, J. Wiles, G. Wyeth, P. Corke, M. Milford, "OpenRatSLAM: an open source brain-based SLAM system", *Autonomous Robots*, April 2013.

[10] M. Milford, G. Wyeth, "Mapping a Suburb with a Single Camera using a Biologically Inspired SLAM System", *IEEE Transactions on Robotics*, vol. 24, pp. 1038-1053, 2008.

[11] J. Engel, T. Schöps, D. Cremers, "LSD-SLAM: Large-Scale Direct Monocular SLAM", *ECCV*, 2014.

[12] J. Engel, J. Sturm, D. Cremers, "Semi-Dense Visual Odometry for a Monocular Camera", *ICCV*, 2013.

[13] A. Glover, W. Maddern, M. Warren, S. Reid, M. Milford, and G. Wyeth, "OpenFABMAP: An open source toolbox for appearance-based loop closure detection.", *ICRA*, 2012.

[14] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, W. Burgar, "g2o: A General Framework for Graph Optimization", *ICRA*, 2011.

[15] N. Zikos, and V. Petridis, "6-DoF Low Dimensionality SLAM (L-SLAM)", *Journal of Intelligent and Robotic Systems*, vol. 79, 2015, pp. 55-72.

[16] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, "FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges", *IJCAI*, 2003.

[17] J. Sturm, N. Engelhard, F. Endres, W. Burgard, D. Cremers, "A Benchmark for the Evaluation of RGB-D SLAM Systems", *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, 2012.