Modern Multi-Agent SLAM Approaches Survey

Kirill Krinkin, Anton Filatov, Artyom Filatov
Saint-Petersburg Electrotechnical University "LETI"
St. Petersburg, Russia
kirill.krinkin@fruct.org, {ant.filatov, art32fil}@gmail.com

***Abstract*—One of the most challenging problems for autonomous mobile platforms is simultaneous localization and mapping (SLAM). This problem is solved more or less for a single agent but applying this task for multiple platforms seems to be perspective. Usage of multi-agent SLAM might be very useful in situations when a single platform has not enough computation resources or can be broken or has low quality sensors. Various new issues appear while solving multi-agent SLAM problem, such as constructing of agent hierarchy, choosing of a SLAM algorithm on each agent etc. The goal of this paper is to provide modern multi-agent SLAM approaches survey.**

I. Introduction

One of the question faced while developing software for mobile platforms is choosing an approach for detecting its state in an environment. In special cases it is possible to provide an actual map of the environment and a locating method to the platform. In other cases there is no knowledge about the map structure, for example, an environment has not been discovered yet. Thus the platform should simultaneously create the map and locate itself there. This problem is known as simultaneous localization and mapping (SLAM) problem.

The examples of solving the SLAM problem for a single platform are presented in [6]-[9]. The classic approach is to use extended Kalman filter (ex. EKF_SLAM [6]), but its execution time increases with extending of the handled environment. Another approach is to use a particle filter where each particle represents a hypothesis of a platform state. The popular strategy is to use graph based SLAM algorithms to provide a dynamic correction for estimated variables during a SLAM process. These ideas could also be applied to multi-agent SLAM with specific modifications.

The multi-agent approach uses several mobile observers and includes an additional task to evaluate platforms relative positions besides issues of single SLAM. The multi-agent architecture provides benefits that are described below.

- An environment could be observed faster as platforms could travel to various directions. It could be applied in situations when time is a critical resource for example for building a rescue route in indoor environment that is on fire.
- In case when a platform is broken down, other could proceed the mission. It is useful for example in a space mission on the Mars surface when each platform might not rich a check point or could lose a connection.
- If two platforms follow crossed trajectories, they are able to correct the map and their trajectories by combining their observations to decrease measurement errors.
- If observers contain the computation unit, the task of building a map and localization of each platform could be distributed between several platforms that increases the performance of the whole system.

Several multi-agent SLAM approaches are considered in this paper and the main ideas of these methods are described. There is a list of properties that are used for a classification:

1) a hierarchy of agents roles;
2) a map representation;
3) a way to estimate relative positions;
4) trajectories & map corrections.

The motivation of this work is to review existing solutions, to identify the differences and to describe their application domains.

The paper is structured as follows: the section II presents a general description of single SLAM and multi SLAM problems; in the section III there are descriptions of existing multi-agent algorithms; the evaluation of considered algorithms is presented in section IV; conclusion and result theses are in the section V.

II. Problem formulation

A. Single SLAM

The SLAM problem formally consists in estimating a world state (a platform position and a map of an environment) using only information from sensors. SLAM algorithms based on different sensors may significantly differ one from each other. The most popular types of observations are:

- video frames presenting a 3D view of an environment, captured by a video camera;
- laser scans presenting a 2D top-down view of an environment, captured by laser rangefinders;
- odometry that represents information about platform's displacement.

The example of a frame that is captured from omnidirectional video camera is presented on Fig. 1. A camera provides frames where landmarks could be extracted and described robustly. There are approaches based on a color difference [12] or a corners extraction [14] etc. Extracted landmarks could displace a representation of a whole map and Kalman filtering [6] could be applied to estimate platforms location.

The example of a laser scan is shown on Fig. 2.

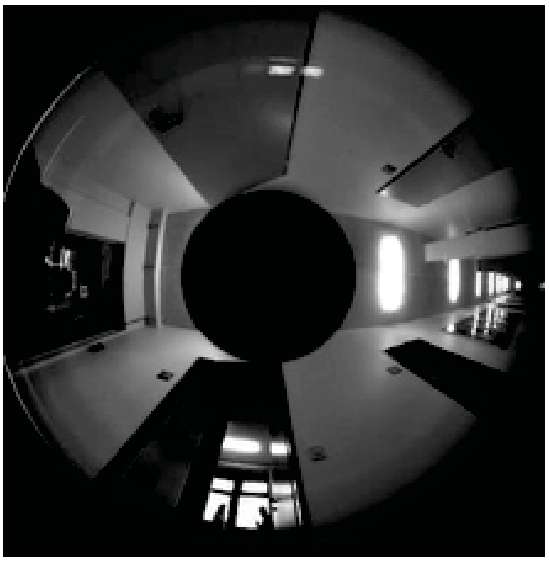


Fig. 1. The example of a scan captured on an omnidirectional video camera

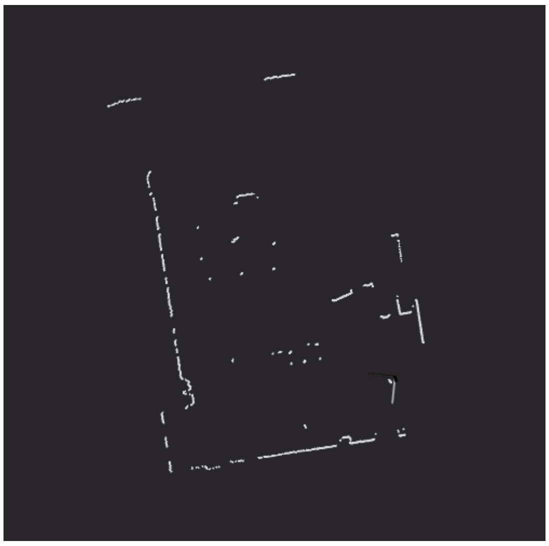


Fig. 2. The example of a laser scan captured from a laser rangefinder

A process of extracting landmarks from a 2D view provided by laser rangefinders is difficult. However there is an algorithm [11] that can solve this problem but it is less robust because a 2D laser view contains less information than a camera 3D frame.

B. Multi-agent SLAM

The multi-agent SLAM task as well as the single SLAM consists in an estimating a world state. But in this case the world state is defined as a map of a whole environment and platforms positions including their relative orientation. This relative orientations are important because it is an only way to combine two maps built with data from two observers. There is approaches that require a prior knowledge about platforms' orientations [1] other try to estimate it in-time if they have a chance [2],[5]. In the both cases methods should provide rules to correct local world state for single observer using data from another one.

Fig. 3 shows a high-level scheme of a typical multi-agent SLAM method.

- *Platform$_1$*, ..., *Platform$_N$* presents executors of single SLAM algorithms. They could be separate computer units or threads from a mainframe etc.
- *Merger* block updates data from a corresponding *Platform* with external data from another *Platform*-s.
- *Common merger* collects all estimated local world states and combines them to produce the output world state and provide data for a local *Platform* to update its state.

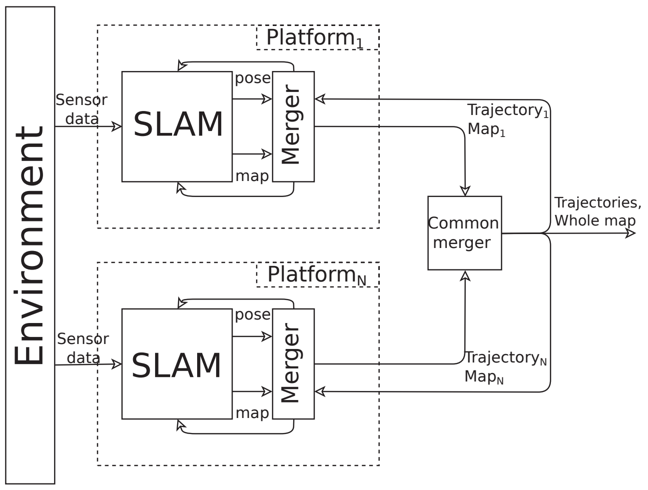


Fig. 3. The high-level multi-agent SLAM scheme

The challenging problem in multi-agent SLAM is a merging the maps that are built by each separate platform. Every multi-agent SLAM method proposes its own algorithm of merging or requires special assumptions for the environment or the world state. For example, following one approach it is required the prior knowledge of platforms' start points and following other – agents should estimate it by themselves. Another problem is to chose the representation of the map. To the best of our knowledge there are three popular types of map setup that are described below.

- A graph representation of a map where each vertex store an unique position of a platform and eadges present the transaction between vertexes [4], [9].
- Landmarks with their covariance matrix. Landmarks may be calculated using captured scans or may be located in the environment manually [2], [5], [6].
- A certainty grid that stores the probability for each cell to be occupied [8], [13].

Every chosen type of a map strongly influences on a merging algorithms.

There is also a question of an interaction of the maps: a map can exist as the one copy that is shared between all platforms or each agent can keep and update its own representation of a map. Moreover the result of a merging can be presented as a common map that is equal for each platform, or every existing map can be updated independently.

At the same time the individual pose updating is an actual problem for multi-agent SLAM because observers could meet in the real world but not in own maps. This meeting initiates a pose correction of the every platform. These corrections could happen as a jump – instantly teleport of a platform to the best found position, or through spreading the correction over several previous poses, but in the last case it is required to track several poses but the last one.

III. Existing solutions

This section contains descriptions of multi-agent SLAM implementations. These algorithms are described in the corresponding papers [1]-[5] and reviews of each of them is presented below.

A. MIT multi-agent SLAM implementation

One of the first work related with multi-agent SLAM is [1] – the MIT implementation expanding the classic Kalman filter SLAM approach to the multi-agent SLAM. The proposed approach is described as a part of a simulation system that is not able to be applied in the real world “as is”.

Following this approach two different roles are determined. The first role is an executor which runs the SLAM algorithm based on classic Kalman filtering, the second role is “The Test Bed” – the supervisor that knows the truth state of the world. This Test Bed also provides observation to each executor when it is ready to get data.

An environment is presented as a set of segments and circles that are determined as features with unique identifier on a map. So when an executor is ready to get an observation, The Test Bed analyzes what obstacles this executor could observe (obstacle ID, a range and an angle) and sends this information to the agent. The agent updates states and a covariation matrix of already observed features and adds new detected features.

All agents begin the execution oriented with the same angle in the world coordinates to avoid rotation while estimating a relative orientation. The start position of any agent is unknown for each other. This positions could be estimated only when one platform directly observes another one, and after that these two executers share the information of observed landmarks that could be transformed to its own coordinates by shifting only by $(\Delta x, \Delta y)$ with no rotation by any angle.

While transaction one agent sends through Test Bed its coordinates, list of new landmarks and built covariance matrix to another agent. After all the data has been received the second agent applies the calculated offset to the coordinates of new obstacles and considers them as new observations, i.e. uses them in Kalman filtering to improve its model.

B. Multi-robot SLAM with unknown initial correspondence

Multi-robot SLAM with Unknown Initial Correspondence is provided in [2] where authors present the implementation of the multi-agent SLAM algorithm based on Extended Kalman filter using visual sensors. The feature of this work is the absence of any prior knowledge for one platform about a start point and an orientation of basis coordinate vectors of any other agent. The single platform keeps its own coordinates and it doesn’t know the world coordinates of its start point. This model is suitable for executing in the real world where the dead certainty about start position and orientation of each platform is unreachable.

There are several platforms that run EKF SLAM independently until they meet. Every platform is equipped with a bright cylinder that can be easily extracted on a video frame. While testing this approach authors used omnidirectional camera as a video sensor, so the agents had no blind areas and they could detect each other if they come close enough regardless of a rotation.

A map of an environment is presented as a vector of landmarks coordinates and their covariance matrix. Updating a map and adding new landmarks is performed by classic EKF SLAM approach. The landmarks represent the corners extracted from laser rangefinders that are also placed on a platform.

When an agent detects another one it estimates an angle between their orientations and a distance between platforms. The distance is measured with a laser rangefinder and the angle can be calculated by the formula that is explained by the Fig. 4.

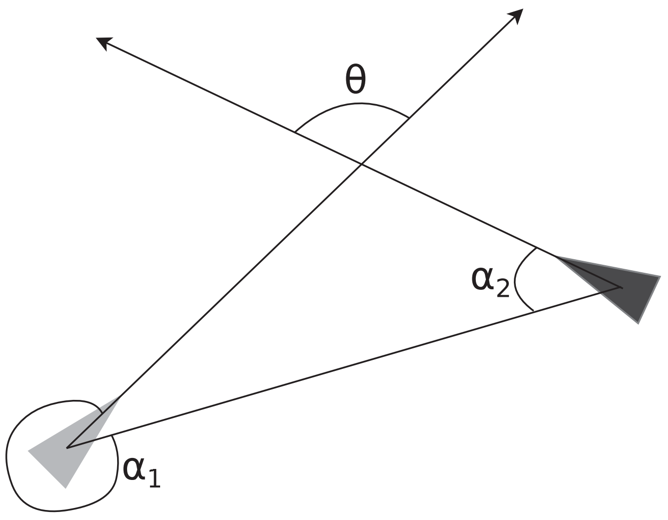


Fig. 4. Calculation of relative orientation angle θ

The angle θ between two platforms’ orientation is calculated by the following formula:

$$\theta = \pi + \alpha_2 - \alpha_1$$

To estimate the distance between platforms, both agents measure the distance and the result is fused taking into account the variances of measurement errors of each agent.

After the relative position has been estimated the agents transfer their own covariance matrices to each other. When the covariance matrix P_2 is received, it is integrated into the agent’s own matrix P_1 directly, i.e. following the formula

$$P_1' = \begin{bmatrix} P_1 & 0 \\ 0 & P_2 \end{bmatrix}$$

Then the algorithm of the Nearest Neighborhood is used to find the duplicate landmarks. The amount of the "closeness" is estimated with Mahalanobis distance. If a duplicate landmark is found in the matrix, it is erased and this algorithm repeats until all duplicate landmarks are found. To make this algorithm more robust the following idea was implemented: the landmarks that are closer to the place of agents are more accurate and the threshold for Mahalanobis distance for them is lower then for the landmarks that are far from the point of the meeting.

C. Collaborative monocular SLAM with multiple MAVs

Collaborative Monocular SLAM with Multiple MAVs presented in [3] is an implementation of a visual multi-agent SLAM algorithm. A video stream is used as an input. This type of sensors is preferred as a video camera is not so heavy as a LIDAR. It is important because Micro Aerial Vehicles (MAVs) were chosen as observers for this type of SLAM.

Every MAV carries a video camera and is equipped with a low power CPU that is not enough to solve a single SLAM problem in the real time. This task is delegated to a server that receives data from the every observer. The observer's CPU is responsible to extract and make a description of features from a camera frame. The information sent to a server presents a set of points with their descriptors defined, for example, as OperSURF features [14].

Every set of points sent to a server is recognized whether it contains new points that have not been observed yet. In this case this frame is called "the key-frame" and is stored in a map, in the another case this set is dropped. The map consists of a set of key frames that have no common points (core frames) and all other key frames (periphery frames). An overlap of a new key frame and key frames in a map is found and then a positions of common features is updated with local Bundle Adjustment [15].

The relative position of MAVs could be defined only after it detects an overlap between two separate maps. So a looping closure operation of one observer or a merging maps from two observers are the same task. To provide the loop closure it could apply a global Bounded Adjustment approach but it is high cost for big maps and moreover could fails while it is looking for a local minimum. So authors suggest to use g2o framework [10] for this optimization. Combining two key frames it simultaneously solves a task of founding relative MAVs positions and map merging.

For indoor environment that presents a surface of approximately 8 by 8 meters the error of locating features (that is delegated to Bundle Adjustment [15]) reaches 0.4 pixels. At the same time a looping closure error reaches 0.01 meters at the end of the testing [3]. The common error for two MAVs observing one room is not bigger than 0.1 meter for 70 seconds execution.

The evaluation using a huge outdoor environment data is not consistent at this work because there is no ground truth for the chosen dataset. The GPS position estimation was used as a ground truth but it provides an error for 5-15 meters so it could not be a sample of an absolute truth location of an observer.

D. Condensed measurements graph based multi SLAM

In this section there is a review of the work [4] where graph based multi-agent SLAM with condensed measurements is described. The idea from this paper seems to be perspective because it has benefits that is described below.

One of the most complicated task during multi-agent SLAM is a looping closure operation – a process of determining whether two agents have visited a same place. This task is solved in graph based single SLAM approaches for determining whether the observer returns to a place that has been visited before. Thus a count of observers is not a big difference for graph based approaches.

The classic graph based single SLAM approach defines a map as a graph. Vertexes of this graph could presents features of an environment or positions of an observer. The second view could be used in cases when features could not been extracted robustly and this representation is used in [4]. Vertexes of the graph presents an observer position and contains observed scan data, and edges presents a transformation between two vertexes. This transformation is flexible for correction, so it could be updated in the future. It makes multi-agent approach to be defined as a single graph based one with many observers. In [4] it is supposed to use several autonomous mobile platforms without any server for collecting data or evaluating platforms positions.

The huge problem faced in multi-agent SLAM is a big data that is to share between agents. So looping closure algorithms take very much time to handle all information. In [4] it is suggested to share "condensed measurements" that is not so heavy. The shared measurements is called "condensed" because one platform provides to another only important piece of information. The transfer process begins after one platform locates another one in its neighborhood. After that platforms make steps of graph SLAM independently and then exchange the handled data. Using RANSAC scan matcher approach [17] every platform could define where stages got from another platform locate on its own map. After that it updates the map with got measurements and connects them with corresponded nodes.

The map updating operation – an optimisation step – happens every time when a looping closure has been found. The task of this optimization is to find the relative transformation between graph vertexes that are included in a loop to make the first and the last vertexes coincide, and to spread a general error between other vertexes in a loop. In [4] the g2o framework [10] is responsible for this optimization step.

E. Cooperative multi-robot map merging using fast-SLAM

There are several works that describe multi-agent Fast-SLAM [5], [16]. In the paper "Cooperative Multi-Robot Map Merging Using Fast-SLAM"" [5] the visual 3D camera is used as a sensor for mobile platforms. The landmarks are represented as bright colored circles that were put on the walls, so there was no problem in extracting landmarks from the scan. Following this approach agents keep their coordinates and in the beginning of execution have no information about any other platforms.

Each agent runs FastSLAM with particle filter where particles represent the hypotheses about world state. That allows to look at the landmarks as at the independent features with no covariance. Every hypothesis has the estimated probability that shows the “closeness” to the truth world state and the hypothesis with the highest probability is considered to describe the world state. Tracking several hypotheses allows to handle situations when the truth world state was estimated wrongly with the little probability.

Agents run single SLAM independently until any two of them appear in a communication radius. Each of them tracks several particles that contain a vector with coordinates of the platform and visited landmarks. When one platform starts a communication it measures the distance and the angle between communicating platforms, and the second agent does the same thing to use these data for estimation of relative position of platforms.

When a communication begins, a platform receives a message that contains a map estimated by another agent, the position of the sender and an observation of itself made by the sender. To send a map the agent should fuse all the particles in one to reduce the time of imposing maps built by two agents. A received map is fused with each particle that takes $O(NM^2)$ where N is an amount of particles and M is an amount of landmarks. Then the method of the nearest neighborhood is used to estimate duplicate landmarks.

The algorithm was tested both on a simulated data and in a real environment. To test on a real data there was a task to make agents to recognize each other in the video camera. To reach this the platforms were equipped with colored marks. Authors of the paper fairly note that the robust algorithm of recognizing of other platforms have to be updated because there might be no opportunity to mark agents with visual bright parts in a real conditions.

IV. Evaluation

The accuracy of a SLAM algorithm can be estimated by a comparison of an output trajectory with the ground truth one. The ground truth trajectory presents a set of positions in the real world where a platform has been located. It is not a simple task to find this values because if there was a truth position of an observer, there would not be a location task. Thus an algorithm’s evaluation faces with the ground truth existence problem every time when they are tested in the real environments.

The problem of the ground truth absence exists for datasets. For example to the best of our knowledge willow garaje dataset [19] are not provided with ground truth. At the same time MIT dataset [18] provide the solution of localization task on the known map instead of ground truth trajectory. Thus the evaluation of SLAM using ground truth might be complicated.

The way to extract the ground truth from a measurement sequence is to simulate these measurements and corresponding errors. Following this the ground truth could be directly provided and it becomes possible to get quantitatively estimation. The world simulation could not be used to get complete evaluation because there it could not provide a detail-full description of the real environment. Moreover the measurement errors, which are approximated as a Gaussian white noise in the simulation, could include more sources and has non-Gaussian structure under the real conditions.

Authors of considered papers evaluate their algorithms quantitatively in three ways:

- do not present this evaluation;
- provide a root-mean-square error (RMSE) with ground truth in a simulation;
- provide the RMSE in the real world.

That algorithms which are not provided with quantitative evaluation are estimated in other ways qualitatively. This variety makes general algorithms evaluation more complicated, thus there is no way to extract huge advantages of one algorithm over another. Table I presents the RMSE values of considered algorithms that could be extracted from corresponding papers.

TABLE I. RMSE values

Algorithms	Trajectory RMSE, m		World
The algorithms of III-A	not presented		
The algorithms of III-B	not presented		
The algorithms of III-C	0.1		real
The algorithms of III-D	2 platforms	1.404	simulation
	4 platforms	1.572	
	8 platforms	1.884	
The algorithms of III-E	0.9		simulation

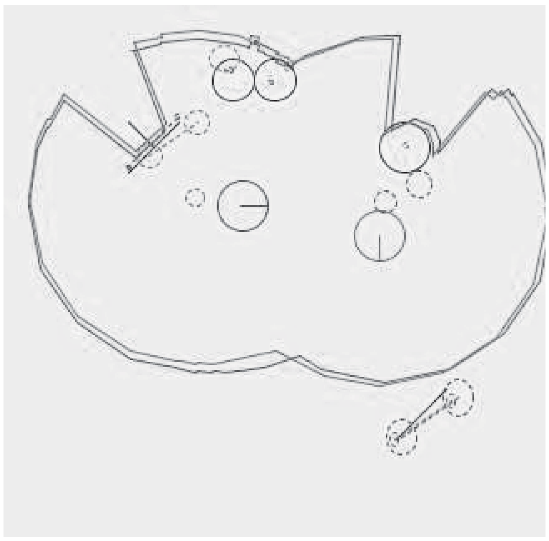
The value of RMSE got from the algorithm of III-C seems to be pretty close to the zero and moreover is provided from the real world. This could be explained with the structure of the world. In [3] it is said that it provides this estimation in the indoor 8x8 meters room and uses a monocular camera as an input sensor. The camera could observe a huge part of this environment and it is possible to extract almost all features of the world and decrease errors of landmarks’ relative orientations. Measurements of other algorithms are provided in bigger environments and observed from longer trajectories so they have bigger values of RMSE.

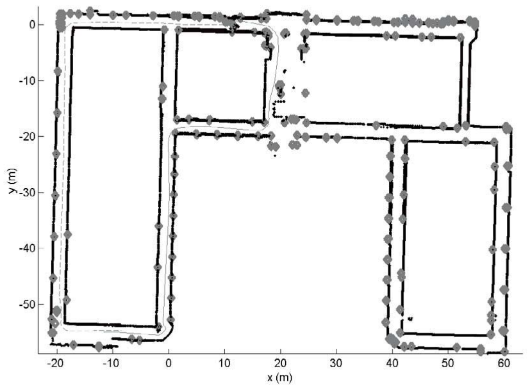
Instead of quantitatively estimation of SLAM execution results there is an approach to estimate them qualitatively. It is possible to estimate the consistencies of built maps. If a map has huge blur or unexpected artifacts, it means that the corresponding approach fails on this data. The figures of the maps (Fig. 5), as well as RMSE, present the simulation world, and the real world if corresponding algorithms were evaluated there.

Fig. 5a represents an output of merging two scans and two different maps of an environment are built. Fig. 5b has artifacts in the middle of the map and in the bottom side, where walls are not matched. The output map presented on Fig. 5c is not a built map but the real one observed from a satellite and used as the ground truth. Trajectories are the output that are superimposed on the map. The walls on Fig. III-D are blurred so errors are hidden there. The map of a room presented on Fig. 5e has the mismatched right wall.

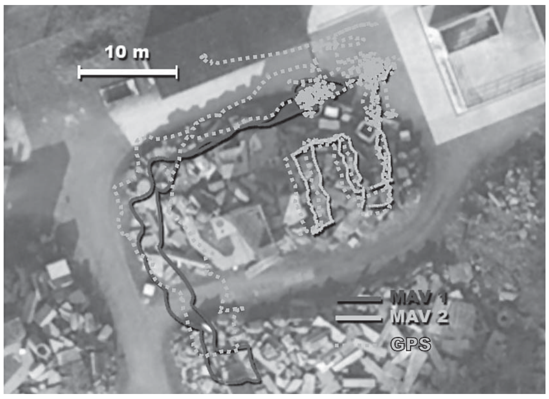
V. Conclusion

In this paper five different approaches for solving the multi-agent SLAM problem were considered. They were chosen for evaluation because each of them has specific features in

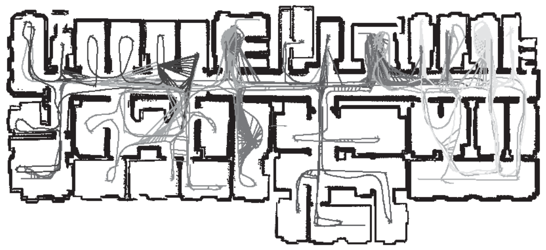

(a) The III-A algorithm



(b) The III-B algorithm



(c) The III-C algorithm


(d) The III-D algorithm

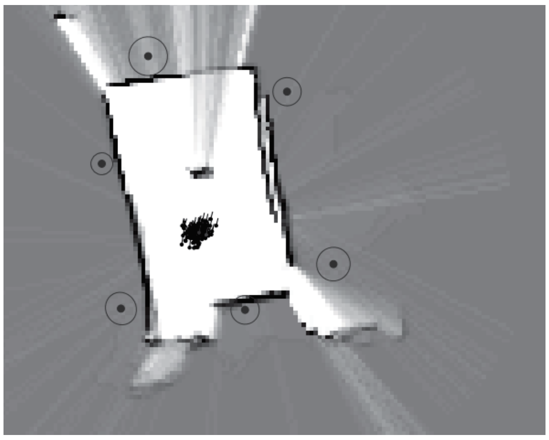

(e) The III-E algorithm

Fig. 5. Maps of algorithms from corresponding papers

an implementation. For example in [5] it is noted that the detection of other platforms on a scan is the very complicated task. In considered papers authors solve this problem on different levels of abstraction. In certain works platforms are marked with bright features that can be easily extracted with a video camera, in other agents don't communicate to each other directly, the task of communication is solved by a supervisor.

The evaluation of described algorithms is provided by a compilation of results presented in the corresponding papers. For the best comparison algorithms ought to be evaluated using one testing data, but considered multi-agent SLAMs have different test bases. The RMSE value and output maps were extracted and combined in common table with notes whether tests were executed in a simulation or the real world. This variety of environment representations makes it difficult to rank them and to choose which of algorithms could produce the most accurate results. However several features are described that are helpful for implementation of any multi-agent SLAM approach:

- The communication of agents for merging maps should happen when they meet that makes them to see the same scans at the moment of communication, thus the process of a relative map orientation evaluation is simplified.
- If after a communication one of the agents clarifies its location it should review its previous path and update the map built on previous steps.
- The final map can be presented in two ways: each agent can keep its own map or there can be the single map that is divided between the agents.

The future goal is to implement a multi-agent SLAM algorithm that includes the most benefits of the considered papers. For example, a server that handles all data from mobile observers may decrease the cost of communications and make the world state unique for all agents. Therefore the SLAM algorithm is executed only by a server while other agents should make observations of an environment and provide them timely. This architecture requires the high performance from a server because it should execute a SLAM algorithm in the real time.

Secondly, it is required to have an opportunity to discard the previous changes of a map if they conflict with many new observations. The most suitable considered approach uses a graph-based algorithm that includes optimization and loop closure steps that may widely effect the map. But the original graph based approach take much computation resources that are limited due to an amount of observers that were mentioned above. So the task is to describe an algorithm that could be executed in the real time.

VI. ACKNOWLEDGMENT

Authors would like to thank JetBrains Research for provided support and materials for working on this research.

REFERENCES

[1] E. Howe and J. Novosad, “Extending SLAM to Multiple Robots”, March, 2005.

[2] X. S. Zhou and S. I. Roumeliotis , “Multi-robot SLAM with unknown initial correspondence: The robot rendezvous case”, *in Proc. of 2006 IEEE International Conference on Intelligent Robots and Systems*, pp. 1785-1792, 2007.

[3] L. Kneip, Ch. Forster, S. Lynen, D. Scaramuzza, “Collaborative monocular SLAM with multiple Micro Aerial Vehicles”, *in Proc. of 2013 IEEE International Conference on Intelligent Robots and Systems*, pp. 3962-3970, 2013.

[4] M. T. Lazaro, L. M. Paz, P. Pinies, J. A. Castellanos, G. Grisetti, “Multi-robot SLAM using condensed measurements”, *in Proc. of 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1069-1076, 2013.

[5] N. Ergin Ozkucur and H. Levent Akin, “Cooperative Multi-Robot Map Merging Using Fast-SLAM”, *in RoboCup 2009*, vol. 5949, pp. 449-460, Heidelberg:Springer, 2010.

[6] G. Zunino; H. I. Christensen, “Simultaneous localization and mapping in domestic environments”, *Conference Documentation International Conference on Multisensor Fusion and Integration for Intelligent Systems. MFI 2001*, pp. 67-72, 2001

[7] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, “Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges”, *in Proc. of the 18th international joint conference on Artificial intelligence*, pp. 1151-1156, 2003.

[8] A. I. Eliazar, R. Parr, “DP-SLAM 2.0”, *in Proc. of the IEEE International Conference on Robotics and Automation*, pp. 1314-1320, 2004.

[9] G. Grisetti, R. Kümmerle, C. Stachniss, W. Burgard, “A Tutorial on Graph-Based SLAM”, *IEEE Intelligent Transportation Systems Magazine*, vol. 2, pp. 31-43, 2011.

[10] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, W. Burgard, “G^2o: A general framework for graph optimization”, *in Proc. of IEEE International Conference on Robotics and Automation*, pp. 3607-3613, 2011.

[11] W. Hess, D. Kohler, H. Rapp, D. Andor, “Real-time loop closure in 2D LIDAR SLAM”, *in Proc. of 2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1271-1278, 2016.

[12] M. S. Nixon, A. S. Aguado, “Feature Extraction and Image Processing”, London: Elsevier, Second edition, 2008.

[13] A. Huletski, D. Kartashov, “A SLAM research framework for ROS”, *Proceedings of the 12th Central and Eastern European Software Engineering Conference in Russia*, Article No.12, 2016.

[14] C. Evans, “Notes on the OpenSURF Library”, University Bristol, 2009.

[15] E. Mouragnon, M. Lhuillier, M. Dhome, F. Dekeyser, and P. Sayd, “3D Reconstruction of Complex Structures with Bundle Adjustment: an Incremental Approac” *in Proc. of 2006 IEEE IEEE International Conference on Robotics and Automation*, 2006.

[16] L. Carlone, M. Kaouk Ng, J. Du “Rao-Blackwellized Particle Filters multi robot SLAM with unknown initial correspondences and limited communication” *in Proc. of 2010 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 243-249, 2010.

[17] M.A. Fischler, R.C. Bolles, “Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography”, Communications of the ACM, 24(6):381-395, 1981.

[18] M. Fallon, H. Johannsson, M. Kaess, J. Leonard “The MIT Stata Center dataset”, *The International Journal of Robotics Research*, vol. 32, no. 14, pp. 1695-1699, 2013.

[19] J. Mason, B. Marthi “An object-based semantic world model for long-term change detection and semantic querying”, *in. Proc of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3851-3858, 2012.