Proceedings of the 2015
IEEE Conference on Robotics and Biomimetics
Zhuhai, China, December 6-9, 2015

Cooperative SLAM on Small Mobile Robots

Nicolai Waniek, *Student Member IEEE*, Johannes Biedermann, Jörg Conradt, *Senior Member IEEE*

***Abstract*— We present a method for simultaneous localization and mapping for robots. The focus of our work is to use multiple small mobile robots which have only limited sensing and computational resources. Our robots each uses a laser pointer and an event based vision sensor to compute distances to its surroundings. The acquired data is used to update an occupancy map which can be shared among many robots at the same time. Here we demonstrate initial results for our proof-of-concept implementation. It runs in real-time using a mobile robot platform with an event based vision sensor for data acquisition. Distance estimates to objects are transferred to a remote computer to build a map of the environment. The results of our work can be used in future technical implementations, and for further investigations into cooperative mapping.**

I. INTRODUCTION

Mapping an environment, computing distances, and goal-directed navigation are ubiquitous activities in the animal kingdom. Honey bees for example are able to locate food and safely navigate back to their hive. There, they discharge their cargo, share the approximate distance to and direction of the food source, and finally fly back to that location to collect more pollen.

Such a behavior inspired science fiction authors and researchers alike for many years. Swarm dynamics gave rise not only to many good novels but also to prominent methods like ant colony optimization. In addition, a recent remarkable demonstration showed how biologically-inspired methods can be used for large quantities of miniature-sized robots to cooperate and form global structures [1].

Although it is debatable if honey bees build maps or not, we were inspired by their collective behavior to address an issue that is still under investigation. How to map an environment with multiple robots at the same time, given that they have to be small, require just a minor amount of power to run for long operation times, use but a few sensors, and have access to only limited computational resources.

Here we present our results towards cooperative mapping using such a restricted mobile system. It consists of a biologically-inspired vision sensor that works akin the human retina and uses this data to solve the problem of simultaneous localization and mapping (SLAM) with the help of a particle filter. We show that we can extend our method to multiple robots, which then cooperatively chart an environment.

First we will highlight related work in Section II, then we will describe the hardware and calibration procedure in Section III. Subsequently, Section IV details the employed algorithms, followed by a description of the experiments and initial results in Section V. We will conclude in Section VI with hints towards future work.

Neuroscientific System Theory, Technische Universität München, 80333 München, Germany. E-mail: nicolai.waniek@tum.de

II. RELATED & PREVIOUS WORK

SLAM for robots is such a vital task. We will thus only highlight a few methods, namely those solutions that are widely used, and some of their extensions.

Many methods use Kalman filters or particle filters to estimate the posterior distribution for the robot pose [2]. Common to most approaches is to store the full posterior and maps of the environment in so called grid maps.

FastSLAM [3] proposed to combine particle filters and Kalman filters recursively to compute the proposal distributions. This approach was highly successfully, and the computational speed can be improved drastically by reducing the number of particles for the cost of increased memory consumption [4].

A variety of other techniques were proposed to counteract the issue that grid map building with particle filters requires huge amounts of memory. For instance, VecSLAM [5] uses a laser range finder to detect obstacles and builds a graph-based representation with the help of vector segments. As a result, the memory requirements are significantly reduced. A general introduction to graph-based methods for mapping can be found in [6].

Many biologically-inspired methods are based on RatSLAM [7]. It solves SLAM by an abstract version of neurons found in the rodent hippocampal formation. The abstract cells, called pose cells, are simulated with a recurrent attractor network that integrates odometric cues. The network activity is then linked to visual data, which in turn is used to detect and solve loop-closures. RatSLAM can map tremendous amounts of space (up to 66km [7]) and is in general suitable for robots with enough processing power. It is, however, unsuitable for small and limited agents that should operate cooperatively.

Recently, a biologically plausible method called HiLAM was proposed that uses a more accurate representation of grid cells and place cells for mapping and path planning [8]. It uses a hierarchical form of grid cells to map the space that an agent is living in, and multi-scale versions of place cells to efficiently plan paths to targets.

None of the research mentioned above addressed cooperative mapping, although the issue is not novel[9]. In fact, recent improvements in graph-based methods show significant progress in the quality of results (see for example [10]). However, almost all of the implementations suffer from at least one of the following issues despite the recent progress: i) they need lots of computations to perform accurately,

978-1-4673-9675-2/15/$31.00 © 2015 IEEE

especially if they process regular camera images as input data, ii) they need lots of memory to store a map or, most importantly, iii) they cannot easily be extended to cooperative mapping with a multi-robot setup.

Recently we demonstrated how to use a biologically-inspired sensor to achieve real-time performance for SLAM using a particle filter on an embedded system [11], [12]. A drawback of the method is the placement of the sensor, which has to look at the ceiling that has to be in a specific distance to the robot. Improving the method by a depth sensor to obtain 3D information yields excellent results [13]. It suffers from the depth sensor's size and its power consumption, though, which limits its application on small mobile robots.

Hence, one remaining issue is to efficiently compute distances. Brandli et al. demonstrated the reconstruction of surfaces with an event based vision sensor[14]. Their approach uses a pulsed laser line and bins the measurements into histograms to reckon distances. In the work presented here we only use a laser pointer. After a calibration step we can directly estimate distances to obstacles.

III. HARDWARE DESCRIPTION

We used a small mobile robot that is equipped with a vision sensor and a laser pointer to address the problems that were mentioned above. In the following, we will describe all important technical parts of the setup as well as the calibration procedure. Calibration is necessary because each robot differs slightly due to the assembly process.

A. Robotic Platform: PushBot

The "PushBot" robots (see Figure 1) are equipped with an embedded Dynamic Vision Sensor (eDVS). This board, which serves both sensor data acquisition and motor control, hosts a Dynamic Vision Sensor (DVS, see Section III-B). It operates using an NXP LPC4337 micro-controller with two 32bit ARM Cortex cores running at $204\,\mathrm{MHz}$. Attached to the back of the robot is a wireless communication module. The sensor has a lens of type BB233 that has an opening angle of $70°$, aperture of F2.0 and a focal length of $3.6\,\mathrm{mm}$.

The robot features a laser pointer at its bottom left. For the sake of simplicity, each of the robots has its individual frequency in the range of $400\,\mathrm{Hz}$ to $600\,\mathrm{Hz}$ at which the laser pointer is active. Reflections of the laser pointer on obstacles can easily be tracked with the eDVS. We can then compute distances to those obstacles because the laser pointer is offset from the sensor's center. Thus, an offset reflection indicates a specific distance. The calibration procedure to turn offsets into distances as well as the tracking algorithm to track blinking light sources are described below.

In addition, the robot has two blinking LEDs attached to its top, one on the front, one on the back. These LEDs are used in a multi-robot scenario to detect other robots. Although the LEDs can be driven at independent frequencies, we keep both LEDs at a frequency of $1000\,\mathrm{Hz}$. Alike the laser pointer, LEDs can easily and efficiently be tracked using the eDVS.

The motors provide information about how many ticks they have turned. For this reason we measure distances

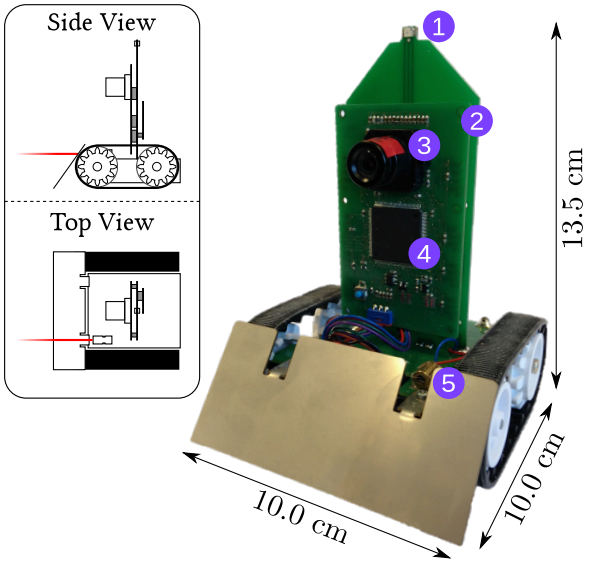

Fig. 1. The PushBot. Our robot platform has the size $10.0\,\mathrm{cm} \times 10.0\,\mathrm{cm} \times 13.5\,\mathrm{cm}$. It is equipped with LEDs on the front and back (1), a DVS control board (2) which hosts a dynamic vision sensor (3) and an NXP LPC4337 µC (4), and a laser pointer (5). The robots communicate using a wireless module attached to the back (not shown in the picture). The inlay shows schematics of the robots, with indicated laser pointer in red.

in motor ticks instead of meters. Although this makes the interpretation of results slightly unfamiliar, transformations from ticks to another system are rendered redundant and can be skipped in all steps. The calibration can easily be expressed in terms of motor ticks, too.

B. Dynamic Vision Sensor (DVS)

The Dynamic Vision Sensor (DVS) is an event based vision chip [15]. In contrast to regular cameras, all pixels operate asynchronously and independently from each other. The sensor size is 128×128 pixels. A single pixel measures changes in illumination and transmits an event e_k as soon as the change raises above or falls below a certain threshold (on- and off-events). Consequently, an event e_k is composed of the pixel location $(x, y) \in [0, 127]^2$, polarity information p about the sign of the change in illumination, and is augmented by a timestamp t. As the pixels operate with a temporal resolution of about $10\,\mu\mathrm{s}$, the eDVS produces streams of events $(e_k, e_{k+1}, \dots)$. The event rate for regular scenes is around 30K events per second, but can easily reach multiple 100K events per second.

High event rates can pose a challenge for the communication medium that is used to transfer events. For instance, using multiple eDVS at the same time within a wireless connection infrastructure to transport events to a remote computer will quickly saturate the network. It is therefore important to move as much computation as possible to the eDVS's embedded microcontroller.

C. Calibration Method and Calibration Results

As mentioned above, we measure distances in motor ticks and not in meters (100 ticks $\approxeq 1\,\mathrm{cm}$). However it is necessary

to establish the mapping between laser pointer position in the input data stream and the motor ticks on a per-robot basis. This is due to small variations introduced during the assembly of the robots. Notably the position of the laser pointer may differ slightly from robot to robot. Detecting the laser pointer in the event stream is described in Section IV-A.

The laser pointer is offset to the sensor's center by a small distance both in x and y direction. This yields an offset of the reflected light beam in the event data that is provided by the eDVS. In turn, we can compute an obstacle distance based on the offset of the reflection from the sensor's center. As the relationship between the x and y coordinates is linear in nature, it is sufficient to examine only one of the offsets. We chose to evaluate the y coordinate offset because the physical offset of the laser pointer in the y direction is slightly larger than the one in x direction. This improved distance estimates for robots that are close to obstacles.

To calibrate the system, we simply start the robot at a specific distance from a wall and keep it driving straight towards this wall while measuring the reflection offset. The result of this procedure for one robot is shown in Figure 2. As expected, the offset follows a logarithmic relationship that depends on the robot's distance to the wall.

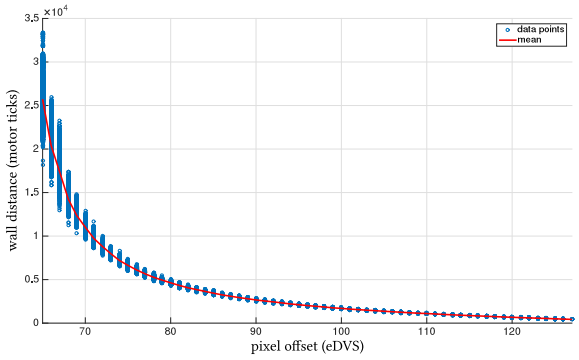

Fig. 2. Calibration measurements. Note that the eDVS pixel indices start at 0 at the top left pixel, but the laser pointer is offset to the bottom left. Hence, the laser pointer reflection will never be seen at pixels below an index of 64. Likewise, the distance towards an obstacle decreases for increasing pixel indices.

We examined the standard deviation of the measurements for all robots to understand the accuracy of the measurements. Again we only show exemplary results for just one robot. As can be seen in Figure 3, the standard deviation quickly increases for obstacles that are further away, corresponding to a smaller pixel offset. Thus the confidence in these measurements decreases. Furthermore we noticed that the laser pointer reflections cannot be seen in arbitrary distances, but are in fact limited to a certain range.

IV. ALGORITHMIC MODULES

Our system is composed of multiple robots and a remote computer. The robots map the environment using their on-board laser pointer by turning 360° and by measuring the distance to obstacles. They transmit the acquired information

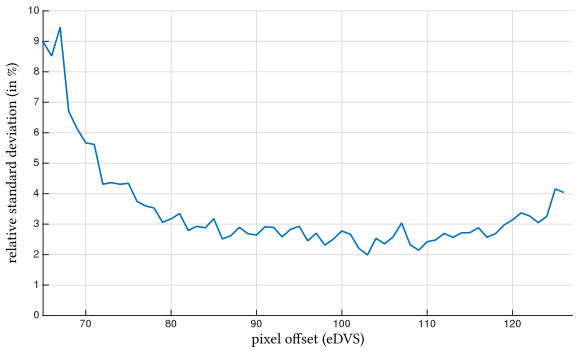

Fig. 3. Horizontal deviation of calibration data. As in Figure 2, the horizontal axis shows the pixel offsets to the sensor's center. The vertical axis shows the standard deviation.

to a remote computer, where it is used to build an occupancy map. The map in turn helps to estimate the robot locations using a particle filter, similar to [2]. Finally, the robot location is used to build a map of visited locations to steer the robots to novel locations. For all experiments we assumed that the environment in which the robots travel is flat.

A. Light Source Tracking

Estimation of distances to obstacles is performed with the help of the on-board laser pointer. As described in Section III-C, the event data produced by a laser pointer and the distance to an object has a certain relationship. Hence, we need to find and track the events generated by the laser pointer in the event stream.

Tracking blinking stimuli in an eDVS data stream is a fairly straightforward task due to the eDVS' high temporal resolution and provided that the blinking frequency is known [16]. As described above, the events from an eDVS are augmented with a timestamp, which helps to initially locate a stimulus by simply comparing timestamps. As soon as a stimulus is located at position $\boldsymbol{p}_t$, where t is a discrete temporal index, it can be updated by subsequent events from the eDVS according to

$$\boldsymbol{p}_{t+1} = (1 - \eta \cdot (w_t + w_{\boldsymbol{p}})) \cdot \boldsymbol{p}_t + \eta \cdot (w_t + w_{\boldsymbol{p}}) \cdot \boldsymbol{p}_e \quad (1)$$

where $\boldsymbol{p}_{t+1}, \boldsymbol{p}_t \in [0, 127]^2$ are the next and current tracker positions, respectively. $\boldsymbol{p}_e \in [0, 127]^2$ represents the 2D location of the novel event. w_t and w_p are temporal and spatial weights. Finally, $\eta \in [0, 1]$ describes the magnitude of influence that a new event has. Obviously the tracker position is updated more quickly for a high value of η but is prone to noisy input. Despite different noise levels from individual eDVS, we found that $\eta = 0.5$ gives the best results during our experiments.

The original approach, presented in [16], computes a tracker update for each event. Consequently, the data rate to report tracked items is the same as when just transferring events. As mentioned above, this can become an issue for the network infrastructure. Hence, we continuously run the

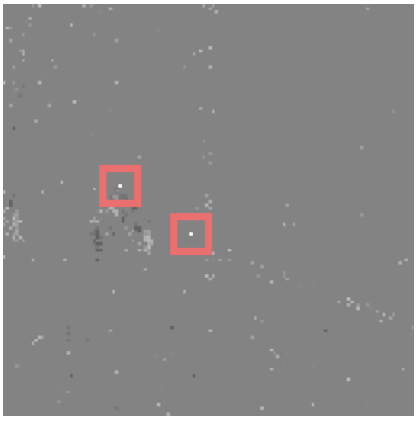

Fig. 4. Visualization of DVS data and tracking results. Bright pixels represent on-events, dark pixels off-events. The tracker identifies two different objects, highlighted by a red box: the left/upper one is another PushBot, the lower/middle object is the robot's own blinking laser pointer. The separation of the different types of inputs is implicit due to the physical arrangement of laser pointers and LEDs on the PushBots. Boxes were added manually, the tracker only reports target locations.

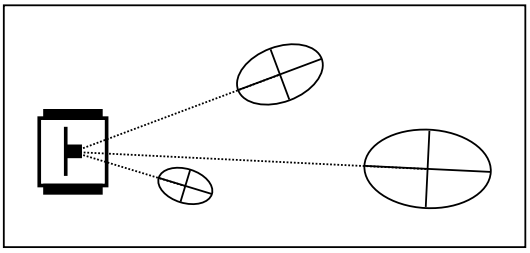

Fig. 5. Top view schematic of a robot detecting three objects. Obstacles influence the Occupancy Map depending on their distance and rotation relative to the robot. The area between the robot and the start of each ellipsoid (dashed line) is negatively weighted, the ellipsoid area is weighted according to a Gaussian distribution.

tracker on board but report the tracked element only every $20\,\mathrm{ms}$. Furthermore we reset the tracker after this time window to compensate for some issues when the tracked object goes out of view. The implementation runs in real-time on the LPC4337, which allows us to drastically reduce the data transfer rate.

We can easily distinguish between laser pointer data and other robots as a result of the physical layout of the PushBots. The LEDs at the top of other robots will always appear in the top half of the sensor, whereas laser pointers will always appear in the lower half. Hence, we simply split the input data into two parts. This implicitly separates the laser pointer and LED data and each can be tracked by its own tracker. Sample results for the tracking are shown in Figure 4.

B. Mapping

Two maps are built using the information provided by the tracker. The first is a grid map, called Occupancy Map, which stores where obstacles like walls or other objects have been detected. The second map, called Visitation Map, stores where the robot has already been according to the pose estimates computed with the help of the particle filter. Both maps have a resolution of 200×200 motor ticks per grid cell, which roughly translates to 2×2 cm.

Given a tracker position $\boldsymbol{p}_t$ and the calibration measurements described above, we can easily translate $\boldsymbol{p}_t$ to a distance d_t in grid units. The free space between an obstacle and the robot is assigned a factor of -0.3. In addition, a positive weighting ellipsoid D is drawn from the two dimensional Gaussian distribution $\mathcal{N}(0, [\sigma_1, \sigma_2])$. Here, $\sigma_1 = 0.1d_t$ and $\sigma_2 = 0.05d_t$. D is then moved to the obstacle location and rotated into its coordinate frame such that D is orthogonal to the robot. A schematic of this approach is shown in Figure 5. Finally, all positive values of the map are normalized to have a maximal value of $+1$, whereas negative areas are kept at -0.3. This normalization procedure helps to minimize the impact of noisy measurements.

Each cell of the Visitation Map counts how often the robot was at the cell's location. Given a pose estimate r_t of the robot's location, we increment the 3×3 area around r_t by one.

C. Particle Filtering

Particle filters are a de facto standard in mapping and as such we will omit a detailed explanation here. The reader is referred to the excellent explanations given in [17] or [18].

The key difference to standard implementations, however, is to augment each particle with a robot ID. Consequently the remote computer only updates those particles which belong to a robot that submits data. Therefore we have multiple tribes of particles "living" on the same map.

D. Path Planning

Our system uses a combination of the Occupancy Map and the Visitation Map to plan a path to the next target. First, we only allow space to be traveled to which is free of any obstacles. Due to the negative weighting of free area in the mapping procedure we can easily select all these parts using the Occupancy Map. Second, we find all local minima in the Visitation Map which reside in the allowed space. Third and finally, we select the local minimum which is closest to the robot as the next target location.

V. INITIAL RESULTS & DISCUSSION

We conducted two main experiments with the robotic system described above. During the first, a single robot had to map the floor of about $12.0\,\mathrm{m} \times 3.3\,\mathrm{m}$ (shown in Figure 6) for $20\,\mathrm{min}$. In the second experiment, two robots had to cooperatively map the same floor for $10\,\mathrm{min}$.

Each experiment was repeated several times to collect data and to answer the following three major questions: i) is the limited robotic setup with only an eDVS and a laser pointer sufficient to map the environment, ii) is the implementation extendable to multiple robots, i.e. can we simply add another robot or does the map degrade due to interfering measurements from multiple robots, and iii) are the results reproducible. Here we report initial results.

Mapping results for one trial of each experiment are displayed in Figure 7. Besides a floor plan, the first row of the figure shows the incremental buildup of the Occupancy Map.

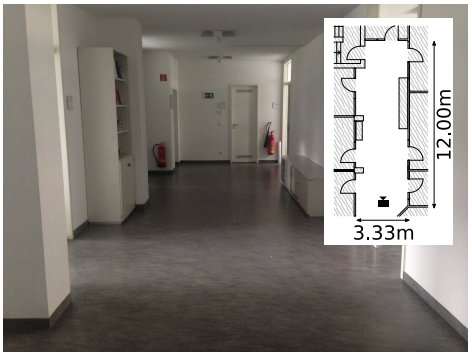

Fig. 6. The hallway that the robots had to map. The inlay shows a floor plan and the location from which the photo was taken. The robots were only allowed to travel the central area, shaded regions on the plan were prohibited.

The second row shows the buildup of the shared Occupancy Map during the second experiment with two robots.

The data demonstrates that the mapping procedure captures the structure of the environment throughout the experiment. Objects like shelves or wall displacements are well preserved. Furthermore, the incremental buildups show that early estimates about distances are already accurate and obstacles are correctly placed. Thus, using a calibrated laser pointer in combination with an event based vision sensor is sufficient for a small robot to build a representation of the environment.

Our implementation runs faster than real-time, which allowed us to perform the second experiment: to map the floor with two robots. Again, the map buildup shows that structures are preserved (bottom row of Figure 7). Like in the first experiment, objects are clearly visibile in the Occupancy Map and the overall arrangement of the floor is preserved. Hence we can state that the approach allows two robots to cooperatively map. However it seems that some parts of the map are less accurately mapped after $10\,\mathrm{min}$ than in the single robot experiment. For instance, the bookshelf on the right side of the floor and the wall on the left side are slightly misplaced. Overall, the map seems to be ”compressed” along the vertical axis, which may be a result of the cooperative mapping. It is yet unclear if the compression effect is introduced by small differences between the robots despite the calibration, though, and future experiments will have to be conducted.

Several repetitions of the experiments generated similar results to the one presented in Figure 7.

Careful inspection of the shape of the Occupancy Map reveals that it is slightly curved. As the curve was convex in all trials, we assume that the reason is due to the data collecting process. To estimate the distances, the robots turn clockwise to collect a full 360° of data. Wheel slipping during this process may introduce noise which then leads to the curvature effect.

Given the results we can answer the initial questions stated above: The limited robotic system is in fact suitable to cooperatively and reproducibly map an environment.

VI. CONCLUSIONS AND FUTURE WORK

In this paper we show how to use small robots to cooperatively map an environment. The method employs a particle filter to estimate robot locations and builds a shared occupancy map. We also provide calibration methods for the robots to learn the relationship between motor ticks and data coming from the biologically-inspired vision sensor. Using a basic path planning strategy the robots are able to navigate within the mapped area and select novel target locations.

The initial results are very promising. In fact, using only a laser pointer and an event based vision senor yields remarkably good results despite the low resolution of the sensor. In addition, our solution performs faster than real-time and is extensible to more agents by simply adding them. In principle, the system can handle an arbitrary number of robots to cooperatively map at the same time.

Some issues need to a addressed, though. For instance we need to identify the origin of the map curvature and, in this context, compensate wheel slipping. As visual feature extraction for event data is still unsolved, we cannot use standard computer vision methods to compute ego-motion and remedy slipping wheels. Furthermore we will examine results for large numbers of cooperatively mapping robots.

We are investigating different representations of the map to improve our method. At the moment, we use only one central occupancy map, but a sparse map would allow to move some parts of the computations to the robots. For instance, a graph-based representation could be cut into sub-maps. A robot traveling in a certain area would need access to only one specific sub-map. Given that the robot could then evaluate its location on its own, it would transmit just novel data to the central computer. This computer in turn would maintain a global view on all agents and identify overlapping regions.

Finally, we study how multiple robots can communicate. At the moment, the required steps to add another robot to the mapping process are manually adjusting the laser pointer frequency and tag of the particle filter. Robots that can communicate with each other will be able to automatically adapt their settings. This will remove the manual step and allow dynamic addition and removal of robots.

ACKNOWLEDGMENTS

Nicolai Waniek and Jörg Conradt are supported by the EU-FET grant GRIDMAP 600725.

REFERENCES

[1] M. Rubenstein, A. Cornejo, and R. Nagpal, “Programmable self-assembly in a thousand-robot swarm,” *Science*, vol. 345, no. 6198, pp. 795–799, 2014. [Online]. Available: http://www.sciencemag.org/content/345/6198/795.abstract

[2] K. P. Murphy, “Bayesian Map Learning in Dynamic Environments,” in *Neural Information Processing Systems*, 1999, pp. 1015–1021.

[3] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, “Fastslam: A factored solution to the simultaneous localization and mapping problem,” in *In Proceedings of the AAAI National Conference on Artificial Intelligence*. AAAI, 2002, pp. 593–598.

[4] G. Grisetti, G. Tipaldi, C. Stachniss, W. Burgard, and D. Nardi, “Fast and accurate SLAM with Rao-Blackwellized particle filters,” *Robotics and Autonomous Systems, Special issue on Simultaneous Localization and Map Building*, vol. 55, no. 1, pp. 30–38, 2007.

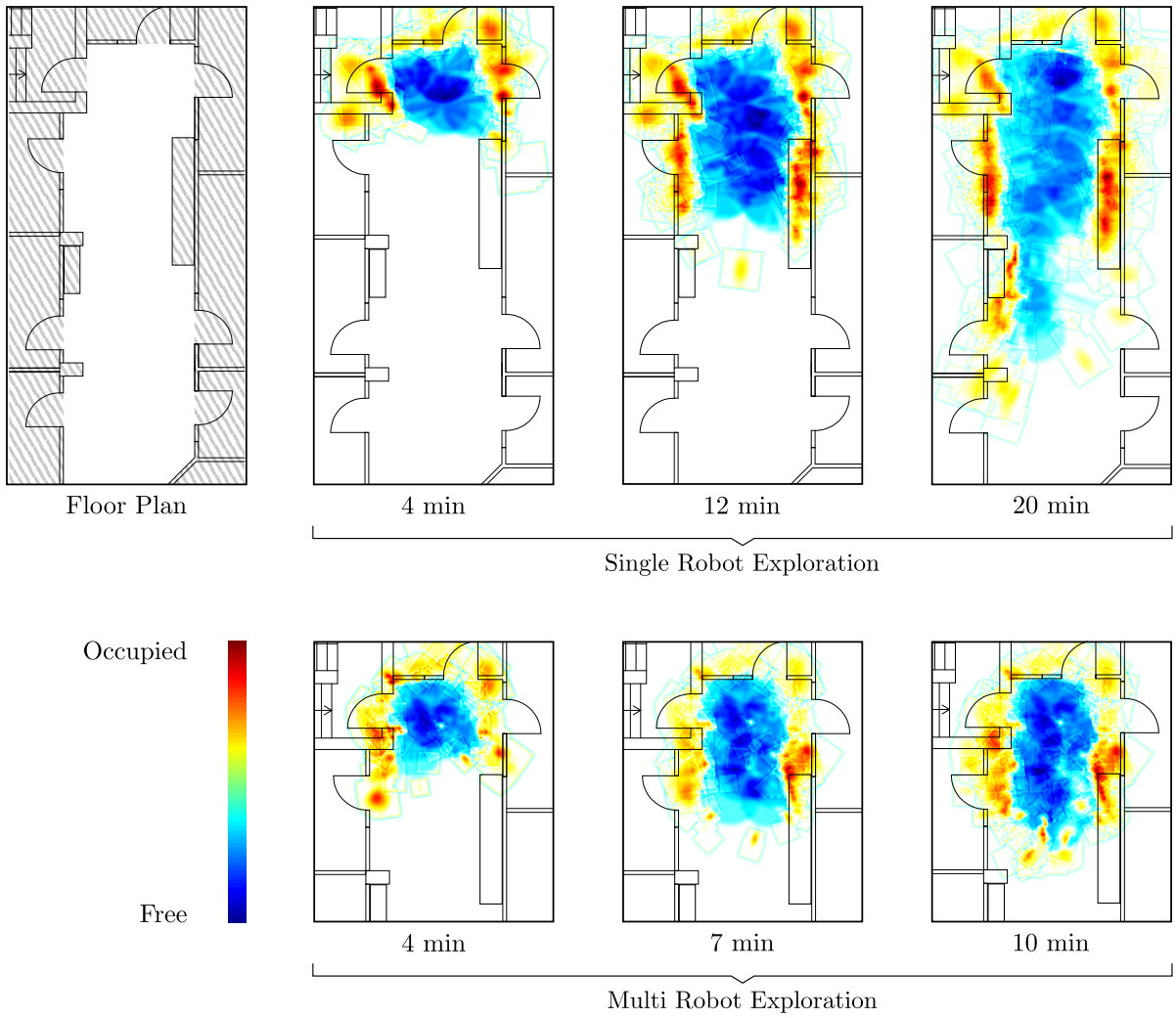

Fig. 7. Mapping results. Top row shows floor plan for reference and incremental Occupancy Map buildup for a single robot, exploring for 20 min. Bottom row shows results for two robots cooperatively mapping for 10 min. Only relevant parts of the map are shown in the bottom row.

[5] H. Sohn and B. Kim, "Vecslam: An efficient vector-based slam algorithm for indoor environments," *Journal of Intelligent and Robotic Systems*, vol. 56, no. 3, pp. 301–318, 2009. [Online]. Available: http://dx.doi.org/10.1007/s10846-009-9313-2

[6] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *Intelligent Transportation Systems Magazine, IEEE*, vol. 2, no. 4, pp. 31–43, winter 2010.

[7] M. Milford and G. Wyeth, "Persistent navigation and mapping using a biologically inspired slam system," *The International Journal of Robotics Research*, vol. 29, no. 9, pp. 1131–1153, 2010. [Online]. Available: http://ijr.sagepub.com/content/29/9/1131.abstract

[8] U. M. Erdem, M. J. Milford, and M. E. Hasselmo, "A hierarchical model of goal directed navigation selects trajectories in a visual environment," *Neurobiology of Learning and Memory*, vol. 117, pp. 109 – 121, 2015, memory and decision making. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1074742714001300

[9] G. Dedeoglu and G. Sukhatme, "Landmark-based matching algorithm for cooperative mapping by autonomous robots," in *Distributed Autonomous Robotic Systems 4*, L. Parker, G. Bekey, and J. Barhen, Eds. Springer Japan, 2000, pp. 251–260.

[10] B. Kim, M. Kaess, L. Fletcher, J. Leonard, A. Bachrach, N. Roy, and S. Teller, "Multiple relative pose graphs for robust cooperative mapping," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, May 2010, pp. 3185–3192.

[11] D. Weikersdorfer and J. Conradt, "Event-based particle filtering for robot self-localization," in *Robotics and Biomimetics (ROBIO), 2012 IEEE International Conference on*, Dec 2012, pp. 866–870.

[12] D. Weikersdorfer, R. Hoffmann, and J. Conradt, "Simultaneous localization and mapping for event-based vision systems," in *Computer Vision Systems*. Springer Berlin Heidelberg, 2013, pp. 133–142.

[13] D. Weikersdorfer, D. Adrian, D. Cremers, and J. Conradt, "Event-based 3d slam with a depth-augmented dynamic vision sensor," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, May 2014, pp. 359–364.

[14] C. Brandli, T. Mantel, M. Hutter, M. Höpflinger, R. Berner, R. Siegwart, and T. Delbruck, "Adaptive pulsed laser line extraction for terrain reconstruction using a dynamic vision sensor," *Frontiers in Neuroscience*, vol. 7, no. 275, 2013.

[15] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128 $\times$ 128 120 db 15 µs latency asynchronous temporal contrast vision sensor," *Solid-State Circuits, IEEE Journal of*, vol. 43, no. 2, pp. 566–576, Feb 2008.

[16] G. R. Müller and J. Conradt, "A miniature low-power sensor system for real time 2d visual tracking of led markers," in *Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on*, Dec 2011, pp. 2429–2434.

[17] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[18] C. Stachniss, *Robotic Mapping and Exploration*, ser. Springer Tracts in Advanced Robotics. Springer, 2009.