Affordable SLAM through the Co-Design of Hardware and Methodology

Stéphane Magnenat, Valentin Longchamp, Michael Bonani, Philippe Rétornaz, Paolo Germano, Hannes Bleuler, and Francesco Mondada

Abstract— **Simultaneous localization and mapping (SLAM) is a prominent feature for autonomous robots operating in undefined environments. Applications areas such as consumer robotics appliances would clearly benefit from low-cost and compact SLAM implementations. The SLAM research community has developed several robust algorithms in the course of the last two decades. However, until now most SLAM demonstrators have relied on expensive sensors or large processing power, limiting their realms of application. Several works have explored optimizations into various directions; however none has presented a global optimization from the mechatronic to the algorithmic level.**

In this article, we present a solution to the SLAM problem based on the co-design of a slim rotating distance scanner, a lightweight SLAM software, and an optimization methodology. The scanner consists of a set of infrared distance sensors mounted on a contactless rotating platform. The SLAM algorithm is an adaptation of FastSLAM 2.0 that runs in real time on a miniature robot. The optimization methodology finds the parameters of the SLAM algorithm using an evolution strategy.

This work demonstrates that an inexpensive sensor coupled with a low-speed processor are good enough to perform SLAM in simple environments in real time.

I. Introduction

The diffusion of consumer robotics appliances would greatly benefit from the integration of SLAM capabilities. Robotic vacuum cleaners, for instance, can radically improve their coverage algorithms using SLAM, as Samsung demonstrated in their Hauzen VC-RE70V. This vacuum cleaner successfully integrates visual SLAM and thus confirms the analysis made by Pirjanian et al. [1]. However, as that paper points out, visual SLAM requires an environment populated with well-illuminated features and a large processing power. Even lightweight visual-SLAM algorithms require laptop-level processing power [2]. Samsung has elegantly fit to these constraints by pointing a camera to the ceiling. This both reduces the dimensionality of the problem to one surface and provides few but robust visual features. The resulting product clearly outperforms competitors. However, some applications do not enjoy such excellent conditions, and must fall back on distance measurements to perform SLAM. For instance search and rescue robots often operate in dark environments with no clear visual features. We therefore believe that there exists a strong need for non-visual SLAM techniques that one can integrate into cheap, small, and low-power robots.

This work was supported by the Swarmanoid and Perplexus projects, both funded by the Future and Emerging Technologies programme (IST-FET) of the European Community, respectively under grants 022888 and 34632. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

Paolo Germano is with LAI-EPFL; the other authors are with LSRO-EPFL. Please send correspondence to Stéphane Magnenat (`stephane at magnenat dot net`), EPFL-LSRO, Station 9, 1015 Lausanne, Switzerland. We thank Basilio Noris for his picture of the experimental marXbot.

II. Related Work

The reliance of most SLAM implementations on bulky and expensive laser scanners hinders their diffusion into mainstream products. Several projects have therefore explored the use of cheap and low-resolution distance sensors.

Schroter et al. [3] used the array of sonar sensors which equips the SCITOS A5 robot. Their work focused on reducing the memory footprint of particle-based gridmap SLAM by sharing the map between several particles. The resulting implementation runs in real-time on laptop-level computers.

Yap et al. [4] also used sonar sensors. They worked with the ActivMedia P3-DX robot, which has less sensors than SCITOS A5. To cope with this sparseness, their SLAM implementation uses a map of line segments instead of a gridmap. Together with a strong assumption that the walls are orthogonal, their solution was able to reconstruct large indoor environments. Their article do not report any performance measurement. In the same direction, Abrate et al. [5] used line extraction to apply SLAM to a Khepera II robot, which only embeds 8 short-range infrared proximity sensors. Like in the work of Yap et al., the environment consists of a small number of orthogonal walls.

These projects are representative of a line of research which focuses on developing SLAM algorithms that fit the features of specific sensors. They all succeed in performing SLAM in loopy environments thanks to robust algorithms. However, these are too computationally intensive to run in an embedded computer, requiring at least laptop-level performances. Gifford et al. [6] have proposed a global approach to address these limitations. They have both designed a robot and implemented a distributed SLAM algorithm which uses a scanning sensor. Their SLAM algorithm uses a particle filter, and they report real-time performances using 15 particles and 3 seconds per update. The authors conclude that their scanner, a simple set of infrared distance sensors mounted on top of a servomotor, does not provide enough information in sparse environments. They also underline the difficulty in finding the right SLAM parameters to fit within the available computational power. Recently, Grzonka et al. [7] performed SLAM experiments on an autonomous indoor flying robot. Albeit they use a laser scanner, their SLAM implementation runs in real-time on the computer of their small flying robot.

In this article, we show how to run SLAM in real time on mobile robots through the co-design of hardware, software,

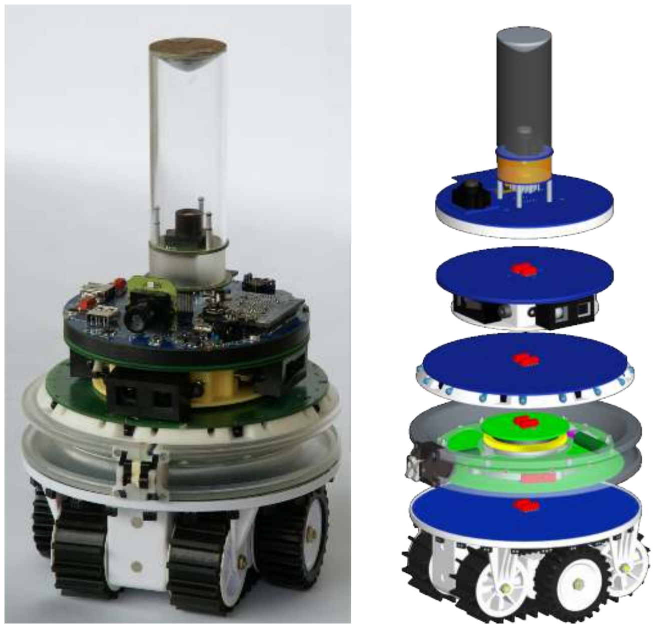

Fig. 1: The marXbot modular mobile robot. The modules are, from base to top: base, connection ring, range and bearing, *rotating scanner*, main processor board with cameras.

and methodology. At the hardware level, a slim rotating distance scanner fits the cost requirements. At the software level, our lightweight SLAM implementation builds upon the work of [8] and runs in real time on smartphone-level processors. At the methodological level, we propose to employ a global optimization algorithm to find the best parameters for the SLAM algorithm.

III. The marXbot robot

The rotating distance scanner that we present in this paper is a module for the marXbot mobile robot (Fig. 1). The marXbot consists of a base of 17 cm of diameter, and various application-dependent modules stacked on top of it. This base provides mobility, energy, and basic sensing. To move, the marXbot uses a pair of treels, a combination of tracks and wheels [9]. These provide good mobility even in rough terrain, at the expense of the precision of odometry. Among other sensors, the base embeds 24 short range infrared proximity sensors and a gyroscope. A 10 Ah lithium polymer battery provides 7 hours of continuous operation when moving around and using the scanner. Several microcontrollers drive the hardware and provide low-level control using the ASEBA framework [10]. A 533 MHz ARM 11 (Freescale iMX31) processor runs Linux to perform high-level control and communicates with the microcontrollers through ASEBA [11].

IV. Hardware of Scanner

A. Mechanical design

To fulfill the requirements of compactness, limited power consumption, and low cost; we decided to design our own rotating distance scanner. Using a commercially available laser scanner would not fit the compactness nor the cost

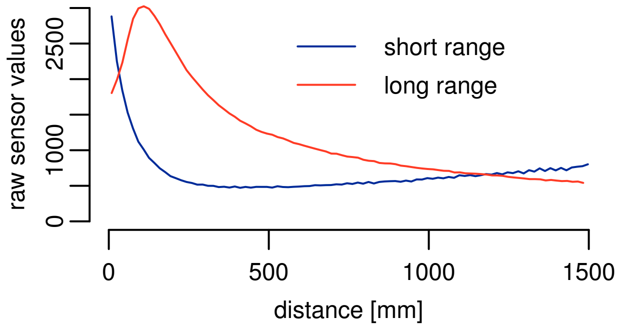

Fig. 2: The response functions of the short and long range sharp sensors.

specification	value
diameter	130 mm
height	29 mm
weight	220 g
power consumption	2 W
cost	390 USD
infrared sensors	2× GP2Y3A001K0F (4–30 cm)
	2× GP2Y3A002K0F (20–150 cm)
global operating distance	4 to 150 cm
maximum scan speed	2 scans/s
angular resolution	3° at 1 scan/s, 6° at 2 scan/s

TABLE I: Characteristics of the rotating distance scanner.

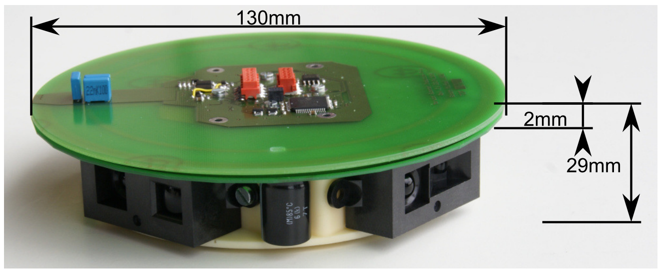

Fig. 3: Overview of the rotating distance scanner module.

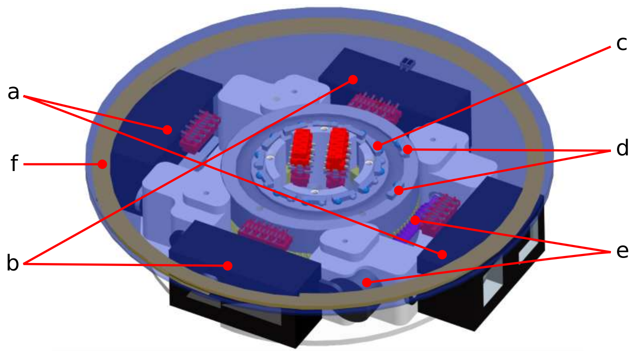

Fig. 4: CAD drawing of the scanner, with semi-transparent PCB. **a:** long range sharp sensors, **b:** short range sharp sensors, **c:** infrared LEDs for data transmission (fixed, 12×), **d:** infrared LEDs for data transmission (rotating, 2×), **e:** rotating motor with a worm gear drive, **f:** induction coils, directly on the PCBs

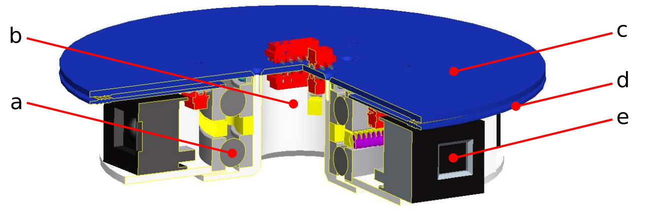

Fig. 5: CAD drawing of the scanner, cut in a 3/4 view. **a:** plastic ball bearing, **b:** hole in the center to pass cables to other modules, **c:** fixed PCB (primary coil), **d:** rotating PCB (secondary coil), **e:** distance sensor

requirement. Fig. 3 shows the final version of the scanner and TABLE I shows its characteristics. Our design is based on 4 infrared sharp distance sensors mounted on a rotating platform. These sensors have a limited range and a dead zone close to the device (Fig. 2), so we couple two sensors of different ranges (40–300 mm and 200–1500 mm) to cover distances up to 1500 mm. The platform rotates continuously to make 360° scans; as it embeds two sensors of each type, the robot gets a full scan every 180°. A motor with a worm gear (Fig. 4) drives the rotation while two plastic ball bearings ensure the guidance. The motor is located in the rotating part, to ease the synchronization between the platform position and scanner's values. This location also fits well within our geometrical constraints and allows for a slim design. To minimize the wear and maximize the life time of the scanner, the fix part transfers energy by induction to the rotating part. They exchange data using infrared light. This solution, albeit more difficult to implement then sliding contacts, is much more reliable and lasting. We have implemented induction directly on two PCB spaced by a gap of 0.8 mm (Fig. 5).

B. *Electronic design*

The electronics of the scanner is distributed between the two PCB (Fig. 6, top). The fixed PCB manages the energy transmission, and the rotating PCB acquires the sensors data and sends them back to the fixed PCB using infrared communication. We can decompose the electronics into two major subsystems: the power and the data transmission. On each PCB, a microcontroller synchronizes the operations of each subsystem.

The single-cell battery of the marXbot provides a voltage in the range of 3.5–4.2 V. The sharp distance sensors demand an input voltage of 5 V. The output voltage from the inductive transfer must thus be higher than 5 V plus the voltage drop in the rectifiers. We designed the inductive transfer for a nominal output voltage before rectifier of 8 V with a peak power transmission of 3.5 W. The primary winding (fixed part) has 2 turns and the secondary winding (rotating part) has 8 turns; the resonant frequency is 228 kHz. We measured an efficiency of $\eta = 0.78$ for the (inductive) transformer with an input voltage of 3.8 V and an output voltage of 5.51 V. The overall efficiency is $\eta = 0.69$, when we consider the H bridge, the rectifier, and the microcontroller of the fixed part.

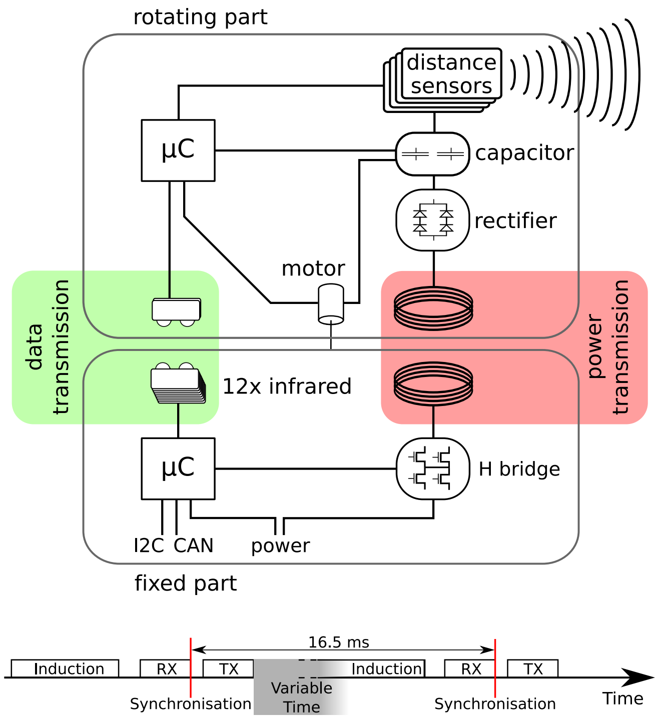

Fig. 6: Electronics of the scanner. Bloc scheme (top) and timing diagram (bottom). In the timing diagram, reception (RX) and transmission (TX) are considered from the point of view of the fixed PCB.

We are satisfied with this efficiency, especially considering that the inductive energy transfer system acts as a voltage step-up for the sharp sensors as well.

The induction generates noise on the power planes. This disturbs infrared communication because its bit rate is 115 kHz, which is close to the frequency of the induction system. This results in erroneous transceivers' outputs. However, we take advantage of large capacitors ($4400\mu F$) that smooth the rectifier's output and thus store energy and shut down the inductive supply during data transmission.

We have implemented bidirectional half-duplex communication between the two PCB using a simple serial transmission with 16-bit cyclic redundancy check. When a microcontroller transmits data, it drives all the infrared transmission LEDs in parallel. The output of the transceivers are simply OR-ed into the RX pin of the destination microcontroller. The rotating PCB sends a message for each sensor acquisition (at 60 Hz) with the 4 sensor values, position of the motor, and the voltage at the output of the rectifier. The latter enables us to regulate the power in the induction's primary coil, to save energy. The fixed PCB sends back a message with the position or speed set-point.

When synchronizing the induction and the infrared communication, a variable time enables us to modulate the power transmitted and to regulate the input voltage of the rotating PCB (Fig. 6, bottom). The target voltage is 6.5 V, which results in a total current consumption of 500 mA at 3.8 V for the whole scanner. The scanner communicates with the rest of the marXbot through a CAN bus using the ASEBA framework.

V. SLAM Implementation

We have adapted FastSLAM 2.0 [8] to the specificities of our hardware. This SLAM implementation estimates the position of the robot and incrementally builds a 2D occupancy-grid map [12] of its surrounding environment. A time step corresponds to a full 360° scan by the rotating scanner (half a turn). Each cell of the occupancy-grid map holds the log odds ratio of the belief that this cell is an obstacle [13]. It consists of a particle filter, where each particle k at each time step t contains the robot position $\boldsymbol{x}_t = (x\ y\ \theta)^T$, the associated weight w_t, and a full map of the environment m_t. The algorithm updates these three values with the new measurements acquired by the scanner in four phases. These phases are: A. the position update, B. the measurement to map matching, C. the occupancy-grid update, and D. the particles resampling.

A. Position update

The rotating scanner only produces enough data for a relevant map matching every half turn. Moreover, we must perform the estimation of the robot position at the same rate as the measurement to map matching. Yet during a half turn of the scanner, the robot receives several odometry measurements. To cope with this discrepancy, we store all the measurements and delay the computation of the robot position. To compute the position, we reconstruct the trajectory by iteratively applying the odometry measurements.

The update of the position $\boldsymbol{x}_t^{[k]}$ knowing the position at the previous time step $t-1$ and the command $\boldsymbol{u}_t$ (odometry measurements) that steered the robot between the two time steps is described by the motion posterior:

$$p(\boldsymbol{x}_t | \boldsymbol{x}_{t-1}^{[k]}, \boldsymbol{u}_t) \tag{1}$$

The marXbot is roughly equivalent to a differential wheeled robot at the level of the motion model. We approximate this motion model by an odometry model in which we decompose the interval $(t-1, t]$ into an initial rotation δ_{rot1}, a translation δ_{trans}, and a final rotation δ_{rot2}. We can directly get δ_{trans} from the motors' encoders as the average of displacement of each treel. However, the tracks introduce non-linear slipping depending on the speed, the acceleration, and the type of surface. The slipping particularly affects the odometry when the robot rotates. We therefore use the gyroscope integrated in the base of the marXbot to measure the changes in orientation.

B. Measurement to map matching

We compute the weight of a particle $w_t^{[k]}$ which is proportional to the likelihood of the measurement $\boldsymbol{z}_t$:

$$w_t^{[k]} \approx p(\boldsymbol{z}_t | \boldsymbol{x}_t^{[k]}, m^{[k]}) \tag{2}$$

To compute the likelihood of each measurement, we project 4 rays oriented like the 4 sensors of the scanner at the time of the measurement onto the particle's internal map and compare the distance measured by the sensor and the one found by reading the map. We back-propagate all the measurements along the trajectory computed at phase A such that matching is done with the estimated robot position at the time of the measurement. The final likelihood is the product of the likelihood of each measurement along the trajectory of the robot in $(t-1, t]$. Since the response functions of the sharp sensors are not injective (Fig. 2), we ignore their values corresponding to invalid distances. The probabilistic nature of the map is sufficient to disambiguate wrong readings from correct ones. We manage to cover the whole $(0, 1]$ m range by dropping the values of short range sensors over 35 cm and the values of long range sensors below 35 cm.

We optimize the robot position knowing the measurement by performing a small Monte-Carlo localization. For each particle, we explore a small space around the final position computed in phase A, following a Gaussian distribution. We perform the measurement to map matching for each candidate position and keep the best match. This operation improves the positioning, and is comparable to having more particles, yet without the memory expense of one distinct map per particle. However, we must project more rays per particle.

C. Occupancy-grid update

We compute the map m which is represented by the posterior:

$$p(m | \boldsymbol{x}_{1:t}, \boldsymbol{z}_{1:t}) \tag{3}$$

We represent m by the set of all grid cells $m = \{m_i\}$, where m_i is a binary variable with $p(m_i = 1)$ representing the probability that an obstacle occupies a cell. This independence assumption allows us to approximate the posterior of the map:

$$p(m | \boldsymbol{x}_{1:t}, \boldsymbol{z}_{1:t}) = \prod_i p(m_i | \boldsymbol{x}_{1:t}, \boldsymbol{z}_{1:t}) \tag{4}$$

For each grid cell, we use the *log odds* representation of occupancy:

$$l_{t,i} = \log \frac{p(m_i | \boldsymbol{x}_{1:t}, \boldsymbol{z}_{1:t})}{1 - p(m_i | \boldsymbol{x}_{1:t}, \boldsymbol{z}_{1:t})} \tag{5}$$

For the sake of efficiency, we update the map for each sensor measurement using a pre-computed update function dependent on the sensor value (Fig. 7). We have pre-computed tables for all sensors values for every sensor (512 KB of data). Like in phase B, we cast a ray from the estimated robot position into the direction of the measure. On this ray, the update function adds information that cells before the measured distance are free of obstacles and that cells at the measured distance are occupied by an obstacle. It adds no information to cells beyond the measured distance. We back-propagate the estimated positions of the measures along the robot trajectory in $(t-1, t]$.

D. Particles resampling

The particles resampling frequency is a parameter of our algorithm. When it resamples particles, the algorithm first sorts them according to their weight. It then draws a new set of particles out of the previous set, with a probability proportional to the weight of the particle. The new set may contain many times the same particle, as particles with a large

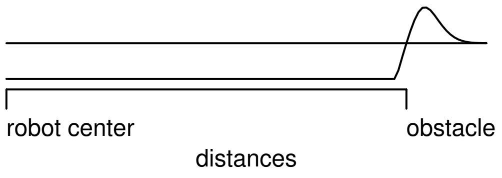

Fig. 7: The update function, whose values are added to the occupancy-grid map.

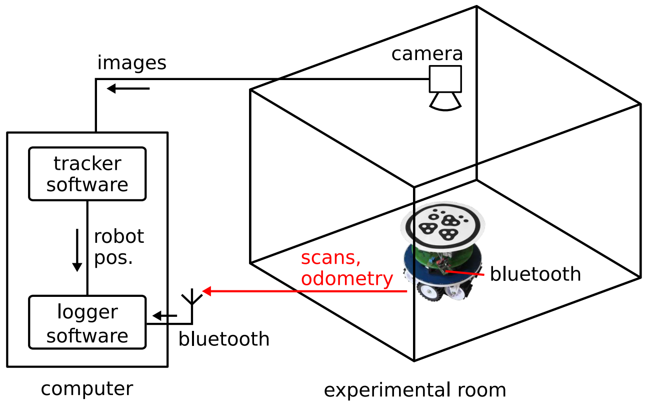

Fig. 8: The experimental setup for SLAM experiments.

weight have a strong probability to be drawn more then once. However, as the position update step introduces randomness, the particles will quickly differentiate.

VI. Experimental Methodology

A. Measuring the quality of SLAM

We run experiments in a room with an overhead camera connected to a robot tracker (Fig. 8). We built the tracker using `libfidtrack` from the reacTIVision project[1] [14]. We measure the quality of the SLAM by comparing the average squared difference between the reconstructed trajectory of the best particle and the real trajectory ($\boldsymbol{T}_{\text{slam}} = \{\boldsymbol{T}^i_{\text{slam}}\}$ and $\boldsymbol{T}_{\text{real}} = \{\boldsymbol{T}^i_{\text{real}}\}$, for i iterating over $S_{\text{T}} = |\boldsymbol{T}_{\text{slam}}| = |\boldsymbol{T}_{\text{real}}|$ tracked positions). However, as both trajectories are expressed in different coordinates, we must first find the set of parameters $\boldsymbol{\theta}_T = \{\theta_\alpha, \boldsymbol{\theta}_d\}$ for the transformation $A(\boldsymbol{T}^i_{\text{slam}}, \boldsymbol{\theta}_T) = \boldsymbol{R}(\theta_\alpha)\boldsymbol{T}^i_{\text{slam}} + \boldsymbol{\theta}_d$ (knowing that $\boldsymbol{R}(x)$ is 2D rotation matrix of angle x) to minimize the distance:

$$\mathrm{d}(\boldsymbol{T}_{\text{slam}}, \boldsymbol{T}_{\text{real}}, \boldsymbol{\theta}_T) = \sum_i^{S_{\text{T}}} (\|A(\boldsymbol{T}^i_{\text{slam}}, \boldsymbol{\theta}_T) - \boldsymbol{T}^i_{\text{real}}\|^2) \quad (6)$$

We find the optimal set $\hat{\boldsymbol{\theta}}_T$:

$$\hat{\boldsymbol{\theta}}_T = \underset{\boldsymbol{\theta}_T}{\mathrm{argmin}}(\mathrm{d}(\boldsymbol{T}_{\text{slam}}, \boldsymbol{T}_{\text{real}}, \boldsymbol{\theta}_T)) \quad (7)$$

The quality of the trajectory is the inverse of the mean of the residual errors:

$$\mathrm{q}(\boldsymbol{T}_{\text{slam}}, \boldsymbol{T}_{\text{real}}) = -\frac{1}{S_{\text{T}}} \sum_i^{S_{\text{T}}} (\|A(\boldsymbol{T}^i_{\text{slam}}, \hat{\boldsymbol{\theta}}_T) - \boldsymbol{T}^i_{\text{real}}\|^2) \quad (8)$$

[1]`http://reactivision.sourceforge.net/`

parameter	default value	mut. σ	best of	best of	best of	best of
ray budget	n.a.	n.a.	8125	16250	32500	65000
dist. error ratio	0.05	0.01	0.13	0.10	0.10	0.12
dist. error const	0.01	0.002	0.014	0.010	0.016	0.006
angle error ratio	0.05	0.01	0.045	0.002	0.064	0.10
angle error const	.01°	0.002°	0.01°	0.01°	0.02°	0.01°
min pos uncertainty	2	0.4	0.41	0.28	0.02	0.05
min angle uncertainty	5°	1°	4.3°	6.9°	8.4°	0.98°
particle resampling f.	1	0.5	1	1	2	5
angle between scans	0	2.5°	1.9°	2.9°	1.8°	3.7°
particle count	1	1	1	1	1	1

TABLE II: Parameters for the SLAM algorithm (left) and their values after optimization (right, best individual of last generation). The particle resampling frequency is irrelevant when the particle count is 1.

B. Optimizing parameters for the SLAM algorithm

The SLAM algorithm depends on multiple parameters (TABLE II). A first set of parameters is related to the error model of the motion model of the robot. They are the constant error, the proportional error, and the minimal uncertainty on position and orientation. A second set of parameters concerns the processing power allocation policy. We have observed that tracing rays on the map consumes most of the processing power (>95 %). Thus to perform SLAM in real time we have a limited ray budget. On the marXbot, when performing 1.5 scan/s, this budget is 65000 rays per scan for a load of 100 %. The parameters related to this budget allocation are the particle count, the minimal angle between scan for measurement to map matching, and the particle resampling frequency.

These parameters affect the quality of the SLAM, but they are not obvious to measure nor compute. We thus propose to learn them from experimental data. Our experimental setup allows the recording of the robot's odometry and scanner data (Fig. 8). We have synchronized the tracker with this telemetry using ASEBA, which allows us to replay any experiment with any set of parameters. We utilize this feature to optimize the set of parameters. To do so, we implement a simple evolution strategy [15] with 25 % elitism. TABLE II gives the standard deviations of Gaussian mutation rate for every parameter.

The quality measure $\mathrm{q}(\boldsymbol{T}_{\text{slam}}, \boldsymbol{T}_{\text{real}})$ is well suited for human interpretation. However, it is highly non-Gaussian: if the robot looses itself during the map creation, the quality will be orders of magnitude worst than in a successful map reconstruction. To alleviate this effect, we let the evolution strategy minimize the following term instead of the quality:

$$\mathrm{e}(\boldsymbol{T}_{\text{slam}}, \boldsymbol{T}_{\text{real}}) = \log(1 - \mathrm{q}(\boldsymbol{T}_{\text{slam}}, \boldsymbol{T}_{\text{real}})) \quad (9)$$

This results in a smoother evolution, because we evaluate each parameter set over 5 recorded experiments in three different environments and take the mean in a log scale.

VII. Experiments and Results

We run 5 experiments of 5 minutes each, in 3 different environments (Fig. 9). We let the marXbot move freely

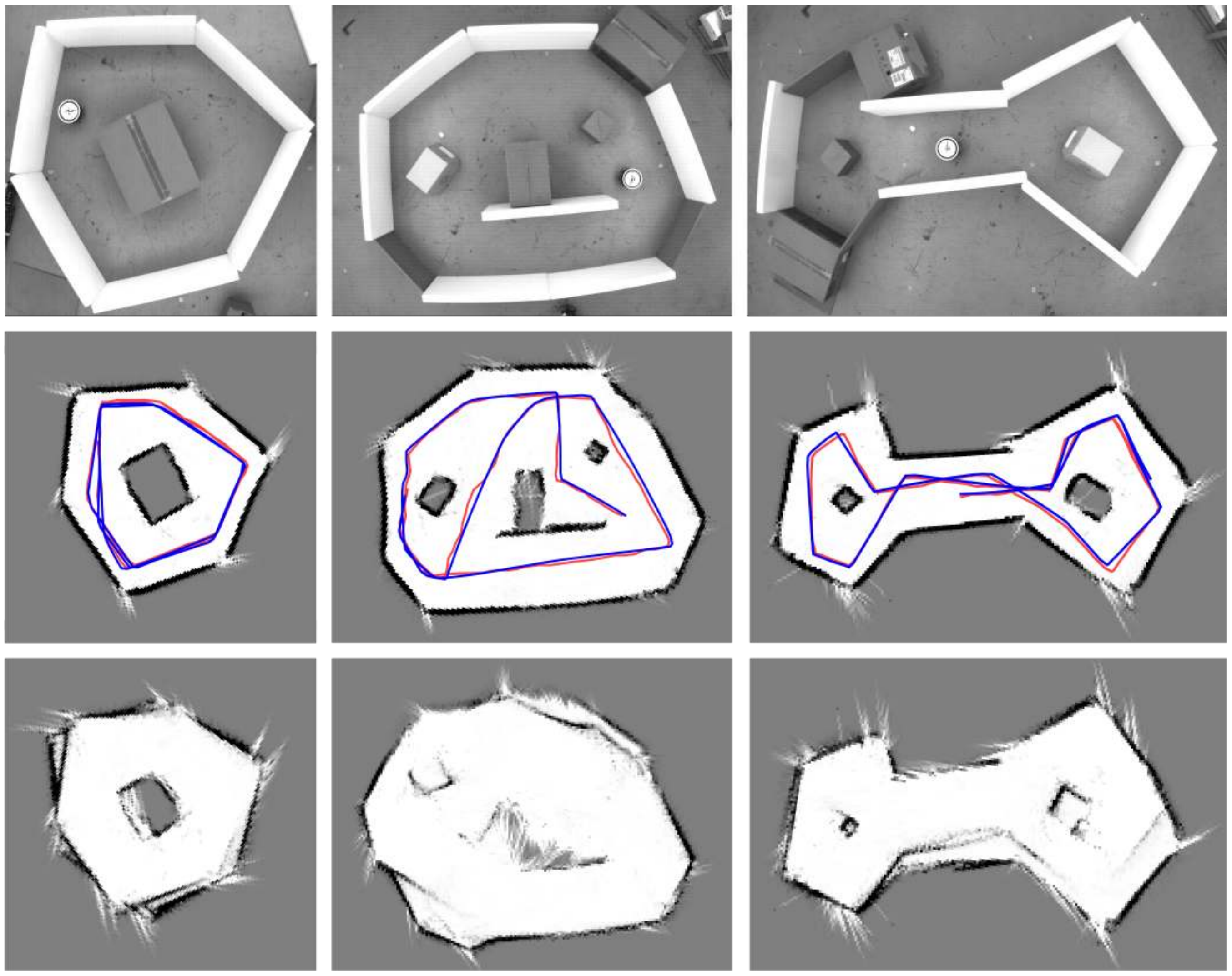

Fig. 9: The different experimental environments (top), and the map built by our SLAM implementation (middle), using a budget of 65000 rays. The SLAM trajectory is in light red while the real trajectory (tracker) is in dark blue. The bottom shows the map reconstruction while ignoring the phase B of our algorithm.

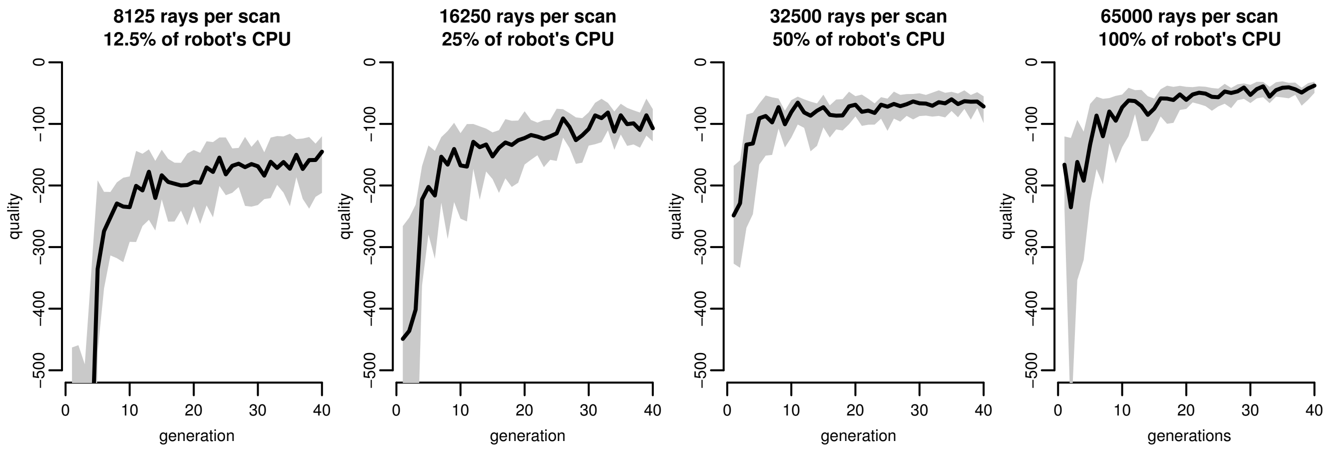

Fig. 10: Optimization of parameters for different ray budgets. We have evolved populations of 48 individuals, over 40 generations, by evaluating each parameter set over 5 recorded experiments for three different environments. The black line represents the median and the gray area represents the interquartile range of quality.

ray budget:	16250	32500	65000
8125 rays; 12.5% of robot's CPU	6.6e-7	2.1e-14	2.2e-16
16250 rays; 25% of robot's CPU		1.0e-4	2.2e-16
32500 rays; 50% of robot's CPU			6.1e-11

TABLE III: P-values of the Mann–Whitney U statistical test between the last generation of evolutions for different ray budgets, for the alternative hypothesis "true location shift is not equal to 0".

and avoid obstacles using its short range proximity sensors. We recorded the robot's scans, odometry, and absolute position from the tracker. We then evolved the parameters corresponding to allocating 1, $\frac{1}{2}$, $\frac{1}{4}$, and $\frac{1}{8}$ of our processing budget to SLAM. As Fig. 10 shows, allocating more processing resources leads to statistically significatively better maps (TABLE III).

The evolution was free to use several particles, to the expense of the quality of the robot position optimization during phase B of the SLAM algorithm. Yet it always kept a single particle, and adapted the minimum uncertainty on position in regards to the available computational power (TABLE II). The more rays were available, the smaller uncertainty the evolution kept. It seems that in our setup, a small number of particles do not hold enough different possibilities to be worth the investment in computational power. However, the robot position optimization reduces the need for particles, as it locally simulates several particles. Nevertheless, we cannot rule out that a longer evolution, with a larger population, and with more experiments per evaluation would lead to the use of more particles. In particular, it would be interesting to allow more computational power and to increase the complexity of the environment to see when multiple particles would get used.

At the qualitative level, we see that all our three environments are well reconstructed (Fig. 9). One exception are the corners, which our scanner tend to see as holes. This is due to the orientation of the sharp sensors and their triangulation-based distance measurement. Corners create reflections which lead to wrong readings from the sensors. We could alleviate this effect by fixating the sensors vertically, but that would triple the height of the scanner. Moreover, we could post-process the grid map knowing that walls are flat [4], and thus work around the problem of the corners.

Several researchers have proposed to take profit of a global optimization algorithm to perform SLAM [16], [17]. However, these works employ the algorithm to estimate the posterior probability distribution over trajectories or maps, which is taken care by our particle filter. To our knowledge, there is no previous work on using an optimization algorithm to find the parameters of the robot's error model and to allocate processing resources.

VIII. Conclusion

In this article, we have demonstrated a SLAM implementation using a low-cost rotating distance scanner. Based on the hardware constraints, we have developed a methodology to optimize the parameters of the software. The optimization has opted for a single-particle SLAM, thus using solely scan matching to build a consistent map. This work demonstrates that an inexpensive sensor coupled with a low-speed processor are good enough to perform SLAM in simple environments in real time.

References

[1] P. Pirjanian, N. Karlsson, L. Goncalves, and E. Di Bernardo, "Low-cost visual localization and mapping for consumer robotics," *Industrial Robot: An International Journal*, vol. 30, pp. 139–144, 2003.

[2] V. A. Sujan, M. A. Meggiolaro, and F. A. W. Belo, "Mobile Robot Simultaneous Localization and Mapping Using Low Cost Vision Sensors," in *Experimental Robotics* (G. E. W. Wolstenholme and M. O'Connor, eds.), vol. 39, pp. 259–266, Springer Verlag, 2008.

[3] C. Schroter, H. Bohme, and H. Gross, "Memory-Efficient Gridmaps in Rao-Blackwellized Particle Filters for SLAM using Sonar Range Sensors," in *Proceedings of the European Conference on Mobile Robots 2007*, pp. 138–143, 2007.

[4] T. Yap and C. Shelton, "SLAM in Large Indoor Environments with Low-Cost, Noisy, and Sparse Sonars," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1395–1401, IEEE Press, May 2009.

[5] F. Abrate, B. Bona, and M. Indri, "Experimental EKF-based SLAM for mini-rovers with IR sensors only," in *Preceedings of 3rd European Conference on Mobile Robots*, European Conference on Mobile Robots, 2007.

[6] C. Gifford, R. Webb, J. Bley, D. Leung, M. Calnon, J. Makarewicz, B. Banz, and A. Agah, "Low-Cost Multi-Robot Exploration and Mapping," in *Proceedings of the IEEE International Conference on Technologies for Practical Robot Applications*, pp. 74–79, IEEE Press, 2008.

[7] S. Grzonka, G. Grisetti, and W. Burgard, "Towards a Navigation System for Autonomous Indoor Flying," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, (Kobe, Japan), pp. 2878–2883, IEEE Press, 2009.

[8] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges," in *IJCAI*, pp. 1151–1156, 2003.

[9] F. Mondada, G. C. Pettinaro, A. Guignard, I. Kwee, D. Floreano, J.-L. Deneubourg, S. Nolfi, L. Gambardella, and M. Dorigo, "SWARM-BOT: a New Distributed Robotic Concept," *Autonomous Robots, special Issue on Swarm Robotics*, vol. 17, no. 2–3, pp. 193–221, 2004.

[10] S. Magnenat, P. Retornaz, M. Bonani, V. Longchamp, and F. Mondada, "Aseba: a modular architecture for event-based control of complex robots," 2009. submitted for publication.

[11] S. Magnenat and F. Mondada, "Aseba Meets D-Bus: From the Depths of a Low-Level Event-Based Architecture," in *IEEE TC-Soft Workshop on Event-based Systems in Robotics (EBS-RO)*, 2009.

[12] A. Elfes, "Using Occupancy Grids for Mobile Robot Perception and Navigation," *Computer*, vol. 22, pp. 46–57, Jun 1989.

[13] S. Thrun, *Probabilistic robotics*. ACM, 2002.

[14] R. Bencina and M. Kaltenbrunner, "The Design and Evolution of Fiducials for the reacTIVision System," in *Proceedings of the 3rd International Conference on Generative Systems in the Electronic Arts*, (Melbourne, Australia), 2005.

[15] H. Beyer, *The theory of evolution strategies*. Springer Verlag, 2001.

[16] T. Duckett, "A Genetic Algorithm for Simultaneous Localization and Mapping," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 434–439, IEEE Press, 2003.

[17] D. J. Feng, S. Wijesoma, and A. Shacklock, "Genetic Algorithmic Filter Approach to Mobile Robot Simultaneous Localization and Mapping," in *9th International Conference on Control, Automation, Robotics and Vision*, pp. 1–6, IEEE Press, Dec. 2006.