2012 International Conference on Indoor Positioning and Indoor Navigation, 13-15th November 2012

A Reference System for Indoor Localization Testbeds

Simon Schmitt, Heiko Will, Benjamin Aschenbrenner, Thomas Hillebrandt and Marcel Kyas
Freie Universität Berlin
AG Computer Systems & Telematics,
Berlin, Germany
{simon.schmitt, heiko.will, benjamin.aschenbrenner, thomas.hillebrandt, marcel.kyas}@fu-berlin.de

***Abstract*—We present a low-cost robot system capable of performing robust indoor localization while carrying components of another system which shall be evaluated. Using off-the-shelf components, the ground truth positioning data provided by the robot can be used to evaluate a variety of localization systems and algorithms. Not needing any pre-installed components in its environment, it is very easy to setup. The robot system relies on wheel-odometry data of a Roomba robot, and visual distance measurements of two Kinects. The Robot Operating System (ROS) is used for the localization process according to a precise pre-drawn floor plan that may be enhanced with Simultaneous Localization and Mapping (SLAM). The system is able to estimate its position with an average error of 6.7 cm. It records its own positioning data as well as the data from the system under evaluation and provides simple means for analysis. It is also able to re-drive a previous test run if reproducable conditions are needed.**

***Index Terms*—Indoor Localization, Robot Operating System, Kinect, Simultaneous Localization and Mapping, Reference System**

I. Introduction

While widely accepted simulations of indoor localization systems and algorithms provide significant insight into their behavior, modeling a sufficient level of detail can often be difficult. But frequent analysis under real-world conditions is essential to optimize new algorithms for best results. Real-world experiments in this domain are hard to evaluate because a Reference System (RS) which provides a more precise localization capability than the System Under Evaluation (SUE) is needed to get ground truth data. With such a system one can evaluate various localization techniques - for example based on time-of-flight measurements. Without such a RS, real-world positions of a SUE would have to be collected by hand, which is difficult and time consuming considering not only the position itself, but also the exact time at which the position was measured or estimated. This is most important if a continuously moving component shall be tracked. The gathered ground truth data and the estimated positions of the SUE can then be synchronized by using timestamps for further analysis. With such a RS, various localization systems, algorithms, and their configurations can easily be evaluated and compared to one another.

In order to evaluate another system, such a mobile RS has to be able to carry the needed components of the SUE. While moving through rooms and corridors, both systems must thereby not interfere in performing their localization. For example, nearby people controlling the RS during a run can disturb a SUE. For that reason, the RS has to be capable of moving autonomous or remote controlled during an experiment. Also, the used radio frequencies and bandwidth must not interfere with the SUE. The RS should not need any pre-installed infrastructure in its environment, e.g. deployed cameras or beacons enabling the localization. Therefore, the RS has to be able to estimate its own position in relation to a given coordinate system. The position estimation error during the performance has to be an order lower than expected errors of a SUE in order to allow a fine grained comparison of different localization approaches.

We present a precise and inexpensive mobile RS capable of performing robust indoor localization on floors in most common office-like environments using Robot Operating System (ROS) [1]. The benefit of this system is that it does not need any pre-installed infrastructure and can easily be used in various buildings. It consists of a Roomba cleaning robot [2] and a laptop which is mounted on top the Roomba. The software running on the computer is able to estimate the robot's position relative to a coordinate frame, which is defined by a pre-drawn floor plan. For the localization, odometry data from the Roomba is combined with visual distance measurements taken by a Kinect depth sensor [3] which is also mounted on the robot. The use of a rack allows the additional mounting of the mobile component of the SUE. The use of open-source software assures a high customizability for hard- and software. We implemented a middleware that allows us to collect data produced by a SUE as well as the ground truth positions from the RS itself. We are able to continuously calculate the robot's position with an average position estimation error of 6.7 cm. The system was evaluated by driving along a grid of known positions while performing self-localization. Furthermore, we show how Simultaneous Localization and Mapping (SLAM) improves localization in rooms essentially, and we argue the need for re-running tests regarding the timing of the initial run.

II. Related Work

Simulation is still dominant when it comes to evaluating indoor positioning algorithms and techniques. When performing real-world experiments in wireless sensor networks, static testing is widely used to measure positions of mobile components.

For example Piras and Cina measured discrete positions along a pre-defined path [4]. They moved a trolley with localization equipment along that path by hand.

Bal, Xue, Shen et al. built a testbed for localization and tracking in wireless sensor networks [5]. They focus on industrial automation environments. Mobile components are mounted to machines, tables, metal shelves and industrial robots. However, they do not automatically collect any ground truth data comparable to a full RS as described.

These approaches do not support mobile testing, which is a requirement for the evaluation process when mobile components are used in a SUE.

Most projects using indoor robotic equipment rely on infrastructural preparation prior to deployment.

For example, Johnson, Stack, Fish et al. use mobile robots with a built-in antenna to evaluate indoor positioning algorithms [6]. They use Emulab, which is a software capable of controlling a network testbed consisting of multiple nodes, and specialized software to control and track the robots. Overhead cameras are used to localize each robot with a mean error of $0.28\,\mathrm{cm}$. However, this is only achieved after preparing and training the tracking system in a constrained environment. They use 6 cameras to track 6 robots in an area of $60\,\mathrm{m}^2$.

Prorok, Arfire, Bahr et al. use among other sensors also an overhead camera to identify multiple robots driving in a constrained environment beneath [7]. The accuracy of this system lies around 1-3 cm, which can also only be achieved after a setup and a calibration stage of the system.

Segura, Hashemi, Sisterna et al. show another system capable of performing localization relying on distributed ultra-wideband modules [8]. The modules emit beacons allowing a robot to perform self-localization with an error under $20\,\mathrm{cm}$.

The drawback of these implementations is their need for some kind of infrastructure. In order to use a mobile RS in common buildings without time consuming preparation or heavy restrictions to space, a fully independent localization system is needed. Another aspect is the usage of ROS, which is not restricted to one scenario only. For example, other robotic hardware could easily be used to allow for higher speeds or a laser rangefinders could be mounted to achieve higher accuracy when measuring distances.

III. Hardware

In order to have a functioning RS at a reasonable expense, we use the low-cost TurtleBot [9] shown in Fig. 1. This TurtleBot consists of a Roomba 531 cleaning robot manufactured by iRobot, Microsoft's Kinect powered by a separate battery and a rack, allowing the user to mount additional components. The Roomba 531 cleaning robot has a differential drive and serves as a basis to a rack allowing for mountable sensors and components of a SUE. A Microsoft Kinect depth sensor is mounted on the rack. While performing localization, a laptop is placed in the rack, that runs the software described in Section IV. The robot is able to drive on any flat floors and over door sills no higher than $0.5\,\mathrm{cm}$. It can sustain a speed of up to $0.5\,\mathrm{m/s}$. The battery pack provides enough power for several test runs in our office building, where it drove a distance of approximately $200\,\mathrm{m}$. The dimensions of $33\,\mathrm{cm}$ in diameter and $40\,\mathrm{cm}$ in height are large enough to carry small mobile components of a SUE.

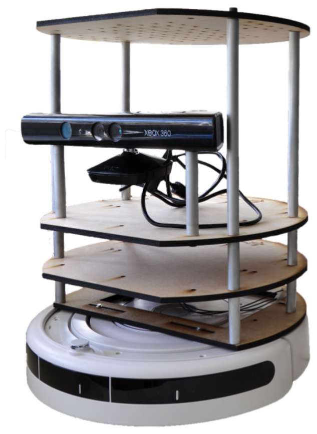
Fig. 1: TurtleBot: Roomba 531 with a mounted Microsoft Kinect.

With the Kinect, Microsoft offers a low-cost depth sensor which is able to estimate distances from about $50\,\mathrm{cm}$ up to $5\,\mathrm{m}$ [10]. We measured a viewing angle of about $58°$. While the accuracy of a Kinect is very high, the distance error of a measurement is proportional to square of the distance [11]. For example, at a distance of $2.5\,\mathrm{m}$ the error is about $4.5\,\mathrm{cm}$, not considering $5\,\%$ of the greatest outliers. At a distance of $5\,\mathrm{m}$, the error is expected to be approximately $19\,\mathrm{cm}$, which is significant. In order to account for that error and to refine the localization and make it more robust, we added a second Kinect to the TurtleBot that points backwards. This ensures that the RS always has objects in its view, even if it is directed in a room larger than $5\,\mathrm{m}$ in diameter, because the rear Kinect points to the wall behind it. This also accounts for people walking along covering one Kinect's view temporarily. As another benefit, the RS can now localize itself in rooms of up to $10\,\mathrm{m}$ in diameter. However, it is not limited to such rooms if additional features like furniture are present. Considering the error at great distance, the system must have objects in its view that are nearby, implying that the user should let the robot always drive along a wall or ensure that mapped obstacles are along the way in order to guarantee that the localization process still works. This satisfies the need of distance measurements in the environment the RS is intended for. Common office-like buildings consist of corridors from 1 to $3\,\mathrm{m}$ in width. While laser rangefinders would greatly enhance the field of view, our localization approach with Kinects still satisfies our own accuracy need. Also laser rangefinders are not as cheap as a Kinect (less than $100\,\mathrm{Euro}$).

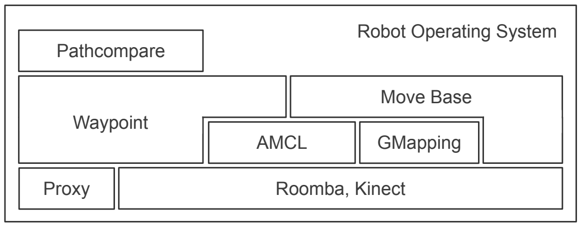

Fig. 2: Important software components in the RS.

IV. Software

The software stack on the on-board computer consists of multiple programs communicating via ROS. Therefore, ROS plays a central part in our RS. The relation between the software components described below is shown in Fig. 2.

A. Robot Operating System

The ROS is an open-source software capable of handling various aspects of typical robot functionalities, like using depth cameras, processing their output or managing a dynamic tree of transformation and rotation information of various objects like the current relation between a map coordinate frame and the robot itself. It enables fast and reliable interchange of messages between programs, possibly running on different computers. That allows for remote surveillance, controlling as well as processing information on another computer. The following paragraphs explain the main components needed by the RS that are already provided by ROS [12].

The RS uses an implementation of Adaptive Monte-Carlo Localization (AMCL) [13] provided by ROS. As shown in Fig. 3, it estimates the robot's position by processing two-dimensional depth information from the Kinect and finding a match in a given two-dimensional pre-drawn floor plan. In a short training stage, AMCL needs a position estimation given by the user in order to find the first match. As the robot moves, a set of nearby positions is maintained that characerizes possible matches in the near future.

The rough position of the robot is calculated using the odometry data provided by a generic Roomba driver. As the robot moves, AMCL uses the odometry data to update the set of possible positions, and aligns the coordinate frame in which the odometry data was recorded with the position estimated by AMCL.

A three-dimensional point cloud is produced by the Kinect's driver. This information is transfered into a two-dimensional depth scan, which is later matched against the floor plan by AMCL.

The ROS offers GMapping as a component which is able to perform SLAM in order to build a map of an environment [14], [15]. To accomplish that, the wheel-odometry data is fused with scans from the Kinect. This map can then be used by AMCL to localize the system accordingly. However, due to the limited range of the Kinect and the quality of our odometry, recorded maps show a significant drift, if a long corridor was

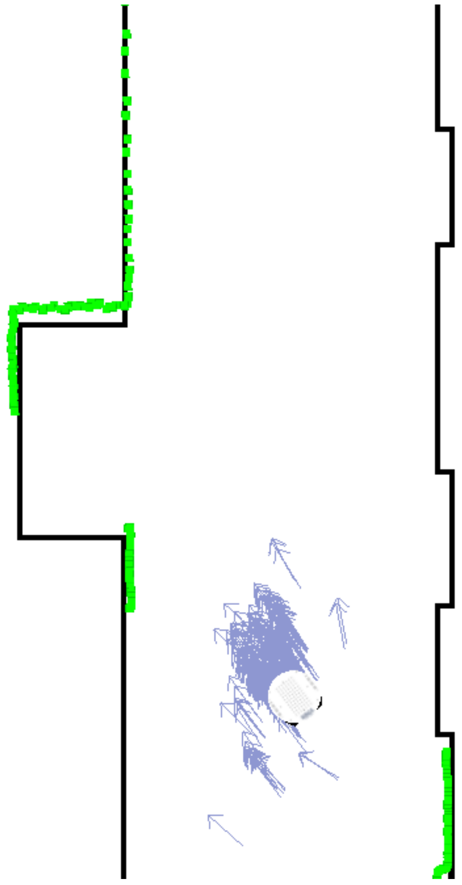

Fig. 3: The TurtleBot in a corridor. The Kinects' distance measurements (green) are matched against the wall and an indentation to a door. The door is drawn as closed. A set of possible positions is displayed around the robot (purple).

mapped. Even so, we show another scenario in section IV-B, that makes use of this software in our RS.

Figure 2 also shows a component called Move Base. It allows the robot to drive automatically to a user-defined destination. This can be used to direct the robot through a building. However, direct user control is also possible and preferable if a specific path shall be driven.

B. Map-Building

In order to localize the RS, AMCL needs a map of the floor the RS is intended to drive on that provides sufficient information to allow self-localization. Sufficient information means that indentations to doors or pillars in a hallway are mapped according to their real position.

Such a map can be recorded using the GMapping software in ROS. With the quality of our wheel-odometry data and the limited range of the Kinect, these maps would be prone to drift. The RS could still be localized in that map, but as shown later in Fig. 4, the positions could not as easily be compared with the SUE. We would need better odometry data or a higher precision of the Kinects in order to build an accurate map with the RS itself.

Therefore, we decided to use a floor plan of the building we obtained before driving. This floor plan has to be accurately scaled in order to be able to compare positions to real-world coordinates. It could be directly used by AMCL for the localization process of the RS. But before using such a floor plan, it might need to be edited regarding doors or markings not belonging to walls or obstacles.

A floor plan does of course not contain furniture and other obstacles. But rooms are typically not empty, therefore these expressive features have to be represented in the map as well. To picture a scenario: the robot may be localized closer to a wall which is behind a locker, if the locker is misunderstood as the wall itself. This would mean that the system is not able to localize itself accurately. Therefore, we enhanced the floor plan of rooms we want to localize the system in. This can be achieved by using GMapping to build a local map of that room or by measuring distances and updating the map by hand. The first method is less prone to drift, than building a map of a whole floor. This local map has to be inserted in the floor plan. Since both maps have to be scaled, this can easily be done.

C. Middleware

ROS brings a lot of features, that can be directly put together to build the RS. To meet additional needs, we developed the following three components, which are started in the ROS ecosystem: The generic Waypoint component, for collecting position information in our RS produced by AMCL and in the SUE. To allow for Waypoint to collect information of a SUE, a Proxy software has to translate the SUEs data. The Proxy itself has to be developed by the user separatly for each SUE. The collected position information can then be merged and analyzed by Pathcompare. Because these software components both run on the same machine, no clocks need to be synchronized. All position information are fused and logged with a timestamp with a resolution of nanoseconds.

The Waypoint software also allows for re-driving a path. If a SUE varies in its performance and thereby in its localization results, it is crucial to be able to repeat that test run multiple times. If the robot was directed through a test area by hand, the recorded data of the RS allows Waypoint to automatically drive the robot along that path again. While driving, Waypoint has to maintain the speed as defined by the original run. If the robot drives fast (which can be a valid evaluation scenario), the SUE may not have enough time to communicate with its infrastructure to establish position estimates due to possible limitations. If the robot's speed is slowed down, the test results of the SUE may change significantly. Therefore, the timing is essential if re-driven paths shall be compared to one another. Algorithm 1 shows the functioning of this component. As a requirement for that algorithm to work, the robot has to be placed near the starting point of the path to re- drive.

While driving, the algorithm continuously computes the linear and angular speeds necessary to reach the next position by dividing the distance/angle to that position and the time left. The time left is computed in line 7 by subtracting the time, which the robot already drove on the path, from the time offset of the next position in the path. That time should be positive, which means that the robot still has time to reach this next position. If the time is negative, it means that the robot should have reached that position in the past. As a limitation, the robot is not allowed to be $1\,\mathrm{s}$ behind. This ensures that the timing of the run can not exceed too much. The robot is considered to have reached a position if it estimates its position in a radius of $0.1\,\mathrm{m}$ of the current targeted position (line 6). If this radius is not yet reached and the time left is less than $0\,\mathrm{s}$, then the algorithm aborts if the distance to the target is greater than $0.4\,\mathrm{m}$, otherwise it targets the next position. The $0.4\,\mathrm{m}$ radius serves as a recovery area, so that the robot still can reach the next target if the current target was slightly missed. The algorithm works best if target positions are in front of the robot. If target positions are at an angle greater than $40°$, the linear speed is reduced to zero to allow the robot to align with the next target. This may look less sophisticated, but considering the small angle differences of position estimates of only a few degrees provided by AMCL, this seems sufficient enough.

Algorithm 1 Re-driving a given path

```
 1: $path \leftarrow [(p_1,t_1),(p_2,t_2),\dots,(p_n,t_n)]$
 2: $t_{\text{path}} \leftarrow t_1$
 3: $t_{\text{run}} \leftarrow \text{now}()$
 4:
 5: for each $(p,t)$ in $path$ do
 6:     while $\text{distanceTo}(p) > 0.1\,\text{m}$ do
 7:         $t_\Delta \leftarrow (t - t_{\text{path}}) - (\text{now}() - t_{\text{run}})$
 8:         $d \leftarrow \text{distanceTo}(p)$
 9:         $a \leftarrow \text{angleTo}(p)$
10:         if $t_\Delta \le -1\,\text{s}$ then
11:             abort
12:         else if $t_\Delta \le 0s \wedge d > 0.4\,\text{m}$ then
13:             abort
14:         else if $t_\Delta \le 0s \wedge d \le 0.4\,\text{m}$ then
15:             break
16:         end if
17:         $v_{\text{linear}} \leftarrow d/t_\Delta$
18:         $v_{\text{angular}} \leftarrow a/t_\Delta$
19:         if $|a| > 40°$ then
20:             $v_{\text{linear}} \leftarrow 0m/s$
21:         end if
22:         $\text{drive}(v_{\text{linear}}, v_{\text{angular}})$
23:     end while
24: end for
25:
26: $\text{drive}(0,0)$
```

After a run was performed, Pathcompare calculates the error of each position, the mean average error (MAE), the root mean squared error (RMSE), the standard deviation, the error variance and a list of the greatest position estimation errors. It writes its results into a comma-separated-values file along with the input data, allowing non-ROS-enabled software to do further analysis. The input datasets are provided in the form of ROS path messages, which are represented as to a list of pairs where each pair (p, t) contains a position p and a timestamp t. In order to evaluate a position p from the SUE's path, we must first find the related position p' on the reference path. Algorithm 2 shows the calculation of the position p', which is found in line 8.

Algorithm 2 Finding reference positions p'

```
1: $reference \leftarrow [(p_1, t_1), (p_2, t_2), \ldots, (p_n, t_n)]$
2: $path \leftarrow [(p_1, t_1), (p_2, t_2), \ldots, (p_n, t_n)]$
3:
4: for each $(p, t)$ in $path$ do
5:     for each $\{(p_n, t_n), (p_{n+1}, t_{n+1})\}$ in $reference$ do
6:         if $t_n \leq t \leq t_{n+1}$ then
7:             $\delta \leftarrow (t - t_n)/(t_{n+1} - t_n)$
8:             $p' \leftarrow p_n + \delta(p_{n+1} - p_n)$
9:             $\mathrm{analyse}(p, p')$
10:            break
11:        end if
12:    end for
13: end for
```

First, a position's timestamp t contained in a pair which shall be evaluated is matched to two corresponding pairs of the reference path containing t_n and t_{n+1}, so that the condition $t_n \leq t \leq t_{n+1}$ holds. The positions p_n and p_{n+1} can then be interpolated to obtain p'. It is a requirement that the robot does not suddenly change its speed. If this is met, the linear interpolation shown here calculates the position p' sufficient enough, considering the small position distances estimated by AMCL, which are up to $25\,\mathrm{cm}$, if the robot has been driving with its full speed.

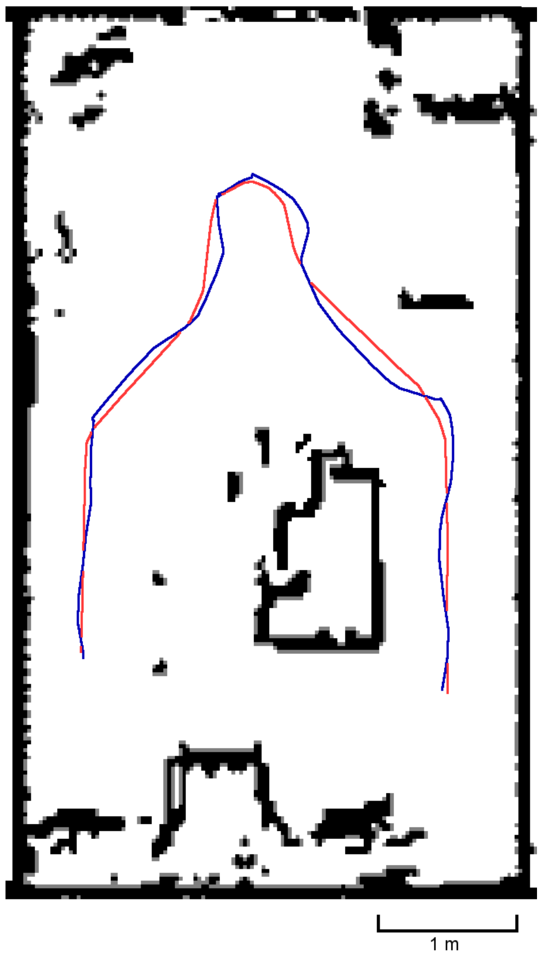

Fig. 5: A precise floor plan of an obstacle-enhanced room. Red shows a recorded path while the re-driven path is displayed in blue.

V. Evaluation

We evaluate the map-building capabilities of the RS, its general position error, applicability in common buildings and path re-drive accuracy. All tests were performed in our office building with a $60\,\mathrm{m}$ long corridor and three vertically branching off corridors of 5 to $20\,\mathrm{m}$ length. There are indentations every few meters which are significant enough for visual localization.

At first, we built maps with GMapping. While resulting maps show inaccuracies of decimeters, they show also a significant drift as shown in Fig. 4. This is due to errors in the wheel-odometry data. Resulting maps could not be used for a RS, because the mapping to real-world coordinates would need additional work. To increase the quality of the results, the robot would have to be equipped with more suitable sensors. However, the result shows also that the local accuracy can be sufficient for a localized map. Figure 5 shows an enhanced section of the floor plan using GMapping.

To evaluate the localization capabilities we marked a 5×9 grid of positions onto the floor at $30\,\mathrm{cm}$ intervals, as shown in Fig. 6. As the robot was directed along the grid, its computed positions were saved when it reached a marked position. In total, 135 measurements were recorded during that test. The results show an average position estimation error of $6.7\,\mathrm{cm}$, with a standard deviation of $4.0\,\mathrm{cm}$. The maximum error was $21.7\,\mathrm{cm}$. As shown in Fig. 7, $96.3\,\%$ of the position errors were equal to or less than $15\,\mathrm{cm}$. $80.0\,\%$ were equal to or less than $10\,\mathrm{cm}$. This error should not interfere with current indoor localization algorithms. However, it could be minimized by using more expensive hardware. The experiment was performed on our floor in a limited area, but due to the repeating topology of the floor, the results are not limited to that area.

As mentioned above, the corridors in which we evaluated our system have significant features to both sides. These features mostly consist of indentations to rooms or pillars. If we would drive in a long corridor without these features, the system would not be able to determine its position accurately, since the wheel-odometry would be the only source with which the system could perform self-localization along the corridor. In short, there have to be enough features in the view of the

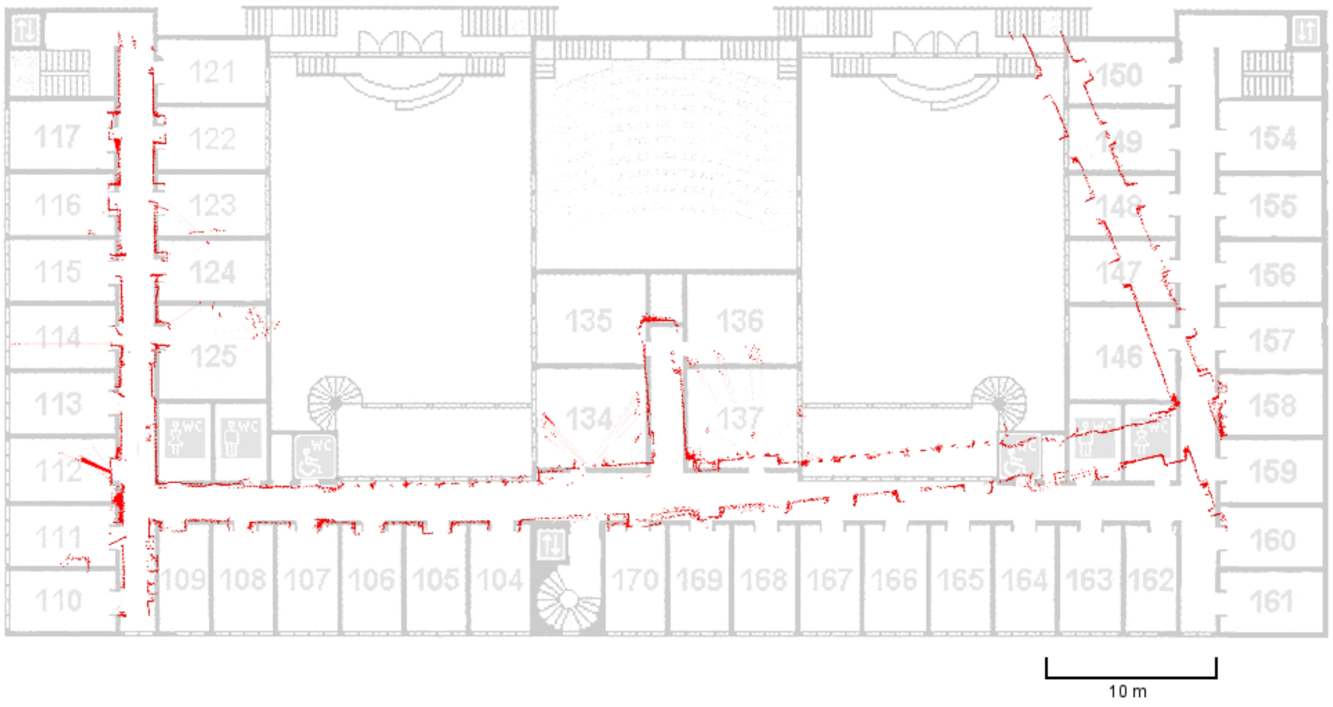

Fig. 4: A recorded map in respect to a floor plan. The displayed overlay in red was created with GMapping.

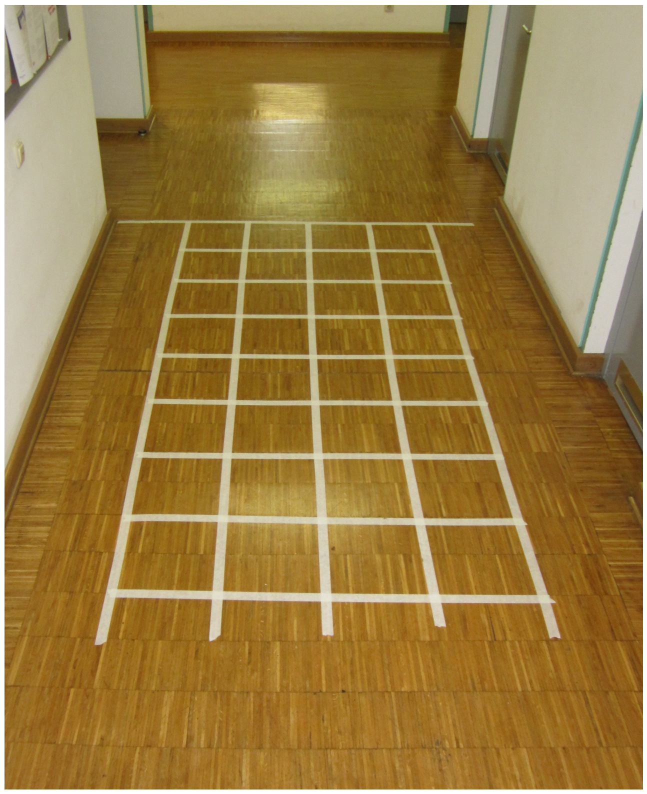

Fig. 6: An evaluation of the position estimation error of the RS in a corridor of our building. 5×9 grid positions are marked in $30\,\mathrm{cm}$ intervals.

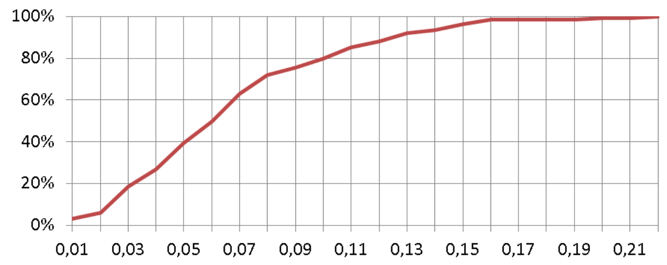

Fig. 7: The error distribution of the localization test. The x-axis shows the position error in meters, while the y-axis depicts the percentage of measurements, that are equal to or less than this position error.

Kinects to correct the position estimate. If the system is to be localized in a room greater than the Kinects' viewing ranges, then multiple positions would qualify as the real one. This limitation can be partly overcome by mapping dynamic environment in rooms as we showed. Nevertheless, this limitation should not apply in common office-like buildings.

Figure 5 shows a recorded drive in one of our office rooms (red). After recording the drive, the RS was put in place to re-drive that path (blue). Re-driven paths show a median distance error of $10.1\,\mathrm{cm}$. The maximum error was about $24.6\,\mathrm{cm}$. The variance is about $0.2\,\mathrm{cm}$. Due to the programming as stated in algorithm 1, the timing error can not exceed $1\,\mathrm{s}$.

VI. Use Case

To explain the usability of the presented system we illustrate a simple use case with an exemplary SUE. For this purpose

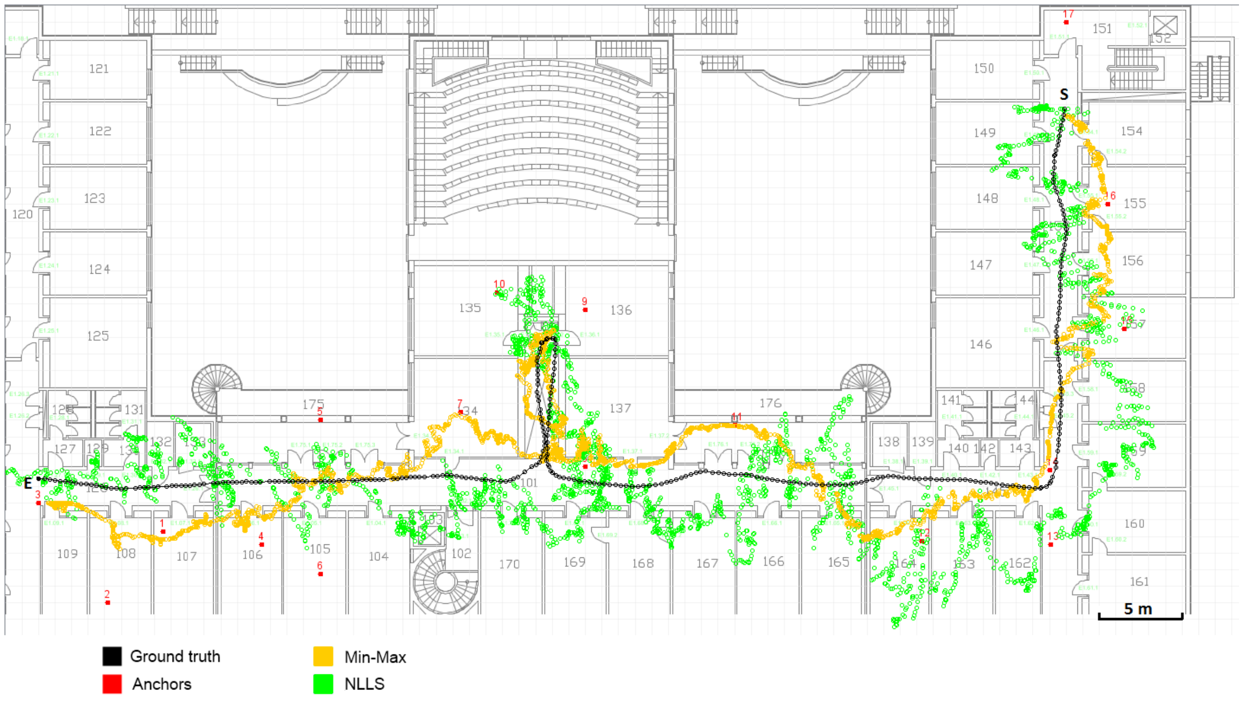

Fig. 8: Position estimates on the second floor of our office building.

we recorded the data of a series of different test runs. The SUE consists of a modified version of the Modular Sensor Board (MSB) A2 [16] node which is equipped with a Nanotron nanoPAN 5375 [17] transceiver. This hardware enables the sensor nodes to measure inter-node ranges using time-of-flight in the 2.4 GHz frequency band. The experiments took place on the first and second floor of our office building during daytime. We conducted several test runs with different paths on every floor to get representative samples of indoor distance measurement conditions using varying anchor counts and inter-anchor distances. As described in Section IV-C we used a proxy to merge the position data of the RS with the ground truth data for further evaluation.

Figure 8 shows one exemplary campaign of the conducted measurements following a route among offices and laboratories with a few people walking around. The starting point is marked “S”, the endpoint is marked “E” and the total length of the path is about $100\,\mathrm{m}$. In this run, we use 17 anchors which are deployed throughout the building. Most of the anchors are placed in office rooms with doors closed. Only a small fraction of nodes are placed on the hallway, in case of Fig. 8, there are four nodes. The path of the RS is plotted in black. We use this reference data to evaluate two common localization algorithms: Multilateration using non-linear least squares (NLLS) [18], [19] and Min-Max algorithm [20], [21]. The results of both algorithms are plotted in Fig. 8 using different colors. Merging the recorded reference data with the data of the SUE which consists of a timestamp and the measured distances to the anchor nodes, we are able to numerically evaluate the performance of both algorithms. Among these metrics are, for instance, the MAE, the RMSE, the standard deviation and the variance of the algorithms. Additionally, we can evaluate the accuracy of the distance measurements using the same metrics and compare how well the algorithms performed relative to the quality of the distance measurements available. For the case of Fig. 8, the nanoPAN achieves ranging precision of around $2.85\,\mathrm{m}$ on average and the RMSE is $4.32\,\mathrm{m}$. However, the ranging error can be as large as $20\,\mathrm{m}$. We even encountered measurement errors up to $75\,\mathrm{m}$ in rare cases.

With the presented SUE, common localization algorithms achieve an average position error of a few meters in indoor scenarios. For instance, NLLS has a MAE of $4.49\,\mathrm{m}$ and Min-Max has a MAE of $2.05\,\mathrm{m}$. Thus, the localization accuracy of the RS with an average position estimation error below $7\,\mathrm{cm}$ is clearly sufficient for evaluation of these algorithms.

VII. Conclusion

We presented a precise RS for indoor localization experiments that is inexpensive and easy to use. The component-based design around ROS allows the easy adaption to other needs and specifications. Because the map-building process is suffering from drift, the use of a precise floor plan is recommended in which the system can localize itself accurately. The system can enhance rooms in the floor plan with obstacles (e.g. furniture) to allow localization where the walls in the floor plan are hidden behind obstacles. A middleware that records a driven path as well as the data produced by the SUE is also presented. To enhance and harden the experimental results test runs can be re-driven multiple times.

The accuracy of the RS is limited by distance measurement errors of a Kinect and the wheel-odometry of the Roomba. Additional sensors like a gyroscope would increase the map-building possibilities so that no precise floor plans of the testing environment have to be used. A laser scanner could increase the field of view significantly, which would greatly enhance not only map-building, but also the localization accuracy. Our RS is able to localize itself in empty rooms smaller than $10\,\mathrm{m}$ in width which should fit most indoor scenarios. When using Kinects, the user has to assure that enough mapped obstacles exist in the field of view to allow a precise localization.

The feasibility of the RS is demonstrated by a use case in our office building. There, the accuracy of the SUE is much worse than the accuracy of the RS ($6.7\,\mathrm{cm}$ vs. $205\,\mathrm{cm}$). Such differences in accuracy are common for radio based localization methods. Consequently, the RS is sufficiently good for evaluating such SUE.

We propose our approach as a cheap and general purpose RS for indoor localization experiments.

References

[1] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “Ros: an open-source robot operating system,” in *ICRA Workshop on Open Source Software*, 2009.

[2] Roomba 531 vacuum cleaner. [Online]. Available: http://www.irobot.com/de/product.aspx?id=127

[3] Microsoft Kinect. [Online]. Available: http://www.xbox.com/en-US/kinect

[4] M. Piras and A. Cina, “Indoor positioning using low cost gps receivers: Tests and statistical analyses,” in *Indoor Positioning and Indoor Navigation (IPIN), 2010 International Conference on*. IEEE, 2010, pp. 1–7.

[5] M. Bal, H. Xue, W. Shen, and H. Ghenniwa, “A test-bed for localization and tracking in wireless sensor networks,” in *SMC*. IEEE, 2009, pp. 3581–3586.

[6] D. Johnson, T. Stack, R. Fish, D. M. Flickinger, L. Stoller, R. Ricci, and J. Lepreau, “Mobile emulab: A robotic wireless and sensor network testbed,” in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, 2006, pp. 1–12.

[7] A. Prorok, A. Arfire, A. Bahr, J. Farserotu, and A. Martinoli, “Indoor Navigation Research with the Khepera III Mobile Robot: An Experimental Baseline with a Case-study on Ultra-wideband Positioning,” in *Proceedings of the IEEE International Conference on Indoor Positioning and Indoor Navigation*, 2010, pp. 1–9.

[8] M. Segura, H. Hashemi, C. Sisterna, and V. Mut, “Experimental demonstration of self-localized ultra wideband indoor mobile robot navigation system,” in *Indoor Positioning and Indoor Navigation (IPIN), 2010 International Conference on*, sept. 2010, pp. 1 –9.

[9] The TurtleBot concept. [Online]. Available: http://www.willowgarage.com/turtlebot

[10] K. Khoshelham, “Accuracy analysis of kinect depth data,” 2011.

[11] Precision of a Microsoft Kinect depth camera. [Online]. Available: http://www.ros.org/wiki/openni_kinect/kinect_accuracy

[12] Robot Operating System. [Online]. Available: http://www.ros.org/

[13] D. Fox, “Kld-sampling: Adaptive particle filters and mobile robot localization,” in *In Advances in Neural Information Processing Systems (NIPS*, 2001.

[14] G. Grisetti, “Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling,” in *In Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA*, 2005, pp. 2443–2448.

[15] G. Grisetti, C. Stachniss, and W. Burgard, “Improved techniques for grid mapping with rao-blackwellized particle filters,” *IEEE Transactions on Robotics*, vol. 23, p. 2007, 2007.

[16] M. Baar, H. Will, B. Blywis, T. Hillebrandt, A. Liers, G. Wittenburg, and J. Schiller, “The scatterweb msb-a2 platform for wireless sensor networks,” no. TR-B-08-15, 09 2008. [Online]. Available: ftp://ftp.inf.fu-berlin.de/pub/reports/tr-b-08-15.pdf

[17] “nanopan 5375 rf module datasheet, berlin, germany, 2009. [online], available: http://www.nanotron.com.”

[18] I. Guvenc, C. Chong, and F. Watanabe, “Analysis of a linear least-squares localization technique in los and nlos environments,” in *Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th*. IEEE, 2007, pp. 1886–1890.

[19] S. Venkatesh and R. Buehrer, “A linear programming approach to nlos error mitigation in sensor networks,” in *Proc. 5th international conference on Information processing in sensor networks*. ACM, 2006, pp. 301–308.

[20] A. Savvides, H. Park, and M. B. Srivastava, “The bits and flops of the n-hop multilateration primitive for node localization problems,” in *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, ser. WSNA ’02. New York, NY, USA: ACM, 2002, pp. 112–121. [Online]. Available: http://doi.acm.org/10.1145/570738.570755

[21] K. Langendoen and N. Reijers, “Distributed localization in wireless sensor networks: a quantitative comparison,” *Comput. Netw.*, vol. 43, no. 4, pp. 499–518, Nov. 2003. [Online]. Available: http://dx.doi.org/10.1016/S1389-1286(03)00356-6