"Off the Grid": Self-Contained Landmarks for Improved Indoor Probabilistic Localization

Eric McCann Mikhail Medvedev Daniel J. Brooks and Kate Saenko
Computer Science Department
University of Massachusetts
Lowell, Massachusetts 01854
Email: {emccann, mmedvede, dbrooks, saenko}@cs.uml.edu

***Abstract*—Indoor localization is a challenging problem, especially in dynamically changing environments and in the presence of sensor errors such as odometry drift. We present a method for robustly localizing a robot in realistic indoor environments. We improve a popular probabilistic approach called Monte Carlo localization, which estimates the robot's position using depth features of the environment and is prone to errors when the topology changes (e.g., due to a moved piece of furniture). We propose a technique that improves localization by augmenting the environment with a set of QR code landmarks. Each landmark embeds information about its 3D pose relative to the world coordinate system, the same coordinate system as the map. Our algorithm detects the landmarks in images from an RGB-D camera, uses depth information to estimates their pose relative to the robot, and incorporates the resulting position evidence in a probabilistic manner. We conducted experiments on an iRobot ATRV-JR robot and show that our method is more reliable in dynamic environments than the exclusively probabilistic localization method.**

I. Introduction

Mobility promises to be the next frontier in robotics, with wide-ranging applications in manufacturing, personal robotics and healthcare. In this paper, we address robot localization, one of the fundamental problems in mobile robotics [1]. Typically, a map of the area is first obtained with Simultaneous Localization And Mapping (SLAM) [2], followed by localization in the resulting map's coordinate frame. Most contemporary localization methods require accurate odometry to maintain an accurate estimate of the robot's position in the world. However, unreliable odometry can negatively affect these methods. Such unreliability need not be as severe as a hardware failure, and can manifest itself, for example, as mere wheel slippage on smooth floor surfaces. Odometry from motor or wheel encoders, an inertial measurement unit, or a camera are all susceptible to drift as a result of error accumulating over time. In indoor environments where Global Positioning Systems (GPS) are not available, odometry is often augmented with sensor measurements of the environment features, such as image features [3]–[5], planar laser scans, and 3D point clouds.

Popular probabilistic methods such as Adaptive Monte Carlo Localization (AMCL) [6] account for the uncertainty in the feature measurements, but can nonetheless be affected by inaccurate odometry. Furthermore, they fail when the input map is inaccurate or the environment is dynamic. Many realistic indoor environments are dynamic, with moving people and furniture causing discrepancy between what the sensor sees and the map. Even slight discrepancy can be sufficient for incorrect hypothetical positions to seem more probably to AMCL than the robot's unknown ground truth.

We present an augmentation to wheel odometry and laser methods, similar to GPS in that it provides absolute position estimates regardless of prior beliefs, but different in that it can function reliably indoors. Our proposed algorithm incorporates an estimate of the robot's position (x, y, and z) and orientation ($roll$, $pitch$, and yaw) obtained using values decoded from 2D barcode landmarks pre-placed in the environment. While several landmark types are feasible, we choose to use QR (Quick Response) codes for their cost effectiveness, ease of creation, error correction, arbitrary text storage ability, and the high availability of libraries with which to decode them.

Our approach consists of the following steps: 1. adaptive local thresholding is applied to the RGB image to enhance the readability of the QR code; 2. resulting binary images are then passed to a QR code reading library to read and find the corners of any QR codes in the image; 3. the coordinates of the corners in the image are then used to look up the 3D positions that correspond to those positions in the point cloud received from a Microsoft Kinect sensor; 4. the robot's pose in the world is then estimated based on the camera relative position of the QR code, and the position of the QR code in the world (encoded in the QR code); 5. if more than one QR code is detected in the same image, their estimates are averaged; lastly, 6. the robot's pose is updated based on the landmark's position using a Kalman filter; this information is then used to reinitialize AMCL's point cloud around the estimated position.

We believe our approach is cost effective, robust, and useful in many indoor mobile robotics applications. We present an experimental evaluation demonstrating promising results in using our algortithm to augment AMCL, most notably with respect to improved reliability in AMCL's degradative cases.

II. Background

Adaptive Monte Carlo Localization (AMCL) is a localization method commonly used in indoor environments by academic researchers. In this work, we employ an open source version included as a component in the Robot Operating System (ROS) navigation package. Based on Thrun's Monte Carlo Localization [6], it uses a particle filter to make inferences about the robot's position based on wheel odometry and on the correspondence between range finder data and what the map should look like from a particular location. While considered to be a reliable form of localization, it requires an accurate

map of the environment to be generated a priori. Substantial changes to the environment, such as moving furniture, can cause problems with proper localization. Furthermore, if wheel odometry values become corrupted, it can cause the particle filter update function to give inaccurate results.

Simultaneous Localization and Mapping (SLAM) algorithms, such as GMapping [2], are another form of localization. With SLAM, the origin of the world coordinates is considered the starting point of the robot, and the map is generated as the robot travels through the environment. While this has the advantage that no map is needed a priori, the coordinates computed at run time will not be consistent unless the robot starts from exactly the same place each time (i.e., location and orientation). SLAM can be used to generate a map for AMCL to later use. GMapping also uses a particle filter with wheel odometry and laser range scanners.

2D barcode assisted localization has been proposed and evaluated previously. Kobayashi [7] used an encoding scheme that allowed for information about the QR code's position, normal vector, and dimensions to be encoded. However, relying entirely on camera calibration to calculate the position of the code relative to the camera and back-solve the robot's position yielded potential errors including up to 10° of yaw. We drastically reduced the amount of information that needs to be encoded in the QR code by using depth information from a Kinect rather than computer vision methods that require the code's physical side length be encoded within it. Additionally, we calculate the normal vector based on the 3D positions of its corners, rather than encoding it in the QR code, once again reducing the quantity of information we need to encode in it. This decrease in information density in the QR codes allows making the QR codes smaller, more redundant, or viewing them from further away. Additionally, employing the Kinect's depth data rather than relying on computer vision methods exclusively has resulted in comparatively higher accuracy, which was reported by Kobayashi as being "not very good" [7].

Many landmark-assisted localization techniques employ sensor fusion [8] to combine readings from other sensors with pose information contained in 2D barcodes. Rather than fuse the landmark estimates with another localization method's estimate, our landmark estimates are used to trump the converged estimate of a probability-based localization method, replacing its point cloud with a point cloud of low-covariance Bayesian distribution centered at the estimated position. Probabilistic methods for localization provide estimates based on a motion model (odometry) and correspondence between sensor readings and previous measurements (often in the form of a map). These estimates often converge on the true position and orientation of the robot, but in various failure cases, they may not. Our calculated robot position does not succumb to the same degradation of accuracy in dynamic situations, so instead of fusing it with a possibly inaccurate probabilistic method, we correct the probabilistic method. By forcing convergence on the known position, we guarantee convergence on a more correct position estimate while codes are in view, and increase the likelihood of the probabilistic method's being correct after we no longer see the QR code, at least temporarily. Our algorithm is intended for use with a sparse distribution of landmarks in the robot's operating environment, relying on AMCL to get between gaps in landmarks, though a sufficiently dense landmark configuration could replace probabilistic localization entirely.

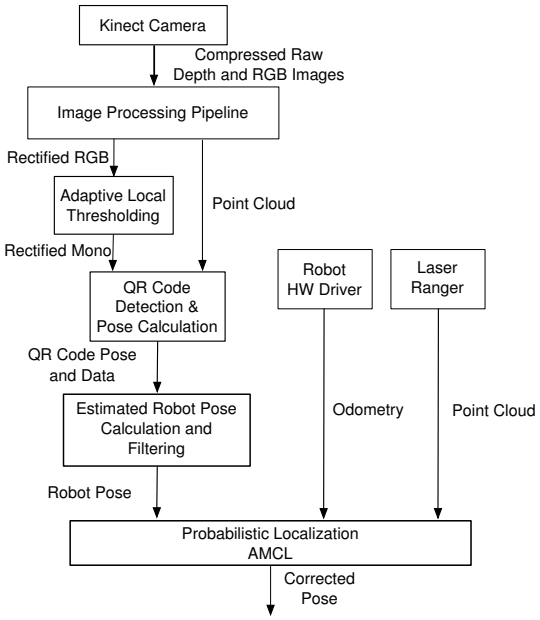

Fig. 1: Overview of our localization method.

III. Implementation

Our approach implements a highly modular design built using ROS Fuerte [9], [10]. Figure 1 describes the main components of our system. First, we pre-process the RGB-D values obtained from the Kinect camera (Section III-B). Adaptive Local Thresholding (Section III-C) is used to filter the RGB image to improve barcode detection before passing it to the QR code detection system (Section III-D). The landmark pose (in the camera frame) and landmark pose decoded from the barcode data (in the world frame) are used to calculate the robot pose in the world coordinate frame; the estimated pose is then used to reinitialize AMCL's hypotheses with low covariance around the calculated pose (Section III-E).

A. Representation of Position in Landmarks

We use QR (Quick Response) Codes as landmarks for our system. QR codes have an easily perceivable orientation, are quickly decoded, and have built-in error correction. Additionally, QR codes can store large amounts of arbitrary information; for example, researchers have investigated the use of QR codes to encode navigation instructions [11], relational information about nearby objects [12], and property information about a particular object [13]. They also provide an inexpensive solution as compared to hardware based landmarks such as active light LEDs (e.g., ByteLight [14], NorthStar [15]) and RFID tags [16]. In comparison to non-self-contained landmark encoding methods, self-contained landmarks are harder to detect at larger distances due to their inherently increased information density. However, unlike non-self-contained landmarks, a priori knowledge of their locations is unnecessary.

We developed a specialized encoding scheme in order to increase the robustness of the QR code detection and decoding.

A QR code can contain up to almost 3Kb of binary data. For a fixed QR code size, only a certain amount of information can be stored to allow decoding with a limited resolution camera at given distance. The minimum possible size of a QR code is 21 by 21 modules, which can store up to 17 decimal digits encoded with maximum redundancy of 30% (Figure 2). Our pose encoding-decoding scheme XXXX-YYYY-ZZZZ-AAAAA uses the first 12 digits to encode the position in 3D space, and last five for orientation.

$$x = \frac{\text{XXXX}}{10} - 500 \qquad \frac{yaw}{9} = \text{AAAAA} \mod 40^2$$

$$y = \frac{\text{YYYY}}{10} - 500 \qquad \frac{pitch}{9} = \frac{\text{AAAAA} - roll}{40} \mod 40$$

$$z = \frac{\text{ZZZZ}}{10} - 500 \qquad \frac{roll}{9} = \frac{\text{AAAAA} - roll - pitch * 40}{40^2}$$

We encode world position coordinates ranging from -500.0 m to 499.9 m in increments of 10 cm, and orientation at angular resolution of 9 degrees.

B. Kinect Data Preprocessing

In order to detect the 3D position of the QR code in the 2D image, we need depth information. While this could be accomplished with a calibrated RGB camera, we used a low-cost RGB-D sensor, a Microsoft Kinect ($110). A benefit of not relying solely on computer vision for the 2D to 3D conversion of the landmark position is that QR code size need not be known a priori or even be consistent between landmarks. The Kinect uses a structured light method to extract depth information for each pixel of the RGB image of a scene.

ROS provides a software package called OpenNI, which packages camera drivers for the Kinect together with ROS's built in Image Pipeline tools. Many robots use power efficient or embedded computers that are not well equipped to deal with image processing. Running OpenNI on the robot can be very computationally expensive, depending on the information requested. To reduce onboard power consumption, we run image processing remotely over a network. The OpenNI camera drivers send compressed image and depth information, along with camera calibration information to a server. The server then rectifies both color and depth images, produces point cloud data from depth information, and sends this information to be processed by downstream components.

C. Adaptive Local Thresholding

The Kinect automatically compensates for light conditions. This introduces variability in image contrast, especially when the camera is facing upwards, due to the lights shining directly into the aperture as the robot passes under them. We have found this to drastically reduce overall system reliability. In favorable lighting conditions (Figure 2 top), with the camera facing a single QR code on the ceiling, detection and pose computation of the QR code occurs at an average of 10.5 times per second on thresholded input, while it only occurs at 2.5 times per second in the un-thresholded image. In unfavorable lighting conditions (Figure 2 bottom), QR code recognition and camera-relative pose calculation occurred an average of 5 times per second with thresholding, and *did not occur without it*.

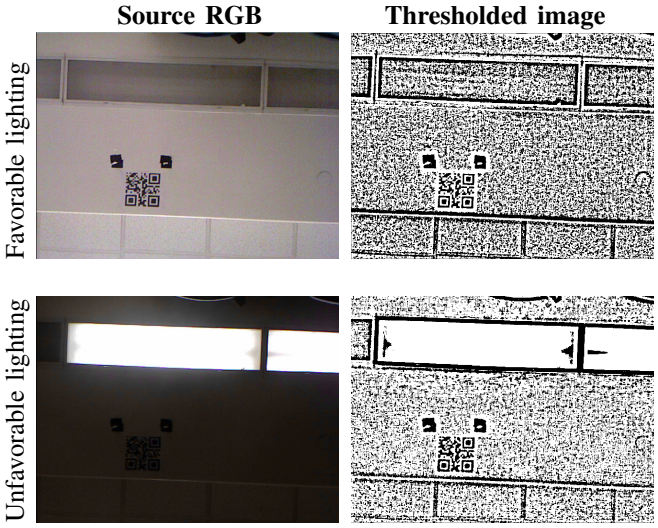

Fig. 2: Adaptive local binary thresholding of 21 module QR code (21 cm $\times$ 21 cm printed size)

We use adaptive local thresholding [17] to mitigate problems that could potentially impede the successful detection of a QR code. Local thresholding allows us to stabilize the contrast of the image to be nearly constant, regardless of lighting conditions. First, every pixel in the RGB bitmap is converted to a single byte of intensity, by taking a weighted average of its values in the red, green, and blue channels. Every pixel is then compared with the average intensity of its neighboring pixels within an empirically determined window ($\pm$5 pixels). If a pixel's intensity is higher than its neighbors' intensities by an empirically determined threshold (0.008), it becomes white in the output image, otherwise, black. As the 2D barcodes are black and white, no useful information is lost by a conversion to gray-scale. We found experimentally that the thresholding increased the frequency of true positive QR code identifications, without introducing any false positives.

As a naive implementation of such a sliding window thresholding algorithm would be prohibitively computationally expensive, we utilize an optimization that reduces the computational complexity from $\mathrm{O}\left(Pixels_{window} * Pixels_{image-size}\right)$ to $\mathrm{O}\left(Pixels_{image-size}\right)$. Our optimized algorithm uses a summed area table [18], alternatively known as an integral image [19]. The integral image precomputes a table containing the sum of all the pixels with x and y pixel coordinates less than or equal to that of the corresponding position in the image, allowing for constant-time lookup of the sum of any region in the image during thresholding. In addition to being extremely efficient once the table has been created, the pre-computation can be done in linear time.

It should be noted that the maximum distance at which the codes could be detected was not limited by the depth sensor, but by the RGB camera resolution (640x480). We observed that the detection rate degraded quickly as the apparent QR module size approached angular camera resolution (about 3 meters in our case).

D. QR Code Detection

There are multiple existing libraries for reading data from QR codes. We selected the ZBar library [20], which we found to be highly reliable under normal lighting conditions and

easy to use. Additionally, the library is capable of finding and decoding multiple QR codes in a single image. The library provides us with four sets of pixel coordinates for each QR code, locating the box containing the QR code in the image.

The decoder begins by passing thresholded images to the ZBar library for barcode identification. If any barcodes are detected, their coordinates are passed to a fast lookup function that returns the 3D camera-centric points in the point cloud that correspond to their 2D pixel coordinates. Due to artifacts (such as "shadows") in the point cloud generation process, not all pixels have corresponding 3D coordinates. Such artifacts manifest themselves as 3D points containing NaNs. When such an artifact is found, a small search is done in the immediate vicinity of the invalid pixel for a valid one within 10 pixels. If such a point is not found, the corner of the QR code is deemed invalid. If all corners are valid, their 3D points are then used to calculate the 3D camera-centric position of the QR code.

To find the 3D orientation of the QR code, we calculate a unit vector normal to three of the four points and positioned in the midpoint of the code. The midpoint is calculated by placing two intersecting vectors between the corners of the code, and finding the midpoint of the shortest line segment perpendicularly connecting the two vectors. We then calculate a final orientation vector, which originates at the midpoint and points in the direction of the midpoint between two of the "front" corners of the code.

E. AMCL + QR Code Correction

All the detected QR code poses are in the Camera Frame of reference. The next step is to resolve the robot pose in the Map Frame using the information stored in the QR code. As described in Section III-A, QR landmark stores its own pose relative to the Map Frame (landmarks are self-localizing). Essentially, a detected landmark's position is known in both the Camera Frame and in Map (World) Frame. The Camera Frame's relationship to the Robot Frame is static. (Figure 3 lower left). In this context the robot pose can be represented as a transform from Map Frame to Robot Frame, and can therefore be computed from the QR code's position in the Map Frame, and its position in the Camera Frame (Figure 3).

ROS uses TF framework to deal with coordinate frames and transforms between them. TF keeps track of all the frames using an acyclic tree structure and provides functionality to compute transforms between any two frames of a connected tree. To resolve the robot pose in the Map Frame, a temporary transform tree is built: $RobotFrame \rightarrow CameraFrame \rightarrow Landmark \rightarrow MapFrame$. The robot's pose in the Map Frame is obtained by walking the temporary tree and accumulating transformations.

If multiple QR codes are detected within a single image, we compute the average as follows. We compute one robot pose estimate in Map Frame per QR landmark on that image, as described above. These robot poses are then averaged to produce a single pose. For example, if there are three QR landmarks on the image, we would average three robot pose estimates to produce a single pose. To stabilize the pose estimates, they are passed through a Kalman filter before being published as the robot's estimated position.

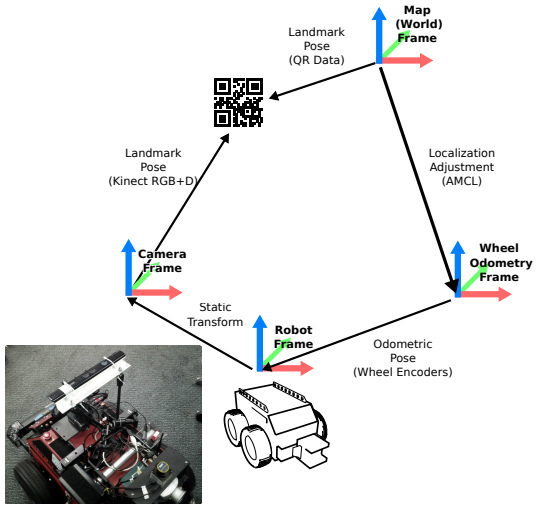

Fig. 3: Coordinate frames and transforms between them. The transform source is parenthesized.

The final step is to update the AMCL belief so it represents the correct (according to the landmark) robot pose. To accomplish this, AMCL's existing hypotheses are cleared, and reinitialized with a low covariance bayesian distribution, centered at the calculated position and orientation. When no QR codes are detected, AMCL functions normally.

IV. Validation

The evaluation experiments are conducted with the robot driving to the set of waypoints while localizing using each of the localization methods.

A. Experiment Setup

1) Robot System: We use iRobot ATRV-JR robot (Figure 3). Onboard computer has 4GB RAM with Intel Core2 Duo at 3.00GHz, and running Ubuntu Server 12.04. The robot has two laser range finders: a SICK LMS200 mounted in front, and Hokuyo URG-04LX faces back. The Microsoft Kinect RGB-D sensor is mounted facing upward. Some of the processing is offloaded to the desktop station with Intel Core i7-2600 CPU at 3.40GHz, 8 GB RAM, running Ubuntu 12.04.

The robot is able to autonomously navigate to a set of waypoints using one of the localization techniques.

2) Environment: We augmented the environment with eleven QR landmarks, each containing their own pose information. The printed markers were 21 cm $\times$ 21 cm, which we placed on the ceiling in two areas. Based on the architectural drawing of the building, we placed six QR landmarks in the hallway (bottom right red area in Figure 4) and five inside an office (top left). The QR codes could be detected at a distances of less than 3 m. It should be noted that the height of the ceiling in the hallway was lower than in the office (2.45 m and 2.68 m, respectively). Additionally, one of the five QR landmarks in the office was placed on a soffit (Figure 2). Finally, it should be noted that the hallway flooring was industrial vinyl tile, and the office a low pile, industrial carpet.

Fig. 4: Locations at which the measurements are taken, numbered in order. Red shaded areas are covered with QR landmarks. (Best viewed in color)

3) Ground Truth Estimation: To evaluate localization algorithms quantitatively, we need to know the ground truth location of the robot to compare against. It is very hard to obtain the exact ground truth; it was outside the scope of our work to use a MoCap system, such as used by Sturm [21] to record one of the localization benchmark datasets. For our evaluation, we estimate ground truth using SICK LMS-200 laser range finder measurements. At each waypoint, the robot's movement was suspended and the laser readings were aligned with the environment map. When the alignment was achieved, the origin of the laser scan was determined to be ground truth. The error was then calculated as the Euclidean distance between the ground truth and the position calculated by the localization algorithm.

B. Procedure

A single experiment run for a condition consisted of 16 waypoints 4 causing the robot to pass through areas with QR landmark coverage at multiple points in the run. Two runs with the same waypoints were conducted per condition. At each waypoint, the estimated error was measured. The error was computed as the difference between the ground truth and the perceived location.

The experiment had 4 conditions:

1) AMCL: We first obtained the baseline ACML performance by conducting two consecutive runs with the standard AMCL used for localization. The map used internally by AMCL was created with GSlam ROS package just prior to the experiment, and no changes were intentionally introduced to the environment.

2) AMCL+QR: Similarly, the performance of AMCL augmented with QR landmarks was obtained, with no changes intentionally introduced to the environment. In these runs, AMCL was reinitialized whenever a QR landmark was visible and could be decoded. Approximate location of QR landmark visibility is marked on Figure 4.

3) AMCL in changed environment: To simulate a dynamic environment, items in the environment were rearranged. The differences before and after the rearrangement is illustrated in Figure 5. There are notably more changes near waypoints 4 and 5, in the office.

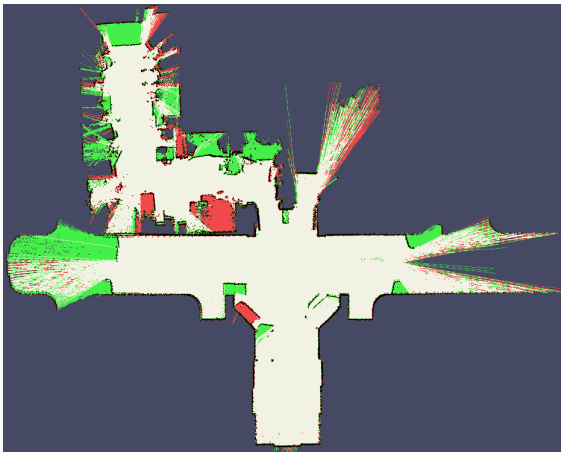

Fig. 5: Changes in the environment: red obstacles are in the original map, green obstacles are in the changed map. (Best viewed in color)

	Mean Error, m	σ
AMCL	0.054	0.028
AMCL+QR	0.094	0.044
AMCL *	0.153	0.135
AMCL+QR *	0.099	0.055

Fig. 6: Total error means across all conditions. Conditions with changed environment marked with an asterisk (*).

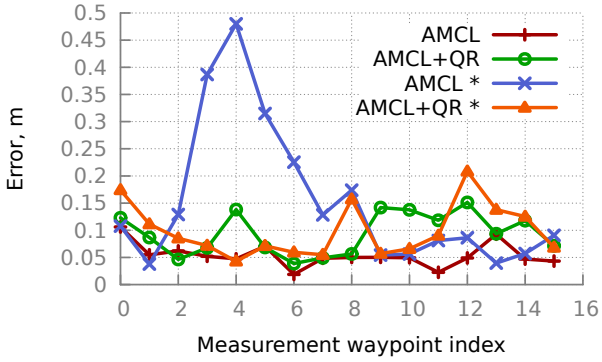

Fig. 7: Measured error across all conditions. Conditions with changed environment marked with an asterisk (*). (Best viewed in color)

4) AMCL+QR in changed environment: As IV-B2, but with the changes to the environment as in IV-B3.

V. Results and Discussion

We performed two runs for each condition, 16 measurements in each run, 8 runs total. The results from our experiments are displayed in Figures 6 and 7.

In Condition 1 (baseline AMCL performance), AMCL showed good performance with an average error of 5.4 cm at the 16 locations measured twice. These results were unsurprising, since AMCL relies on having an accurate map and this run was performed just after the environment was mapped.

Condition 2 (AMCL with QR code correction) did not perform as well as AMCL alone, with an average error of 9.4 cm. At first glance, a 4 cm increase in error introduced by our correction algorithm may seem counter productive. However, it should be noted that for many dynamic indoor environments, a 4 cm error in localization may be completely reasonable. We believe this error was the result of accumulated error in the hand placement of the QR codes and the calibration of our Kinect sensor.

Condition 3 (AMCL in a changed environment) highlights AMCL's failure scenario. The error was much higher near waypoints 4 and 5, which was an area of high local difference compared to before rearranging was performed. The maximum and average errors for this condition were 48 cm and 15.3 cm, respectively. Again, these results should be unsurprising, as AMCL's accuracy is entirely dependent on correlation between the map and range readings. It is interesting to note the environment was modified the most near waypoint 4, where AMCL had the most trouble localizing (See Figures 5 and 7). AMCL's estimates only reconverged later due to the environment not being as heavily modified at those waypoints. It should be further noted that the changes made to the environment for this condition were rather minimal in nature when compared to dynamic environments in the real world, such as warehouses, museums, or shopping malls.

Condition 4 (AMCL in a changed environment with QR code correction) showcases the advantage of using our correction algorithm. The error in this condition was much lower than that of Condition 3, averaging only 9.9 cm. Overall, we believe the proposed method has the potential to improve probabilistic localization (AMCL) for the cases when it fails, e.g., for highly dynamic environments.

One limitation of our work is that we tested a single failure case – a non-static environment. There are a number of extensions to this work, particularly in the areas of crowded environments (e.g., a shopping mall [22], a museum field trip) and also for long-term operation of a robot in a space.

VI. Conclusions and Future Work

We hope to further optimize our algorithm to a point at which it can be run entirely on our robot's onboard computer, and even eventually be run on Atom-powered hardware such as a FitPC. This in part should be possible with ongoing improvements in the area of RGB-D sensing. These sensors are also getting cheaper; for nearly one hundreds dollars, it is possible to replace the functionality of a commercial indoor localization system.

We will further investigate the distribution and placement of QR codes in the environment. We intend to move the RGB-D sensor from an upward-facing to a forward-facing configuration, and evaluate localization accuracy with QR codes located on walls in a crowded public space.

Acknowledgements

The authors would like to thank Dr. Holly A. Yanco for use of her laboratory and robot.

References

[1] S. Thrun, "Robotic mapping: A survey," *Exploring artificial intelligence in the new millennium*, pp. 1–35, 2002.

[2] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007.

[3] H. Andreasson, A. Treptow, and T. Duckett, "Localization for mobile robots using panoramic vision, local features and particle filter," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. IEEE, 2005, pp. 3348–3353.

[4] S. Se, D. Lowe, and J. Little, "Mobile robot localization and mapping with uncertainty using scale-invariant visual landmarks," *The international Journal of robotics Research*, vol. 21, no. 8, pp. 735–758, 2002.

[5] ——, "Vision-based global localization and mapping for mobile robots," *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 364–375, 2005.

[6] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust Monte Carlo Localization for mobile robots," *Artificial Intelligence*, vol. 128, no. 1-2, pp. 99–141, 2000.

[7] H. Kobayashi, "A new proposal for self-localization of mobile robot by self-contained 2d barcode landmark," in *Proceedings of SICE Annual Conference*. IEEE, 2012, pp. 2080–2083.

[8] D. Tick, J. Shen, Y. Zhang, and N. Gans, "Chained fusion of discrete and continuous epipolar geometry with odometry for long-term localization of mobile robots," in *IEEE International Conference on Control Applications (CCA)*. IEEE, 2011, pp. 668–674.

[9] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009.

[10] Willow Garage. (2013, Jan.) Robot operating system (ROS). [Online]. Available: http://www.ros.org

[11] T. Takahashi, M. Shimizu, and M. Okaya, "A navigation method of service robots at shelters," in *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE, 2011, pp. 105–109.

[12] Y. Xue, G. Tian, B. Song, and T. Zhang, "Distributed environment representation and object localization system in intelligent space," *Journal of Control Theory and Applications*, vol. 10, no. 3, pp. 371–379, 2012.

[13] P. Wu, L. Kong, and S. Gao, "Holography map for home robot: an object-oriented approach," *Intelligent Service Robotics*, pp. 1–11, 2012.

[14] bytelight indoor location. With light. [Online]. Available: http://www.bytelight.com

[15] Evolution Robotics. (2005) NorthStar. Low-cost, indoor localization. [Online]. Available: http://web.archive.org/web/20101216070615/http://www.evolution.com/products/northstar.pdf

[16] M. Boccadoro, F. Martinelli, and S. Pagnottelli, "Constrained and quantized kalman filtering for an RFID robot localization problem," *Autonomous Robots*, vol. 29, no. 3, pp. 235–251, 2010.

[17] B. Banko. (2011) Realtime webcam sudoku solver. [Online]. Available: http://www.codeproject.com/Articles/238114/Realtime-Webcam-Sudoku-Solver

[18] F. C. Crow, "Summed-area tables for texture mapping," *SIGGRAPH Comput. Graph.*, vol. 18, no. 3, pp. 207–212, Jan. 1984. [Online]. Available: http://doi.acm.org/10.1145/964965.808600

[19] K. Derpanis, "Integral image-based representations," *Department of Computer Science and Engineering, York University, Paper*, 2007.

[20] J. Brown. (2011, Jul.) ZBar bar code reader. [Online]. Available: http://zbar.sourceforge.net

[21] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems," in *Proc. of the IEEE Int. Conf. on Intelligent Robot Systems (IROS)*, 2012.

[22] K. Zheng, D. F. Glas, T. Kanda, H. Ishiguro, and N. Hagita, "Supervisory control of multiple social robots for navigation," in *Proceedings of the 8th ACM/IEEE international conference on Human-robot interaction*. IEEE Press, 2013, pp. 17–24.