Reliable Kinect-based Navigation in Large Indoor Environments

Mihai Jalobeanu, Greg Shirakyan, Gershon Parent, Harsha Kikkeri, Brian Peasley and Ashley Feniello

Microsoft Research

Abstract—***Practical mapping and navigation solutions for large indoor environments continue to rely on relatively expensive range scanners, because of their accuracy, range and field of view. Microsoft Kinect on the other hand is inexpensive, is easy to use and has high resolution, but suffers from high noise, shorter range and a limiting field of view. We present a mapping and navigation system that uses the Microsoft Kinect sensor as the sole source of range data and achieves performance comparable to state-of-the-art LIDAR-based systems. We show how we circumvent the main limitations of Kinect to generate usable 2D maps of relatively large spaces and to enable robust navigation in changing and dynamic environments. We use the Benchmark for Robotic Indoor Navigation (BRIN) to quantify and validate the performance of our system.***

I. INTRODUCTION

2D mapping and navigation for indoor mobile robots using single-line laser range-finders is a well-studied problem with efficient algorithms and proven implementations, including commercial ones [1]. However, these solutions are relatively expensive, the cost being dominated by the cost of the LIDAR.

Since Kinect is an order of magnitude cheaper than single-line LIDARs, we wanted to see if Kinect could provide a viable low-cost alternative for 2D mapping and navigation. This question appears to have been investigated only superficially so far, particularly when it comes to navigation in dynamic environments. A common approach is to combine a few center rows from the depth image generated by Kinect, cut off the depth at 8m or less to remove the noisiest readings and feed the resulting scan line into an existing 2D mapping and navigation implementation [2]. Not surprisingly, the results obtained with this approach are disappointing, since the method exposes the weaknesses of Kinect (smaller horizontal field of view, range limitations) without explicitly leveraging its strengths (higher resolution and larger vertical field of view).

We introduce a complete recipe for reliable Kinect-based mapping and navigation. Our approach focuses on increasing the useful range and field of view of the depth data acquired from Kinect, in principal by integrating readings over time into an always up-to-date, sliding-window local map. This is done during mapping as well as navigation, and informs both the motion model and the observation model of both systems. Our method is inspired by Kinect Fusion [3], but is applied in 2D rather than 3D to reduce the computational burden.

Specifically, our contributions are:

- A sensor calibration method that eliminates most of the systematic noise in the depth image.
- A scan alignment procedure that combines a truncated signed distance function (TSDF) grid map with a particle filter operating at frame rate.
- A method for making use of depth readings that are extremely noisy (>10% error) during mapping and localization. This allows us to extend the useful range of Kinect depth data to 20m.
- A method for increasing the field of view (FOV) of Kinect during navigation by generating virtual 360 degree views from incremental local maps.

We include experimental results based on the Benchmark for Robotic Indoor Navigation (BRIN) [4] that quantify the performance of our approach relative to state-of-the-art systems.

It should be noted that, while all our experiments have been conducted with Kinect v1, we expect the method and findings to apply equally well to Kinect v2.

II. RELATED WORK

Consumer 3D sensors like Microsoft Kinect and its successors have dramatically simplified access to reliable and accurate 3D data, and as a result spurred a lot of research activity in computer vision and robotics. Simultaneous Localization and Mapping (SLAM) in particular has seen some notable developments. Newcombe et al. [3] demonstrated remarkable real-time 3D reconstruction results with a frame-to-model matching approach that aligns dense depth data with a voxel map using ICP. Real-time performance requires GPU-acceleration and as a result the volume that can be reconstructed is limited by the available GPU memory, with small office environments being the practical limit. The method was later extended to larger environments by Whelan et al. [5] by allowing the region of space mapped by Kinect Fusion to vary and by paging the model data out to a mesh representation. An RGB-only variant called DTAM that also uses frame-to-model matching but is based on dense RGB data and a photometric distance was also proposed by Newcombe et al. [6]. A number of successful sparse methods have also been developed, notably the RGB-D SLAM system introduced by Engelhard et al. [7], where sparse local features extracted from RGB data are augmented with their depth. Frame-to-frame alignment is performed in a two-step process, using

RANSAC and ICP. Dryanovski et al. [8] developed an efficient alternative using frame-to-model matching.

In 2D SLAM, motion estimation from alignment of LIDAR scans is part of most SLAM front-ends. Related to our method, the approach by Hahnel et al. [9] uses an occupancy grid map and a cost function that considers only the cells that fall under the beam-endpoints, and performs gradient descent to minimize the cost function. Similarly, Kohlbrecher et al. [10] use the spatial derivative of the occupancy map for gradient descent and rely on a multi-scale approach to avoid local minima.

The idea of using sliding-window local maps to localize on a global map has been introduced by Gutmann and Konolige [11], who use incremental scan matching to generate local maps and then correlation to determine overlap.

The use of particle filters for SLAM goes back many years, with a significant body of work dedicated to Rao-Blackwellized filters (where each particle has its own map) introduced first by Murphy [12], and extended by Montemerlo et al. [13], Hahnel et al. [9], Eliazar and Par [14] and later by Grisetti et al. [15]. These approaches demonstrate remarkable results not only in scan alignment and incremental mapping but also in closing relatively large loops.

Among the navigation methods that take full advantage of the Kinect depth data is the work done by Biswas et al. [16], where planes are extracted from the depth image and projected in the horizontal plane to match against a map. While this method assumes preexisting maps, building maps from large plane registration has been addressed by Pathak et al. [17].

Kinect depth calibration was covered by Teichman et al. [18], who show the need for individual sensor calibration and perform calibration of long-range readings using SLAM results from close readings, and Smisek et al. [19], who attempt to eliminate vertical banding in the depth distortion based on the assumption that they are static.

III. Our Method

Our approach has three main ingredients: a method for processing depth frames to de-noise them and to extract a relevant subset of the data, a method for continuous incremental mapping to generate reliable local maps from extremely noisy readings and a method for integrating the local maps into the navigation pipeline (primarily localization, but also path planning and path tracking). In particular, localization is done based on a virtual 360 degree view extracted from local maps. Depth processing, incremental mapping, extraction of the 360 degree view and localization are all performed at frame rate (30Hz).

A. Notation

Our method deals with three distinct coordinate systems:

- The world domain in $\boldsymbol{R}^3$. Within world space, a global frame of reference is chosen for simplicity to be the start pose of the robot, with the x and y axes parallel to the floor.
- The image domain in $\boldsymbol{I}^2$, with the coordinate system origin in the top-left corner and x and y being pixel coordinates. The function $u(x,y)$ represents the depth map generated by the Kinect sensor after rectification, relating a point $\boldsymbol{p} \in \boldsymbol{I}^2$ to a depth measurement. The transformation from $\boldsymbol{R}^3$ to $\boldsymbol{I}^2$ is given by the pin-hole camera projection.
- The map domain in $\boldsymbol{I}^2$, where x and y are grid map coordinates. The origin of the map is in the bottom-right corner. A point $\boldsymbol{p_g}$ in the global frame is projected onto the xy plane and transformed to map coordinates by a similarity transform $T_{M.}$

In world space, the 3DOF robot pose is represented by the rigid transformation T_R, and the 3DOF pose of the camera relative to the robot by the transformation T_C, such that a point $\boldsymbol{p}$ in camera frame is transformed to global frame coordinates by $\boldsymbol{p_g} = T_R T_C \boldsymbol{p}$.

B. Calibration

While the Kinect depth sensor can generate long-range readings of over 20m, it exhibits significant noise beyond its main operational range of 4m. We discuss treatment of random error in Section C below. The dominant type of systematic error is a radial error that distorts the depth map in a doughnut shape.

The severity of the systematic error is sensor-specific and increases with distance, being as high as 10% at 10m. We were able to eliminate most of this error using a semi-autonomous calibration procedure. It consists of positioning the robot at various known distances from the wall and learning a mapping $\boldsymbol{D}(u, x, y)$ between the distance $u(x, y)$ reported by the sensor for a given image point $[x, y]^T$ in I and the actual distance to the wall d. Ground truth distance to the wall is determined using a consumer-grade single point laser range finder rigidly attached to the robot and aligned with the Kinect camera. Alignment is achieved by pointing the range finder to a marker placed at the center of the IR image. In the data collection phase, the robot increases its distance d from the wall in small increments from 0.5 m to 4m. At each step we automatically align the sensor to the wall by minimizing the deviation from the mean of each pixel in the depth map.

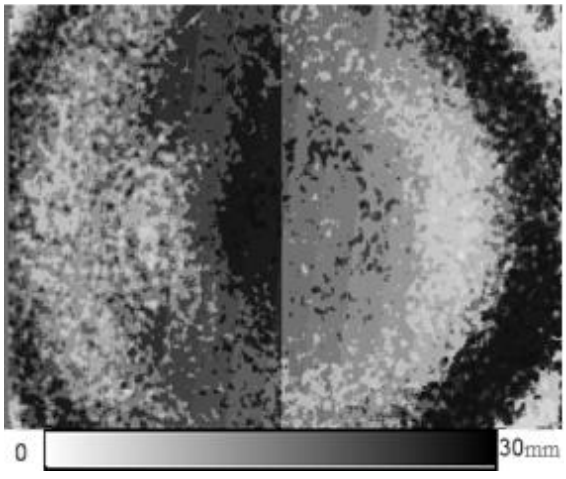

Figure 1. Depth image of a flat wall at 1.5m exhibiting radial error and vertical banding effects. Deviation from ground truth is shown on grayscale dial.

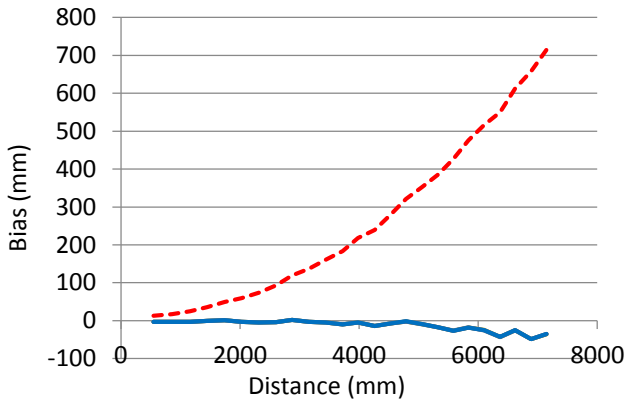

Figure 2. Average frame-wide bias relative to distance, before (dotted line) and after (solid line) calibration, computed on a training dataset and a validation dataset respectively.

We then remove banding effects from each frame (see below) and integrate multiple frames to eliminate random noise. The clean depth image obtained this way is fed into our learning algorithm together with the ground truth distance *d*.

The learning algorithm is a simple parameter estimator for a family of quadratic functions of the form $D_k(u(x,y)) = a_k u(x,y)^2 + b_k u(x,y) + c_k$, where k represents tiled 16x16 regions of $\boldsymbol{I}^2$, similar to [19]. The 16x16 size was chosen experimentally, by varying the tile size from 1x1 to 64x64.

Once the error function for each region is learned using data captured from stepping through 0.5-4m range, we extrapolate depth reading correction values for distances of up to 20m. We use step sizes of 5%-15% of distance from the wall with good results, as opposed to fixed size step. This ensured a denser sampling at closer distances, effectively giving higher weight to readings at closer range during polynomial fitting, to avoid degrading sensor accuracy.

An often overlooked problem in Kinect depth calibration procedures such as [18] and [19] is the presence of dynamic vertical banding artifacts in the depth image. The number of vertical bands and their effects appear to be changing with low frequency even after many hours of sensor operation. It would be a mistake to factor them into the sensor bias correction function, since the behavior is scene-specific and the effects will be different later, at runtime. Thus, we remove banding effects from each frame during calibration by detecting discontinuities along the horizontal and adjusting depth values in each column such that the average frame-wide reported distance to the wall is the same.

Note that we also calibrate the intrinsic parameters of the IR camera (in particular the center of the sensor) using the calibration procedure described in [20].

C. *Depth frame processing*

The Kinect depth frame has a resolution of 640x480 pixels. The depth image can contain any number of pixels for which no depth estimate is available (because of shadows, objects that are closer than 0.5m, obstacles that are too far or open space). We discard all such invalid pixels, as well as any depth readings greater than 20m.

The Kinect uses rolling-shutter cameras, and as a result the generated depth image exhibits significant shearing effects when panning. We use the robot motion to determine the angular speed of the camera around the vertical axis and we correct rolling-shutter effects by shifting the rows in the depth image, as described in [21].

Once the depth map is corrected and aligned, we eliminate from the image all pixels that are either floor readings or are above a given height (2m). This way we retain a slice of the world parallel to the floor and of fixed thickness, extending out to the end of the depth range. Our method of floor extraction operates in image space on each individual frame, by approximating the floor plane based on the readings in the bottom 10% of the depth image and extrapolating it out to the rest of the depth map. Specifically, the method iterates over each column that has enough valid depth readings in the bottom 48 rows and fits a plane to these depth readings. The columns that have a small enough standard deviation from the candidate planes are used to determine the final floor plane, through majority voting. If the majority is less than 75%, the frame is discarded and the floor plane extracted from the previous frame is used instead. Other floor extraction methods, such as the plane detector in [22], would work equally well.

Finally, we walk the resulting image in column order, looking for the smallest value from each column. The value represents the point closest to the camera. We smooth the value by applying bilateral filtering to its neighborhood [23], and retain the resulting depth value in the corresponding entry of a 640x1 array. The array represents the depth profile of the depth frame, matching the corrected Kinect depth map column for column.

D. *Local map representation*

Similar to KinectFusion, our method relies on the TSDF representation of the 2D model. Thus, our map m_k obtained at step k is a fixed-size grid in which each cell contains the signed distance value F_k from the cell to the closest obstacle, together with an associated weight W_k:

$$m_k(x, y) = [F_k(x, y), W_k(x, y)] \quad (1)$$

The distance function is the same projective TSDF used by KinectFusion, which is an approximation of a true signed distance function, theoretically correct only at the zero crossing. In practice, the averaging effect obtained from incremental updates from a moving sensor makes the approximation sufficient.

This representation has two important advantages over the classical occupancy grid:

- It allows extraction of coherent contours even after integrating very noisy readings, with sub-cell resolution.
- It is directional, in that cells with non-zero values don't appear as obstacles when seen from a different angle. Only the zero-crossings are interpreted as obstacles.

The process of integrating a new depth profile into the map for a given camera pose estimate consists of visiting and updating every cell in the map that could be affected by the new reading (given the camera's horizontal field of view and the current maximum depth reading). Each candidate cell is re-projected onto the camera to find its corresponding depth profile entry (nearest neighbor). The cell value is updated with the difference η between the distance from the camera to the cell and the corresponding value in the depth profile line (i.e. the distance from the camera to the obstacle along the same ray), truncated and normalized by the truncation threshold μ. As a result, cells very close to an obstacle get a value close to 0, while cells in front get a positive value increasing away from the surface. Cells behind the obstacle, up to the truncation threshold distance, are updated with a negative value decreasing away from the surface:

$$\Psi(\eta) = \begin{cases} \min\left(1, \frac{\eta}{\mu}\right) \operatorname{sgn}(\eta), & \text{iff } \eta \geq -\mu \\ \text{null}, & \text{otherwise} \end{cases} \quad (2)$$

The cells with value <1 make up the support region of the surface. The truncation threshold μ is scaled with the distance to the obstacle along the ray, so that obstacles farther away get a larger support region. This helps in integrating the large noise present in long-range readings.

Given a new depth profile S_k obtained at time k from the current robot position $\boldsymbol{p_0}$, the projective TSDF at point $\boldsymbol{p}$ is given by:

$$F_{S_k}(p) = \Psi(S_k(T_S p) - \|p - p_0\|_2) \quad (3)$$

$$W_{S_k}(p) = \frac{1}{\|p - p_0\|_2^{\,2}} \quad (4)$$

The map cell is then updated according to:

$$F_k(p) = \frac{F_{k-1}(p) * W_{k-1}(p) + F_{S_k}(p) * W_{S_k}(p)}{W_{k-1}(p) + W_{S_k}(p)} \quad (5)$$

$$W_k(p) = W_{k-1}(p) + W_{S_k}(p) \quad (6)$$

The weight associated with each update is a function of distance from the camera to the cell being updated. Since the weight of a cell is ever increasing, it also captures the number of times the cell was seen so far. The weight plays an important role in incrementally improving the obstacle contours as more observations are added to the map.

E. Incremental mapping

The critical ingredient of our method is the ability to maintain a correct local map from which a 360 degree contour can be easily extracted. The contour is then used as a scan line instead of the actual observation during navigation. Our method for incremental mapping is frame-to-model matching. The approach used by Kinect Fusion is to extract the surface from the current model using ray-tracing and to re-project it onto a virtual camera. Computing the cost function becomes a pixel-wise operation between the Kinect depth map and the depth map obtained via re-projection of the model, but the ray tracing operation is expensive. We opted for a simpler and cheaper way to compute an approximate version of the cost function by using the TSDF directly. Given a camera pose P_k, a depth profile S_k obtained from the latest camera frame k, and the current map m_k, we denote with $C(m_k, P_k, S_k)$ the cost function representing how well the observed depth profile S_k and the predicted contour match.

Figure 3. TSDF map generated using our incremental mapping method.

$$C(m_k, P_k, S_k) = \sum_{p \in S_k} |(F_k * W_k)(T_M T_R T_C p)| \quad (7)$$

When it comes to pose estimation via minimizing the cost function, Kinect's reduced field of view and high noise levels make single-solution search methods like gradient descent or ICP [24] of limited usefulness, particularly once the depth image is reduced to a scan line. There are many cases in which the robot faces an area devoid of salient features, resulting in a sequence of observations that cannot be properly aligned with the model. Presence of moving obstacles in the field of view can also present an alignment problem (outlier detection can help only to some extent). In such cases single-solution methods fail and need external input to recover. Our approach is to use a multi-theory search method instead, implemented as a particle filter that approximates the probability distribution over the robot pose given the robot motion, the most likely map and a set of observations, similar to [25]. The filter is updated at frame rate, and the average lifespan of a particle is <10 seconds. This is in contrast with Rao–Blackwellized particle filter implementations, which keep multiple theories alive (together with their associated maps) until a loop closure is detected, typically by updating the filter with low frequency [9], [15]. Hence, we refer to our method as an incremental mapping method rather than a SLAM solution.

Because computing the distance function is cheap (linear in the horizontal resolution of the sensor), and because we only maintain the map generated by the maximum likelihood solution, the particle filter can maintain and update thousands

of particles at frame rate. The filter update is also fully parallelizable, since particles of the same generation are independent of each other. In our tests, the ability to compute many more candidate poses appears to outweigh the fact that the cost function is a relatively coarse approximation of fitness when compared to the ray-traced approach.

As with any incremental mapping method, our method suffers from drift over long distances and/or time intervals. To ensure that our local map remains consistent in the vicinity of the robot, we update it using a sliding window approach. We maintain a queue of the observations currently integrated into the map. The queued observations are removed from the map (and from the queue) once they become too old (>120s) or once the robot traveled a maximum distance (20m). The removal operation is the inverse of the map update operation:

$$F_k(p) = \frac{F_{k-1}(p) * W_{k-1}(p) - F_{S_k}(p) * W_{S_k}(p)}{W_{k-1}(p) - W_{S_k}(p)} \quad (8)$$

$$W_k(p) = W_{k-1}(p) - W_{R_k}(p) \quad (9)$$

F. Contour extraction

To compensate for the small field of view of the Kinect, we generate, after each map update, a virtual observation in the form of a dense 360 degree depth profile. For a given robot position and orientation, the visible obstacle contours are extracted from the current map using ray-tracing, by visiting the map cells along each ray in a dense set of rays within a given field of view. The method iterates over map cells in order of increasing distance from the observer for as long as the current cell value is positive. When a negative value is encountered, the surface has been crossed and its exact position can be computed by linearly interpolating between the positive and negative values. The resulting position is added to the point set that describes the surface. The positive and negative values are themselves interpolated from the surrounding cells. Once all the desired rays in the field of view have been traced, the computed point set describes the surface as seen from the given pose with the given field of view.

Note that, unlike Kinect Fusion, our ray tracing visits every cell along the ray. We found the Kinect Fusion approach of space skipping to be unreliable when applied to our maps, due to the nature of the robot motion. In particular, when moving along a corridor, the generated support region of the walls is thin because of the sharp viewing angle. When the robot rotates, changing the viewing direction, the space skipping method ends up jumping over the thin support region, missing the surface.

G. Global map

The global map used for navigation is represented as a grid map containing a true signed distance function. The global map is generated in an offline step from data recorded while manually driving the robot around the target environment. Loop closure information is provided by pressing a button during recording. An initial estimate of the trajectory of the robot is obtained using the incremental mapping procedure, and is then corrected in a graph optimization step based on the loop closure information. A TSDF map is generated using the updated trajectory and is then transformed to a true distance function by ray-tracing from every map cell and retaining the minimum distance to the extracted contour.

H. Navigation

The key ingredient of our localization method is the generation, at frame rate, of a virtual 360 degree scan line from the most recent observation and a local map that is always up to date. The 360 degree contour is extracted from the local map by ray-tracing as described in Section F. Then, the most recent observation is used to replace the section of the scan line that it overlaps with (after a simple conversion of the observation from pin-hole model to angular model). The resulting combined scan line makes it look as if the robot was equipped with a 360 degree LIDAR. However, our scan line is denser, containing 3600 readings, or 10x more than most LIDARs.

Localization is performed using a particle filter with the same cost function $C(m_k, P_k, S_k)$ described earlier, this time applied to the true distance function map. The motion update of the filter is based on the motion estimate output of the incremental mapping subsystem.

To handle changes to the environment such as furniture being moved, path (re)planning is performed with high frequency against a combined map obtained by overlaying the most recent local map over the global map, given the most likely pose from the localization subsystem. While fairly simplistic and theoretically prone to live-locks due to oscillating alignment of the maps (and the resulting oscillation of the planned path), this approach appears to work well in practice. Nonetheless, this area merits more attention and future work.

Dynamic obstacles such as people are handled at path tracking level, using a purely reactive obstacle avoidance strategy consisting of a short sequence of random rotations and translations towards open space. The random behavior is designed to eliminate the risk of live-locks between obstacle avoidance and path tracking.

IV. Experimental Results

We conducted several experiments to compare our method with existing state-of-the-art implementations and to quantify the benefits of our continuous mapping technique. For reasons of size and drive train similarities with our hardware prototype, we chose to compare our platform with two similar platforms running complete mapping and navigation solutions: TurtleBot 2 (from Yujin Robot) and Adept Pioneer 3DX. The TurtleBot is an open source platform based on iRobot Create. It is equipped with a Microsoft Kinect, and runs Linux and ROS [26]. The Pioneer 3DX is a commercial platform produced by Adept Mobile Robots. It is equipped with a Sick LIDAR and runs Adept's proprietary ARNL navigation software [1].

It became obvious early on that our TurtleBot system (using the default mapping and navigation solution) was not able to handle the large space we picked for our benchmark, and thus we removed it from the test. The Adept Pioneer on the other hand proved to be a capable contender. Note that in

our experiments we used the research version of the Adept ARNL software.

A. Benchmarking method

We used the BRIN method (Benchmark for Robotic Indoor Navigation) to quantify the performance of our navigation method in large environments. The benchmark method was introduced in [4] and was presented in detail in [27].

To summarize, we picked a large environment consisting of four areas: office, hallway, lounge and atrium. The lounge and atrium share an open space of more than 30 x 30 m. The lounge area was densely furnished, while the atrium was mostly empty. We chose 8 interesting places in the environment and marked one waypoint in each place by applying an overhead fiducial. We ensured that the robot visited these waypoints during mapping, and we captured the position of the robot on the generated map, as well as the position of the robot in the real world, relative to the fiducial associated with the waypoint being visited. This last step was done using an up-facing camera mounted on the robot and the ground truth software described in [28]. Note that the up-facing camera was used solely for ground truth information and was not available to the navigation software.

Once the initial map was created and the waypoints were recorded, the robot was instructed to visit the waypoints in a particular sequence designed to maximize exposure to environmental challenges. The track length of the sequence was about 250m. The sequence was repeated 12 times, for a total of 3km, and every time a different set of modifications was made to the environment, such as people walking by or blocking the robot, doors being closed, obstacles placed in the path and furniture being disturbed or moved to new locations. For a complete description of the benchmark procedure see [4].

The main metric used to compare the tested systems was the total number of failures. Failure was defined as the inability of the robot to reach the next waypoint in the set within a given amount of time. In all failure cases observed it was fairly obvious that the robot could not recover and would never reach the waypoint. After a failure, the robot was reset to the next waypoint and the test resumed.

We ran the tested platforms one after the other for each sequence of points, thus guaranteeing that the systems were exposed to the same changes in the environment. The tests were run in late afternoon and evening, to eliminate the possibility of interference.

B. Results

Table 1 below quantifies the contribution of the mapping and localization strategy described in this paper. It compares the benchmark results obtained in two distinct benchmark runs, performed a few weeks apart in the same environment. The two runs used the same hardware platform but different versions of the navigation systems. In the first run we used a simpler version of our navigation stack which implemented the same depth frame processing method but relied directly on the resulting 60° scan line to localize against a static map. As expected, the majority of the failures were due to loss of localization, either when the furniture was moved (lounge) or

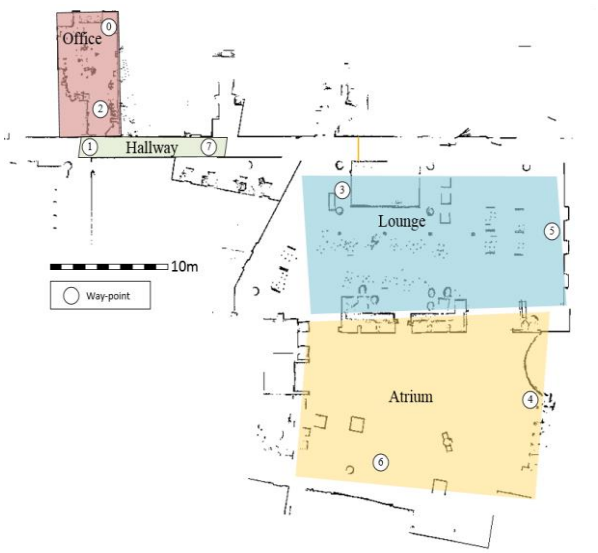

Figure 5. Map of the environment in which the benchmark was conducted.

when encountering people in areas with sparse features (traversing the atrium).

The second run used the system described in this paper and showed a dramatic improvement across the board. The always up-to-date local map and the virtual 360° view generated from it helped a lot in maintaining good localization even when the environment changed, resulting in a five-fold increase in positioning accuracy.

Because these were distinct benchmark sessions, the results are normalized by the performance of the reference platform in each session, as required by BRIN.

Table 2 shows the results of our prototype, MSR-P1, and the Pioneer 3DX robot. Our system outperformed the Pioneer in all regards except speed. Accuracy of the two systems is almost the same, even though the LIDAR used by the Pioneer system is 10x more accurate than the Kinect, has 5x the range and 6x the field of view. Note that failures are not reflected in the positioning error, since measuring the positioning error requires the robot to arrive at the landmark to begin with.

Metric	**No local map, 60° FOV**	**Local map, 360° FOV**
Number of failures*	2.40	0.56
Mean time to failure	0.23	3.12
Max time to failure	0.43	2.55
Mean dist. to failure	0.13	2.01
Max dist. to failure	0.27	1.82
Average speed	0.50	0.64
Positioning error*	5.22	1.05

TABLE 1. Comparison of navigation results with and without the incremental mapping method, using the MSR-P1 platform. The results are normalized with the performance of the reference platform, and are thus unit-less. Larger values are better, except for items marked with *.

Metric	MSR-P1	Pioneer
Number of failures	5	9
Mean time to failure	2265 s	726 s
Max time to failure	5023 s	1971 s
Mean dist. to failure	367 m	183 m
Max dist. to failure	860 m	472 m
Average speed	0.16 m/s	0.25 m/s
Positioning error	0.23m ±0.2m	0.22m ±0.1m

TABLE 2. Comparison of results obtained with MSR-P1 running our method and the Pioneer 3DX running the ARNL navigation system.

While the Pioneer software was better in navigating a static environment, especially in the large atrium, the ability of our robot to deal with environment changes was ultimately the main differentiator in the overall score.

The breakdown of the failures of MSR-P1 is as follows: 1 software crash, 3 localization failures and 1 localization divergence. The Pioneer failures were: 1 software hang, 5 localization failures, 1 path oscillation for more than 5 minutes, 1 not able to traverse path plotted around a new obstacle and 1 failure to detect a low obstacle.

V. CONCLUSION

In this paper we have shown that the Microsoft Kinect depth camera is a viable low-cost alternative to LIDARs for indoor navigation scenarios. We have devised a method for correcting systematic error in sensor readings and for coping with large random noise in long-range Kinect data. We introduced a method for extracting better statistics about obstacle proximity. Finally, we have shown how, by performing continuous incremental mapping and by using a TSDF map representation, we can compensate for the limited field of view of the Kinect sensor. Our experimental results demonstrate that the resulting system can outperform existing state-of-the-art indoor navigation systems that use LIDARs, particularly when it comes to overall reliability in dynamic environments. Despite the LIDARs significant advantage in accuracy, range and field of view over Kinect, our system achieves similar positioning accuracy and has a significantly lower failure rate.

References

[1] "Adept Mobile Robots ARNL Software," [Online]. Available: http://robots.mobilerobots.com/wiki/ARNL.

[2] "Turtlebot navigation package," [Online]. Available: http://wiki.ros.org/turtlebot_navigation.

[3] R. A. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim and A. Fitzgibbon, "KinectFusion: Real-time dense surface mapping and tracking.," in *International symposium on Mixed and augmented reality (ISMAR)*, 2011.

[4] C. Sprunk, J. Röwekämper, G. Parent, L. Spinello, G. D. Tipaldi, W. Burgard and M. & Jalobeanu, "An Experimental Protocol for Benchmarking Robotic Indoor Navigation.," in *International Symposium on Experimental Robotics (ISER)*, 2014.

[5] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard and J. McDonald, "Kintinuous: Spatially extended kinectfusion," 2012.

[6] R. A. Newcombe, S. J. Lovegrove and A. J. Davison, "DTAM: Dense tracking and mapping in real-time," in *International Conference on Computer Vision (ICCV)*, 2011 .

[7] N. Engelhard, F. Endres, J. Hess, J. Sturm and W. Burgard, "Real-time 3D visual SLAM with a hand-held RGB-D camera.," in *RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum*.

[8] I. Dryanovski, R. G. Valenti and J. Xiao, "Fast visual odometry and mapping from rgb-d data," in *International Conference on Robotics and Automation (ICRA)*, 2013.

[9] D. Hahnel, W. Burgard, D. Fox and S. Thrun, "An efficient FastSLAM algorithm for generating maps of large-scale cyclic environments from raw laser range measurements," in *International Conference on Intelligent Robots and Systems (IROS)*, 2003.

[10] S. Kohlbrecher, O. v. Stryk, J. Meyer and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," in *International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 2011.

[11] J.-S. Gutmann and K. Konolige, "Incremental mapping of large cyclic environments," in *International Symposium on Computational Intelligence in Robotics and Automation*, 1999.

[12] K. P. Murphy, "Bayesian Map Learning in Dynamic Environments," *NIPS*, pp. 1015-1021, 1999.

[13] M. Montemerlo, S. Thrun, D. Koller and B. Wegbreit, "FastSLAM: A factored solution to the simultaneous localization and mapping problem.," *AAAI/IAAI*, pp. 593-598, 2002.

[14] A. Eliazar and R. Parr, "DP-SLAM: Fast, robust simultaneous localization and mapping without predetermined landmarks," *IJCAI*, vol. 3, pp. 1135-1142, 2003.

[15] G. Grisetti, G. D. Tipaldi, C. Stachniss, W. Burgard and D. Nardi, "Fast and accurate SLAM with Rao–Blackwellized particle filters," *Robotics and Autonomous Systems*, vol. 55, no. 1, pp. 30-38, 2007.

[16] J. Biswas and M. Veloso, "Depth Camera Based Indoor Mobile Robot Localization and Navigation," in *International Conference on Robotics and Automation (ICRA)*, 2012 .

[17] K. Pathak, A. Birk, N. Vaskevicius, M. Pfingsthorn, S. Schwertfeger and J. Poppinga, "Online three-dimensional SLAM by registration of large planar surface segments and closed-form pose-graph relaxation.," *Journal of Field Robotics*, pp. 52-84, 2010.

[18] A. Teichman, S. Miller and S. Thrun, "Unsupervised intrinsic calibration of depth sensors via SLAM," in *Robotics: Science and Systems*, 2013.

[19] J. Smisek, M. Jancosek and T. Pajdla, "3d with kinect," in *IEEE Workshop on Consumer Depth Cameras for Computer Vision*, 2011.

[20] Z. Zhang, "A flexible new technique for camera calibration," *Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330-1334., 2000.

[21] E. Ringaby and P.-E. Forssen, "Scan rectification for structured light range sensors with rolling shutters.," in *International Conference on Computer Vision (ICCV)*, 2011.

[22] D. Holz, S. Holzer, R. B. Rusu and S. Behnk, "Real-time plane segmentation using RGB-D cameras," in *RoboCup 2011: Robot Soccer World Cup XV*, Springer Berlin Heidelberg, 2012, pp. 306-317.

[23] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Sixth International Conference on Computer Vision*, 1998.

[24] F. Lu and E. Milios, "Robot pose estimation in unknown environments by matching 2d range scans," *Journal of Intelligent and Robotic Systems*, vol. 18, no. 3, pp. 249-275, 1997.

[25] S. Thrun, W. Burgard and D. Fox, "A real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping," in *International Conference on Robotics and Automation*, 2000.

[26] "TurtleBot," [Online]. Available: http://www.turtlebot.com/.

[27] "Benchmark for Robotic Indoor Navigation," [Online]. Available: http://research.microsoft.com/en-us/projects/brin/.

[28] H. Kikkeri, G. Parent, M. Jalobeanu and S. Birchfield, "An Inexpensive Methodology for Evaluating the Performance of a Mobile Robot Navigation System," in *International Conference on Robotics and Automation (ICRA)*, 2014.