# Localization and Navigation of the CoBots Over Long-term Deployments

Joydeep Biswas and Manuela M. Veloso

***Abstract*—For the last three years, we have developed and researched multiple collaborative robots, CoBots, which have been autonomously traversing our multi-floor buildings. We pursue the goal of long-term autonomy for indoor service mobile robots as the ability for them to be deployed indefinitely while they perform tasks in an evolving environment. The CoBots include several levels of autonomy, and in this paper we focus on their localization and navigation algorithms. We present the Corrective Gradient Refinement (CGR) algorithm, which refines the proposal distribution of the particle filter used for localization with sensor observations using analytically computed state space derivatives on a vector map. We also present the Fast Sampling Plane Filtering (FSPF) algorithm that extracts planar regions from depth images in real time. These planar regions are then projected onto the 2D vector map of the building, and along with the laser rangefinder observations, used with CGR for localization. For navigation, we present a hierarchical planner, which computes a topological policy using a graph representation of the environment, computes motion commands based on the topological policy, and then modifies the motion commands to side-step perceived obstacles. The continuous deployments of the CoBots over the course of one and a half years have provided us with logs of the CoBots traversing more than $130km$ over $1082$ deployments, which we publish as a dataset consisting of more than $10$ million laser scans. The logs show that although there have been continuous changes in the environment, the robots are robust to most of them, and there exist only a few locations where changes in the environment cause increased uncertainty in localization.**

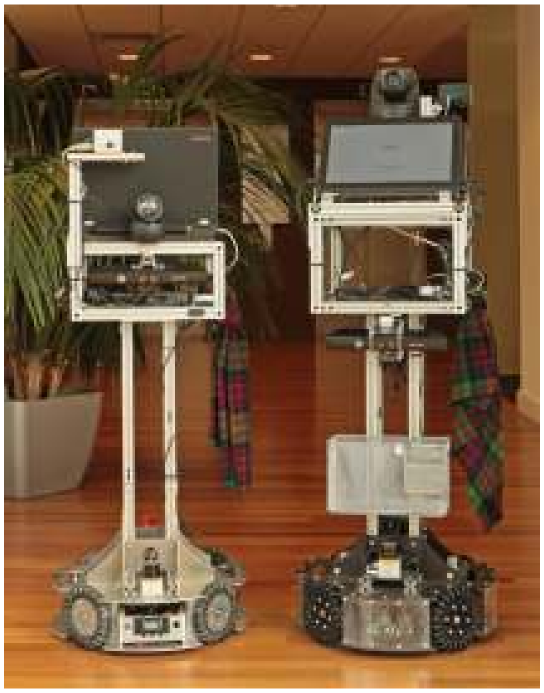


Fig. 1: The two CoBots, CoBot1 (left) and CoBot2 (right) that have been deployed in our office environment since September 2011.

## 1. Introduction

We pursue the goal of long-term autonomy for indoor service mobile robots, as the ability for the robots to be deployed indefinitely in an evolving environment. Our Collaborative Robots (CoBots) autonomously perform tasks on multiple floors of our office building, including escorting visitors, giving tours, and transporting objects. Two CoBots have been deployed in our building since 2010, and for the period of September 2011 to January 2013, they have logged data while autonomously traversing more than $130km$ over 1082 deployments and a total run time of 182 hours. The CoBot robots rely on the tight integration of a number of autonomous components, including a *symbiotic human-robot relationship* [Rosenthal et al., 2010], [Rosenthal and Veloso, 2012], the capability of seeking information from the web [Samadi et al., 2012], and the core localization and navigation algorithms [Biswas and Veloso, 2010], [Biswas et al., 2011], [Biswas and Veloso, 2012] that enable the robot to traverse the environment without human supervision or chaperoning. In this article, we focus on the localization and navigation algorithms that enable the CoBots to reliably operate autonomously over time, as demonstrated by the analysis of the extensive sensor logs that have collected. The localization and navigation algorithms we contribute are general to any mobile robot equipped with similar sensing modalities. The sensor logs comprise, to the best of our knowledge, the largest (in terms of number of observations, distance traversed, and duration) data set of an autonomous indoor mobile robot to date, and we make them publicly available[1] in the hope that they will prove valuable to the researchers in the robotics community.

The CoBots (Fig. 1) are four-wheeled omnidirectional robots, [2] purposefully simple, equipped with an inexpensive

Joydeep Biswas is with the Robotics Institute, and Manuela M. Veloso is with the Computer Science Department, School of Computer Science, Carnegie Mellon University (e-mail:{joydeepb, veloso}@cs.cmu.edu). This work was supported by the National Science Foundation award number NSF IIS-1012733.

[1]Logs of CoBots' deployments may be downloaded from: http://www.cs.cmu.edu/~coral/cobot/data.html

[2]The CoBot robots were designed and built by Michael Licitra, mlicitra@cmu.edu, with the base being inspired by the hardware of the CMDragons small-size soccer robots [Bruce et al., 2007], also designed and built by Licitra.

short-range laser rangefinder sensor, and an inexpensive depth camera. An on-board tablet provides the computational platform to run the complete algorithms for sensing and control, and also provides a graphical and speech-based interface for users.

For their localization and navigation, the CoBots sense the environment through depth cameras and laser rangefinders. The depth images observed by the depth camera are filtered by Fast Sampling Plane Filtering (FSPF) [Biswas and Veloso, 2012] to extract points that correspond to planar regions. A 2D vector map is used to represent the long-term features (like walls and permanent fixtures) in the environment as line segments. The localization algorithm uses an analytic ray cast of the vector map to find correspondences between the line segments in the map and the observations including plane-filtered points from depth images and laser rangefinder observations. These correspondences are then used to update the predicted location of the robot (also including odometry) using Corrective Gradient Refinement (CGR) [Biswas et al., 2011]. CGR extends Monte-Carlo Localization (MCL) [Dellaert et al., 1999] by using analytically computed state-space derivatives of the observation model to refine the proposal distribution prior to the observation update.

The navigation algorithm on the CoBots uses a two-level hierarchy. At the high level, the algorithm uses a topological graph of the traversable paths in the environment to plan a topological policy for a given destination. This topological policy is converted to motion commands using the "Next Maximal Action" algorithm [Biswas and Veloso, 2010]. At the low level, the motion commands are modified by the obstacle avoidance algorithm to side-step around obstacles by making use of the omnidirectional drive capability of the CoBots.

This article is organized as follows. Section 2 reviews related work on localization and navigation algorithms used for indoor mobile robots, and robots that have been deployed in the past in human environments. Section 3 describes the Fast Sampling Plane Filtering (FSPF) algorithm used to extract planar points from depth images, Section 4 describes the analytic ray casting algorithm used to find correspondences between observations and the map, and Section 5 describes the Corrective Gradient Refinement (CGR) algorithm used to compute location estimates of the robot based on the observations and their correspondences with the map. Section 6 presents the hierarchical navigation algorithm of CoBot. Section 7 presents the long-term deployments of the CoBots, the sensor logs collected in the process, and provides instructions on how to obtain this dataset. Section 9 concludes with a summary of lessons learnt from the deployments of CoBot, and directions for future work.

## 2. Related Work

We first briefly review the state of the art in robot localization and mapping, robots deployed in human environments, approaches to long-term autonomy of robots, and publicly available datasets for long-term autonomy. We then contrast the related work with our approach, and highlight the contributions of this article.

### *1. Localization and Mapping*

Kalman Filter [Kalman, 1960] and Monte Carlo Localization (MCL) [Dellaert et al., 1999] based algorithms comprise the vast majority of localization algorithms used in practice today. Due to the nonlinear nature of robot localization, Kalman Filter based algorithms use one of several nonlinear variants [Lefebvre et al., 2004]. Sensor modalities used with these algorithms include geometric beacons [Leonard and Durrant-Whyte, 1991], SONAR [Jetto et al., 1999], and laser rangefinder [Roumeliotis and Bekey, 2000]. Monte Carlo Localization (MCL) [Dellaert et al., 1999] represents the probability distribution of the robot's location with a set of discrete samples that are propagated through a probabilistic prediction function based on a motion model of the robot, and either the observed odometry or command actions, followed by a observation based weighting function and a resample step. The MCL algorithm, though successful in most cases, suffers from a few limitations, including the inability to scale efficiently between large variations in uncertainty, and the difficulty of recovering from incorrect estimates (the "kidnapped robot" problem). KLD-sampling [Fox, 2001] extends MCL by dynamically varying the number of particles used, based on the uncertainty as determined by the spread of the proposal distribution. Sensor Resetting Localization (SRL) [Lenser and Veloso, 2000] probabilistically injects samples from the observation likelihood function into the proposal distribution when the observation likelihood drops, thereby allowing faster recovery from the kidnapped robot problem than MCL. Corrective Gradient Refinement (CGR) [Biswas et al., 2011] uses analytically computed state-space derivatives of the observation model to refine the proposal distribution prior to the observation update, thus allowing the samples to sample densely along directions of high uncertainty while confining the samples along directions of low uncertainty.

Many approaches to Simultaneous Localization and Mapping (SLAM) have been proposed to date and surveyed [Thrun et al., 2002], [Durrant-Whyte and Bailey, 2006], [Bailey and Durrant-Whyte, 2006]. The Rao-Blackwellized Particle Filter (RBPF) [Doucet et al., 2000] in particular has been shown to be successful in solving the SLAM problem by decoupling the robot's trajectory from the estimate of the map, and sampling only from the trajectory variables while maintaining separate estimators for the map for each sample. Graph SLAM [Thrun and Montemerlo, 2006] represents the trajectory of the robot as a graph with nodes representing the historic poses of the robot and map features, and edges to represent relations between the nodes based on odometry and matching observations.

### *2. Robots Deployed In Human Environments*

Shakey the robot [Nilsson, 1984] was the first robot to actually perform tasks in human environments by decomposing tasks into sequences of actions. Rhino [Buhmann et al., 1995], a robot contender at the 1994 AAAI Robot Competition and Exhibition, used SONAR readings to build an occupancy grid map [Elfes, 1989], and localized by matching its observations to expected wall orientations. Minerva [Thrun et al., 1999] served as a tour guide in a Smithsonian museum. It used laser scans and camera images along with odometry to construct two maps, the first being an occupancy grid map, the second a textured map of the ceiling. For localization, it explicitly split up its observations into those corresponding to the fixed map and those estimated to have been caused by dynamic obstacles. Xavier [Koenig and Simmons, 1998] was a robot deployed in an office building to perform tasks requested by users over the web. Using observations made by SONAR, a laser striper and odometry, it relied on a Partially Observable Markov Decision Process (POMDP) to reason about the possible locations of the robot, and to reason about the actions to choose accordingly. A number of robots, Chips, Sweetlips and Joe Historybot [Nourbakhsh et al., 2003] were deployed as museum tour guides at the Carnegie Museum of Natural History in Pittsburgh. Artificial fiducial markers were placed in the environment to provide accurate location feedback for the robots. The PR2 robot at Willow Garage [Oyama et al., 2009] has been demonstrated over a number of "milestones" where the robot had to navigate over 42 km and perform a number of manipulation tasks. The PR2 used laser scan data along with Inertial Measurement Unit (IMU) readings and odometry to build an occupancy grid map using GMapping [Grisetti et al., 2007], and then localized itself on the map using KLD-sampling [Fox, 2001].

### 3. *Long-Term Autonomy*

Recognizing the need for robots to adapt to changes in the environment in order to remain autonomous over extended periods of time, a number of approaches have been proposed to dynamically update the map of the environment over time. One such approach is to maintain a history of local maps over a number of time scales [Biber and Duckett, 2005]. The Independent Markov Chain approach [Saarinen et al., 2012] maintains an occupancy grid with a Markov chain associated with every bin to model their probabilities of remaining as well as transitioning from occupied from unoccupied, and vice-versa. Dynamic Pose Graph SLAM [Walcott-Bryant et al., 2012] maintains a pose graph of all observations made by the robot, and successively prunes the graph to remove old observations invalidated by newer ones, thus keeping track of an up-to-date estimate of the map. For long-term autonomy using vision, the approach of "Visual Experiences" [Churchill and Newman, 2012] uses a visual odometry algorithm along with visual experiences (sets of poses and feature locations), and successively builds a topological map as sets of connected visual experiences.

### 4. *Datasets for Long-Term Autonomy*

To tackle the problems of long-term autonomy, datasets of robots navigating in environments over extended periods of time are invaluable, since they provide insight into the actual challenges involved and also serve to evaluate the strengths and limitations of existing algorithms. Gathering such datasets is difficult since it requires robots to be deployed over long periods of time, which would require considerable human time unless the robots were autonomous. In addition to the data itself, annotation of the data (such as ground truth and places where the robot had errors) is crucial to evaluating the success of algorithms geared towards long-term autonomy. A five-week experiment at Örebro University, Sweden[3] [Biber and Duckett, 2009] provided a five-week long dataset of a robot driven manually around in an office environment, traversing $9.6$ km, and over $100,000$ laser scans. Datasets of a robot recording omni-directional images in an office environment[4], an office room[5] and a lab environment[6] were recorded at University of Lincoln [Dayoub et al., 2011].

### 5. *Contrast and Summary of Contributions*

Our approach to localization uses a vector map, which represents long-term features in the environment as a set of line segments, as opposed to occupancy grids or sets of raw observations as used by other localization and SLAM algorithms. We introduce the Corrective Gradient Refinement (CGR) algorithm that extends Monte Carlo Localization and allows efficient sampling of the localization probability distribution by spreading out samples along directions of higher uncertainty and consolidating the samples along directions of lower uncertainty. We also introduce the Fast Sampling Plane Filtering (FSPF) algorithm, a computationally efficient algorithm for extracting planar regions from depth images. CoBot's localization uses CGR with these observed planar regions, along with laser rangefinder observations. The FSPF-CGR localization with a vector map is a novel approach that we demonstrate on the CoBots. Although the CoBots do not perform SLAM or track map changes online, the localization and navigation algorithms that we present are robust to most changes in the environment. We present an extensive dataset of the robot autonomously navigating in the building while performing tasks. To the best of our knowledge this dataset is the longest publicly available dataset of an autonomous indoor mobile robot, in terms of duration, distance traversed, and volume of observations collected. In summary, the contributions of this article include

1) the Fast Sampling Plane Filtering (FSPF) algorithm for extracting planar regions from depth images,
2) the Corrective Gradient Refinement (CGR) algorithm, which extends MCL and allows better sampling of the localization probability distribution,
3) results from long-term deployments demonstrating the robustness of the above algorithms, and
4) an extensive dataset of robot observations and run-time estimates gathered by the CoBots over these deployments.

[3]https://ckan.lincoln.ac.uk/dataset/ltmro-4
[4]https://ckan.lincoln.ac.uk/dataset/ltmro-1
[5]https://ckan.lincoln.ac.uk/dataset/ltmro-2
[6]https://ckan.lincoln.ac.uk/dataset/ltmro-3

## 3. Fast Sampling Plane Filtering

The CoBots use observations from on-board depth cameras for both localization, as well as safe navigation. Depth cameras provide, for every pixel, color and depth values. This depth information, along with the camera intrinsics consisting of horizontal field of view $f_h$, vertical field of view $f_v$, image width $w$ and height $h$ in pixels, can be used to reconstruct a 3D point cloud. Let the depth image of size $w \times h$ pixels provided by the camera be $I$, where $I(i, j)$ is the depth value of a pixel at location $d = (i, j)$. The corresponding 3D point $p = (p_x, p_y, p_z)$ is reconstructed using the depth value $I(d)$ as $p_x = I(d)\left(\frac{j}{w-1} - 0.5\right)\tan\left(\frac{f_h}{2}\right)$, $p_y = I(d)\left(\frac{i}{h-1} - 0.5\right)\tan\left(\frac{f_v}{2}\right)$, $p_z = I(d)$.

With the limited computational resources on a mobile robot, most algorithms (*e.g.*, localization, mapping) cannot process the full 3D point cloud at full camera frame rates in real time. The naïve solution would therefore be to sub-sample the 3D point cloud, for example by dropping one out of $N$ points, or by dropping frames. Although this would reduce the number of 3D points being processed, it would end up discarding information about the scene. An alternative solution is to convert the 3D point cloud into a more compact, feature-based representation, like planes in 3D. However, computing optimal planes to fit the point cloud for every observed 3D point would be extremely CPU-intensive and sensitive to occlusions by obstacles that exist in real scenes. The Fast Sampling Plane Filtering (FSPF) algorithm combines both ideas: it samples random neighborhoods in the depth image, and in each neighborhood, it performs a RANSAC-based plane fitting on the 3D points. Thus, it reduces the volume of the 3D point cloud, it extracts geometric features in the form of planes in 3D, and it is robust to outliers since it uses RANSAC within the neighborhood.

FSPF takes the depth image $I$ as its input, and creates a list $P$ of $n$ 3D points, a list $R$ of corresponding plane normals, and a list $O$ of outlier points that do not correspond to any planes. Algorithm 1 outlines the plane filtering procedure. It uses the helper subroutine $[\text{numInliers}, \hat{P}, \hat{R}] \leftarrow \text{RANSAC}(d_0, w', h', l, \epsilon)$, which performs the classical RANSAC algorithm over the window of size $w' \times h'$ around location $d_0$ in the depth image, and returns inlier points and normals $\hat{P}$ and $\hat{R}$ respectively, as well as the number of inlier points found. The configuration parameters required by FSPF are listed in Table I.

FSPF proceeds by first sampling three locations $d_0$,$d_1$,$d_2$ from the depth image (lines 9-11). The first location $d_0$ is selected randomly from anywhere in the image, and then $d_1$ and $d_2$ are selected from a neighborhood of size $\eta$ around $d_0$. The 3D coordinates for the corresponding points $p_0$, $p_1$, $p_2$ are then computed (line 12). A search window of width $w'$ and height $h'$ is computed based on the mean depth ($z$-coordinate) of the points $p_0$, $p_1$, $p_2$ (lines 14-16) and the minimum expected size $S$ of the planes in the world. Local RANSAC is then performed in the search window. If more than $\alpha_{in} l$ inlier points are produced as a result of running RANSAC in the search window, then all the inlier points are added to the list $P$, and the associated normals (computed using a least-squares fit on the RANSAC inlier points) to the list $R$. This algorithm runs a maximum of $m_{max}$ times to generate a list of maximum $n_{max}$ 3D points and their corresponding plane normals.

**Algorithm 1** Fast Sampling Plane Filtering

```
1:  procedure PlaneFiltering($I$)
2:      $P \leftarrow \{\}$                          ▷ Plane filtered points
3:      $R \leftarrow \{\}$                          ▷ Normals to planes
4:      $O \leftarrow \{\}$                          ▷ Outlier points
5:      $n \leftarrow 0$                             ▷ Number of plane filtered points
6:      $k \leftarrow 0$                             ▷ Number of neighborhoods sampled
7:      while $n < n_{max} \wedge k < k_{max}$ do
8:          $k \leftarrow k + 1$
9:          $d_0 \leftarrow (\text{rand}(0, h-1), \text{rand}(0, w-1))$
10:         $d_1 \leftarrow d_0 + (\text{rand}(-\eta, \eta), \text{rand}(-\eta, \eta))$
11:         $d_2 \leftarrow d_0 + (\text{rand}(-\eta, \eta), \text{rand}(-\eta, \eta))$
12:         Reconstruct $p_0$, $p_1$, $p_2$ from $d_0$,$d_1$,$d_2$
13:         $r = \frac{(p_1 - p_0) \times (p_2 - p_0)}{||(p_1 - p_0) \times (p_2 - p_0)||}$   ▷ Compute plane normal
14:         $\bar{z} = \frac{p_{0z} + p_{1z} + p_{2z}}{3}$
15:         $w' = w\frac{S}{\bar{z}}\tan(f_h)$
16:         $h' = h\frac{S}{\bar{z}}\tan(f_v)$
17:         $[\text{numInliers}, \hat{P}, \hat{R}] \leftarrow \text{RANSAC}(d_0, w', h', l, \epsilon)$
18:         if numInliers $> \alpha_{in} l$ then
19:             Add $\hat{P}$ to $P$
20:             Add $\hat{R}$ to $R$
21:             numPoints ← numPoints + numInliers
22:         else
23:             Add $\hat{P}$ to $O$
24:         end if
25:     end while
26:     return $P, R, O$
27: end procedure
```

| Parameter | Value | Description |
|---|---|---|
| $n_{max}$ | 2000 | Maximum total number of filtered points |
| $k_{max}$ | 20000 | Maximum number of neighborhoods to sample |
| $l$ | 80 | Number of local samples |
| $\eta$ | 60 | Neighborhood for global samples (in pixels) |
| $S$ | $0.5m$ | Plane size in world space for local samples |
| $\epsilon$ | $0.02m$ | Maximum plane offset error for inliers |
| $\alpha_{in}$ | 0.8 | Minimum inlier fraction to accept local sample |

TABLE I: Configuration parameters for FSPF

Once the list of planar points $P$ observed by the depth camera has been constructed, these points, along with the points observed by the laser rangefinder sensor, are matched to the expected features on the map. The map representation, and the observation models are the subjects of the next section.

## 4. Vector Map And Analytic Ray Casting

The map representation that we use for localization is a vector map: it represents the environment as a set of line segments (corresponding to the obstacles in the environment), as opposed to the more commonly used occupancy grid [Elfes, 1989] based maps. The observation model therefore has to compute the line segments likely to be observed by the robot given its current pose and the map. This is done by an analytic ray cast step. We introduce next the representation of the vector map and the algorithm for analytic ray casting using the vector map.

The map $M$ used by our localization algorithm is a set of $s$ line segments $l_i$ corresponding to all the walls in the environment: $M = \{l_i\}_{i=1:s}$. Such a representation may be acquired by mapping (*e.g.*, [Zhang and Ghosh, 2000]) or, as in our case, taken from the architectural plans of the building.

Given this map and the pose estimate of the robot, to compute the observation likelihoods based on observed planes, the first step is to estimate the *scene lines* $L$ which are sections of the lines on the map likely to be observed. This ray casting step is analytically computed using the vector map representation.

The procedure to analytically generate a ray cast at location $x$ given the map $M$ is outlined in Algorithm 2. The returned result is the scene lines $L$: a list of non-intersecting, non-occluded line segments visible by the robot from the location $x$. This algorithm calls the helper procedure $\mathrm{TrimOcclusion}(x, l_1, l_2, L)$, which accepts a location $x$, two lines $l_1$ and $l_2$ and a list of lines $L$. TrimOcclusion trims line $l_1$ based on the occlusions due to the line $l_2$ as seen from the location $x$. The list $L$ contains lines that yet need to be tested for occlusions by $l_2$. There are in general 4 types of arrangements of $l_1$ and $l_2$, as shown in Fig. 2:

1) $l_1$ is not occluded by $l_2$. In this case, $l_1$ is unchanged.
2) $l_1$ is completely occluded by $l_2$. $l_1$ is trimmed to zero length by TrimOcclusion.
3) $l_1$ is partially occluded by $l_2$. $l_1$ is first trimmed to a non occluded length, and if a second disconnected non occluded section of $l_1$ exists, it is added to $L$.
4) $l_1$ intersects with $l_2$. Again, $l_1$ is first trimmed to a non occluded length, and if a second disconnected non occluded section of $l_1$ exists, it is added to $L$.

**Algorithm 2** Analytic Ray Cast Algorithm

```
1:  procedure AnalyticRayCast($M, x$)
2:      $\hat{L} \leftarrow M$
3:      $L \leftarrow \{\}$
4:      for $l_i \in \hat{L}$ do
5:          for $l_j \in L$ do
6:              $\mathrm{TrimOcclusion}(x, l_i, l_j, \hat{L})$
7:          end for
8:          if $||l_i|| > 0$ then        ▷ $l_i$ is partly non occluded
9:              for $l_j \in L$ do
10:                 $\mathrm{TrimOcclusion}(x, l_j, l_j, \hat{L})$
11:             end for
12:             $L \leftarrow L \cup \{l_i\}$
13:         end if
14:     end for
15:     return $L$
16: end procedure
```

The analytic ray casting algorithm (Algorithm 2) proceeds as follows: A list $\hat{L}$ of all possible lines is made from the map $M$. Every line $l_i \in \hat{L}$ is first trimmed based on occlusions by lines in the existing scene list $L$ (lines 5-7). If at least part of $l_1$ is left non occluded, then the existing lines in $\hat{L}$ are trimmed based on occlusions by $l_i$ (lines 9-11) and $l_i$ is then added to the scene list $L$. The result is a list of non occluded, non-intersecting scene lines in $L$. Fig. 3 shows an example scene list on the real map.

Thus, given the robot pose, the set of line segments likely to be observed by the robot is computed. Based on this list of line segments, the points observed by the laser rangefinder and depth image sensors are related to the map using the projected point cloud model, which we introduce next.

Let the observed 2D points from the laser rangefinder sensor, along with their estimated normals, be represented by the lists $P_L$ and $R_L$ respectively. Since the map on which the robot is localizing is in 2D, we project the 3D filtered point cloud $P$ and the corresponding plane normals $R$ onto 2D to generate a 2D point cloud $P_D$ along with the corresponding normalized normals $R_D$. Points that correspond to ground plane detections are rejected at this step. Let the pose of the

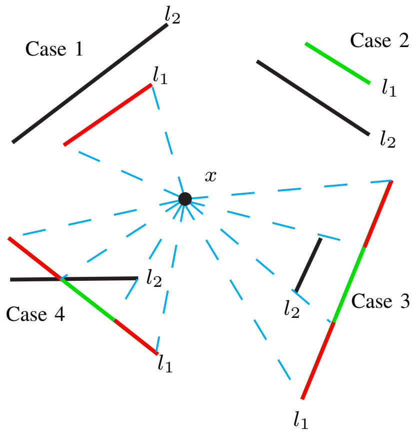


Fig. 2: Line occlusion cases. Line $l_1$ is being tested for occlusion by line $l_2$ from location $x$. The occluded parts of $l_1$ are shown in green, and the visible parts in red. The visible ranges are bounded by the angles demarcated by the blue dashed lines.

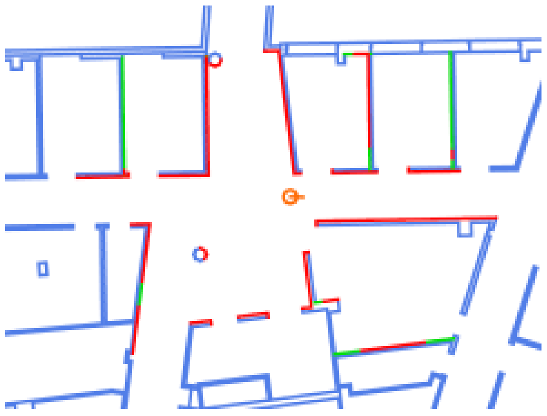


Fig. 3: An example result of analytic ray casting from the robot location marked in orange. The lines in the final scene list $L$ are shown in red, the original, un-trimmed corresponding lines in green, and all other lines on the map in blue.

robot $x$ be given by $x = \{\mathrm{x}, \mathrm{y}, \theta\}$, the location and orientation of the robot. The observable scene lines list $L$ is computed using an analytic ray cast. The observation likelihood $p(y|x)$, where the observation $y$ are point clouds observed by the laser rangefinder $P_L$ and the depth camera $P_D$ respectively, is computed as follows:

1) For every point $p_i^L$ in $P_L$, line $l_i$ ($l_i \in L$) is found such that the ray in the direction of $p_i^L - x_1$ and originating from $x_1$ intersects $l_i$.
2) Points for which no such line $l_i$ can be found are discarded.
3) Points $p_i^L$ for which the corresponding normal estimates $r_i^L$ (from $R_L$) differ from the normal to the line $l_i$ by a value greater than a threshold $\theta_{max}$ are discarded.
4) The perpendicular distance $d_i^L$ of $p_i^L$ from the (extended) line $l_i$ is computed.
5) The perpendicular distance $d_i^D$ of points $p_i^D \in P_D$ (observed by the depth camera) from corresponding line segments are similarly computed.

The total non-normalized observation likelihood $p(y|x)$ is then given by

$$p(y|x) = \left(\prod_{i=1}^{n_L} \exp\left[-\frac{(d_i^L)^2}{2 f_L \sigma_L^2}\right]\right)\left(\prod_{i=1}^{n_D} \exp\left[-\frac{(d_i^D)^2}{2 f_D \sigma_D^2}\right]\right). \quad (1)$$

Here, $\sigma_L$ and $\sigma_D$ are the standard deviations of distance measurements made by the laser rangefinder and the depth camera respectively, and $f_L : f_L > 1$ and $f_D : f_D > 1$ are discounting factors to discount for the correlation between observations. Although the observation likelihood function treats each observed point independently, in reality every observed point is correlated with its neighbors since they are most likely to be observations of the same object, and hence most likely to have similar values of $d_i$. Smaller values of $f_L$ and $f_D$ result in more abrupt variations of $p(y|x)$ as a function of location $x$, while larger values produce smoother variations of $p(y|x)$. In our implementation we set $f_L$ and $f_D$ to the number of points observed by the laser rangefinder and depth camera sensor, respectively.

In Eq. 1, the observation likelihood function penalizes the *perpendicular* error $d_i$ between each observed point and the lines on the map rather than the error along the ray of the observation. This is because when surfaces are observed at small angles of incidence, the error along the rays will be significantly large even for small translation errors in the pose estimate, while the small perpendicular errors in $d_i$ more accurately represent the small translation errors.

The observation likelihoods thus computed are used for localization using the Corrective Gradient Refinement (CGR) algorithm, which we describe next.

## 5. Corrective Gradient Refinement

The belief of the robot's location is represented as a set of $m$ weighted samples or particles, as in MCL [Dellaert et al., 1999]: $Bel(x_t) = \{x_t^i, w_t^i\}_{i=1:m}$. Recursive updates of the particle filter can be implemented via a number of methods, notably among which is the sampling/importance resampling method (SIR) [Gordon et al., 1993]

In the MCL-SIR algorithm, $m$ samples $x_{t-}^i$ are drawn with replacement from the prior belief $Bel(x_{t-1})$ proportional to their weights $w_{t-1}^i$. These samples $x_{t-}^i$ are used as priors to sample from the motion model of the robot $p(x_t|x_{t-1}, u_{t-1})$ to generate a new set of samples $x_t^i$ that approximates $p(x_t|x_{t-1}, u_{t-1})Bel(x_{t-1})$. Importance weights for each sample $x_t^i$ are then computed as

$$w_t^i = \frac{p(y_t|x_t^i)}{\sum_i p(y_t|x_t^i)}. \quad (2)$$

Although SIR is largely successful, several issues may still create problems:

1) Observations could be "highly peaked," *i.e.* $p(y_t|x_t^i)$ could have very large values for a very small subspace $\epsilon$ and be negligible everywhere else. In this case, the probability of samples $x^i$ overlapping with the subspace $\epsilon$ is diminishingly small.
2) If the posterior is no longer being approximated well by the samples $x^i$, recovery is only possible by chance, if the motion model happens to generate new samples that better overlap with the posterior.
3) The required number of samples necessary to overcome the aforementioned limitations scales exponentially in the dimension of the state space.

The Corrective Gradient Refinement (CGR) algorithm addresses the aforementioned problems in a computationally efficient manner. It refines samples locally to better cover neighboring regions with high observation likelihood, even if the prior samples had poor coverage of such regions. Since the refinement takes into account the gradient estimates of the observation model, it performs well even with highly peaked observations. This permits CGR to track the location of the robot with fewer particles than MCL-SIR.

CGR iteratively updates the past belief $Bel(x_{t-1})$ using observation $y_t$ and control input $u_{t-1}$ as follows:

1) Samples of the belief $Bel(x_{t-1})$ are propagated through the motion model, $p(x_t|x_{t-1}, u_{t-1})$ to generate a first stage proposal distribution $q^0$.
2) Samples of $q^0$ are refined in $r$ iterations which produce intermediate distributions $q^i, i \in [1, r-1]$ using the gradients $\frac{\delta}{\delta x} p(y_t|x)$ of the observation model $p(y_t|x)$.
3) Samples of the last proposal distribution $q^r$ and the first stage proposal distribution $q^0$ are sampled using an acceptance test to generate the final proposal distribution $q$.
4) Samples $x_t^i$ of the final proposal distribution $q$ are weighted by corresponding importance weights $w_t^i$, and resampled with replacement to generate $Bel(x_t)$.

We explain the four steps of the CGR algorithm in detail.

### 1. The Predict Step

Let the samples of the past belief, $Bel(x_{t-1})$ be given by $x_{t-1}^i$. These samples are then propagated using the motion model of the robot to generate a new set of samples $q^0 =$

$\left\{x^i_{q^0}\right\}_{i=1:m}$ as $x^i_{q^0} \sim p(x_t|x^i_{t-1}, u_{t-1})$. This sample set $q^0$ is called the first stage proposal distribution, and takes time complexity $\mathcal{O}(m)$ to compute.

### *2. The Refine Step*

The Refine step is central to the CGR algorithm. It corrects sample estimates that contradict the observations $y_t$, *e.g.*, when the sensor observations indicate that the robot to be in the center of the corridor, but the sample estimates are closer to the left wall. This results in the CGR algorithm sampling less along directions that have low uncertainty in the observation model while preserving samples along directions of high uncertainty.

For the CGR algorithm, estimates of the first order differentials (the gradients) of the observation model, $\frac{\hat{\delta}}{\delta x}p(y_t|x)$ must to be computable. Given these gradients, the Refine step performs gradient descent for $r$ iterations with a step size $\eta$, generating at iteration $i$ the $i$-th stage proposal distribution. Algorithm 3 outlines the Refine step.

**Algorithm 3** The Refine step of CGR

```
1: Let $q^0 = \left\{x^j_{q^0}\right\}_{j=1:m}$
2: for $i = 1$ to $r$ do
3:     $q^i \leftarrow \{\}$
4:     for $j = 1$ to $m$ do
5:         $x^j_{q^i} \leftarrow x^j_{q^{i-1}} + \eta \left[\frac{\hat{\delta}}{\delta x}p(y_t|x)\right]_{x=x^j_{q^{i-1}}}$
6:         $q^i \leftarrow q^i \cup x^j_{q^i}$
7:     end for
8: end for
```

Performing multiple iterations of gradient descent allows the estimates of the gradients of the observation model to be refined between iterations, which results in higher accuracy.

After the Refine step, samples from the $r$-th stage distribution are compared to the samples from the first stage proposal distribution by an acceptance test to generate the final proposal distribution $q$, as we now present.

### *3. The Acceptance Test Step*

To generate the final distribution $q$, Samples $x^i_{q^r}$ from the $r$-th stage distribution are probabilistically chosen over the corresponding samples $x^i_{q^0}$ of the first stage distribution proportional to the value of the acceptance ratio $r^i$, as

$$r^i = \min\left\{1, \frac{p(y_t|x^i_{q^r})}{p(y_t|x^i_{q^0})}\right\}. \tag{3}$$

This Acceptance Test allows the algorithm to probabilistically choose samples that better match the observation model $p(y_t|x)$. If the Refine step does not produce samples with higher observation likelihood than the samples in the first stage distribution, the final proposal distribution $q$ will have a mixture of samples from $q^0$ and $q^r$. Furthermore, if the Refine step results in most samples having higher observation likelihood than the samples in the first stage distribution, the final proposal distribution $q$ will consist almost entirely of samples from $q^r$. The acceptance test thus ensures that instantaneously high weights due to possibly incorrect observations do not overwhelm the Belief distribution in a single step.

Samples from the final proposal distribution $q$ thus generated are weighted by importance weights, and resampled to compute the latest belief $Bel(x_t)$ in the Update step.

### *4. The Update Step*

The importance weights for the CGR algorithm are different from those of the MCL-SIR algorithm, since in the CGR algorithm, the proposal distribution $q$ is not the same as the samples of the motion model. To derive the expression for the importance weights of the CGR algorithm, we first factor out the belief update as:

$$Bel(x_t) \propto p(y_t|x_t)p(x_t|u_{t-1}, Bel(x_{t-1})) \tag{4}$$

$$p(x_t|u_{t-1}, Bel(x_{t-1})) = \int p(x_t|x_{t-1}, u_{t-1})Bel(x_{t-1})dx_{t-1} \tag{5}$$

The proposal distribution from which the belief $Bel(x_t)$ is computed, is $q$. Hence, the non-normalized importance weights $w^i_t$ for samples $x^i_t$ in $q$ are given by:

$$w^i_t = \frac{p(y_t|x^i_t)p(x^i_t|u_{t-1}, Bel(x_{t-1}))}{q(x^i_t)} \tag{6}$$

In this expression, $p(y_t|x^i_t)$ is computed using the observation model (see Section 4) using the latest sensor observations and the pose estimates for the different particles, and the terms $p(x^i_t|u_{t-1}, Bel(x_{t-1}))$ and $q(x^i_t)$ are computed by kernel density estimates at the locations $x^i_t$ using the samples from $q^0$ and $q$ for support of the kernel density, respectively. The kernel density function should be wide enough such that areas with a high sample density do not bias the Belief update, but should be narrow enough to preserve samples from visually different locations. We use a Gaussian kernel with a $2\sigma$ value equal to the radius of the robot. Since the importance weight accounts for the motion model from the term $p(x^i_t|u_{t-1}, Bel(x_{t-1}))$ as well as the refined proposal distribution ($q(x^i_t)$), the CGR algorithm avoids "peak following" behavior that contradicts the motion model.

The samples in $q$ are resampled in proportion to their importance weights using low variance resampling [Thrun et al., 2005] to obtain the samples of the latest belief, $Bel(x_t)$. The update step has time complexity $\mathcal{O}(m^2)$. It is quadratic in the number of particles since it requires computation of kernel density functions.

Thus, given the motion model $p(x_t|x_{t-1}, u_{t-1})$, the observation model $p(y_t|x)$, the gradients of the observation model $\frac{\delta}{\delta x}p(y_t|x)$ and the past belief $Bel(x_{t-1})$, the CGR algorithm computes the latest belief $Bel(x_t)$.

### *5. Localization Accuracy*

To compare localization using CGR to MCL-SIR, we logged sensor data while driving the robot around the map, traversing a path 374m long. The true robot location was manually annotated by aligning the sensor scans to the map. This data was then processed offline with varying number of particles to compare success rates, accuracy and run times for CGR as well as MCL-SIR. At the beginning of every trial, the particles were randomly initialized with errors of up to ±4m and ±40°. The number of particles $m$ was varied from 2 to 202, with 80 trials each for CGR and MCL-SIR for each value of $m$. The value of $r$, the number of iterations in the Refine step was set to 3 for all trials.

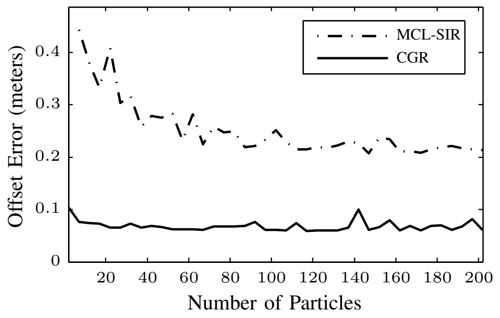


Fig. 4: The offset errors from the true robot locations for the MCL-SIR and CGR algorithms

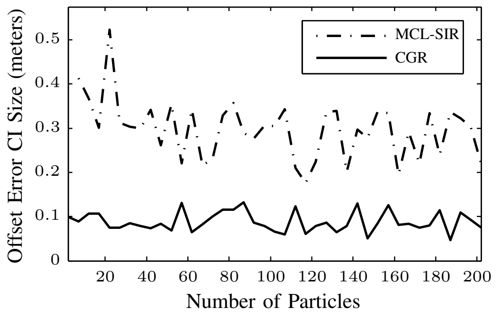


Fig. 5: The size of the 70% confidence intervals for the offset errors for the MCL-SIR and CGR algorithms

Fig. 4 shows the mean localization errors for the two algorithms for different numbers of particles. The graph shows that localization using CGR is consistently more accurate than when using MCL-SIR. In particular, localizing using CGR with 20 particles has smaller mean errors than localizing using MCL with even 200 particles.

Another advantage of the CGR algorithm over MCL-SIR is that the CGR algorithm has lower variance across trials. Fig. 5 shows the 70% confidence interval sizes for the two algorithms, for different numbers of particles. The variance across trials is consistently smaller for the CGR algorithm.

## 6. Navigation

CoBot is capable of navigating on and between multiple floors. To navigate between floors, CoBot pro-actively asks for human assistance [Rosenthal and Veloso, 2012] to press the elevator buttons. On a single floor, CoBot uses a hierarchical two-level navigation planner. At the high level a topological graph map is used to compute a policy, which is then converted to a sequence of actions based on the current location of the robot. At the low level, a local obstacle avoidance planner modifies these actions to side-step perceived obstacles.

### 1. Topological Graph Planner

CoBot uses a graph based navigation planner [Biswas and Veloso, 2010] to plan paths between locations on the same floor of the building. The navigation graph $G$ is denoted by $G = \langle V, E\rangle$, $V = \{v_i = (x_i, y_i, w_i)\}_{i=:n_V}$, $E = \{e_i = (v_{i1}, v_{i2}) : v_{i1}, v_{i2} \in V\}_{i=:n_E}$. The set of vertices $V$ consists of $n_V$ vertices $v_i = (x_i, y_i)$ that represent the location of the ends and intersections of hallways. The set of edges $E$ consists of $n_E$ edges $e_i = (v_{i1}, v_{i2})$ that represent navigable paths of width $w_i$ between vertices $v_{i1}$ and $v_{i2}$. Fig. 6 shows the navigation graph for a section of a floor in the building.

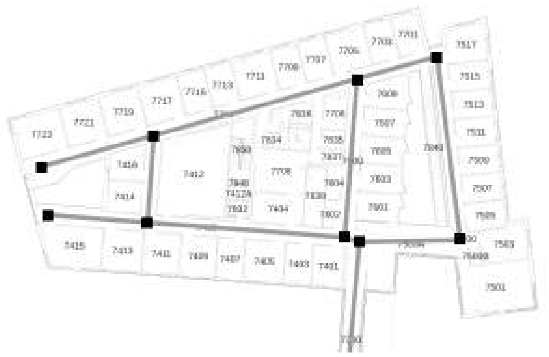
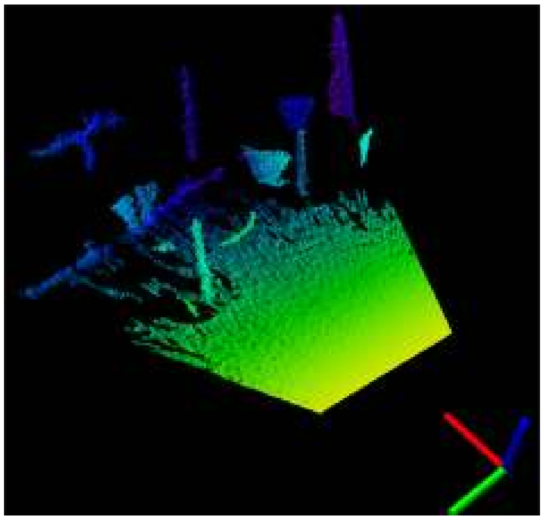


Fig. 6: Navigation graph of a part of a floor in the building. Vertices are shown in black squares, and connecting edges in grey lines.

Given a destination location $l_d = (\mathrm{x}_d, \mathrm{y}_d, \theta_d)$, the navigation planner first finds the projected destination location $l'_d = (\mathrm{x}'_d, \mathrm{y}'_d, \theta_d)$ that lies on one of the edges in the graph. This projected destination location is then used to compute a topological policy using Dijkstra's algorithm for the entire graph [Biswas and Veloso, 2010]. The navigation planner projects the current location $l = (x, y, \theta)$ onto the graph and then executes the topological policy until the robot reaches the edge on which $l'_d$ lies, and then drives straight to the location $l_d$. Thus, the navigation planner navigates between start and end rooms given by the task executor.

### 2. Obstacle Avoidance

While executing the navigation plan, CoBot performs obstacle avoidance based on the obstacles detected by the laser rangefinder and depth cameras. This is done by computing open path lengths available to the robot for different angular directions. Obstacle checks are performed using the 2D points detected by the laser rangefinder, and the down-projected points in the plane filtered and outlier point clouds generated by FSPF from the latest depth image.


(a)

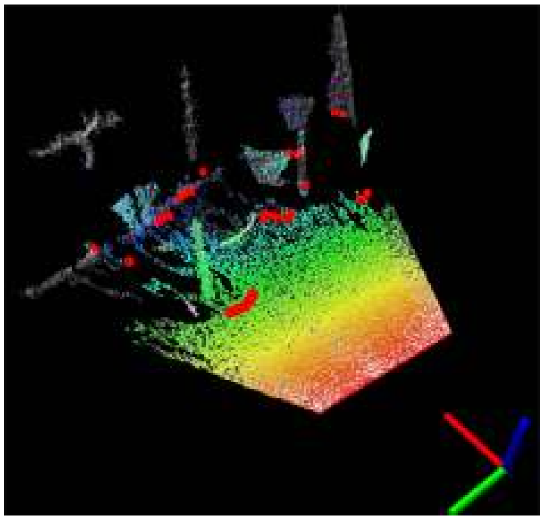

(b)

Fig. 7: Obstacle avoidance: The full 3D point cloud (a) and (b) the sampled points (shown in color), along with the open path limits (red boxes). The robot location is marked by the axes.

Let $P$ be the set of points observed by the laser rangefinder as well as sampled from the latest depth image by FSPF. Let $r$ be the robot radius, $\theta_d$ the desired direction of travel, and $\hat{\theta}$ a unit vector in the direction of $\theta$. The open path length $d(\theta)$ and the chosen obstacle avoidance direction $\theta^*$ are hence calculated as

$$P_\theta = \left\{p : p \in P \wedge ||p - p \cdot \hat{\theta}|| < r\right\}, \tag{7}$$

$$d(\theta) = \min_{p \in P_\theta} \left(\max(0, ||p \cdot \hat{\theta}|| - r)\right), \tag{8}$$

$$\theta^* = \arg\max_{\theta} \left(d(\theta)\cos(\theta - \theta_d)\right). \tag{9}$$

Fig. 7 shows an example scene with two tables and four chairs that are detected by the depth camera. Despite randomly sampling from the depth image, all the obstacles are correctly detected, including the table edges. The computed open path lengths from the robot location are shown by red boxes.

In addition to avoiding obstacles perceived by the laser rangefinder and depth cameras, CoBot restricts its location to virtual corridors of width $w_i$ for every edge $e_i$. These virtual corridors prevent the CoBots from being maliciously shepherded into unsafe areas while avoiding obstacles. Furthermore, our environment has numerous obstacles that are invisible to the robot (Section 7.2), and the CoBots successfully navigate in the presence of such invisible hazards by maintaining accurate location estimates, and by bounding their motion to lie within the virtual corridors.

## 7. Long-term Deployments of the CoBots

Since September 2011, two CoBots have been autonomously performing various tasks for users on multiple floors of our building, including escorting visitors, transporting objects, and engaging in semi-autonomous telepresence. We present here the results of the autonomous localization and navigation of the CoBots over their deployments.

### 1. *Hardware*

The CoBots are custom-built robots with four-wheel omni-directional drive bases, each equipped with a Hokuyo URG-04lx short-range laser rangefinder and a Microsoft Kinect depth camera sensor. The laser rangefinders have a maximum range of $4m$ and a viewing angle of $240°$, and are mounted at ankle height on the drive bases. The depth camera sensors are mounted at waist height, and are tilted downwards at an angle of approximately $20°$ so as to be able to detect any obstacles in the path of the robot. Both CoBots are equipped with off-the-shelf commodity laptops with dual-core Intel Core i5 processors, and the robot autonomy software stack (localization, navigation, obstacle avoidance, task planner and server interface) share the computational resources on the single laptop. CoBot2 has a tablet form-factor laptop, which facilitates touch-based user interaction. For purposes of telepresence, the CoBots have pan-tilt cameras mounted on top of the robots.

### 2. *Challenges*

Over the course of their regular deployments, the CoBots traverse the public areas in the building, encountering varied scenarios (Fig. 8), and yet accurately reach task locations (Fig. 9). Using localization alone, the robot is repeatably and accurately able to slow down before traversing bumps on the floor (Fig. 8c), stop at the right location outside offices (Fig. 9a-b) and the kitchen (Fig. 9c), as well as autonomously get on and off the elevators in the building (Fig. 9d). There are areas in the building where the only observable long-term features are beyond the range of the robot's sensors, requiring the robot to navigate these areas by dead reckoning and still update its location estimate correctly when features become visible later. For safe navigation, the CoBots have to detect tall chairs and tables undetectable to the laser rangefinder sensors but visible to the depth cameras. Despite these challenges, the CoBots remain autonomous in our environment, and rarely require human intervention.

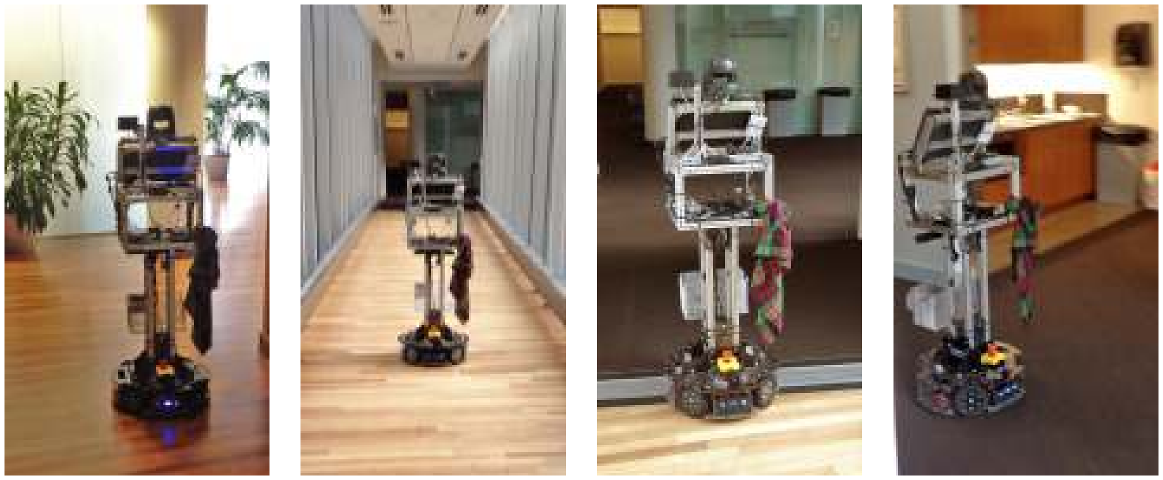

(a) (b) (c) (d)

Fig. 8: Localization and navigation in varied scenarios, including (a) in an open area with glass walls, (b) while crossing a bridge, (c) slowing down to cross a bump, and (d) negotiating a sharp turn.

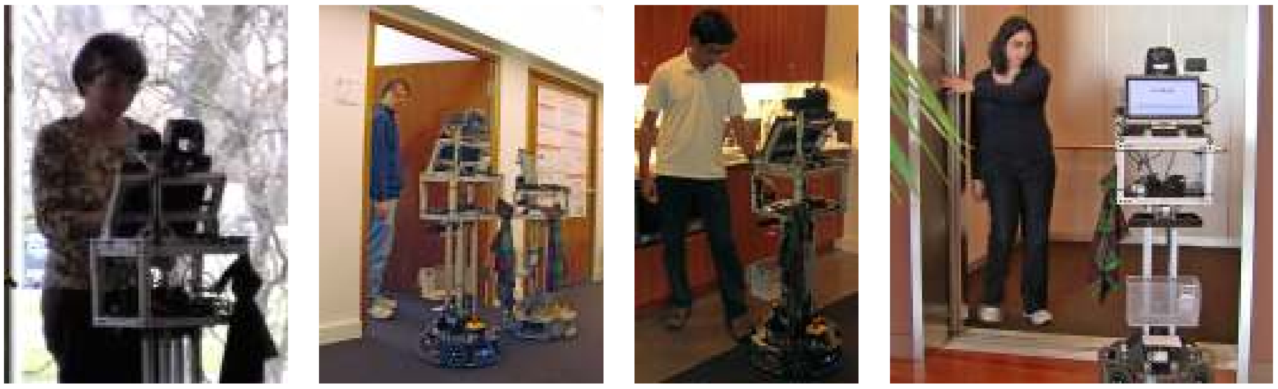

(a) (b) (c) (d)

Fig. 9: Reliable robot positioning at task locations, including (a) outside an office door, (b) at a transfer location, (c) at the kitchen, and (d) while taking the elevator.

### *3. Sensor Logs*

The CoBots (over the course of their regular deployments) have been logging sensor data, as well as the reported location of the CoBots (as computed by the on-board localization algorithms) and the locations where the localization was reset. The sensor data logged include the observed robot odometry and the laser rangefinder observations. The observed depth images were not recorded due to storage space limitations on the robot, but the CoBots relied on them both for localization and safe navigation. These logs show that the CoBots have traversed more than $131km$ autonomously over a total of 1082 deployments to date. The sensor logs and the maps of the building are available for download from our website http://www.cs.cmu.edu/~coral/cobot/data.html. Table II summarizes the contents of the sensor logs.

| Property | Value |
|---|---|
| Total duration | 260 hrs |
| Total distance traversed | 131 km |
| Total number of deployments | 1082 |
| Total number of laser rangefinder scans | 10,387,769 |

TABLE II: Data of sensor logs over CoBots' deployments from September 2011 to January 2013.

### *4. Results*

The CoBots have autonomously navigated on floors $3-9$ of the Gates Hillman Center (GHC) and floor 4 of Newell Simon Hall (NSH) at Carnegie Mellon University. Fig. 10 shows the combined traces of the paths traversed by the CoBots on the different floors. Different floors were visited by the CoBots with different frequencies, as listed in Table III.

Along with the raw sensor logs, the locations at which the localization estimates of the CoBots were reset, were also logged. Fig. 11shows the distribution of the locations where the location of the CoBots had to be reset on floor GHC7. These locations include the startup locations where the CoBots were initialized, locations outside the elevators where the map for localization was switched when the robot had determined that it had successfully entered a different floor, and locations where the localization estimates had to be manually reset due to localization errors. By excluding the startup locations and

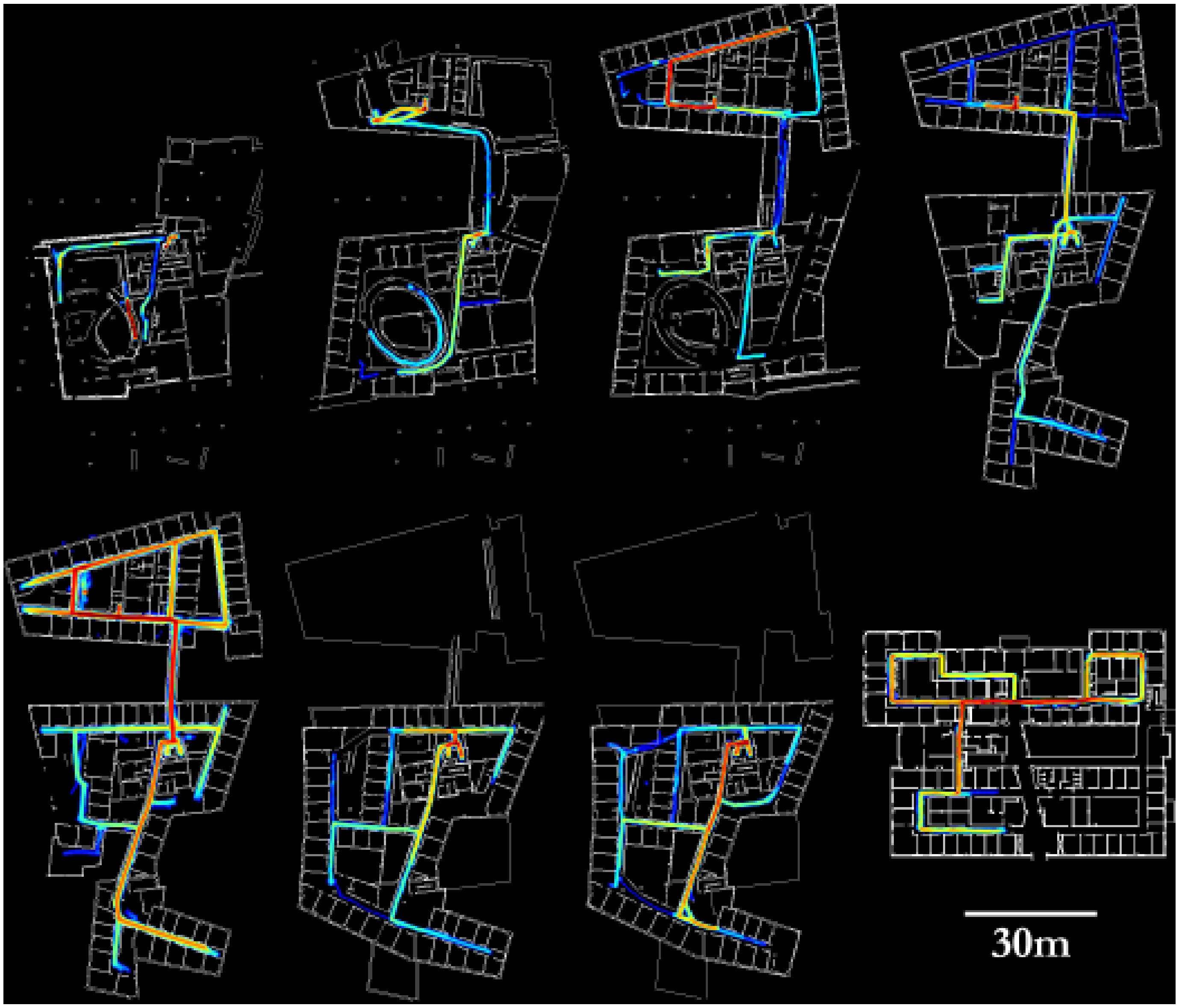


Fig. 10: Combined traces of the paths traversed by the CoBots on floors GHC3 – 6 (top row) and GHC7 – 9, NSH4 (bottom row). Locations on the map are color-coded by the frequency of visits, varying from dark blue (least frequently visited) to red (most frequently visited). Since the CoBots visited each floor a different number of times, the color scales are different for each floor.

| Floor | Times visited |
|---|---|
| GHC3 | 21 |
| GHC4 | 50 |
| GHC5 | 13 |
| GHC6 | 95 |
| GHC7 | 751 |
| GHC8 | 52 |
| GHC9 | 58 |
| NSH4 | 42 |

TABLE III: Breakup of the times each floor was visited by the CoBots.

map switch locations, and normalizing the reset counts at each location by the number of times that the CoBots traversed them, the error rate of the CoBots is computed, as a function of location on the map. As Fig. 12 shows, the locations where the CoBots occasionally require manual intervention are few in number and clustered in specific areas. The region with the highest error rate on floor GHC7 had 10 errors out of the 263 times the CoBots visited that region, for an error rate of 0.038. Although there are many areas in the building with significant differences between expected observations based on the map and the actual observations, and also many areas with constant changes due to changing locations of movable objects, only a small number of them actually present difficulties for the autonomy of the CoBots. We discuss the causes for these infrequent failures in Section 8.2. We have not observed any correlation between the lengths of paths traversed by the robot, but as shown in Fig. 12, there is a correlation between certain specific locations and localization errors.

### 5. *Failures*

Over the course of deployments there have been a few unexpected failures due to unforeseen circumstances. These were caused by hardware failures, sensing failures and algorithmic limitations. It is important to note that in all of these cases except one, the failure mode of the robot was to stop safely and send an email to the research group mentioning that the robot had been unexpectedly stopped, and the estimated location where this event occurred.

*1) Hardware Failures:* On CoBot1, we had one instance before we started using the depth camera sensor, where the laser rangefinder stopped functioning and returned maximum ranges for all the angles. This led the robot to hit a wall corner before a bystander pressed the emergency stop button on the robot. Since this event, we have been investigating automated approaches to detect un-modeled and unexpected errors by monitoring the safe operational state space of the robot using redundant sensors [Mendoza et al., 2012].

Our building has a number of connecting bridges with expansion joints at the interfaces of these bridges (Fig. 8c). When the CoBots run over the expansion joints at their normal speed of $0.75m/s$, occasionally there is a momentary break of electrical contact to the USB ports on the computer, which causes the Linux kernel to de-list the USB devices. The CoBot bases are designed to halt if no new command is received within $50ms$, so when the USB ports get de-listed on the computer, the robot comes to a halt. We circumvented this problem by enforcing a maximum robot speed of $0.3m/s$ for the navigation edges that straddle such bumps. Even though the lengths of these edges with speed limits are just $0.5m$ long, due to the accuracy of the localization, the CoBots always correctly slow down before encountering the bumps and we have not since encountered any problems crossing the bumps.

*2) Sensing Errors:* The laser rangefinders and depth cameras on the CoBots are rated only for indoor use, and are not rated for operation in direct sunlight. However, in some parts of the buildings, at some specific seasons and times of the day, the CoBots do encounter direct sunlight. The direct sunlight saturates the sensor elements, causing them to return invalid range readings. Our obstacle avoidance algorithm treats invalid range readings from the laser rangefinder, and large patches of invalid depth from the depth cameras as being equivalent to minimum range. We chose to err on the side of caution in this case, and when the CoBots detect invalid range or depth readings, they come to a stop and wait. If the sensors do not recover from saturation within 5 minutes, the CoBots email the research group with the last location estimate, stating that they have been stopped for more than 5 minutes without progress towards their tasks.

*3) Algorithmic Limitations:* The long-term deployments of the CoBots have been invaluable in testing the algorithms on the robot in a wide number of scenarios. We have encountered a few cases that are currently not addressed adequately by our algorithm, and which motivate our ongoing research.

In one scenario, during an open house, CoBot2 was crossing a bridge with glass walls and heat radiators near the floor. The glass walls are invisible to both the laser rangefinder as well as the depth camera sensors. The radiators, being poorly reflective, are invisible to the depth camera sensors but visible to the laser rangefinder for small angles of incidence. Given the partial laser rangefinder observations, CGR was accurately able to disambiguate the location of the robot perpendicular to the glass walls, but when it neared the end of the bridge, it was uncertain of the distance it had traversed along the length of the bridge due to wheel slip. It observed a large group of visitors at the end of the bridge, and the group as a collective was mis-labelled as a vertical plane by FSPF. Since the particle cloud overlapped with a location at the end of the bridge that genuinely had a wall, the location estimates incorrectly converged to that location. As a result, the robot expected a corridor opening earlier, and came to a stop when the obstacle avoidance detected that there was an obstacle (the wall) in front of it.

This failure raises an important question: how should a robot reason about *what* it is observing, and what objects are there in the environment? Currently most localization, mapping and SLAM algorithms are agnostic to *what* they are observing, but not all observable objects in the environment provide useful information for localization. Therefore, a robot should not add observations of moving and movable objects, like humans, bags and carts to its localization map, and if it observes such objects, it should not try to match them to the map. In the long run, a richer model of the world would clearly benefit robot localization and mapping. In the short term, we resort to measures like stricter outlier checks. A stricter plane fit check for FSPF results in fewer false positives, but also causes false negatives of planar surfaces with poorly reflective surfaces.

## 8. Lessons Learnt

The continued deployments of the CoBots have provided us with valuable insights into the strengths and limitations of our algorithms. We list the lessons learnt under the sub-categories of Sensors, Localization, Navigation and System Integration.

### 1. *Sensors*

The CoBots are equipped with a laser rangefinder and a depth camera sensor each, which have very different characteristics.

The laser rangefinder provides range readings in millimeters with an accuracy of about $1-5$ cm depending on the reflecting surface properties, and a wide field of view of $240°$. As a relatively inexpensive laser rangefinder, its maximum range is limited to $4m$. The wide field of view allows the robot to more effectively avoid obstacles, and the accurate readings enable accurate localization. However, it is difficult to distinguish between objects (*e.g.*, a pillar *vs.* a person) with the laser rangefinder alone, which leads to occasional false data associations for localization. Furthermore, since the laser rangefinder is fixed at a particular height, it is unable to detect a number of obstacles like tall chairs and tables.

The depth camera sensor provides depth values with an accuracy that degrades with distance (about 1cm at a depth of 0.75m, degrading to about 4cm at a depth of 5m). The field of view of the sensor is constrained to about $55°$

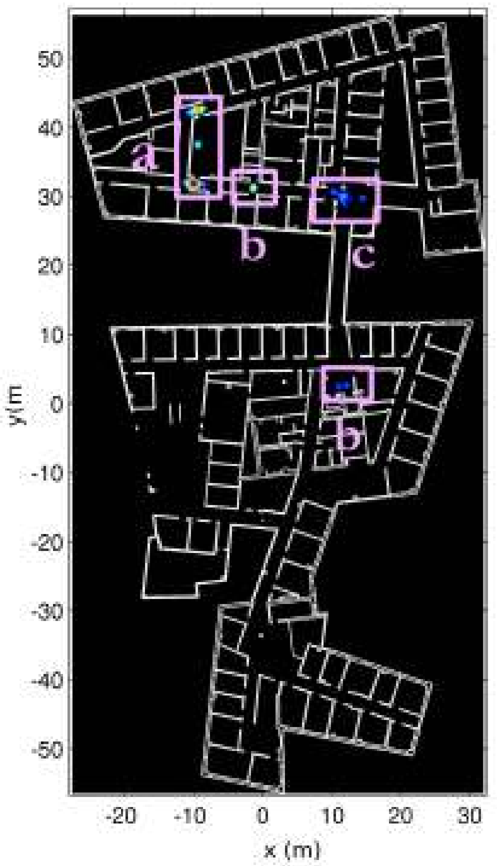


Fig. 11: Locations on floor GHC7 where the localization estimates of the CoBots were reset over all the deployments. These include a) the startup locations of the CoBots, b) locations where the CoBots switched maps after taking the elevator, and c) locations with localization errors.

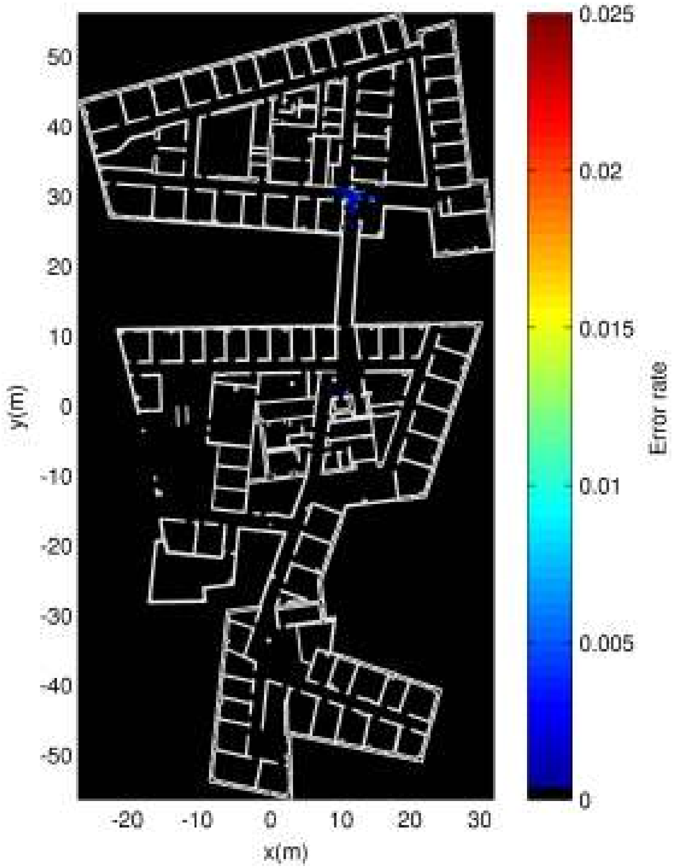


Fig. 12: Error rates of localization for the CoBots on floor GHC7. Locations with no errors (including locations not visited) are shown in black, and the error rates are color-coded from blue(lower error rates) to red(higher error rates).

horizontally and $40°$ vertically. Due to its relatively narrow field of view, the depth camera sensor observes fewer features for localization compared to the laser rangefinder, resulting in higher uncertainty of localization. However, by performing plane filtering on the depth images (Section 3), map features like walls can easily be distinguished from moving and movable objects like people and chairs, thus dramatically reducing the number of false data associations for localization [Biswas and Veloso, 2013]. The depth images also allow the robot to effectively avoid hitting obstacles, like chairs and tables with thin legs that are missed by the laser rangefinder.

Neither the laser rangefinder, nor the depth camera sensors on the CoBots, are rated for use in direct sunlight. As a result, occasionally, when the CoBots encounter direct bright sunlight, they detect false obstacles and the obstacle avoidance algorithm brings the robots to a stop.

## 2. *Localization*

The sensors used for localization on the CoBots have limited sensing range, and cannot observe the entire length of the hallway that the CoBot is in, for most of the time. Therefore, it is up to the localization algorithms to reason about the uncertainty parallel to the direction of the hallway. This is in stark contrast to a scenario where a robot with a long-range sensor (like the SICK LMS-200 laser rangefinder, with a maximum range of 80m) is able to observe the entire length of every hallway (the longest hallway in the GHC building is about 50m in length), and hence is able to accurately compute its location with a single reading. The decision to use inexpensive short-range sensors is partly motivated by cost, since we wish to eventually deploy several CoBots, and also because we wish to explore whether it is possible to have our algorithms be robust to sensor limitations. Despite the limitation of the sensor range, the CoBots repeatedly stop at exactly the right locations in front of office doors, and always follow the same path down hallways (when there are no obstacles). In fact, in a number of hallways, the repeated traversal of the CoBots along the exact same paths has worn down tracks in the carpets.

The CGR algorithm (Section 5) is largely to credit for the repeatable accuracy of localization. When traversing down hallways, CGR correctly distributes particles along the direction of the hallway (as governed by the motion model), and correctly limits the spread of the particles perpendicular to the hallway (due to the observations of the walls). This effectively reduces the number of dimensions of uncertainty of localization from three (position perpendicular to the hallway, position parallel to the hallway, and orientation) to two (position parallel to the hallway and the orientation), thus requiring much fewer particles than would have been required for the same degree of accuracy by MCL-SIR. CGR also allows quicker recovery from uncertain localization when new features become visible. For example, when travelling through an area with few observable features, the particles will have a larger spread, but when the robot nears a corridor intersection, CGR is able to effectively sample those locations that match the observations to the map, thus quickly converging to the true location.

There are occasional errors in localization, and most of these errors are attributed to erroneous data association of the observations, as discussed earlier (Section 7.5.3). Open areas, where most of the observations made by the robots consist of movable objects, remain challenging for localization. In such areas (like cafes, atria, and common study areas in our buildings), even if the map were updated by the robot to reflect the latest locations of the movable objects (*e.g.*, chairs, tables, bins), the next day the map would again be invalid once the locations of the objects changed.

### *3. Navigation*

In our experiences with extended deployment of the CoBots, we have come to realize that a conservative approach to navigation is more reliable in the long term as compared to a more unconstrained and potentially hazardous approach. In particular, the obstacle avoidance algorithm (Section 6) uses a local greedy planner, which assumes that paths on the navigation graph will always be navigable, and can only be blocked by humans. As a result, the planner will not consider an alternative route if a corridor has a lot of human traffic, but will stop before the humans and ask to be excused. Furthermore, due to the virtual corridors, the robot will not seek to side-step obstacles indefinitely. While this might result in longer stopped times in the presence of significant human traffic, it also ensures that the robot does not run into invisible obstacles (Section 7.2) in the pursuit of open paths. Although there exist many hallways with glass walls, the robot has never come close to hitting them, thanks to its reliable localization and virtual corridors.

One drawback of the constrained navigation is that if the localization estimates are off by more than half the width of a corridor intersection (which is extremely rare, but has occurred), the obstacle avoidance algorithm will prevent the robot from continuing, perceiving the corner of the walls at the intersection as an obstacle in its path. In such circumstances, issuing a remote command to the robot (via its telepresence interface) to turn and move to the side is sufficient for the robot to recover its localization (by observing the true locations of the walls), and hence the navigation as well.

### *4. System Integration*

As an ongoing long-term project, the CoBots require significant automation in order to ensure continued reliable operation. During deployments, the CoBots are accessible remotely via a telepresence interface that allow members of the research group to examine the state of the robot from the lowest (sensor) levels to the highest (task execution) levels. When a CoBot is blocked for task execution due lack of human responses to interaction, it automatically sends and email to the research group mentioning its latest location estimate, task status, and reason for being blocked.

The sensor feeds of the robot (except the depth camera images, since they are too voluminous) are logged, along with the estimated state of the robot, task list, and any human interactions. During robot startup, several health and monitoring checks are performed automatically on the robots, including:

1) Auto-detecting the serial and USB ports being used by all the devices, including the sensors and motor controllers,
2) Checking that all the sensor nodes are publishing at the expected rates,
3) Checking the extrinsic calibration of the sensors by performing consistency checks and detection of the ground plane,
4) Checking that the central task management server is accessible and is responding to requests for task updates, and
5) Verifying that the robot battery level is within safe limits.

There are several nightly scripts that execute on the robots as well as the central server, including:

1) Compressing and transferring the deployment logs of the day from each robot to the central server,
2) Running a code update on all the robots to pull the latest version of the code from the central repository and recompiling the code on the robot,
3) Processing the deployment logs on the server to generate synopses of the locations visited, distance traversed, and errors encountered (if any) during the day by all the robots, and
4) Emailing the synopses of the day's deployments to the developers.

We are currently at the point where the physical intervention required to manually unplug the charger from the robot is the bottleneck in deploying the CoBots. Therefore we are exploring designs for an automatic charging dock that will be robust to small positioning errors of the robot, durable in order to withstand thousands of cycles per year, and yet be capable of transferring sufficient power to charge the robot base as well as the laptop at the same time.

## 9. Conclusion

In this article, we have presented the localization and navigation algorithms that enable the CoBots to reliably and autonomously perform tasks on multiple floors of our buildings. The raw sensor observations made by the CoBots during the long-term autonomous deployments have been logged, and these logs of the CoBots demonstrate the robustness of the localization and navigation algorithms over extensive autonomous deployments. Despite the presence of dynamic obstacles and changes to the environment, the CoBots demonstrate resilience to them, save some infrequent errors. These errors are confined to a few areas, and we will be exploring strategies for autonomously recovering from such instances in the future.

## 10. Acknowledgements

We would like to thank Brian Coltin and Stephanie Rosenthal for their concurrent work on CoBot, in particular the task scheduler and human interaction behaviors of CoBot, which have been vital to the successful deployment of the CoBots.

## References

[Bailey and Durrant-Whyte, 2006] Bailey, T. and Durrant-Whyte, H. (2006). Simultaneous localization and mapping (SLAM): Part II. Robotics & Automation Magazine, IEEE *13*, 108–117.

[Biber and Duckett, 2005] Biber, P. and Duckett, T. (2005). Dynamic maps for long-term operation of mobile service robots. In Proc. of Robotics: Science and Systems (RSS) pp. 17–24,.

[Biber and Duckett, 2009] Biber, P. and Duckett, T. (2009). Experimental analysis of sample-based maps for long-term SLAM. The International Journal of Robotics Research *28*, 20–33.

[Biswas et al., 2011] Biswas, J., Coltin, B. and Veloso, M. (2011). Corrective gradient refinement for mobile robot localization. In Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on pp. 73–78, IEEE.

[Biswas and Veloso, 2010] Biswas, J. and Veloso, M. (2010). Wifi localization and navigation for autonomous indoor mobile robots. In Robotics and Automation (ICRA), 2010 IEEE International Conference on pp. 4379–4384, IEEE.

[Biswas and Veloso, 2012] Biswas, J. and Veloso, M. (2012). Depth camera based indoor mobile robot localization and navigation. In Robotics and Automation (ICRA), 2012 IEEE International Conference on pp. 1697–1702, IEEE.

[Biswas and Veloso, 2013] Biswas, J. and Veloso, M. (2013). Multi-Sensor Mobile Robot Localization For Diverse Environments. In RoboCup 2013: Robot Soccer World Cup XVII. Springer.

[Bruce et al., 2007] Bruce, J., Zickler, S., Licitra, M. and Veloso, M. (2007). Cmdragons 2007 team description. In Proceedings of the 11th International RoboCup Symposium, Atlanta, USA.

[Buhmann et al., 1995] Buhmann, J., Burgard, W., Cremers, A., Fox, D., Hofmann, T., Schneider, F., Strikos, J. and Thrun, S. (1995). The mobile robot Rhino. AI Magazine *16*, 31.

[Churchill and Newman, 2012] Churchill, W. and Newman, P. (2012). Practice makes perfect? managing and leveraging visual experiences for lifelong navigation. In Robotics and Automation (ICRA), 2012 IEEE International Conference on pp. 4525–4532, IEEE.

[Dayoub et al., 2011] Dayoub, F., Cielniak, G. and Duckett, T. (2011). Long-term experiments with an adaptive spherical view representation for navigation in changing environments. Robotics and Autonomous Systems *59*, 285–295.

[Dellaert et al., 1999] Dellaert, F., Fox, D., Burgard, W. and Thrun, S. (1999). Monte carlo localization for mobile robots. In Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on vol. 2, pp. 1322–1328, IEEE.

[Doucet et al., 2000] Doucet, A., De Freitas, N., Murphy, K. and Russell, S. (2000). Rao-Blackwellised particle filtering for dynamic Bayesian networks. In Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence pp. 176–183, Morgan Kaufmann Publishers Inc.

[Durrant-Whyte and Bailey, 2006] Durrant-Whyte, H. and Bailey, T. (2006). Simultaneous localization and mapping: part I. Robotics & Automation Magazine, IEEE *13*, 99–110.

[Elfes, 1989] Elfes, A. (1989). Using occupancy grids for mobile robot perception and navigation. Computer *22*, 46–57.

[Fox, 2001] Fox, D. (2001). KLD-sampling: Adaptive particle filters and mobile robot localization. Advances in Neural Information Processing Systems (NIPS) .

[Gordon et al., 1993] Gordon, N. J., Salmond, D. J. and Smith, A. F. (1993). Novel approach to nonlinear/non-Gaussian Bayesian state estimation. In IEE Proceedings F (Radar and Signal Processing) pp. 107–113, IET.

[Grisetti et al., 2007] Grisetti, G., Stachniss, C. and Burgard, W. (2007). Improved techniques for grid mapping with rao-blackwellized particle filters. Robotics, IEEE Transactions on *23*, 34–46.

[Jetto et al., 1999] Jetto, L., Longhi, S. and Venturini, G. (1999). Development and experimental validation of an adaptive extended Kalman filter for the localization of mobile robots. Robotics and Automation, IEEE Transactions on *15*, 219–229.

[Kalman, 1960] Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. Transactions of the ASME–Journal of Basic Engineering *82*, 35–45.

[Koenig and Simmons, 1998] Koenig, S. and Simmons, R. (1998). Xavier: A Robot Navigation Architecture Based on Partially Observable Markov Decision Process Models. In Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems, (D. Kortenkamp, R. B. and Murphy, R., eds), pp. 91 – 122. MIT Press.

[Lefebvre et al., 2004] Lefebvre, T., Bruyninckx, H. and De Schutter, J. (2004). Kalman filters for non-linear systems: a comparison of performance. International journal of Control *77*, 639–653.

[Lenser and Veloso, 2000] Lenser, S. and Veloso, M. (2000). Sensor resetting localization for poorly modelled mobile robots. In Int. Conf. on Robotics and Automation IEEE.

[Leonard and Durrant-Whyte, 1991] Leonard, J. and Durrant-Whyte, H. (1991). Mobile robot localization by tracking geometric beacons. Robotics and Automation, IEEE Transactions on *7*, 376–382.

[Mendoza et al., 2012] Mendoza, J. P., Veloso, M. and Simmons, R. (2012). Motion Interference Detection in Mobile Robots. In Proceedings of IROS'12, the IEEE/RSJ International Conference on Intelligent Robots and Systems, Vilamoura, Portugal.

[Nilsson, 1984] Nilsson, N. (1984). Shakey the robot. Technical report DTIC Document.

[Nourbakhsh et al., 2003] Nourbakhsh, I., Kunz, C. and Willeke, T. (2003). The mobot museum robot installations: A five year experiment. In Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on vol. 4, pp. 3636–3641, IEEE.

[Oyama et al., 2009] Oyama, A., Konolige, K., Cousins, S., Chitta, S., Conley, K. and Bradski, G. (2009). Come on in, our community is wide open for Robotics research! In The 27th Annual conference of the Robotics Society of Japan vol. 9, p. 2009,.

[Rosenthal et al., 2010] Rosenthal, S., Biswas, J. and Veloso, M. (2010). An effective personal mobile robot agent through symbiotic human-robot interaction. In Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1 pp. 915–922, International Foundation for Autonomous Agents and Multiagent Systems.

[Rosenthal and Veloso, 2012] Rosenthal, S. and Veloso, M. (2012). Mobile Robot Planning to Seek Help with Spatially-Situated Tasks. In Proceedings of the Twenty-Sixth Conference on Artificial Intelligence (AAAI-12), Toronto, Canada.

[Roumeliotis and Bekey, 2000] Roumeliotis, S. and Bekey, G. (2000). Segments: A layered, dual-kalman filter algorithm for indoor feature extraction. In Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on vol. 1, pp. 454–461, IEEE.

[Saarinen et al., 2012] Saarinen, J., Andreasson, H. and Lilienthal, A. J. (2012). Independent Markov chain occupancy grid maps for representation of dynamic environment. In Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on pp. 3489 –3495,.

[Samadi et al., 2012] Samadi, M., Kollar, T. and Veloso, M. (2012). Using the Web to Interactively Learn to Find Objects. In Proceedings of the Twenty-Sixth Conference on Artificial Intelligence (AAAI-12), Toronto, Canada.

[Thrun et al., 1999] Thrun, S., Bennewitz, M., Burgard, W., Cremers, A., Dellaert, F., Fox, D., Hahnel, D., Rosenberg, C., Roy, N., Schulte, J. et al. (1999). MINERVA: A second-generation museum tour-guide robot. In Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on vol. 3, IEEE.

[Thrun et al., 2005] Thrun, S., Burgard, W. and Fox, D. (2005). Probabilistic robotics.

[Thrun and Montemerlo, 2006] Thrun, S. and Montemerlo, M. (2006). The graph SLAM algorithm with applications to large-scale mapping of urban structures. The International Journal of Robotics Research *25*, 403–429.

[Thrun et al., 2002] Thrun, S. et al. (2002). Robotic mapping: A survey. Exploring artificial intelligence in the new millennium , 1–35.

[Walcott-Bryant et al., 2012] Walcott-Bryant, A., Kaess, M., Johannsson, H. and Leonard, J. J. (2012). Dynamic pose graph SLAM: Long-term mapping in low dynamic environments. In Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on pp. 1871 –1878,.

[Zhang and Ghosh, 2000] Zhang, L. and Ghosh, B. (2000). Line segment based map building and localization using 2D laser rangefinder. In IEEE Int. Conf. on Robotics and Automation.