A Mobile Robot for Autonomous Scene Capture and Rendering

Article

CITATION
1

READS
18

3 authors, including:

Rui Wang
Linyi University
214 PUBLICATIONS **4,750** CITATIONS

SEE PROFILE

Roderic A. Grupen
University of Massachusetts Amherst
221 PUBLICATIONS **3,102** CITATIONS

SEE PROFILE

A Mobile Robot for Autonomous Scene Capture and Rendering

Blake Foster, Rui Wang, and Roderic Grupen
University of Massachusetts Amherst

Abstract— **In recent years, projects such as Google Street View have contributed to widespread metrology of open human environments. Despite the ubiquity of such projects, image-capturing remains a labor-intensive process. In this work, we apply a new image-based heuristic to capture indoor scenes with an autonomous mobile robot. We task the robot to take a set of photographs that is sufficient to build a relatively complete 3D model of its environment. Our key insight is that the completeness of the the map, at any given viewpoint, is closely correlated with the quality of a synthesized image. Our heuristic exploits this relationship to choose a strategic set of new locations for the robot to visit. Because our heuristic is based on a realtime rendering algorithm, it is easily accelerated on the GPU, which allows us to evaluate a large set of candidate locations quickly. Our algorithm starts with an initial set of photos obtained through random exploration, which may not show a large part of the scene. We then compute a sparse 3D point cloud, and randomly pick a number of candidate viewpoints in the region bounded by the 3D points. Finally, we select the best viewpoints according to our metric, and send the robot to those locations to gather more photos. The process can be repeated, until no unexplored regions remain. When the exploration is finished, we construct a complete 3D model of the scene, which we use to create a virtual tour.**

I. INTRODUCTION

In the past decade, projects such as Google Street View have contributed to widespread metrology of open human environments. Such projects motivate a number of challenges, including adaptive view selection and smooth view interpolation. The human brain is remarkably adept at the former. Given even a small set of images of a new environment, a human can easily identify additional viewpoints that would reveal hidden areas. In this work, we attempt to give that same ability to a simple mobile robot, built from a Lego Mindstorms kit and a point-and-shoot camera. We task the robot to capture a set of photographs that is sufficient to build a relatively complete 3D model of an indoor scene. Our key insight is that the completeness of the map, at any given viewpoint, is closely correlated with the quality of a synthesized image. This relationship elicits an efficient heuristic for choosing new areas to explore.

Initially, our robot has no knowledge of its surroundings. We begin by building a rough map via random exploration. In this stage, the robot moves semi-randomly, capturing photos at intervals. We then use bundle adjustment [25] to build an initial map, consisting of a sparse set of 3D points and the location of each photo.

We use the initial map to guide a second, adaptive exploration step. In this stage, the robot attempts to capture photos from previously unvisited or undersampled areas. To this end, we select a set of candidate locations, and evaluate each location with a heuristic that estimates the quality of a synthesized image. Our heuristic is based on a real time rendering algorithm, which enables acceleration on a modern GPU. The computational cost of evaluating each candidate viewpoint is equivalent to rendering one low-resolution image. On a modern GPU, our implementation can can perform up to 155 evaluations per second.

After evaluating each candidate viewpoint, we select the best locations for further exploration. The robot then navigates to its goals, tracking SIFT [19] features to ensure that it stays on course. This provides a new set of photos, which fill in gaps in the robot's map. If necessary, the process can be repeated, until the robot's map is complete. When the exploration is finished, we construct a complete 3D model of the scene, which we use to create a virtual tour.

II. RELATED WORK

SLAM. Se et al. [23] use SIFT features as landmarks in near realtime SLAM. Their system runs on a mobile robot with a trinocular vision system. Successfully matching a feature in all three cameras provides a 3D location relative to the robot. These 3D landmarks are tracked across multiple frames. In [24], they extend their approach to the *kidnapped robot problem*, in which a robot must determine its location without any initial clues beyond its current sensor data.

Davison et al. [6] use a single camera to track features in a vision-based realtime SLAM system. Their system obtains 3D locations for features by matching to previous frames.

Sabe et al. [22] use stereo vision to guide a humanoid robot as it autonomously explores its environment. Their focus is mainly on obstacle avoidance. Their system detects the ground plane, and assumes that regions classified as ground may be safely walked on. The robot maintains an egocentric navigation grid, in which each cell contains the probability that an obstacle is present. We use a similar system for our robot.

Gutmann et al. [26] use a similar approach to path-planning. Their navigation grid keeps track of the estimated floor height. They use a cost function based on both the floor height and the probability of finding an obstacle to plan their robot's path.

Eade and Drummond [9] use a graph-based approach to monocular SLAM. Their system avoids the computational cost of global bundle adjustment by partitioning the landmarks into clusters. They first optimizing the landmark positions in each cluster, and then run a global optimization on the cluster locations. In [10], they extend their approach to allow for multiple connected components in the graph.

This allows the robot to recover when it becomes lost. In cases where the robot recognizes a previously-seen location, their system allows for loop-closing.

Unlike previous SLAM research, our goal is to build a dense reconstruction of the scene, consisting of millions of points. We therefore depart from the usual goal of realtime mapping as the robot explores, and opt for offline bundle adjustment. Our main contribution is a heuristic defined by the quality of a synthesized image, which we use to meet our goal of complete scene reconstruction.

Structure From Motion. Advances in multi-view stereopsis make image-based reconstruction a viable alternative to manual modeling or laser scanning. Yang et al. [28] present a realtime stereo vision system that estimates the geometry of the scene in front of an array of stationary Webcams. Their system uses the GPU to choose a depth at each pixel from a fixed set of possible depth values. They get good results for scenes that contain mostly fronto-parallel planar surfaces, but their system fails in scenes that contain complex geometry at many depth levels or repetitive patterns. Gallup et al. [13] use multiple plane orientations to improve the performance of this approach on sloped surfaces. Merrell et al. [20] introduced a realtime system for merging rough depth maps obtained via plane-sweeping. Given sufficiently many views captured in video, their system provides excellent results.

Pan et. al [21] present an interactive vision-based modeling system. They require the user to rotate an object in front of a Webcam, and use features tracked across multiple frames to build a 3D model. Their system works well for small objects with simple geometry, but is not applicable to larger scenes captured with moving cameras.

Bundler [25] can efficiently compute robust geometry from images taken with many different cameras under a variety of lighting conditions. Goesele et al. use Bundler in conjunction with a 3D photo-browsing system, but they do not attempt to solve the problem of view-interpolation. In [14], they obtain dense reconstructions from Bundler's output. We use Bundler to compute the camera locations and sparse 3D points that make up our robot's initial map.

Patch-based multiview stereo (PMVS) [12] computes a dense point cloud from a set of photos taken by calibrated cameras. In some scenes, the point cloud is dense enough to be used directly by a simple point-based rendering system. In cases where the point cloud is too sparse, they use Poisson surface reconstruction [17] to construct a mesh. We use their open source PMVS software to build our final dense reconstruction, after calibrating the cameras with Bundler.

Image-Based Rendering. While a wide variety of image-based rendering systems have been proposed, all existing systems require either detailed geometry or many photographs. In either case, the data-gathering phase requires specialized equipment or extensive human labor.

Light-field rendering [18] and the lumigraph [15] both attempt to capture all the light within a scene. Both approaches achieve photorealistic rendering in real time, but require huge numbers of photographs. Additionally, light-field rendering requires a complex capturing rig. Surface light fields [27] require a smaller number of photos, but rely on a laser scanner to obtain accurate geometry. Debevec et al. [8] render architectural scenes captured in only a few photos with the help of approximate geometry. They employ a user-assisted modeling system to obtain rough geometric models, and use projective texture mapping to render the models from novel viewpoints.

In unstructured lumigraph rendering [5], Buehler et al, recognize that image-based rendering techniques lie on a continuum from detailed geometry and few photographs to no geometry and many photographs. They propose an approach that works for any point on the continuum. Given a large number of photographs, their system functions like the lumigraph. Given a 3D model and a few photographs, their system functions like view-dependent texture mapping [7]. Our rendering quality heuristic is motivated by their algorithm.

Aliaga et al. [2] also use a robot to capture images for view synthesis. Their robot captures thousands of photos at regularly-spaced locations, thereby eliminating the need for accurate geometry. Our system differs from theirs in that we only capture a sparse set of photos, which we use to compute accurate scene geometry.

III. Hardware

Our robot is built with a Lego Mindstorms NXT kit. The NXT's 32-bit ARM7 processor handles the low-level motor control and sensor communication. Due to the computational complexity of multiview stereopsis, we run all of our control code on a host PC. We use the Mindstorms NXT Toolbox for MATLAB [3] to control the motors and read sensor data over the NXT's built-in Bluetooth. The robot's sensors include a compass sensor, an ultrasonic rangefinder, mechanical double bumpers on the front and rear, and sensors to monitor the camera status.

The camera is a Cannon A590 running the CHDK [1]. The shutter is controlled by a simple CHDK script that monitors the 5V pin in the camera's USB port. When we wish to take a photo, we command the NXT to turn on one of its three motor ports, which connects to the USB port through a voltage regulator. This in turn causes the CHDK script to trigger the shutter. The photo is then transmitted to the host PC with an Eye-Fi wireless SD card. Figure 1 shows a photograph of our robot. The attached video shows a demo of our robot in action.

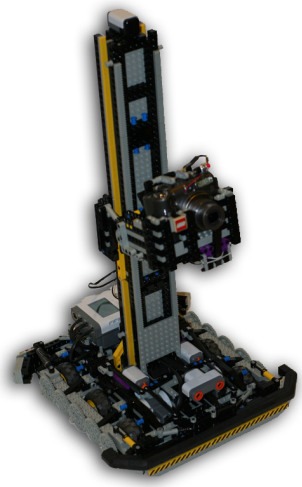

Fig. 1. Our robot

IV. Algorithms

Our robot gathers images in two stages. In the *random exploration* stage, the robot roams semi-randomly, taking photos at intervals. We use these photos to construct an initial map. In the *adaptive exploration* stage, the robot uses its map to determine where it needs to gather more data, and navigates to those areas with its vision system.

A. Random Exploration

SLAM systems update both a robot's location and map in realtime as it explores. Our random exploration stage differs from SLAM in that we do not attempt either localization or mapping until after the exploration completes. We take this approach because our goal is to build a dense reconstruction of the scene, consisting of far more points than a typical SLAM system would track. Global bundle adjustment is a necessary step in the reconstruction pipeline, so for simplicity we allow bundle adjustment to recover the locations of the initial set of photos. The computational cost of bundle adjustment is not a limitation, since, if desired, SLAM-based navigation could be used up until the final reconstruction phase, with local geometry estimated from small clusters of photos serving as geometric proxies during goal selection.

During the random exploration stage, the robot attempts to capture circles of photos, in which it turns in place while taking pictures at roughly 10° intervals. Since the camera is positioned off the robot's axis of rotation, these images provide good baselines for stereo matching. The robot records readings from its ultrasonic and compass sensors along with each photo. When a circle completes, it turns to the compass heading at which it saw the longest range on the ultrasonic sensor, and drives for a random distance before starting another circle.

We do not attempt to avoid obstacles as the robot explores. Instead, we rely on the robot's bumper to detect collisions. When the robot encounters an obstacle, it backs up, turns in a random direction, and continues on a straight path. Should an obstacle fail to trigger the mechanical bumper, the robot will detect that the motors have stalled, and proceed as if the bumper were triggered.

Typically we allow the robot to take around 100 photos in the random exploration stage. While we could reconstruct the scene from fewer photos, capturing view-dependent lighting effects (e.g. shiny reflections) necessitates a larger set of images. When the random exploration stage completes, we use Bundler [25] to compute the 3D location of each photo and a cloud of 3D scene points. Bundler works by matching SIFT [19] features in tracks across multiple images. Thus a 3D point is associated with a SIFT descriptor in each image in which it is identified. We use these SIFT features as landmarks for navigation in the second stage.

B. Adaptive Exploration

Our robot does most of its work in the adaptive exploration stage. The goal of the adaptive exploration stage is to identify areas that require more photographs, travel to those areas, and capture more images.

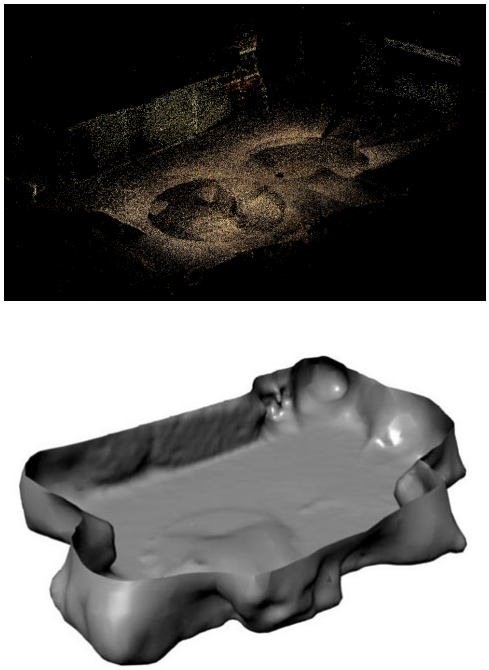

Fig. 2. Top: A point cloud from Bundler. Bottom: The resulting mesh, generated with Poisson surface reconstruction. The details in the interior of the room are missing, but our heuristic does not require accurate geometry.

Choosing a goal. Our adaptive exploration stage begins by computing a geometric model of the visible portions of the scene. To this end we estimate a normal at each 3D point (based on the positions of nearby points), and employ Poisson surface reconstruction [17] to create a mesh. This mesh both defines the robot's boundaries and serves as a geometric proxy in our rendering-quality heuristic. While this mesh is typically only a rough approximation of the true geometry, our heuristic does not require a highly accurate geometric model. Figure 2 shows a point cloud and the resulting mesh.

Intersecting the mesh with the plane defined by the camera positions yields a rough approximation of the boundaries of the robot's space. We choose our goal locations from a set of randomly-selected points within this boundary. At each point, we evaluate a heuristic that estimates the quality of a synthesized image. The heuristic is based on unstructured lumigraph rendering [5]. Given a location L and angle θ, we first position a virtual camera C at L with orientation θ. Next, using a modern GPU, we rasterize the mesh with C. We use a shader that encodes the 3D location of each pixel in the red, green, and blue color channels. This provides the 3D location P of the projected point on the mesh for each pixel p. Ideally, if we were to render the scene from C, we would want to find a source camera C_i at location L_i that captures the ray $R(t) = P + t \cdot \vec{r}$, where $\vec{r} = (L - P)/||L - P||$. The color of the pixel in C_i that captures $R(t)$ then tells us the amount of light reflected from the scene point P in the direction of L.

While it is unlikely that any of the photos will perfectly capture $R(t)$, some of them may capture rays that are very close. Our heuristic attempts to encode exactly how close

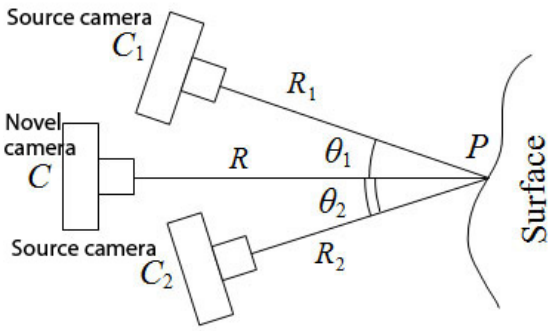

Fig. 3. An example of our heuristic with two source views. The score for the pixel in camera C intersected by $\vec{r}$ is the minimum of the two angles θ_1 and θ_2 between R and the rays R_1 and R_2 from the scene point P to source cameras C_1 and C_2, respectively. In each novel viewpoint, we compute a score for every pixel, and take the average as the view score.

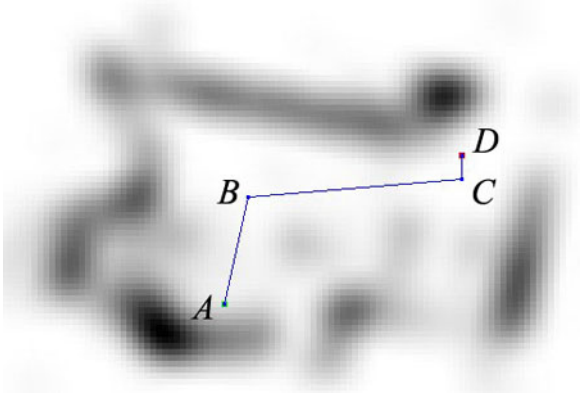

Fig. 4. An example navigation grid and path from A to D via intermediate notes B and C. Darker cells have higher costs. The costs are computed with a kernel density estimator, which gives the grid a blurred appearance.

we are to capturing the ray through each pixel. For each camera C_i (located at L_i) that can see P, we construct the ray $R_i(t) = P + t \cdot \vec{r_i}$ from P through L_i. The angle $\theta_i = \arccos(\vec{r} \cdot \vec{r_i})$ then gives us a measure of how close C_i comes to capturing $\vec{r}$. The smallest angle found in all the views that can see P becomes the score for pixel p. We compute the score for every pixel in each of the randomly-selected novel views. The actual score for each view is the mean of the per-pixel scores. Figure 3 shows an example with two source cameras C_1 and C_2.

We compute the per-pixel scores in parallel on the GPU with Nvidia's CUDA, and then compute the mean on the CPU. With 100 source views, we can evaluate a new viewpoints at roughly 155 frames per second on a GeForce GTX 480. We could possibly achieve higher speed by parallelizing the mean, but the per-pixel scores dominate the computation. Finally, we select the views with the best scores, and send the robot to gather more photos at these locations.

Path-Planning. Our path-planning system takes a similar approach to [22] and [26]. We create a navigation grid (typically about 100 cells in the larger dimension) centered on the bounding box of the previously explored area. Each grid cell c is then assigned a score $s(c)$ determined by the number of above-ground 3D points that it contains. Our coordinate system places all the cameras in the x-y plane, so we can identity above-ground points by setting a threshold on the z-coordinate. We set a conservative threshold of $z = 0$ in all our experiments.

The cost $K(c)$ of entering a cell represents the probability that c contains an obstacle. We precompute the costs of all cells with a kernel density estimator. In a neighborhood N around c, the cost $K(c)$ is given by:

$$K(c) = \frac{1}{|N|} \sum_{c_i \in N} w(c, c_i) \cdot s(c_i) \quad (1)$$

In our experiments, the weights $w(c, c_i)$ were Gaussian, which allowed for efficient precomputation of the cell costs via a separable convolution.

We find a path to our goal location with an A* search on the grid, using the cell costs as weights. We allow the robot to move to any of the 8 neighbors of its current cell. To avoid a long sequence of short moves, we simplify the path when the search completes. First, we merge sequences of moves at the same angle into single longer moves. Then we recursively remove path nodes that can be skipped over without increasing the path cost by more than a constant multiple of the total cost of the original move sequence.

Figure 4 shows the navigation grid for the office scene, and an example path from A to D.

Path-Following. Our robot maintains a target node as it follows its path. Initially, the robot is at the starting node A and the target is the second node B. When the robot comes within a threshold distance of its target node, the next node along the path becomes the new target node, until the robot reaches its goal D. Since the initial map is only a rough approximation of the scene geometry, it is possible that the path to the target is occluded. If our robot encounters an obstacle, it drives a short distance off of its path and then computes a new path to its goal. If the robot encounters more than k obstacles, we assume that our geometric model is inadequate to find a route to the goal (or the goal unreachable), and move on to the next goal. We used $k = 5$ in our experiments.

Accurate path-following depends on the ability to determine the robot's location. Our robot's odometry is inaccurate (mainly due to slippage during turns), and thus we rely on vision for localization. Localization begins by taking a photo, and extracting SIFT features. We then match these new SIFT features to the existing SIFT features associated with the current 3D points obtained from Bundler. We use best bin first search (BBF) [4] to accelerate the matching. Because the point cloud is static, we precompute a balanced KD-Tree, and reuse it every time we wish to localize a new photo. Matching 1,000 features to a KD-Tree containing 160,000 features takes less than 250 milliseconds on a Core i7 920 CPU.

To ensure that our system is robust to bad matches, we use RANSAC to compute the camera parameters from the point correspondences. On each RANSAC iteration, we first use the DLT algorithm [16, §4.1] to estimate a 3×4 camera

matrix P. We then use an RQ decomposition [16, §6.2.4] to extract the camera intrinsic matrix K, rotation matrix R, and translation vector T from P, so that

$$P \approx K \cdot \begin{bmatrix} R & T \end{bmatrix} \tag{2}$$

where K—written in terms of the focal length f, image width w, and image height h—has the form

$$K = \begin{bmatrix} f & & w/2 \\ & f & h/2 \\ & & 1 \end{bmatrix} \tag{3}$$

Due to error in P, (2) is never exact. Moreover, factorizing P can introduce additional error, since the upper triangular matrix obtained via the RQ decomposition does not necessarily take the form of K. We therefore apply a nonlinear optimization to f, R, and T, to minimize the reprojection error of the matched points. The rotation matrix and translation vector from the best consensus set give us the position and orientation of the robot.

C. Reconstruction

When the robot has finished gathering new photos, we regenerate a dense 3D model which includes the newly discovered areas. We first run Bundler again, to obtain precise locations of the new photos and sparse 3D points in previously unexplored parts of the scene. We then use the open-source patch-based multiview stereo software [12] to obtain a dense reconstruction.

V. Evaluation and Results

A. Scene Reconstruction

We tested our robot in an office, which contained a variety of challenges to navigation and localization, including furniture, specular surfaces, and textureless areas. The robot took 108 photos during the random exploration stage, and 153 additional photos during the adaptive exploration stage. Figure 7 shows the resulting point clouds from Bundler before and after the adaptive exploration stage. To better illustrate the completeness of the reconstruction, we also generated dense points clouds with PMVS after each stage.

B. Evaluation of Navigation Accuracy

We evaluated the robot's navigation accuracy in a small test area in our lab. We hung a Webcam from the ceiling, to provide a view of the test area directly overhead. We marked a grid on the floor, which we used to remove perspective distortion from the Webcam image. Then we allowed the robot to run its random exploration stage in the test area, taking a photo with the Webcam every time the robot took a photo. We manually marked the robot's location in each overhead image, which gave us a ground truth heading and orientation in every photo. To measure the accuracy of the camera locations computed by Bundler, we found a similarity transformation that minimized the sum of squared distances between the Bundler camera locations and the ground truth locations. We then computed both the position and heading error between Bundler's results and the ground truth. Since stereo reconstruction algorithms typically have more error along the line of sight, we also computed the line of sight and transverse components of the position error. Figure 5b shows histograms of the results.

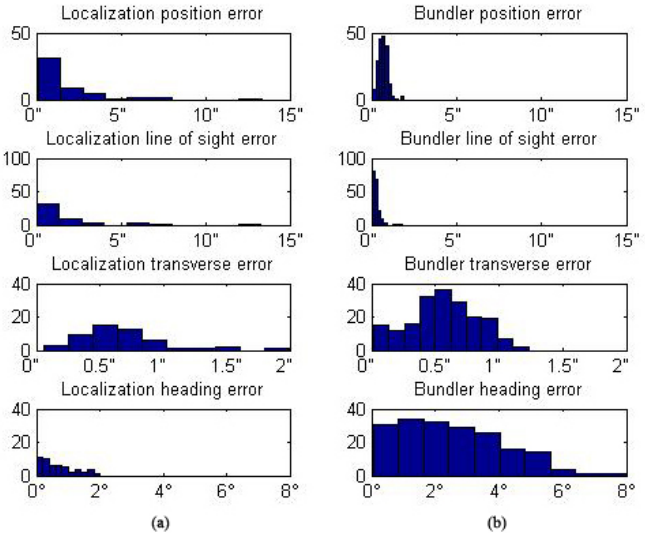

Fig. 5. (a): Histograms of the position error (in inches) and the heading error (in degrees) for vision-based localization from 70 new viewpoints. (b): Histograms of the position error (in inches) and the heading error (in degrees) for the photos in the initial map.

To evaluate the robot's vision-based localization, we placed the robot near a wall in our test area, and took 70 more photos as the robot slowly backed to the opposite wall. We used the map obtained from Bundler to compute the location of each photo. We then used the same similarity transformation to map the locations of the new photos into the ground truth coordinate frame, and compared the results to manually-marked locations and headings. Figure 5a shows histograms of the results.

On average, the position error was about two inches and the heading error about a degree. This is likely within the margin of error on our ground-truth positions.

C. View Synthesis

The photos gathered by our robot are naturally well-suited for image-based rendering. We implemented a simple view-interpolation system based on unstructured lumigraph rendering [15], using the dense point cloud directly as a geometric proxy. We handle holes in the rasterized point cloud by interpolating the depths from nearby pixels. We tested our rendering system on a set of 261 images taken by the robot in an office. Figure 6 shows several novel viewpoints. There is some blurring due to slight inaccuracies in the recovered geometry. If highly accurate view-interpolation is desired, we would recommend a plane-based reconstruction algorithm such as [11].

D. Limitations and Future Work

We have demonstrated a simple and inexpensive system that can adaptively capture photos for scene reconstruction.

Fig. 6. Synthesized images from a set of photos taken by the robot.

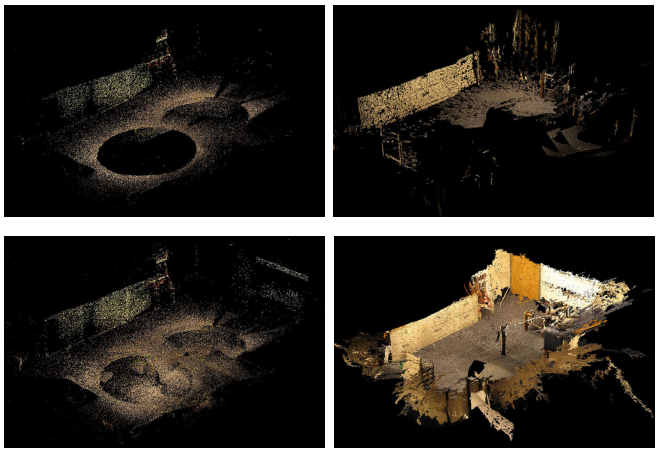

Fig. 7. Top left: The point cloud after the random exploration stage. Top right: The resulting dense point cloud computed with PMVS. Bottom-left: The point cloud after the adaptive exploration stage. Bottom right: The resulting dense point cloud computed with PMVS.

Our robot is guided by a heuristic that is well suited for both dense reconstruction and image-based rendering. Currently, we use two stages in the scene reconstruction, the random exploration stage and the adaptive exploration stage. If the adaptive exploration stage misses parts of the scene, we can repeat the process as many times as necessary. One possible direction for future work would be to dynamically update the map during the adaptive exploration stage, and stop automatically when it is complete.

Currently, evaluating our heuristic requires offline bundle adjustment before the adaptive exploration stage. Ideally, we would avoid global bundle adjustment until we build the final 3D model. Our heuristic requires only rough geometry, so it would be conceivable to obtain approximate geometry by running bundle adjustment on small clusters of images as the robot explores. We could then use SLAM-based methods for navigation, and eliminate the need for the random exploration stage.

REFERENCES

[1] *CHDK*, 2007–2010, `http://chdk.wikia.com/wiki/CHDK`.

[2] D. Aliaga, T. Funkhouser, D. Yanovsky, and I. Carlbom, *Reconstructing building interiors from images*, Proc. IEEE Visualization, 2002, pp. 331–338.

[3] Linus Atorf, Alexander Behrens, Achim Knepper, Robert Schwann, Rainer Schnitzler, Johannes Ball, Thomas Herold, and Aulis Telle, *RWTH - Mindstorms NXT toolbox for Matlab*, 2007–2009, `http://www.mindstorms.rwth-aachen.de`.

[4] Jeffrey S. Beis and David G. Lowe, *Shape indexing using approximate nearest-neighbour search in high-dimensional spaces*, CVPR, 1997, pp. 1000–1006.

[5] Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen, *Unstructured lumigraph rendering*, SIGGRAPH, 2001, pp. 425–432.

[6] Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, and Olivier Stasse, *Monoslam: Real-time single camera slam*, **29** (2007), no. 6, 1052–1067.

[7] Paul Debevec, Yizhou Yu, and George Boshokov, *Efficient view-dependent image-based rendering with projective texture-mapping*, Tech. report, 1998.

[8] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik, *Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach*, SIGGRAPH, 1996, pp. 11–20.

[9] Ethan Eade and Tom Drummond, *Monocular slam as a graph of coalesced observations*, ICCV, 2007, pp. 1–8.

[10] ———, *Unified loop closing and recovery for real time monocular slam*, BMVC, 2008.

[11] Yasutaka Furukawa, Brian Curless, Steven M. Seitz, and Richard Szeliski, *Reconstructing building interiors from images*, ICCV, 2009, pp. 80–87.

[12] Yasutaka Furukawa and Jean Ponce, *Accurate, dense, and robust multi-view stereopsis*, PAMI **1** (2009), no. 1, 1–8.

[13] D. Gallup, J. Frahm, P. Mordohai, Q. Yang, and M. Pollefeys, *Real-time plane-sweeping stereo with multiple sweeping directions*, CVPR, 2007, pp. 1–8.

[14] Michael Goesele, Noah Snavely, Brian Curless, Hugues Hoppe, and Steven M. Seitz, *Multi-view stereo for community photo collections*, ICCV, 2007, pp. 1–8.

[15] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen, *The lumigraph*, SIGGRAPH, 1996, pp. 43–54.

[16] Richard Hartley and Andrew Zisserman, *Multiple view geometry in computer vision*, 2nd ed., Cambridge University Press, 2006.

[17] M. Kazhdan, M. Bolitho, and H. Hoppe, *Poisson surface reconstruction*, Proc. of SGP, 2006, pp. 61–70.

[18] Marc Levoy and Pat Hanrahan, *Light field rendering*, SIGGRAPH, 1996, pp. 31–42.

[19] David G. Lowe, *Object recognition from local scale-invariant features*, ICCV, 1999, p. 1150.

[20] P. Merrell, A. Akbarzadeh, L. Wang, P. Mordohai, J. Frahm, R. Yang, D. Nistér, and M. Pollefeys, *Real-time visibility-based fusion of depth maps*, ICCV, 2007, pp. 1–8.

[21] Q. Pan, G. Reitmayr, and T. Drummond, *ProFORMA: Probabilistic Feature-based On-line Rapid Model Acquisition*, BMVC, 2009.

[22] Kohtaro Sabe, Masaki Fukuchi, Jens-Steffen Gutmann, Takeshi Ohashi, Kenta Kawamoto, and Takayuki Yoshigahara, *Obstacle avoidance and path planning for humanoid robots using stereo vision*, ICRA, vol. 1, 2004, pp. 592–597.

[23] S. Se, D. Lowe, and J. Little, *Vision-based mobile robot localization and mapping using scale-invariant features*, ICRA, 2001, pp. 2051–2058.

[24] Stephen Se, David G. Lowe, and James J. Little, *Vision-based global localization and mapping for mobile robots*, IEEE Transactions on Robotics **21** (2005), 364–375.

[25] Noah Snavely, Steven M. Seitz, and Richard Szeliski, *Photo tourism: exploring photo collections in 3d*, ACM Trans. Graph. **25** (2006), no. 3, 835–846.

[26] Jens steffen Gutmann, Masaki Fukuchi, and Masahiro Fujita, *Real-time path planning for humanoid robot navigation*, IJCAI, 2005, pp. 1232–1237.

[27] Daniel N. Wood, Daniel I. Azuma, Ken Aldinger, Brian Curless, Tom Duchamp, David H. Salesin, and Werner Stuetzle, *Surface light fields for 3d photography*, SIGGRAPH, 2000, pp. 287–296.

[28] Ruigang Yang and Marc Pollefeys, *Multi-resolution real-time stereo on commodity graphics hardware*, CVPR, 2003, pp. 211–218.