Real-time 3D reconstruction for mobile robot using catadioptric cameras

Romain Rossi, Xavier Savatier, Jean-Yves Ertaud, Bélhacène Mazari

Real-Time 3D Reconstruction for Mobile Robot Using Catadioptric Cameras

Romain Rossi, Xavier Savatier, Jean-Yves Ertaud, Bélahcène Mazari
Research Institute for Embedded Systems (IRSEEM),
Technopôle du Madrillet - Av. Galilée - BP10024
76801 Saint Etienne du Rouvray Cedex - France
E–mail: {rossi, savatier, ertaud, mazari}@esigelec.fr.

***Abstract*—This paper presents a 3-second 3D reconstruction algorithm able to process a dense geometric approximation of the surrounding environment. Image acquisition is done by a stereoscopic panoramic system with two color catadioptric cameras mounted on a mobile robot. An algorithm running on a Graphical Processing Unit (GPU) processes the 3D reconstruction in real-time. As the camera system moves, new views of the scene are used to improve the model of the scene thanks to an incremental algorithm. Then, the performance of our approach is evaluated using a synthetic image sequence**

***Index Terms*—GPU, CUDA, 3D reconstruction, voxel, catadioptric camera, computer vision**

I. Introduction

Determining the 3D geometric structure of an unknown environment from pictures is a typical problem in computer vision. Being able to do so in real-time can simplify a variety of tasks like obstacle avoidance and path finding for mobile robots or other unmanned vehicles. In the case of remote-controlled vehicles, displaying a 3D model of the scene to the operator can improve their awareness of the situation, it can also help them to perform complex tasks. The two main parts of 3D reconstruction (image acquisition and data processing) need to be performed quickly in order to keep the 3D model of the scene up-to-date with currently occurring events. In order to minimize the acquisition time of visual data of the surrounding environment, wide field-of-view imaging systems can be used. Panoramic pictures can be taken with a variety of vision systems, like rotating cameras [1], camera networks [2], or catadioptric cameras [3].

Rotating cameras provide very-high resolution pictures (up to 9000×59000 pixels) with a low frame-rate (8 seconds per frame)[4]. However, their mechanical moving parts make it difficult to embed them into autonomous robots and the very low frame-rate makes them unsuitable for real-time applications. Camera networks, where multiple perspective cameras are combined, can be a solution for panoramic acquisition. The main drawbacks of these systems are the synchronisation issues and the high number of generated pictures. Using catadioptric cameras is another panoramic imaging solution . They are made of a perspective camera looking towards a rotationally symmetric mirror, they are compact, without moving parts and provide a 360° field-of-view in one shot allowing a high acquisition rate.

To extract 3D information from a scene, at least two different viewpoints are needed. A stereoscopic configuration can be achieved using two identical cameras with intersecting fields-of-view. In the case of panoramic cameras, a vertical alignment maximizes the field-of-view common to both sensors. Images with different viewpoints can also be taken by moving the vision sensor, provided the new position of the cameras can be estimated. An incremental algorithm can use these new viewpoints to improve the model of the scene.

Processing the 3D model of the scene in real-time is another difficult challenge. With the recent development of multicore CPUs (Intel Core2 / AMD Phenom : 2 to 4 cores) and many-core GPUs (nVidia GeForce 8xxx/9xxx : 96 to 240 cores) parallel algorithms using these processors can be many times more efficient than sequential algorithms. Processing algorithms have to be carefully adapted to the processor architecture in order to maximize performance.

In this paper, our complete acquisition and processing system is presented. Section II presents our stereoscopic panoramic sensor and its calibration. The proposed 3D reconstruction method is introduced in section III and its implementation is detailed in section IV. Experimental validation using synthetic data is presented in section V and these results are discussed in section VI before the conclusion.

II. Stereoscopic Panoramic Sensor Design and Calibration

The stereoscopic panoramic system developed for our experiment is composed of two vertically stacked catadioptric sensors (see Fig.1). The field-of-view of each sensor is $360° \times 106°$ and the vertical alignment maximizes the common field-of-view for both sensors.

Each sensor is composed of a Prosilica GC1380C camera, a Neovision H3S hyperbolic mirror and a Fujinon HF9H1-1B lens. The camera provides a 1360×1024 pixels resolution in color, but only the central 1024×1024 pixels are useful because of the mirror's shape (see Fig. 2). It can sustain a frame rate of 20 frames per seconds through the gigabit Ethernet connexion.

The distance from the mirror to the camera is approximately 9 centimeters, so a single sensor measures about $20 \times 6 \times 6$ centimeters. The baseline of the system is 228 millimeters

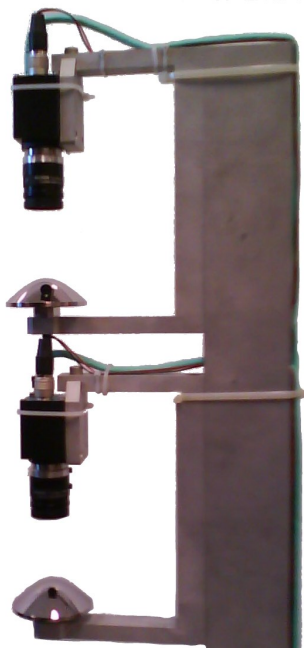

Figure 1. Stereoscopic catadioptric vision system

Figure 2. Sample picture from a catadioptric sensor

and the complete system measures $43 \times 6 \times 20$ centimeters, including the mechanical support.

Previous research provides several options for geometric calibration of catadioptric sensors. One of these solutions developed by [5] and [6] uses an equivalent projection sphere. We used this model which is valid for all central catadioptric sensors, such as our design. It relies on the acquisition of a checkerboard calibration pattern to process the camera's intrinsic parameters and the relative position of the two sensors.

Complete information on the design, construction and calibration of this vision system can be found in [7].

III. 3D Reconstruction Method

Processing a complete 3D scene reconstruction in real-time is a demanding task in terms of processing power and memory bandwidth. We designed our reconstruction method to run specifically on a nVidia GPU, exploiting its massively parallel processor and large bandwidth memory. On these kind of processor architectures called SIMD (Single Instruction stream, Multiple Data streams) according to Flynn's taxonomy [8], it is important to carefully design the data access pattern to avoid conflicts which can result in dramatic performance penalties. The best way to avoid such conflicts is to split the process into independent tasks which do not need data from other running tasks to complete, so the algorithm is "data-parallel". Our method was designed, from the beginning, to be fully data-parallel.

A. 3D Information Extraction from Pictures

The most studied 3D modelling methods in the field of catadioptric imaging are stereo-vision methods, for example [9] or [10]. Using two cameras or a single mobile camera, these methods search for a set of remarkable features (points, edges, corners) on the reference image and try to find matching features on the other image(s). 3D position of the point is triangulated using the coordinates of the matching features and the cameras' parameters. This category of methods can be referred as "2D to 3D" methods, because matching is done on the picture (2D) and then the scene point (3D) is determined based on this information. While these methods provide accurate results, they usually use global optimisation techniques which are computationally intensive. Another drawback of these methods is the inefficiency of features matching when the cameras are widely spaced.

Other 3D reconstruction methods exist such as Voxel Coloring [11] and derived methods [12], [13] which can also be referred to as "3D to 2D" methods. A virtual volume containing the scene is sub-divided into volume elements named *voxels*. Each voxel is then projected onto the images of the scene and the projections are tested for similarities. A typical test is the color similarity of all projections of a given voxel. These methods perform well even with a large baseline and randomly-distributed viewpoints. The regular 3D scene structure lends itself to task parallelization which can improve performance on a parallel processor.

B. Proposed Method

In order to maximize performance for the 3D reconstruction, we decided to design a "3D to 2D" modelling method using a regular 3D array to store the model of the scene. This data organisation helps to improve data access speed and workload distribution across parallel processors because the memory address of a voxel is solely determined by its coordinates. However, this uses more memory whereas hierarchical structures like octrees are more compact.

Our 3D reconstruction method is inspired from Voxel Coloring [11] and uses similar principles and hypotheses. First of all, the scene to model is included in a Volume Of Interest (VOI), an imaginary bounding box of the 3D scene model. This VOI is sub-divided by a 3D regular grid into voxels. Depending on the application, different VOI size and voxel size can be used. In our case, voxels are cubes of all equal size.

The reconstruction process mainly consists of labeling each voxel as *opaque* (the voxel lies on the surface of a scene's

object) or *transparent* (the voxel represents unoccupied space) and determining the color of the opaque voxels. Finally, the set of opaque voxels and their color make up the 3D model of the scene.

The scene's lighting is approximated using the Lambert illumination model, meaning a voxel located on an object's surface is seen with the same color from every viewpoint, except in the case of occlusions. On the contrary, a voxel located in an unoccupied region of the scene will be seen as different colors depending on the viewpoint. Therefore to process the 3D model of the scene, the colors of each voxel seen from different viewpoints are compared. This process known as photo-consistency estimation is the key to this reconstruction method.

Typical photo-consistency estimation methods process the projection of a given voxel onto every picture of the scene where it is visible. A statistical analysis of the colors in each projection of the voxel is done and the voxel is declared opaque if the variance (or standard deviation) is below a certain threshold. Our experimental setup uses only two cameras and a voxel is tested with only two color samples, so a statistical analysis makes no sense. To test color similarity, our algorithm calculates the euclidean distance between the two colors in the RGB 3D space and compares it to a threshold.

To accurately estimate the photo-consistency of a voxel, it is necessary to determine if the voxel can be seen by the cameras. This is known as visibility estimation. It is necessary to avoid taking into account the voxel's projections that are occluded by another opaque voxel, which may introduce errors into the photo-consistency estimation. In order to accurately estimate the visibility of a voxel, the state of all voxels on the sight-line between the camera and the considered voxel must be known; this introduces data-dependencies which must be worked around to calculate this in parallel for every voxels.

In order to estimate the visibility of a voxel in a data-parallel way, we developed a visibility estimator using an occupancy map of the scene. The occupancy map is a 2D projection of the previous state of the reconstructed scene on an horizontal plane. The visibility of a voxel is then determined by counting the number of occupied cells in the occupancy map between the ground point located below the cameras and the ground point below the considered voxel. Because it's evaluated in 2D, this visibility estimation is less accurate but a lot faster than a 3D visibility estimation. Furthermore, this operation can easily be parallelized without many data-access conflicts.

IV. Implementation

Before the detailed description of the algorithm, an overview is presented to outline the reconstruction process in Fig.3. The camera system is calibrated before starting reconstruction so the projection of a 3D scene point onto each image can be processed. The exact position of the vision system in the scene is supposedly known for each image acquisition.

The initialization step clears the visibility and occupation maps, so each voxel in the scene is considered visible and will be updated with data from the first pair of images.

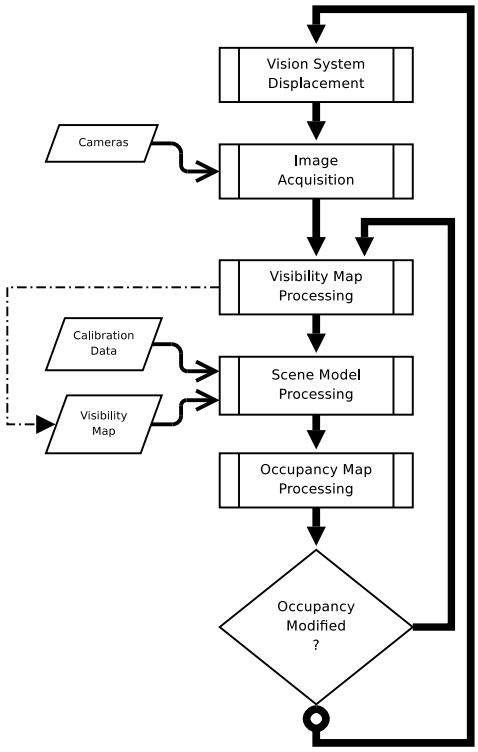

Figure 3. 3D Reconstruction process

During reconstruction, only the last pair of pictures and the previous model of the scene are used. Previous pairs of pictures are not retained.

Here is a detailed description of each of the steps for the 3D reconstruction process.

a) Vision System Displacement: The mobile robot moves the vision system to a known location in the scene. The robot's sensors, or another external measurement system, are used to estimate the position. In the case of a simulation, the different viewpoint locations are established when the synthetic images are generated by the ray-tracing software.

b) Image Acquisition: Image acquisition is triggered by the PC through the Ethernet network connecting the two cameras. Pictures are transferred to the computer for the reconstruction process. In the case of simulation, PovRay is used to render an image sequence of a synthetic scene. The 3D reconstruction software reads these images in sequence, simulating a real camera. No pre-processing is done on the pictures, except for a conversion from the raw format to 24-bit RGB bitmap picture format. The pictures are stored in the GPU RAM so the CUDA program running on the GPU can access them.

GPU hardware texturing units allow a bi-linear interpolation to be performed on the pictures at the same time a read-access is done. Thanks to wired logic functions this operation requires no extra time. Using this feature, performance can be greatly improved.

c) Visibility Map Processing: The 2D visibility map is processed using the current camera position in the scene and

the occupancy map. The visibility information is the number of occupied cells in the occupancy map on the straight line from the camera to the considered voxel. The visibility map is a 2D map so the same visibility information applies for a column of voxels along the vertical axis. Fig.5 illustrates the relationship between the occupancy and the visibility maps.

To accurately process the discrete path on the occupancy map, the Bresenham line algorithm is used [14]. This incremental algorithm, generally used to plot lines on computer screens, is very efficient for finding which cells intersect with a straight line on a 2D array. Along this line, the number of non-empty cells on the occupancy map is counted and the result makes an occlusion estimation. The visibility estimation stored in the corresponding cell of the visibility map is derived from the occlusion estimation using a simple subtraction $Visibility = 255 - Occlusion$. Visibility values outside the range $[0; 255]$ are saturated.

Visibility map processing is performed by a CUDA kernel (GPU program) running on the GPU. One processing thread is launched for every cell in the visibility map. The resulting visibility map is stored in the global memory of the GPU which can be read and written from the GPU.

d) Scene Model Processing: The scene model processing algorithm is detailed in Algo.1. A voxel stores two pieces of information : a 24-bit RGB color and a visibility index which indicates "how well" the voxel was seen when it was last updated. In the reconstruction step, a voxel is updated only if its current visibility is greater than the stored one, meaning there are fewer occlusions between the camera and the voxel from the current point of view.

The photo-consistency measure used is a simple distance in the 3D RGB color space which is processed according to (1). The voxel's center is projected onto both images and the colors of these projections are calculated with bi-linear interpolation. A simple Euclidean distance is calculated and the result is compared to a threshold in order to determine the state (opaque or transparent) of the voxel. The voxel's color is calculated by averaging the colors of all its projections. This color is stored as well as the new visibility index in the 3D scene model. A transparent voxel is indicated using the color value $(0; 0; 0)$ and is treated specially by the program.

$$d(C_0, C_1) = \sqrt{(R_0 - R_1)^2 + (G_0 - G_1)^2 + (B_0 - B_1)^2} \quad \text{with } C_i = (R_i; G_i; B_i) \text{ an RGB color} \quad (1)$$

The reconstruction is stored in the global memory of the GPU. A typical voxel grid ($500 \times 500 \times 200$ voxels, 4 bytes per voxel) takes about 200 MB in RAM.

e) Occupancy Map Processing: The occupancy map is a 2D array where each cell contains occupancy information for one vertical row of voxels, stored on an 8-bit integer. The occupancy information is the number of opaque voxels in this voxel column.

A sample scene with the corresponding occupancy map is presented in Fig.4 to illustrate the processing of this information.

Algorithm 1 Scene model processing algorithm

- FOREACH voxel V (in parallel for all voxels)
  - IF current visibility for V is better than stored visibility for V
    - Process projections of V on both image planes
    - Read the corresponding pixels in images
    - Calculate voxel color
    - Test photo-consistency
    - IF V is transparent
      - Set voxel color to (0;0;0)
    - END IF
    - Store new voxel color for V
    - Store new visibility index for V
  - END IF
- END FOREACH

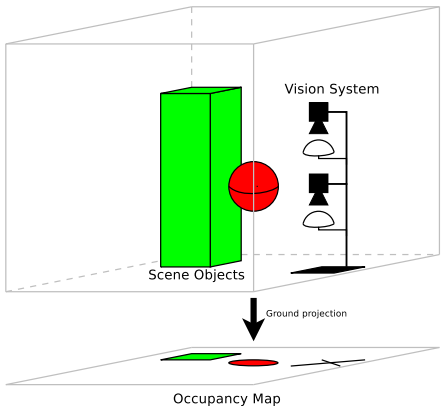

Figure 4. Sample scene and occupancy map. Camera position in the occupancy map is marked only for reference.

The occupancy map processing is also done in the GPU. One thread is launched per occupancy cell which counts the number of opaque voxels in one column, stores this value in the occupancy map, and raises a flag if a change is made.

f) Detecting Changes: When an update of the occupancy map is performed, the previous value is tested to detect any changes. If there is a change, it indicates a change in visibility.

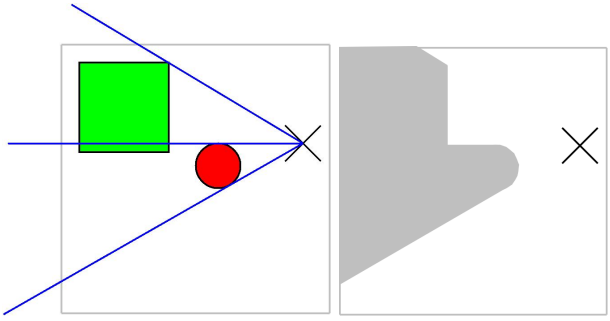

Figure 5. Sample occupancy map (left) and corresponding visibility map (right), top view. The cross marks the position of the cameras. The gray area indicates non-visible voxels on the visibility map.

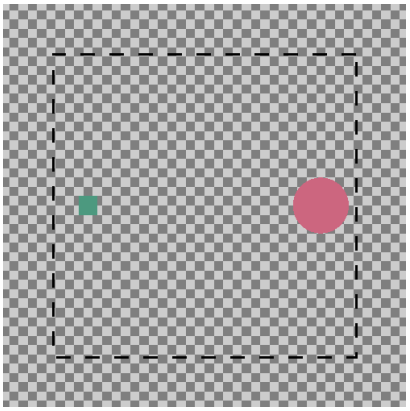

Figure 6. Top view of the synthetic scene. The sensor path for the image sequence appears as a black dashed line.

In this case, a software flag is raised to indicate the need to perform another iteration of the reconstruction process in order to take into account these changes. This is necessary to ensure the updating of all previously hidden voxels unveiled when an opaque voxel is updated to transparent state. The reconstruction algorithm starts a new iteration from the visibility map processing step.

V. Results

The reconstruction technique explained in this paper was implemented on a personal desktop computer. The implementation used a nVidia 8800GTS GPU with 320MB of embedded RAM with the help of CUDA 2.2. The PC is equipped with an Intel Core2 Q6600 CPU with 4GB of RAM and runs Ubuntu 8.04.1 x64 server edition.

A synthetic scene was used to validate the proposed method. The PovRay ray-tracing software was used to generate the image sequence from the scene description. The list of sensor positions was generated at the same time.

A. Synthetic Scene Description

This synthetic scene contains a very tall green box and a red sphere in a gray room. The floor has a checkerboard pattern and realistic lighting casts shadows on the walls and objects. The green column is 150 cm tall and 20×20 cm wide and on the ground. The sphere's diameter is 60 cm, and it's floating 10 cm above the ground.

A top view of the scene is presented in Fig. 6. The objects' positions are clearly visible as well as the floor. The vision system does not appear in this view.

Fig. 7 presents a sample pair of pictures from the sequence. The complete sequence contains forty pairs of pictures rendered from different positions along a square path around the scene. The scene size is 5×5 m and 3 m high. Camera system displacement is in 375 mm steps.

B. Reconstruction results

Fig.8 shows the algorithm in action. A 2D slice of the 3D reconstruction is shown at different stages during the incremental reconstruction. Step 1 shows the scene state just after the processing of the first pair of images. The complete sequence is composed of forty steps, only six are shown.

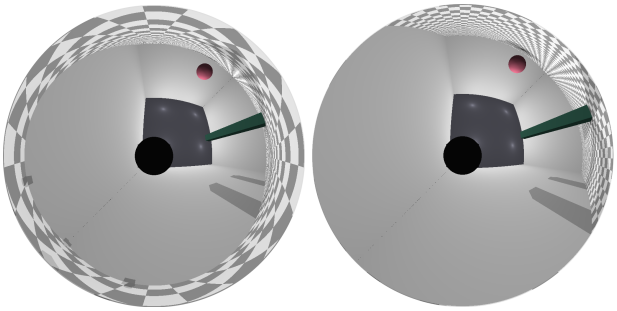

Figure 7. Camera views of the scene (left : lower camera, right : upper camera)

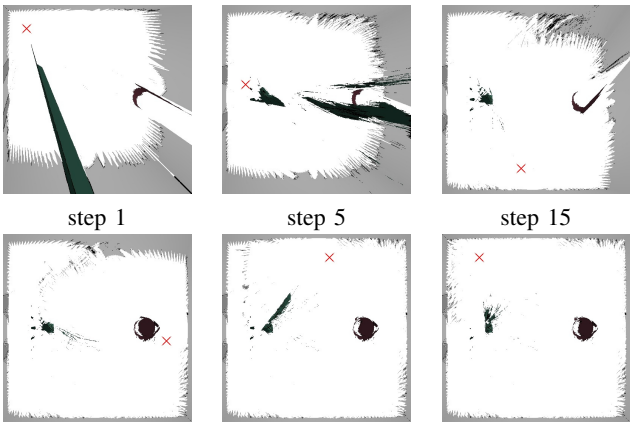

Figure 8. Ongoing incremental reconstruction (top view of an horizontal slice, at successive steps as the sensor moves around the scene). The slice shown is 50 cm above the floor. White indicates transparent voxels. The red cross marks the location of the cameras at each step.

Fig 9 shows the final model of the scene. Horizontal slices of the 3D model, taken at increasing heights in the scene VOI are presented. This view allow objects measurement along the vertical axis.

The scene VOI was set to a size of $5 \times 5 \times 2$ m and the voxel size was $1 \times 1 \times 1$ cm so the VOI consisted of $500 \times 500 \times 200$ voxels.

The reconstruction time was about 3 seconds on average for each new view, including all the steps for the 3D reconstruction process and data transfer to and from the GPU. A previous implementation using only a single thread running on the CPU needed about 20 minutes to perform the same tasks.

VI. Discussion

In step 1, the reconstruction is estimated solely on photo-consistency, as visibility and occupation information are not yes available. This first reconstruction contains many errors, the objects are reconstructed as long "cones" from the cameras' centers to the border of the VOI. This effect is due to weak texturing of the objects present in this scene and is well studied in [11]. When the camera system moves around the scene, incorrect voxels are updated : the green voxels behind

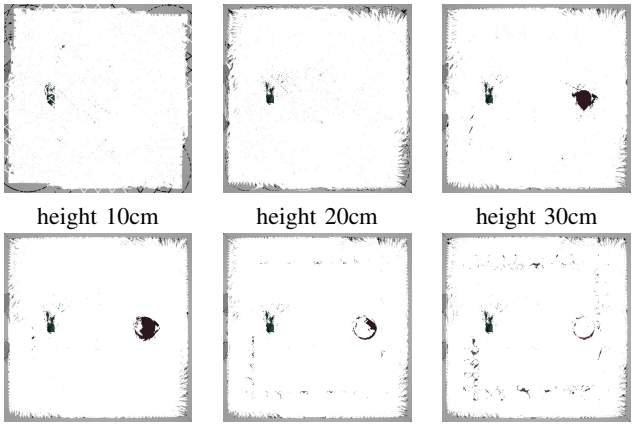

Figure 9. Final reconstructed scene (Top view of horizontal slices taken at increasing heights). White indicates transparent voxels.

the box disappear as well as the gray voxels in front of the walls.

Photo-consistency performs a lot better on the sphere thanks to the color gradient caused by the shadows, so only the surface voxels are initially set opaque. As the camera moves around the sphere, the previously hidden side of the sphere is seen and reconstructed.

When the camera returns to its first position, the whole scene has been seen and the reconstruction is complete. There is some noise in the final reconstruction, but the shapes of scene objects were successfully depicted.

In the reconstructed scene, the sphere object appears as a $68 \times 66 \times 58$ cm voxel group, in the wider part (including neighboring erroneous voxels). Considering the real size of the sphere, this gives a measurement error of 13%.

The column is reconstructed as a 30×30 cm object during most steps in the incremental reconstruction, however a region of incorrect voxels is added nearby in the last few steps. Most of the errors in the final model of the scene came from the poor discrimination of the photo-consistency algorithm. The simple RGB color distance used is easily mislead by uniformly-colored regions or non-textured objects.

VII. Conclusion

In this paper we presented a fast algorithm for 3D volumetric reconstruction of an unknown environment using catadioptric cameras and a fast visibility estimator. The presented implementation efficiently uses a massively-parallel processor such as a GPU thanks to the fully data-parallel algorithm used.

The incremental method gradually improves the model of the scene as new views of the scene are made available. Each reconstruction was done in a very short time (about 3 seconds for every new view) which can be considered real-time for an autonomous robot with limited moving speed. While the resulting reconstruction is not a photo-realistic representation of the scene and contains some errors, the overall shapes of the objects are depicted.

The main drawback of our algorithm is the necessity to know the location of the vision system at each image acquisition. An improvement would be to develop a location estimator using the previous state of the 3D scene model and odometry information from the robot's internal sensors.

References

[1] L. Smadja, R. Benosman, and J. Devars, "Cylindrical sensor calibration using lines," in *International Conference on Image Processing*, vol. 3, October 2004, pp. 1851–1854.

[2] H. Tanahashi, K. Yamamoto, C. Wang, and Y. Niwa, "Development of a stereo omnidirectional imaging system (sos)," in *26th IEEE IECON*, vol. 1, Nagoya, Japan, October 2000, pp. 289–294.

[3] S. Baker and S. K. Nayar, "A theory of single-viewpoint catadioptric image formation," *International Journal of Computer Vision*, vol. 35, no. 2, p. 175–196, 1999.

[4] [Online]. Available: http://www.panoscan.com/MK3/ReadMore.html

[5] C. Geyer and K. Daniilidis, "A unifying theory for central panoramic systems and practical implications," in *ECCV*, 2000.

[6] J.-P. Barreto, "General central projection systems : Modeling, calibration and visual servoing," PhD Thesis, University of Coimbra, Dept. of Electrical and Computer Engineering University of Coimbra, October 2003.

[7] R. Boutteau, X. Savatier, J.-Y. Ertaud, and B. Mazari, "An omnidirectional stereoscopic system for mobile robot navigation," *Sensors & Transducers Journal, Special Issue on Robotic and Sensors Environments*, vol. 5, pp. 3–17, 2009.

[8] M. J. Flynn, "Some computer organizations and their effectiveness," *IEEE Transactions on Computers*, vol. 21, no. 9, pp. 948–960, September 1972.

[9] M. Lhuillier, "Automatic structure and motion using a catadioptric system," *Computer Vision and Image Understanding (CVIU)*, vol. 109, pp. 186–203, 2008.

[10] B. Micusik and T. Pajdla, "Autocalibration & 3d reconstruction with non-central catadioptric cameras," in *IEEE Conf. on Computer Vision and Pattern Recognition*, vol. 1, June 2004, p. 58–65.

[11] S. M. Seitz and C. R. Dyer, "Photorealistic scene reconstruction by voxel coloring," *International Journal of Computer Vision*, vol. 35, no. 2, pp. 151–173, November 1999.

[12] K. Kutulakos and S. Seitz, "A theory of shape by space carving," *International Journal of Computer Vision*, vol. 38, no. 3, pp. 199–218, 2000.

[13] W. B. Culbertson, T. Malzbender, and G. Slabaugh, "Generalized voxel coloring," in *IEEE Workshop on Vision Algorithms Theory and Practice helded in conjunction with ICCV*, vol. 1883/2000, 1999, p. 100.

[14] J. E. Bresenham, "Algorithm for computer control of a digital plotter," *IBM Systems Journal*, vol. 4, no. 1, p. 25–30, January 1965.