Parallel Implementation of Roadmap Construction for Mobile Robots using RGB-D Cameras [⋆]

Marco Negrete [∗] Jesús Savage [∗] Jesús Cruz [∗] Jaime Márquez [∗]

[∗] *Universidad Nacional Autónoma de México*

Abstract: This paper describes a method to construct roadmaps for service robots' navigation using vector quantization. Point clouds are acquired from a RGB-D camera and are processed in parallel in a GPU programmed with CUDA. Every point in the cloud is transformed into the canonical horizon in order to obtain the plane that represents the floor. A point is considered to be free space if its Z component is less than a given constant. Vector quantization technique is used to partition the free space into regions and the centroids of such regions become the nodes of a roadmap, that is used by a service robot to navigate. To test the proposed method, several experiments were performed in an indoor environment.

Keywords: Mobile robots, vector quantization, clustering, parallel processing

1. INTRODUCTION

Service robots must have the capability of navigating in dynamic environments in order to accomplish their purpose: helping humans in everyday domestic tasks (Chen et al., 2014). Navigation in dynamic environments implies the need for a reactive system that allows the robot to avoid obstacles, but, since service robots must execute complex tasks, it is also necessary a world representation for allowing the robot to plan high level tasks. Thus, a service robot's navigation system must be reactive (in real time) to allow a safe navigation and, at the same time, it must be capable of high level task planning.

Roadmaps are useful for mobile robots' navigation in structured environments. If such roadmaps are constructed with a fast enough sampling time and based on information extracted from robot's sensors, they can be used for obstacle avoidance. There are several techniques for roadmap construction when a geometrical representation of the objects in the environment is available, for example, Voronoi diagrams (Latombe, 1991), visibility maps (Lozano-Pérez and Wesley, 1979) or probabilistic roadmap methods (Kavraki et al., 1996). If a geometric representation is not available, it can be obtained by vector quantization (VQ) methods and, based on this representation, it is possible to build a Voronoi diagram.

This paper describes a parallel implementation of the construction of roadmaps using vector quantization techniques. To build the roadmap, we process the point cloud acquired from a Kinect device to separate free and occupied space. Then, both the free and occupied space are clustered. Centroids and sizes of the occupied space clusters are used as a geometrical representation of the objects in the environment. Centroids of the free space are used as nodes of the roadmap. Point clouds are processed in parallel in a GPU programmed with CUDA. We also implemented the whole process in serially, using C++, in order to compare processing times and test the effectiveness of the parallel implementation. The proposed method was tested in an indoor environment in the context of the Robocup @Home tests.

2. THE SERVICE ROBOT JUSTINA

Justina is a service robot built at the Biorobotics Lab of the Engineering School of the National University of Mexico. This robot and its predecessors have been participating in the Rocobup@Home league since 2006 performing several task like cleaning up a room, serving drinks and several other tasks that the human beings ask for. It is based on the ViRbot architecture for the operation of mobile robots (Savage et al., 1998). Justina has several sensors: two laser range finders, a Kinect sensor, a stereo camera and a directional microphone. Also, Justina has encoders in each motor (mobile base, head and its two arms). The proposed method uses the Kinect sensor and head encoders. Figure 1 shows the robot Justina and the position of its sensors and actuators.

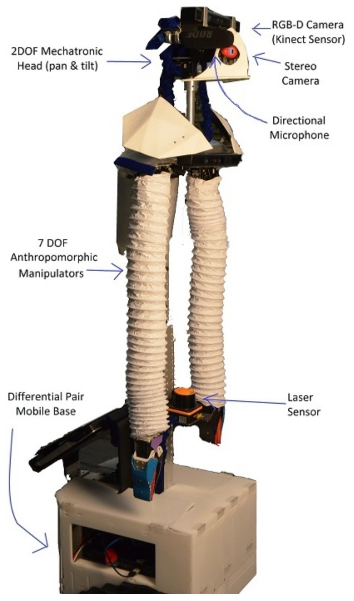
RGB-D Camera (Kinect Sensor)
2DOF Mechatronic Head (pan & tilt)
Stereo Camera
Directional Microphone
7 DOF Anthropomorphic Manipulators
Differential Pair Mobile Base
Laser Sensor

Fig. 1. The service robot Justina

[⋆] This work was partly supported by PAPIIT-DGAPA UNAM under Grant IN-107609 and by CONACYT

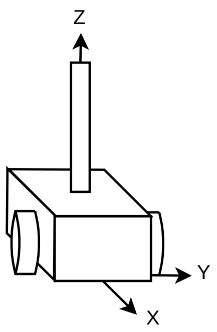

Fig. 2. The robot's frame

3. SEPARATION OF FREE AND OCCUPIED SPACE

RGB-D cameras provide information through an RGB image and a point cloud that represents the spatial position of each pixel of the captured image. This research uses only the spatial information which comes as a set R of triplets of the form $r_j = (x_{screen_j}, y_{screen_j}, d_j)$, where $(x_{screen_j}, y_{screen_j})$ is the pixel location in the image and d_j, the distance to the object on the line of sight. Then, the point cloud S of the cartesian positions $s_j = (x_j, y_j, z_j)$ of the objects w.r.t. the Kinect's plane, can be obtained with the transformation

$$s_j = Mr_j \tag{1}$$

with M, the matrix of intrinsic parameters of the Kinect camera, provided by the OpenNI libraries (Ope, 2010).

The Kinect sensor is mounted in the robot's mechatronic head. This head has two degrees of freedom: pan and tilt. It is built with Dynamixel servomotors whose encoders allow to measure the head orientation.

To transform the point cloud to the robots frame, the following homogeneous transformation is used:

$$p_j = \begin{bmatrix} cos\phi & 0 & sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -sin\phi & 0 & cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos\theta & -sin\theta & 0 & H_x \\ sin\theta & cos\theta & 0 & H_y \\ 0 & 0 & 1 & H_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} s_j \tag{2}$$

where s is calculated according to (1), and (θ, ϕ) are the head pan and tilt respectively, with a positive pan when the head is pointing to the left and a positive tilt when head is looking down, and (H_x, H_y, H_z) is the position of the head w.r.t to the robot's frame. Each point of the resulting point cloud $P = p_j$ is expressed w.r.t. the robot's frame, whose position is showed in figure 2.

A point p_j is classified as free space if its z component is less than a constant K_h and, as occupied space, otherwise.

4. VECTOR QUANTIZATION

Vector Quantization (Linde et al., 1980) is commonly used for data compression in telecommunications and digital signal processing. In the field of robotics, it is also used to compress data and get a smaller but significative set of data. In this research, we use VQ to cluster the free and occupied space and, based on this clusters, to construct a roadmap.

Given a point cloud P, i.e., a set of N_v vectors $p_j = (x_j, y_j, z_j); j = 1, ..., N_v$ that represent the position in space, a set of centroids that represents this vectors is found.

A collection of centroids is called a codebook and it is designed from a long training sequence that is representative of all vectors p_j to be encoded. The codebook is created with the Linde-Buzo-Gray (LGB) algorithm (Linde et al., 1980), as follows:

(1) Initialization: Find an initial coodebook D_1, with only one centroid C_1 which is obtained by averaging all vectors p_j. Let $m = 1$ be the current iteration and $L_m = 1$, the current number of centroids C_i in the codebook D_m.
(2) Disturbing centroids: For each centroid C_i, $i = 1, ..., L_m$ in the current codebook D_m, obtain two new centroids by adding a disturbance $\pm\psi$ of small magnitude. That is, the new codebook D_{m+1} will contain $L_{m+1} = 2L_m$ new centroids.
(3) Given a codebook $D_m = C_1, ..., C_{L_m}$, assign each vector p_j into the nearest cluster R_k, whose corresponding centroid is C_k. Determining the nearest cluster is made with some measurement that satisfies the conditions to be a distance function $d_j = d(p_j, C_k)$. In this research we use the Euclidean distance.
(4) For each cluster R_k, recompute its centroid C_k by averaging all vectors p_j belonging to R_k.
(5) If the difference between the average distance $\overline{d_t} = \frac{1}{N_v}\sum d(p_j, C_k)$, in iteration t, between vectors p_j and their corresponding centroids C_k, and the previous average distance $\overline{d_{t-1}}$, is greater than a constant, i.e. $\left|\overline{d_t} - \overline{d_{t-1}}\right| > \epsilon$, then go to 3.
(6) If $L_m < L_d$, go to 2.

Where L_d is the desired codebook size (number of regions in the environment) and it is chosen with a tradeoff between computation time limitations for real time operation and the desired precision. In this work, we use $|\psi| = 0.01$, $\epsilon = 0.03$ and $L_d = 64$.

The LGB algorithm is applied to cluster both the free and occupied space obtained as described in the previous section, i.e., a total of 128 centroids is calculated, 64 for the free space and 64 for the occupied space. Figure 3, left and center, shows the resulting clusters.

5. ROADMAP CONSTRUCTION

To represent the environment, we consider each centroid of the occupied space as the centroid of a rectangular object of 0.2[m]x0.2[m]. Paths are calculated under this assumption. After the free and occupied spaces are separated and clustered, the roadmap is built following the next algorithm:

Input:

- N: Number of nodes in the roadmap (it should be a power of 2)
- P: Centroids of the quantized occupied space
- C: Centroids of the quantized free space

Output:

- Roadmap G(V, E)
- V: Nodes of the roadmap
- E: Edges of the roadmap

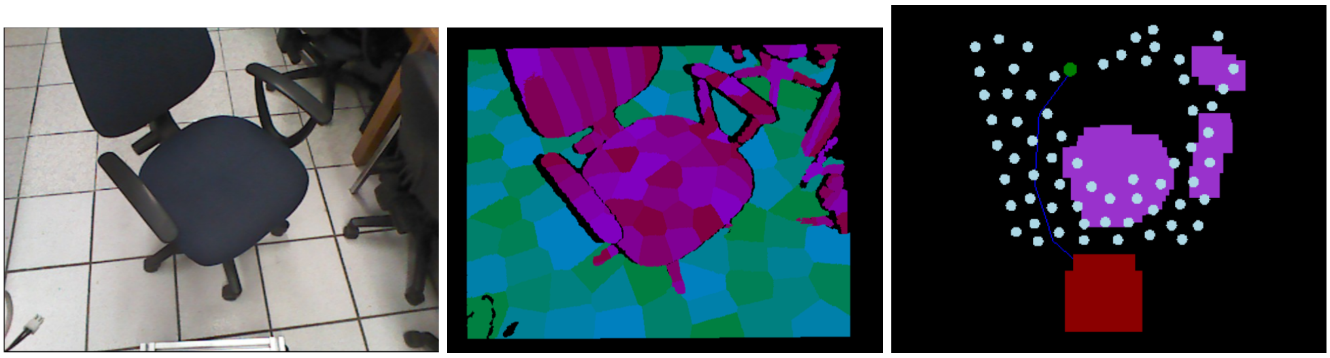

Fig. 3. **Left:** Original RGB image captured by the Kinect. **Center:** Free space clusters are colored in green and occupied space clusters, in purple. The black regions are those points with no depth information. **Right:** Resulting environment representation. Blue dots represent the nodes (free space centroids) used to calculate a path. Purple rectangles represent obstacles and the blue lines are the path calculated by Dijkstra algorithm to reach the goal point, colored in green.

Steps:

```
(1) $E \leftarrow 0$
(2) $V \leftarrow C$
(3) for all $v \in V$ do
(4)     for all $v\prime \in V$ do
(5)         if $e(v, v\prime) \notin V$ and $Vis(v, v\prime, p) \neq NV \forall p \in P$
(6)             $E \leftarrow E \cup e(v, \prime)$
(7)         end if
(8)     end for
(9) end for
```

Where $e(v, v\prime)$ represents the edge between nodes v and $v\prime$ and $Vis(v, v\prime, p)$ is a function that determines if it is possible to reach the node v from node $v\prime$ without crashing with the obstacle whose centroid is p. NV means Not Visible and function Vis return this value when node v is not visible from $v\prime$.

Once the roadmap is constructed, a path to the goal point is calculated using the Dijkstra algorithm. Figure 3, right, shows in blue lines the resulting path to reach the goal point, colored in green.

6. PARALLEL IMPLEMENTATION

The transformation of the point cloud to the canonical horizon, separation of free and occupied space and clustering were implemented in parallel using CUDA, which is a toolkit designed specifically to develop parallel applications on Nvida GPU's. Calculations of the roadmap edges and the Dijsktra algorithm were implemented in serial.

In this research, we use a GPU Nvida Quadro 2000, which has 192 cores, 1.25 GHz and 1GB RAM. We use the CUDA Toolkit 5.0 for the parallel processing, OpenNI 1.5 to capture data from the Kinect sensor, OpenCV 2.4.8 for basic operations on the image (coloring free and occupied space) and Visual Studio 2010 for the general implementation.

7. EXPERIMENTAL RESULTS

Figure 3 shows the results of the whole roadmap construction. The image on the left shows the original image captured with the Kinect device. Since the Kinect is mounted on the robot's head, this image is in the robot's point of view. The figure on the center shows the free and occupied space clusters. Free space is colored in green and occupied space in purple. Regions in black are those points with no depth information. This image is also in the robot's point of view. Image on the right shows the environment as it is represented by the robot. Since navigation is made only in two dimensions, this image is the 2D projection of the free and occupied space clusters. Nodes of the resulting roadmap are drawn in blue and obstacles (obtained from centroids of the occupied space) are drawn in purple. The blue lines are the path calculated by de Dijkstra algorithm to reach the goal point, colored in green.

Serial Time [s]	Parallel Time [s]
1.341	0.078
1.373	0.078
1.341	0.078
1.388	0.077
1.404	0.078
1.357	0.078

Table 1. Comparison of processing time for the serial and parallel implementations.

The complete process is performed all time, so that, a new path to avoid an obstacle is always available.

In order to compare the effectiveness of the parallel implementation, the clustering was also implemented in serial. Table 1 shows the times taken to process one video frame, of six frames, both for the parallel and serial clustering. It can be observed that the parallel implementation is significatively faster than the serial one. The mean time for the parallel execution was 0.078 [s] and it was until 18 times faster than the serial one. This allows a safe navigation since the maximum robot speed is 1.0 [m/s].

The resulting roadmaps have been tested in the service robot Justina, in the context of the Robocup@Home tests, which involve structured indoor environments as described in Chen et al. (2014).

8. CONCLUSIONS

This paper described a method to construct roadmaps for service robots' navigation by clustering the data captured for an RGB-D camera, such as the Kinect sensor. Roadmaps built with this technique have characteristics similar to the ones developed using Voronoi diagrams, but with more flexibility to choose the number and size of the regions in the environment. Commonly, clustering is an expensive process, considering the computation time, and some times it is not suitable to be implemented for obstacle avoidance in robots' navigation. Nevertheless, with the parallel implementation the time taken to process one frame was reduced significatively, allowing the implementation for a safe navigation of a service robot. Finally, the whole process was tested in the service robot Justina in the context of the Robocup @Home tests.

REFERENCES

(2010). *OpenNI User Guide*. OpenNI organization. URL `http://www.openni.org/documentation`. Last viewed 19-01-2011 11:32.

Chen, K., Holz, D., Rascon, C., Ruiz del Solar, J., Shantia, A., Sugiura, K., Stückler, J., and Wachsmuth, S. (2014). Robocup@home2014: Rules and regulations. `http://www.robocupathome.org/rules/2014_rulebook.pdf`.

Kavraki, L.E., Svestka, P., Latombe, J.C., and Overmars, M.H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4), 566–580.

Latombe, J.C. (1991). *Robot Motion Planning.* Kluwer Academic.

Linde, Y., Buzo, A., and Gray, R.M. (1980). An algorithm for vector quantizer design. *Communications, IEEE Transactions on*, 28(1), 84–95.

Lozano-Pérez, T. and Wesley, M.A. (1979). An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10), 560–570.

Savage, J., Billinghurst, M., and Holden, A. (1998). The virbot: a virtual reality robot driven with multimodal commands. *Expert Systems with Applications*, 15(3), 413–419.