

# Time-of-flight camera based probabilistic polygonal mesh mapping

Louis-Kenzo Cahier    Toru Takahashi    Tetsuya Ogata    Hiroshi G. Okuno

Graduate School of Informatics, Kyoto University

## Introduction

In the past decade, time-of-flight cameras – optical cameras that measure the depth of each pixel – have been made available. Combined with a traditional color camera, this type of sensor provides Red-Green-Blue-Depth (RGBD) images at high frequencies. With the mass-produced Microsoft Kinect introduced in late 2010, 30Hz 640 \* 480 RGB images are available for less than \$200. This source of information, while not exempt of problems, is what the stereo vision community has been striving to obtain for years.

We believe this new wealth of information, combined with state-of-the-art Simultaneous Localization And Mapping (SLAM) techniques such as visual SLAM [1], sets the stage for a practical revolution in robotics and artificial intelligence. Indeed, we now have at our disposal methods for sensing geometric, visual and auditory information, all spatialized in the same global reference frame and bound together by SLAM. Because those modalities are the dominant forms of information for humans, robots now have all the essential tools for moving in and interacting with the physical world (geometry), recognizing common objects (vision), and communicating with humans (audition).

As a sensor itself, time-of-flight technology has several advantages and disadvantages. At around 10 frames per second, it is possible to collect 3D data of a scene much faster than a pan-tilt laser scan. This increase in speed is significant especially for scene reconstruction for mobile robots, which cannot wait the 5-10s required to obtain one 3D scan from a laser. The major drawback of TOF technology is its high frame-to-frame noise, especially off reflective surfaces. It is this unreliability that we attack in this paper, to improve the TOF sensor's utility for robotics, vision, and other fields.

In the following sections, we describe a new method to transform TOF data into a probabilistically smoothed polygonal mesh. We first explain how the robot could internally represent its surroundings as a mesh representation. Then we describe a method for maintaining this mesh in an online manner, through triangulation of point estimates and converging to stable values through maximum likelihood estimation.

## 1 Scene representation

The scene is represented internally by the robot as a polygonal mesh possibly having several connected components. The mesh is defined as a pair  $M = (V, T)$ , where  $V$  is a set of vertices, and  $T$  a set of triangles. Each vertex is simply a point in  $\mathbb{R}^3$ , and each triangle is a triple  $(x_i, y_i, z_i) \in \mathbb{N}_+^3$  of indices to vertices in  $V$ .

The triangular nature of the mesh representation is not relevant; the fact that it is a representation of the environment's geometric surface, persists through time, is built incrementally, and can be arbitrarily simplified or complexified as attention is focused or drifts away are the essential properties of our scene representation. This work

focuses on the means to construct and maintain a triangular mesh representation efficiently from high-frequency depth as typically returned by time-of-flight cameras, ignoring – but preparing for – object segmentation and more synthetic properties such as generic descriptors of shape, visual appearance, auditory emissions, articulated degree-of-freedom model, etc.

## 2 Probabilistic Polygonal Mesh

Let us suppose that at time step  $t$  we have a 3D mesh representing part of the environment of the robot. A new depth scan  $S_t \in \mathcal{M}_{n,p}(\mathbb{R}^3)$  is acquired. The points in  $S_t$  are all sensed as the result of light bouncing off some surface and returning through the optical center  $O$  of the sensor. This type of data is more structured and richer than general point clouds, for which we don't have the ray along which light for the sensed pixel passed. In particular, we know that the normal of the real surface at each sensed point makes an angle less than  $90^\circ$  with the ray of light that bounced off it towards the sensor.

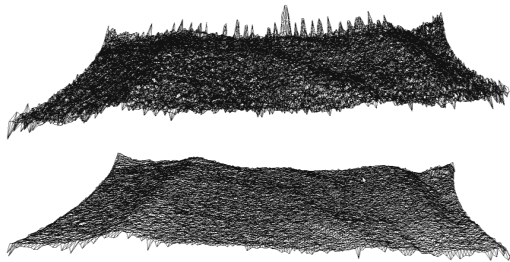
For each element  $S_t(x, y)$  of  $S_t$ , a virtual ray is cast going through it from the position of the sensor's optical center, which may give rise to an intersection point  $I_t(x, y)$ .

**Incremental triangulation** We then consider all triangles of the forms  $S_t(x, y)S_t(x, y + 1)S_t(x + 1, y)$  or  $S_t(x + 1, y)S_t(x, y + 1)S_t(x + 1, y + 1)$ ; any such triangle is added to the mesh  $M$  if none of its vertices have an intersection found in the ray casting step, and it satisfies a continuity criterion. This continuity criterion aims at separating estimates that are neighboring in pixel space, but whose observed points are in fact not neighbors on the same surface. This happens for example when observing 2 objects roughly aligned with the sensor, creating an overlap.

Moreover, neighboring estimates are sensed through 2 projection lines that cross in  $O$  but get farther apart as the distance from the sensor increases, making neighboring estimates of a given surface grow apart as it gets farther. A simple application of Thales' intercept theorem shows us the distance between estimates is proportional to the distance of the object. We account for this by computing a continuity factor for each edge of the triangles,  $(d_{max} - d_{min})/d_{min}$ , which is proportional to the drop in distance of the estimates, but inversely proportional to the distance of the object. We empirically set a maximum threshold of 2.5% for the continuity factor of the edges of potential triangles.

This procedure incrementally builds a mesh from an empty start, by creating new connected components when parts of the environments are newly observed. However, estimates whose ray intersected the existing mesh are not used during the triangulation step.

**Online vertex estimation** Each estimate  $S_t(x, y)$  with an intersection point is a candidate for updating the position of the intersected face's vertices. For each such face vertex  $V$ , we compute the angle  $\alpha = \angle VOS_t(x, y)$ , the angular gain  $\gamma = 1 - \min(\alpha/\rho, 1)$  ( $\rho$  is the angular



**Figure 1:** Example comparing directly triangulated data (above) with a PPM built over several scans (below).

resolution of the sensor, here  $0.24^\circ$ ), the innovation vector  $\vec{I} = (\overline{OS}_i(x, y) - \overline{OV})\overline{OV} / \overline{OV}$  and the error signal.

$\gamma$  is called gain because it is a number between 0 and 1 used to modulate how much an estimate will be incorporated as a new estimate for a vertex. This is because the value of each pixel does not only measure the distance along a line that passes through the center of the pixel, but rather integrates the distances of points viewable from the optical center through the extent of the pixel. We notice that if  $\alpha > \rho$ ,  $\gamma = 0$ , which means that the estimate's distance integration area doesn't contain  $V$  at all and the influence is null; conversely, if  $\alpha = 0$ ,  $\gamma = 1$  and the estimate will be fully incorporated. The innovation vector is colinear to  $\overline{OV}$ , and its magnitude equal to the difference in measured depth.

One possibility to incorporate the estimate  $S_i(x, y)$  is to move candidate vertex with the lowest  $\alpha$  using the innovation vector modulated by the angular gain:  $V = V + \gamma\vec{I}$ .

**Maximum Likelihood vertex estimation** However, this would not guarantee convergence even for a zero-mean noise sensor. Rather, we propose using a maximum likelihood estimation from successive estimates, for each vertex. We modify the mesh data structure to add a real number to each vertex, that we call the effective number of observations  $\Gamma$ ; when a vertex is first added as part of a new triangle,  $\Gamma = 1$ .

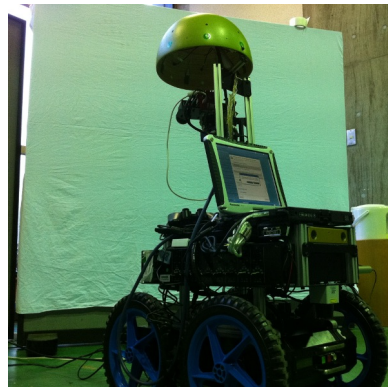
Looking at the lifecycle of a vertex that receives  $N$  innovation vectors  $\vec{I}_1, \dots, \vec{I}_N$  with angular gains  $\gamma_1, \dots, \gamma_N$ , we now incorporate new measurements as  $\Gamma_n = \Gamma_{n-1} + \gamma_n$  and  $V_n = \frac{\Gamma_{n-1}V_{n-1} + \gamma_n(V_{n-1} + \vec{I}_n)}{\Gamma_n}$ , which is the online weighted arithmetic average of the  $(V_{i-1} + \vec{I}_i)_{i \in \{1, \dots, n\}}$  for the vertex, i.e. the maximum likelihood estimation.

### 3 Real-time Implementation

We implemented our PPM algorithm on our custom-made mobile robot named Kappa, which is equipped with a Swiss Ranger TOF camera as a distance sensor. The code is written as a C++ ROS node for portability to other systems. Mesh processing was performed on an off-board computer with two Intel Xeon Quadcores 2.4 GHz, though only no parallel processing was used in particular for multi-core optimization.

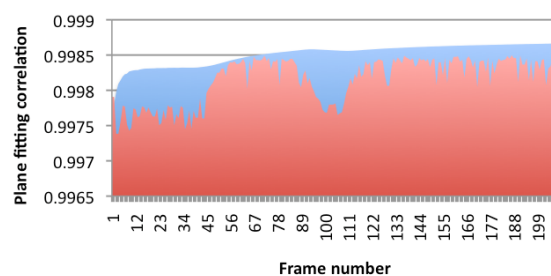
### 4 Evaluation

**Experimental setup and results** In our experiment, we placed a white wall approximate 1.3m away from the



**Figure 2:** Experimental setup, with robot facing white wall.

### Flat surface: Comparison of PPM to Raw



**Figure 3:** Comparison of plane reconstruction performance between PPM (blue) and raw data (red) over 200 frames

robot, and faced the TOF camera towards it such that the image filled the sensor's field of view entirely. The robot remained still during the entire experiment. Our goal here is to how well the raw data represents a relatively flat surface, versus our PPM. We recorded 200 frames of TOF camera images, and tested the two methods as follows: 1) extract the mesh generated by raw point cloud data or the PPM 2) fit a plane to the vertices of the mesh using linear least squares 3) extract the correlation coefficient between the mesh and the plane. The results are plotted in Fig. 3. The higher the coefficient, the better the mesh fits the plane. Note the quick increase and stability of the PPM correlation over the lower and more erratic precision of the raw point clouds.

### 5 Conclusions and future work

We have given a probabilistic method to generate stable 3D data from a time of flight camera, using a mesh representation. Our future work includes integration of new meshes, such that the mobile robot can connect its current representation of the world to any new data it sees. [2] [3]

This research was supported in part by Global Centers Of Excellence (GCOE).

### References

- [1] N. Karlsson, E. di Bernardo, J. Ostrowski, L. Goncalves, P. Pirjanian, and M. Munich, "The vSLAM Algorithm for Robust Localization and Mapping," *ICRA*, pp. 24–29, 2005.
- [2] Z. C. Marton, R. B. Rusu, and M. Beetz, "On fast surface reconstruction methods for large and noisy point clouds," *ICRA*, pp. 3218–3223, May 2009.
- [3] R. Rusu, Z. Marton, N. Blodow, A. Holzbach, and M. Beetz, "Model-based and learned semantic object labeling in 3D point cloud maps of kitchen environments," in *iros*, pp. 3601–3608, IEEE, 2009.