Object tracking in 3D using depth map

Conference Paper · January 2012

CITATION
1

READS
479

2 authors, including:

Zoltan Tomori
Slovak Academy of Sciences
159 PUBLICATIONS **780** CITATIONS

SEE PROFILE

Object tracking in 3D using depth map

Radoslav Gargalík*
Inst. of Computer Science
University of Pavol Jozef Šafárik
Jesenná 5
040 01 Košice, Slovakia

Zoltán Tomori†
Inst. of Experimental Physics
Slovak Academy of Sciences
Watsonova 47
040 01 Košice, Slovakia

Abstract

Our work is devoted to object tracking in 3D. An object can move within a region which is defined by walls and static obstacles (in our case an animal within a cage). We present an approach which is based on three steps. The first step consists of floor detection. The second step focuses on region of interest detection within which an object can move. During the last step an object is segmented using the region of interest which was detected in the previous step. Tracking itself is done by center of gravity calculation throughtout all frames in video. All three steps mentioned above are done automatically.

CR Categories: D.0 [Software]: General— [J.0]: Computer Applications—General

Keywords: tracking, region of interest detection, depth map, segmentation

1 Introduction

Tracking in 3D is the critical problem in many areas of computer vision. 3D data can be acquired by various systems like SLR stereo rig, Time of Flight (TOF) camera, LIDAR, 3D scanner based on structured light etc.. In our work we use Microsoft Kinect device to obtain depth map of the scene in real time and to convert it to the 3D point cloud. Microsoft Kinect represents revolutionary low-cost 3D sensor for Xbox gaming console. It was launched recently and achieved a big commercial success. Support for programmers appeared shortly after it (Microsoft Kinect SDK, OpenNI, OpenKinect, Freenect). In our work we decided to exploit Microsoft Kinect SDK. Using the Kinect device we can represent our scene as RGB-D image (RGB image plus the Depth map). The other form of 3D representation is point cloud image consisting of a set of (x, y, z) values in real units (e.g. meters) where z is the distance of pixel (x, y) from the camera. Kinect depth sensor consist of IR transmitter and IR camera pair. The transmitter projects small dots (speckles) into the surrounding scene and the IR camera acquires image and compares their position with the reference one. The depth is calculated from the displacement of the individual speckles.

*e-mail: radoslav.gargalik@student.upjs.sk
†e-mail:tomori@saske.sk

2 Acquisition

We use Kinect device to obtain the RGB-D images of the scene over the time. An example of such scene image is illustrated on figure 1 which shows a behavioural test of minipig. The animal tracking quantifies its activity studied at the Institute of Animal Physiology SAS in Košice. The resolution of the RGB image is 640x480 and the resolution of the depth image is 320x240. Using the built-in features of Microsoft SDK we convert obtained RGB-D images of the scene to the point clouds. Each point in a cloud has three coordinates (x, y, z) which are measured in milimeters.

Figure 1: Example of RGB-D image of our scene captured by Kinect. Top - RGB image, bottom - depth image.

3 Processing

Our approach consist of three steps: floor plane detection, region of interest detection and object segmentation.

3.1 Floor plane detection

Each plane can be represented by the following equation

$$ax + by + cz + d = 0 \tag{1}$$

where $\vec{N} = (a,b,c)$ is a normal vector of plane and d is the distance of a plane from the origin. Considering the Kinect measurement inaccuracy we use the modified version of the previous equation

$$|ax + by + cz + d| \leq T_f \tag{2}$$

where T_f is threshold which defines the maximum distance of a point from the plane, such a point is considered to be in a plane. In our experiments we use $T_f = 60$ which means that the maximum distance is 60 milimeters.

To detect a floor plane automatically we use one of the RANSAC algorithm modification described in [Chum and Matas 2008][1]. We iteratively choose three random points from the point cloud of scene and construct a plane from them. Then we count the number of points from the scene point cloud which are part of a floor using the equation 2. The whole process is repeated iteratively I_f times. In our experiments the constant I_f was set to 100 which generaly should be enough to detect the plane which contains the most points from the point cloud of the scene. We assume that a floor plane contains the most points from the point cloud according to other planes presented in the scene (e.g. walls). If the condition is not fulfilled the user has a possibility to select a floor plane manually.

After the floor plane was successfully detected (or manually selected if needed) we remove all points from the point cloud which are part of the floor. The result of this step depicts figure 2.

Figure 2: Floor plane automatically detected and removed from the scene.

3.2 Detection of region of interest (ROI)

The next step in our approach is a limitation of an area within which an object can move (region of interest). We limit that area by polygon which is located on the floor. In most cases the ROI is polygon which is not convex, especialy in the case where some static obstacles are located in the scene. The main idea of the ROI detection is to project all the points from the point cloud (except the ones which are part of the floor) parallely to the floor plane and find the polygon representing the ROI.

[1]Even though the Microsoft Kinect has a built-in functionality to detect a floor plane, it is strictly bound to skeleton tracking. Given that we do not use skeleton tracking in our application it is not possible to use Kinect for a floor plane detection purpose.

As figures 1 and 2 illustrate, an object itself is located inside the ROI. We would need to remove an object from the scene to properly detect the ROI but the exact position of an object is unknown at this point. To eliminate this problem we will project parallely only those points from the point cloud which are not part of the floor plane and satisfy the following equation

$$|ax + by + cz + d| \geq H_m \tag{3}$$

where the constant H_m represents the maximum height of an object. In other words, we project parallely those points which distance from the floor plane is at least H_m. In our experiments the maximum height of an object was half a meter, so the constant H_m was set to 500 milimeters.

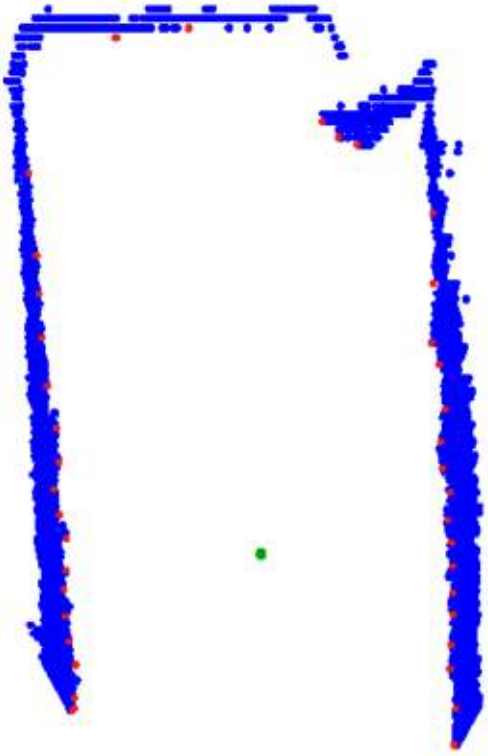

Figure 3: Parallel projection of points to the floor plane (blue points), calculated centroid (green dot) and vertices of a polygon represented ROI (red points).

After all the points from the point cloud which satisfy equation 3 were projected to the floor plane (blue points on figure 3, we use the following algorithm to find the ROI:
First we calculate the centroid of projected points (green circle on figure 3). Let $C = (cx, cy)$ be that centroid. Let $\alpha_i = 360/k$ be an incremental angle, where k is the number of itnerations. For each iteration we define angle $\alpha = \alpha + \alpha_i$, starting with $\alpha = 0$ degrees. In each iteration step we construct a line L, where an angle between the up vector $\vec{U} = (0,-1)$ and line L is α (see figure 4). Now for each projected point $P_i = (x_i, y_i)$ we construct a vector $\vec{CP_i} = (x_i - cx, y_i - cy)$ and we calculate the value V_{P_i} using the equation

$$V_{P_i} = K_\alpha |\sphericalangle(L, \vec{CP_i})| + K_D |\vec{CP_i}| \tag{4}$$

where the $|\sphericalangle(L, \vec{CP_i})|$ is an angle between line L and vector $\vec{CP_i}$ (angle β on figure 4). From all projected points we choose the best candidate for ROI polygon vertex, which has the smallest value V_{P_i}. In our experiments $K_\alpha = 100$ and $K_D = 1$.

To improve the speed of our algoritm we construct another line $L_\perp$ which is perpendicular to line L (see blue line on figure 4) and we use equation

$$ax_i + by_i + c < 0 \tag{5}$$

to skip those points in each iteration which are on the other side of the line $L_\perp$ as line L. The resulting vertices of the ROI of one of our scene are depicted on figure 3 (red points).

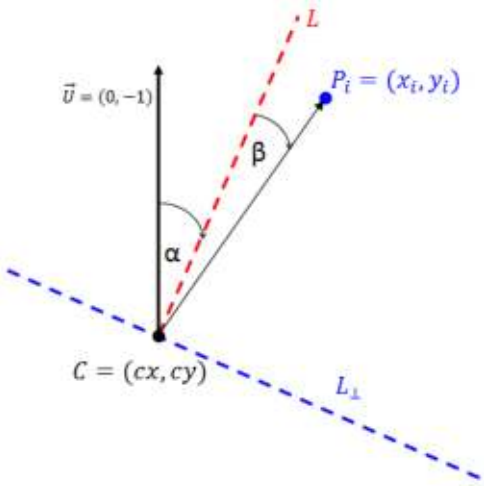

Figure 4: Explanation of the algorithm for computing ROI.

3.3 Object segmentation

In the previous step we detected the polygon which represents the ROI. All vertices of the ROI polygon lie on the floor plane. We segment an object in two steps.

In first step we project all points (except those which are part of the floor plane) in a point cloud to the floor plane. For each projected point we test if it is inside the ROI polygon or not. We use Ray-casting algorithm to check the position of a point in regard to ROI polygon. Points which are not inside the ROI polygon are automaticaly rejected (walls, static obstacles, ...). If a point is inside the ROI polygon, we do one more check to determine if a point is not very near to either side of the ROI polygon. If the point is very near then it is also rejected. The result can be seen on figure 5.

Points which are left in a cloud can be divided into two groups: object points and noise points. To eliminate noise points and to finally segment an object, we project points from 3D back to depth map and use Breadth-first search algorithm (well known from graphs) to count the number of components of such a graph. Components are then filtered based on their size (number of points in a component). Small components are rejected (less than 500 poits in our experiments). The component which contains the most points represents the object points. The result of the segmentation step is illustrated on figure 6.

3.4 Object tracking

After the object was successfully segmented we can easily calculate its center of gravity. Tracking of the object is then reduced to tracking its center of gravity over the time.

Figure 5: Points inside the ROI polygon projected back to depth map. We have object points and noise points.

Figure 6: Final segmentation of the object points.

4 Future work

The essential step in our approach is automatic detection of the floor plane. We do so by using the RANSAC algorithm which in some cases do not return optimal result. If the floor plane is not detected correctly, then the final result of our approach will be significantly worse. The user still has the possibility to select the floor plane manualy in our software but in the future we want to replace the RANSAC algorithm with some better solution.

Also, our approach assumes that obstacles in the scene are static and they are located only in the corners of the scene. In the future we plan to try to improve our solution to handle non static obstacles in the scene.

5 Acknowledgement

This work was supported by VVGS-PF-2012-60 grant and VEGA 2/0191/11.

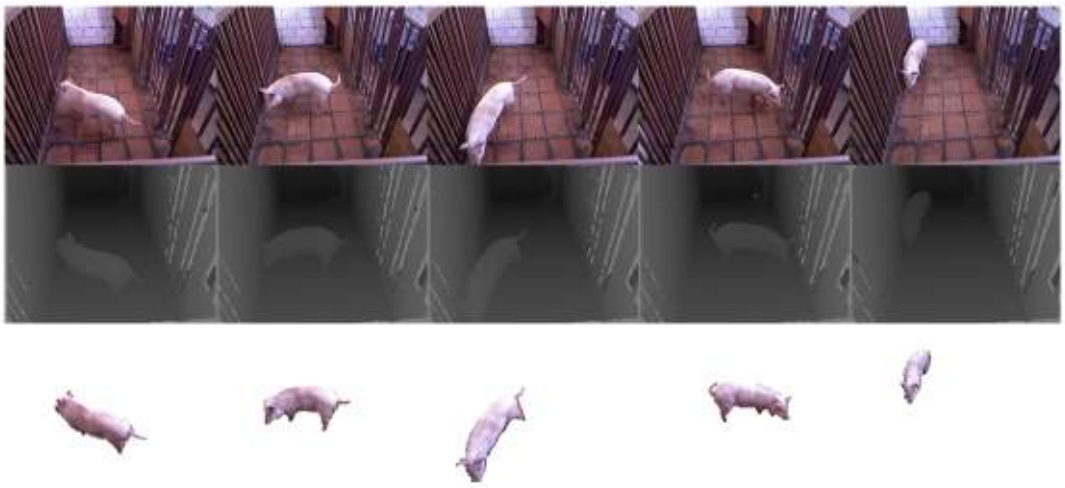

Figure 7: Some of our results. RGB images (top row), depth maps (middle row) and segmented objects (bottom row) of our scenes.

References

CHUM, O., AND MATAS, J. 2008. Optimal randomized ransac. *IEEE Trans. Pattern Anal. Mach. Intell. 30*, 8, 1472–1482.

DAL MUTTO, C., ZANUTTIGH, P., CORTELAZZO, G. M., MUTTO, C. D., ZANUTTIGH, P., AND CORTELAZZO, G. M. 2012. Fusion of depth data with standard cameras data. In *Time-of-Flight Cameras and Microsoft Kinect*, SpringerBriefs in Electrical and Computer Engineering. Springer US, 69–91. 10.1007/978-1-4614-3807-6-5.

GOESELE, M., MATSUSHITA, Y., SAGAWA, R., AND YANG, R., Eds. 2011. International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission, 3DIMPVT 2011, Hangzhou, China, 16-19 May 2011, IEEE.

METAXAS, D. N., QUAN, L., SANFELIU, A., AND GOOL, L. J. V., Eds. 2011. IEEE International Conference on Computer Vision Workshops, ICCV 2011 Workshops, Barcelona, Spain, November 6-13, 2011, IEEE.

MUTTO, C. D., ZANUTTIGH, P., CORTELAZZO, G. M., AND MATTOCCIA, S. 2011. Scene segmentation assisted by stereo vision. In Goesele et al. [Goesele et al. 2011], 57–64.

SILBERMAN, N., AND FERGUS, R. 2011. Indoor scene segmentation using a structured light sensor. In Metaxas et al. [Metaxas et al. 2011], 601–608.

SMISEK, J., JANCOSEK, M., AND PAJDLA, T. 2011. 3d with kinect. In Metaxas et al. [Metaxas et al. 2011], 1154–1160.