POINT CLOUD ALIGNMENT WITH ICP

W. G. C. W. Kumara[1], S. S. J. Gamaralalage[2]

[1] Department of Computer Science and Information Engineering, National Central University, Taiwan (R. O. C), Email: chinthakawk@gmail.com
[2] Department of Industrial and Systems Engineering, Ohio University, Athens, Ohio, USA, Email: jayasinghegss@gmail.com

ABSTRACT

Point cloud is a set of (*x*, *y*, *z*) coordinates of the points of an object or a scene captured using a depth camera. Color information may also contain per each point. Microsoft Kinect is a low cost depth sensor, which is capable of capturing color information as RGB, and depth information, which together called RGB-D information in the literature. We captured the RGB-D information of objects from different camera positions and viewing directions using multiple Microsoft Kinect version 2 sensors. Point clouds are then generated using the RGB-D information. We present the procedure of aligning these different point clouds using Iterative Point Cloud (ICP) algorithm. Detailed information and comparison of the performance of ICP is presented with several objects.

***Key words*:** ICP, point clouds, alignment

1. INTRODUCTION

3D entertainments are getting famous in the world since the last several years. Most of the recent movies released in 3D as well. Many world renowned TV manufacturers like Samsung, Sony, LG, now have multi view 3D TVs. 3D content creation for these 3D capable devices is another field of research. Mostly depth sensing methods in parallel with color sensing is used in such 3D content creation. Several consumer level 3D capturing devices are now widely used by the researches and application developers. Microsoft Kinect v1 and v2, Creative Senz3D, Leap Motion are several famous such depth sensors. Specifications of these devices are given in Table 1 (figures are not in scale). Sens3D, Kinect v1, and v2, provide both color and depth image, which is mostly referred as RGB-D image or stream. In the other side, leap motion provides only a thin wedge shaped depth map. Another noticeable difference between Kinect v1 and v2 is, v1 is capable of vertical camera rotation of ±27° with a motor control, while v2 does not have that capability. Leap Motion from Leap Motion Incorporated is designed for simulation of a computer input device, such as, mouse and keyboard. With the availability of the v2 sensor Microsoft phased out the sale of the original Kinect for Windows sensor in 2015 [1]. Microsoft Kinect v2 sensor can be currently purchased for about 149.98 USD [2]. All the sensors given in the Table 1 are near range sensors. Velodyne LiDAR (Light Detection and Ranging) introduces several middle range depth scanners as shown Figure 1 and as compared in Table 2 . All these sensors can range around 100 m and specially capable of scanning 360° horizontal. Their smallest sensor VLP-16 is only 830 grams in weight and can be purchased for 7,999.00 USD [3].

Table 1: Comparison of depth sensors

		Senz3D [4]	**Kinect v1 [5]**	**Kinect v2 [5]**	**Leap Motion [6]**
Resolution	**RGB**	1280 ×720	640 ×480	1920 ×1080	-
	Depth	320 ×240	320 ×240	512 ×424	150° field of view
Frame rate (fps)		30	30	30	290
Range		0.5-3.25 ft.	0.4-4.5 m	0.5-4.5 m	2.5-60 cm
Raw data		Yes	Yes	Yes	No
Front view					

Figure 1: Comparison of Velodyne LiDAR products (images are not in scale) [3]

Table 2: Comparison of Velodyne LiDAR sensors [3]

	HDL-64E	HDL-32E	VLP-16
Channels	64	32	16
Range (m)	120	80-100	100
Points/second (million)	2.2	0.7	.3
Horizontal FOV (°)	360	360	360
Vertical FOV (°)	26.8	±20	±15
Angular resolution (azimuth) (°)	0.08	-	-
Accuracy (cm)	<2	±2	-
Vertical Resolution (°)	~0.4	-	-

2. METHODOLOGY

2.1. Point Cloud

Point cloud is a set of 3d (*x*, *y*, *z*) coordinates of an object. As we use colored point clouds here, we need to represent color of each point as well. Hence, a point is represented as a vector with 6 elements, 3 for *x*, *y*, *z* coordinates and 3 for RGB color. There are several ways of representing point clouds. One famous file format is PLY format. Figure 1 shows a part of a PLY file. First few lines are the header, which explains the format representation of the PLY file. Third line shows the number of points represented in this point cloud file. Then the representation of a point is explained. In our case, we set the value of alpha component to 0 for every point, as we do not use that value.

```
ply
format ascii 1.0
element vertex 49152
property float x
property float y
property float z
property uchar red
property uchar green
property uchar blue
property uchar alpha
end_header
-0.010346 -0.007755 0.017000 90 110 94 0
-0.010054 -0.007755 0.017000 80 104 90 0
-0.010022 -0.007755 0.017000 79 104 92 0
-0.009990 -0.007755 0.017000 78 105 89 0
-0.009957 -0.007755 0.017000 80 105 89 0
-0.009925 -0.007755 0.017000 79 105 90 0
…
```

Figure 2: A part of an example PLY file

We generate point clouds using the RGB-D information from Microsoft Kinect v2. RGB-D stands for RGB information provided with depth information for each point. Algorithm 1 shows the points clouds generation process based on Kinect input information. Depth information provided by Kinect depth camera is used for the generation of the point clouds. We can clearly see the size differences of the color and depth images of the Kinect in Figure 3. As the resolutions of the color and depth images are different, Microsoft Kinect SDK provides functionalities to project each image to other image's space. As an example we first project the depth map in to the 3D or the camera space to get the real world 3d coordinates of each point, and then the color image is projected to the depth image space to get the relevant color information. Figure 4 shows an example point cloud generated.

Algorithm 1: Point clouds generation from Kinect

```
input: $I_{RGBo}$ // color image from Kinect
       $I_D$ // 13 bit original depth image from Kinect
output: $PC$ // points cloud generated
begin
  // matrix to hold points cloud
  $PC$ = zeros(size($I_{RGB}$,1),size($I_{RGB}$,2),6);
  // mapping color image to depth space
  // function NuiTransformDepthImageToSkeleton
  $I_{RGB} \leftarrow I_{RGBo}$
  // mapping depth image to camera (3d) space
  // NuiImageGetColorPixelCoordinatesFromDepthPixelAtResolution
  $I_D \leftarrow I_{Do}$
  for $r$ = 1:size($I_{RGB}$,1)
    for $c$ = 1:size($I_{RGB}$,2)
      $PC(r,c,1) = I_D(r,c).x/I_D(r,c).w$; // $w$: depth width
      $PC(r,c,2) = I_D(r,c).y/I_D(r,c).w$;
      $PC(r,c,3) = I_D(r,c).z/I_D(r,c).w$;
      $PC(r,c,4) = I_{RGB}(r,c,1)$;
      $PC(r,c,5) = I_{RGB}(r,c,2)$;
      $PC(r,c,6) = I_{RGB}(r,c,3)$;
    end
  end
  return $PC$;
end
```

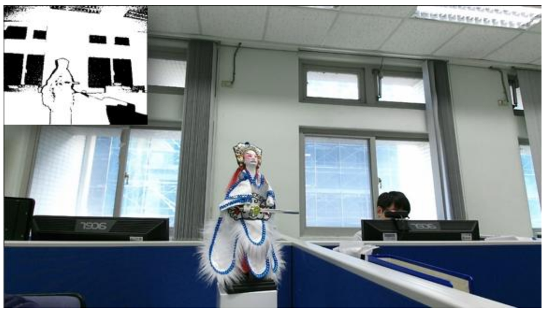
Figure 3: Example color and depth image of the Kinect v2

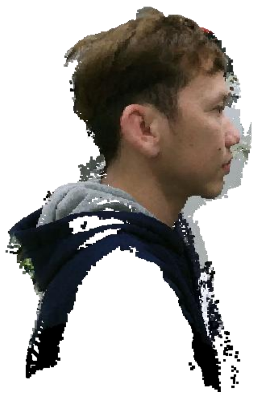

Figure 4: An example point cloud

2.2. ICP

To merge two point clouds of the same object, registration is necessary. ICP (Iterative Closest Point) algorithm is used to register two point clouds into the same coordinate system. ICP finds a transformation matrix for the source cloud to transform to target cloud minimizing the error in similar point pairs. One incremental transformation matrix (T_{inc}) [7] corresponding to an iteration can be expressed as,

$$T_{inc} = [r \mid t] = \begin{bmatrix} 1 & a & -g & t_x \\ -a & 1 & b & t_y \\ g & -b & 1 & t_z \end{bmatrix}, \qquad (01)$$

where, r and $t = ([t_x, t_y, t_z]^T)$ presents the rotation and translation matrices. t_x, t_y and t_z stands for the rotations along each axis. α, β and γ represents yaw, pitch, and roll angles respectively. ICP takes two point clouds as input and output the integrated transformation matrix for the registration. Since the Euclidean Distance calculates the difference of two point clouds, the first step is to find the point-to-point correspondences of these two point clouds. Inevitably there may be outliers, some points that cannot find correspondences from the other point clouds and are discarded. According to [8], there are several strategies to calculate correspondences between two point clouds as, closet points, normal shooting, and projection.

The standard ICP algorithm solves an $L2$-error minimization problem, defined as follows [9]. Let two 3D point sets $\mathbf{X} = \{\mathbf{x}_i\}$, $i = 1, ..., M$ and $\mathbf{Y} = \{\mathbf{y}_j\}$, $j = 1, ..., N$, where $\mathbf{x}_i, \mathbf{y}_j \in \mathbb{R}^3$ are point coordinates, be the data point set and model point set respectively. The aim is to estimate a rigid motion with rotation $\mathbf{R}$ and translation $\boldsymbol{t}$, which minimizes the following $L2$ error E:

$$E(\mathbf{R}, \boldsymbol{t}) = \sum_{i=1}^{M} e_i(\mathbf{R}, \boldsymbol{t})^2 = \sum_{i=1}^{M} \left\| \mathbf{R}\mathbf{x}_i + \boldsymbol{t} - \mathbf{y}_{j*} \right\|^2, \qquad (02)$$

where $e_i(\mathbf{R}, \boldsymbol{t})$ is the per-point residual error for $\mathbf{x}_i$. The point $\mathbf{y}_{j*} \in \mathbf{Y}$ is denoted as the optimal correspondence of $\mathbf{x}_i$, which in the context of ICP is the closest point to the transformed $\mathbf{x}_i$ in $\mathbf{Y}$, i.e.

$$j* = \underset{j \in \{1,\dots,N\}}{\arg\min} \left\| \mathbf{R}\mathbf{x}_i + \mathbf{t} - \mathbf{y}_j \right\|. \qquad (03)$$

Given initial transformation $\mathbf{R}$ and $\boldsymbol{t}$, the ICP algorithm iteratively solves the above minimization via alternating between estimating the transformation in Eq. (01), and finding the closest-point matches by Eq. (02). Due to such iterative nature, ICP can only guarantee the convergence to a local minimum.

3. RESULTS

Original ICP algorithm is modified to include the color information as well in our method. Only the distances to find the corresponding points are used in the original ICP algorithm. But in the our modified ICP algorithm, we consider a pair as a corresponding pair, if the color difference (for each RGB) is less than 20 and the distance is less than a threshold of 0.001. This provides better results as it considers color information as well in finding pairs. As there are many points in the point cloud, we merge or remove some pairs during the alignment based on the intra-pair distance. First, the distance between the points in each pair each calculated (intra-pair distance, d). Figure 5 shows and example histogram of intra-pair distance. For this example, mean value of the intra-pair distance of all the pairs is $d_m = 0.0030345$. If the $d > d_m$, the pair is removed, and if $d \leq d_m$ the pair is merged.

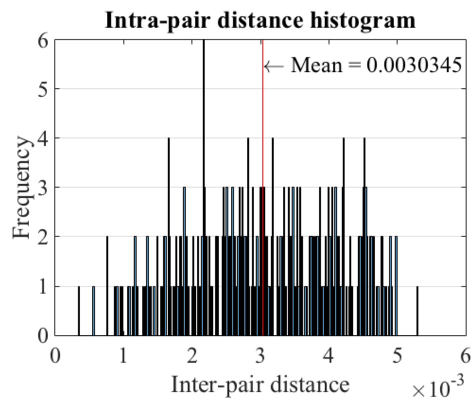

Figure 5: Intra-pair distance histogram

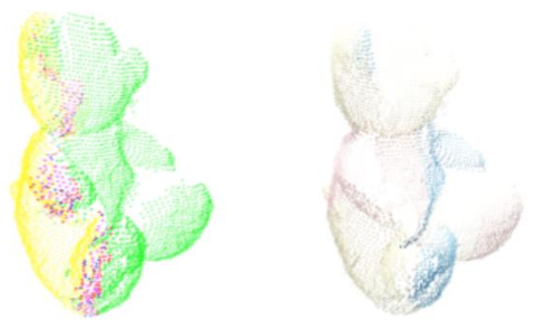

Figure 6: An example ICP alignment of two views captured with Kinect v2 (Left column shows static colors and right column shows the original colors. Source (green), target (yellow), Red: duplicates in source (removed), Blue: duplicates in target (removed), Magenta: merged duplicates).

Figure 6 shows an example alignment series of different point clouds. Alignment of point clouds captured from Kinect 1 and 2 are shown. In the left column point clouds are shown in static colors to differentiate the points of each cloud. Source cloud is in green and target cloud is in yellow. In ICP we iteratively move the source cloud to align with the target. Red colored points shown are corresponding point of the duplicate pairs in source that comply the condition $d > d_m$ and are removed during the alignment. Blue colored points are the corresponding point in the target and also removed. Pairs that satisfy the condition $d \leq d_m$ are shown in magenta and are merged during the alignment. Right column shows the resultant point cloud with original colors.

As explained during the duplicate removal ICP alignment process number of points in the aligned point clouds reduces. There were 3,543 and 3,353 points in the point clouds of Kinect 1 and 2 respectively. So there are 6,896 points in the aligned and merged point cloud if we do not remove the duplicates. But after duplicate processing there are only 6,555 points.

4. CONCLUSION

Point clouds that represent the *x*, *y*, and *z* coordinates of the captured objects are created using multiple RGB-D sensors Microsoft Kinect v2. These multiple point clouds taken at different viewpoints and directions of the same object are then aligned using ICP algorithm. Color difference and the distance threshold are used to find the corresponding points of two point clouds. Mean of the intra-pair distance value is used to merge or remove the correspondences in the resultant point cloud. Future works are planned on noise removal of the point clouds and further improving the alignment.

5. REFERENCES

[1] Kinect for Windows Product Blog, "*Original Kinect for Windows sensor sales to end in 2015*," [Online] Available: https://blogs.msdn.microsoft.com/ kinectforwindows/2014/12/30/original-kinect-for-windows-sensor-sales-to-end-in-2015/. [Accessed: 23-Apr-2016].

[2] Microsoft Store, "*Buy Kinect for Xbox One*," [Online]. Available: http://www.microsoftstore.com/store/msusa/en_US/pdp/Kinect-for-Windows-Developer-Bundle/productID.314513600.[Accessed: 23-Apr-2016].

[3] "*Products*." [Online]. Available: http://velodynelidar.com/products.html. [Accessed: 23-Apr-2016].

[4] Creative Store - Asia, "*Creative Senz3D*," [Online]. Available: http://asia.creative.com/p/ web-cameras/creative-senz3d. [Accessed: 23-Apr-2016].

[5] "*Developing with Kinect*." [Online]. Available: https://developer.microsoft.com/en-us/windows/ kinect/develop. [Accessed: 22-Apr-2016].

[6] Leap Motion, "*Leap Motion*." [Online]. Available: https://www.leapmotion.com/. [Accessed: 23-Apr-2016].

[7] X. Zabulis and K. Daniilidis, "*Multi-camera reconstruction based on surface normal estimation and best viewpoint selection*," in 3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004. Proceedings. 2nd International Symposium on, 2004, pp. 733–740.

[8] D. Gallup, J.-M. Frahm, P. Mordohai, Q. Yang, and M. Pollefeys, "*Real-time plane-sweeping stereo with multiple sweeping directions*," in Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on, 2007, pp. 1–8.

[9] J. Yang, H. Li, and Y. Jia, "*Go-icp: Solving 3d registration efficiently and globally optimally*," in Proceedings of the IEEE International Conference on Computer Vision, 2013, pp. 1457–1464.