Project Report

CS676: Image Processing and Computer Vision

Localization and Mapping of 3D interior space using Depth Camera

By:

Vempati Anurag Sai

Sudhanshu Tamhankar

Guided by:

Prof. Dr. Amitabha Mukherjee

Table of Contents

1. Introduction

We aim to rapidly create a detailed 3D reconstruction of an indoor scene. We move a Microsoft Kinect across the scene and gather the colour and depth data. Each depth image would give more information regarding the scene and helps in the betterment of the surface reconstructed.

This problem would basically include tracking of the camera followed by surface reconstruction. Tracking is accomplished by just using the Depth images to find the necessary transformation between the camera and the global coordinate system. Once the necessary transformations are found all the 3D points are mapped into a common coordinate system. We then construct an isosurface from the dense 3D point cloud. The colour images are then used to map colour to the surface.

Reconstructing geometry using active sensors, passive cameras, online images, or from unordered 3D points are well studied areas of research in computer graphics and vision [Izadi 11]. The problem of Simultaneous Localization and Mapping (SLAM) which aims to track a robot/user while simultaneously mapping the surroundings is extensively studied in the recent years.

Over the last decade, range images have grown in popularity and found increasing applications in fields including medical imaging (PET), object modelling, and robotics. In mobile robotics, the availability of range sensors capable of quickly capturing an entire 3Dscene has drastically improved the state of the art. Even in the challenging problems like autonomous driving, fast-scanning laser range sensors are used instead of cameras to perform obstacle avoidance, motion planning and mapping.

This particular method we discuss offers an advantage over the traditional methods in that they generally involve offline reconstruction techniques. Moreover in techniques like Structure from Motion (SfM) or RGB plus Depth (RGBD) techniques, sparse scene feature detection is used for tracking. Even the holes in the surface formed due to inadequate sensor data can be filled with this particular approach.

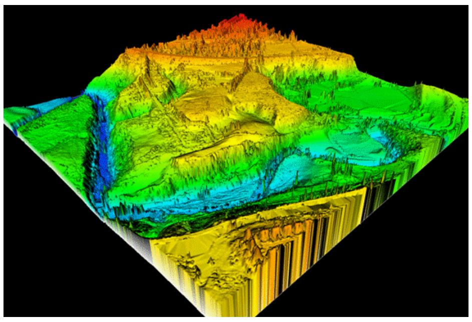
Our problem is actually a subset of KinectFusion [Izadi 11] problem which enables a user holding and moving a standard Kinect camera to rapidly create detailed 3D reconstructions of an indoor scene and even allow interaction with the scene. We further simplify the task by just considering static scene.

2. Applications

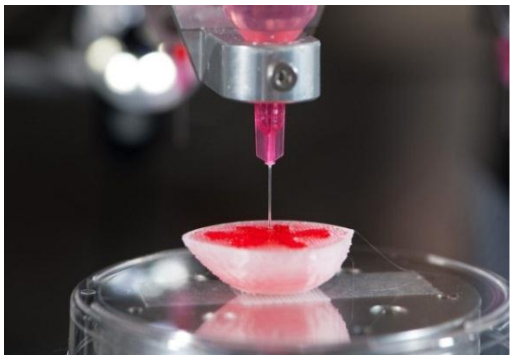
Few important applications of this technique are:

- **3D printing**: A process of making three dimensional solid objects from a digital model. 3D printing is achieved using *additive processes*, where an object is created by laying down successive layers of material

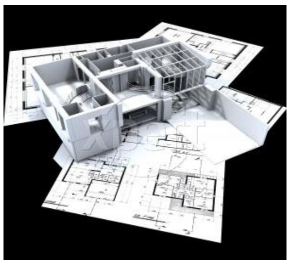
- **3D geological models:** 3D Representations of the Earth's terrain



- **Bio printing:** 3D printing technology was being studied by biotechnology firms and academia for possible use in tissue engineering applications where organs and body parts are built using inkjet techniques. Layers of living cells are deposited onto a gel medium or sugar matrix and slowly built up to form three dimensional structures including vascular systems



- **Software Architectural Models:** The architecture industry uses them to demonstrate proposed buildings and landscapes



3. Camera Tracking

Since various depth readings come from various viewpoints, they all must be brought to a common coordinate system. This requires camera position to be tracked.

3.1 Image to point cloud

Depth images in the dataset [CVPR] are provided as uint-16 (16-bit resolution) images. The depth reading at a particular pixel can found by dividing pixel intensity by 5000 (since, a value of 5000 corresponds to one meter). The conversion from pixel location [u,v] to (x, y, z) in 3D space can be computed as follows:

$$x = (u - c_x) * z / f$$
$$y = (v - c_y) * z / f$$

where,
(c_x, c_y) is the centre of the image (319.5, 239.5 for a 480 x 640 image)
'f' is the focal length of the camera.

3.2 The ICP Algorithm

The Iterative Closest Point (ICP) algorithm was introduced in 1991 by Chen and Medioni and in-dependently by Besl and McKay and it was further developed by various researchers. It's an algorithm to find the transformation between two point clouds. It does so by finding a correspondence between subsets of points from the two point clouds. So, given two point clouds, P_1 and P_2, ICP will compute the translation and rotation matrices, TT and TR, such that

$$P_1 = TT + TR * P_2$$

Many solutions have been proposed to address the issues related to ICP over the years, which has led to several versions of the same algorithm [Rusinkiewicz 01], varying in efficiency or speed. But overall, the six distinct stages present in most of the implementations of the algorithm are as follows:

a) **Selection**
It might be beneficial of only some of the points are considered before applying the ICP algorithm. Techniques like outlier filtering to stabilize the selection, random or uniform sampling to reduce the number of points (and consequently the computation time)

b) **Matching**
Matching is perhaps the most costly stage in ICP in terms of computation. Different techniques can be used to speed up nearest neighbour searching like Delaunay triangulations or kD- trees which provide greatly enhanced look up times at the cost of some increased processing.

c) **Weighting**
The matched point pairs from the previous step are now assigned weights. A matched point pair may be weighed based on their colour value at the pair, distance between them, curvature or tangent normal directions.

d) **Rejecting**
Point pairs can be rejected after matching step. This can be done with a statistical evaluation of the nearest neighbour distances. For eg the 10% worst pairs may be rejected based on some metric.

e) **Error Metric computation**
Error metric defines the objective function that is to be minimized in every iteration of the algorithm. One of the possible metrics can be expressed as

$$E = \sum_{all\ pairs\ pi} (\, p_{i1} - p_{i2}\,)^2$$

where p_i are the matched point pairs.
Of the various metrics, point-point (sum of squared difference between points of P_1 and their corresponding points in P_2) and point-plane (sums the distances of data points of P1 to the tangent planes in which the points of P2 reside) metrics are widely used.

f) **Minimization of Error metric**
Once the error metric is fixed, then methods for minimization methods beneficial for a particular error metric can be considered which may be guaranteed to converge for that particular case. Singular Value Decomposition (SVD) and some other non-linear methods are popular for this step.

Matlab code for ICP has been taken from [ICP].

3.3 Pose estimation

In the tracking phase, a rigid 6DOF transform is computed to closely align the current point cloud with the previous frame, using an implementation of the Iterative Closest point (ICP) algorithm. Relative transforms are incrementally applied to a single transform that defines the global pose of the Kinect.

Terminology:

^{1}P – The coordinates of point P with respect to coordinate system '1'
$^{1}R_{2}$ – The rotation matrix that takes coordinates in system '1' to system '2'
$^{1}T_{2}$ – The translation vector from system '2' to system '1'
O_{1} – Origin of coordinate system '1'

We start with the assumption that the relative pose of the camera at initial time instant is known (i.e. $^{G}\mathbf{R}_{1}$ and $^{G}\mathbf{O}_{1}$ are known). In general the motion capture data specifies the rotation in the form of a quaternion. A simple conversion has to be computed as follows:

Conversion from quaternion to Rotation matrix:

If q = a + **bi** + **cj** + **dk** is a quaternion, the corresponding R matrix would be,

$$\begin{pmatrix} a^2+b^2-c^2-d^2 & 2bc-2ad & 2bd+2ac \\ 2bc+2ad & a^2-b^2+c^2-d^2 & 2cd-2ab \\ 2bd-2ac & 2cd+2ab & a^2-b^2-c^2+d^2 \end{pmatrix}.$$

At ith time instant we have a point cloud $\{P_i\}$. If we consider a particular point (say 'P') on the surface, we have the coordinates of that point as seen with respect to the camera coordinate system at that particular instant which is ^{i}P. The ICP algorithm gives the transformation TR and TT which transform ^{i}P to ^{i+1}P i.e.

$$^{i+1}P = \mathbf{TR} * {}^{i}P + \mathbf{TT}$$

So as per the terminology, **TR** = $^{i+1}\mathbf{R}_{i}$ and **TT** = $^{i+1}\mathbf{O}_{i}$
The pose of the camera at i+1th instant can be evaluated as follows:

$$^{G}\mathbf{R}_{i+1} = {}^{G}\mathbf{R}_{i} * {}^{i}\mathbf{R}_{i+1} = {}^{G}\mathbf{R}_{i} * (\mathbf{TR})^{-1}$$
$$^{G}\mathbf{O}_{i+1} = {}^{G}\mathbf{O}_{i} - {}^{G}\mathbf{R}_{i+1} * {}^{i+1}\mathbf{O}_{i} = {}^{G}\mathbf{O}_{i} - {}^{G}\mathbf{R}_{i+1} * \mathbf{TT}$$

The Rotation matrix can now be converted back to a quaternion.

Conversion from Rotation matrix to quaternion:

$$\begin{pmatrix} q_0^2 \\ q_1^2 \\ q_2^2 \\ q_3^2 \end{pmatrix} = \frac{1}{4}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 \\ -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 \end{pmatrix}\begin{pmatrix} a_{11} \\ a_{22} \\ a_{33} \\ 1 \end{pmatrix}$$

$$\begin{aligned} \operatorname{sgn}(q_0) &= +1 \\ \operatorname{sgn}(q_1) &= \operatorname{sgn}(a_{32} - a_{23}) \\ \operatorname{sgn}(q_2) &= \operatorname{sgn}(a_{13} - a_{31}) \\ \operatorname{sgn}(q_3) &= \operatorname{sgn}(a_{21} - a_{12}) \end{aligned}$$

where a_{ij} is i,j th element of Rotation matrix.

Once we have the necessary transformations, all the point clouds can be transformed back to Global coordinate system to get a dense sampling of the scene.

4. Surface Reconstruction

An isosurface is used to construct surface from the dense point cloud we obtained. Isosurface is the 3D version of contour plots. It tries to connect through all the points with similar value (called isovalue). It can be imagined as a lattice immersed in a sticky liquid which tries to bridge the gaps in between the points. By assigning a fixed isovalue to all the points in the point cloud and allowing the value to decay in various directions, an isosurface can be obtained. The isosurface has a nice property to fill in tiny holes in the surface.

4.1 Colouring the surface

Since the colour images and the depth images obtained from the Kinect are accurately calibrated, there exists a one-to-one correspondence between the colour image and the depth image. Each point in a point cloud is assigned the colour of the pixel corresponding to that point. To remove any kind of dullness present due to limited lighting conditions, the colour ranges of all the three components R, G and B have been scaled to the maximum range [0-255] for an 8-bit image.

4.2 Visualization

To visualize the surface, Mesh Lab software is used. We used MeshLab because it's very simple to use yet sophisticated to handle large data. Some of our surfaces had around 20 million points. This software takes input in many formats like .ply, .obj etc. We have used .ply in our project. Various properties of the isosurface like surface vertices, faces, normals and colour intensities are written into a .ply file as explained ahead:

A .ply file can be easily written in ASCII format. A typical file would look like this [PLY] :

File header

```
Ply                                       { indication that it's a .ply file
format ascii 1.0                          { specify format
comment this is a comment                 { comments
element vertex 8                          { define "vertex" element, 8 of them in file }
property float x                          { vertex contains float "x" coordinate }
property float y                          { y coordinate is also a vertex property }
property float z                          { z coordinate, too }
element face 6                            { there are 6 "face" elements in the file }
property list uchar int vertex_index      {"vertex_indices" is a list of ints }
end_header                                { delimits the end of the header }
0 0 0                                     { start of vertex list }
0 0 1
0 1 1
0 1 0        Vertices
1 0 0
1 0 1
1 1 1
1 1 0
4 0 1 2 3                                 { start of face list }
4 7 6 5 4                                 { number of vertices forming the face followed by
4 0 4 5 1                                   corresponding vertex numbers forming the face}
4 1 5 6 2        Faces
4 2 6 7 3
4 3 7 4 0
```

An element (vertex/face/colour/normal etc.) is mentioned as follows:
element <element name> <number of occurrences in file>

Each element have one or more properties associated with it (xyz coordinates for the vertices/ RGB intensities for colour etc.). They are mentioned as:
property <data type> <property name 1>
property <data type> <property name 2>
…
…

We can add elements like colour, normals etc. to get a better surface in MeshLab. More on the can be found in [PLY].

Since Mesh Lab has compatibility only with the binary_big_endian mode of a .ply file. With the help of some C++ code from an open source library [Libply] to convert between the two formats, the ACII file generated by out code is converted to a binary one, which can be read into Mesh Lab to visualize the results.

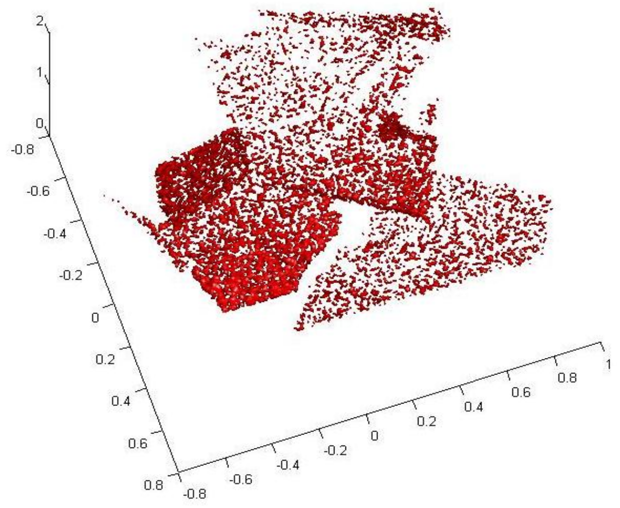
5. Results

Tracking Results:

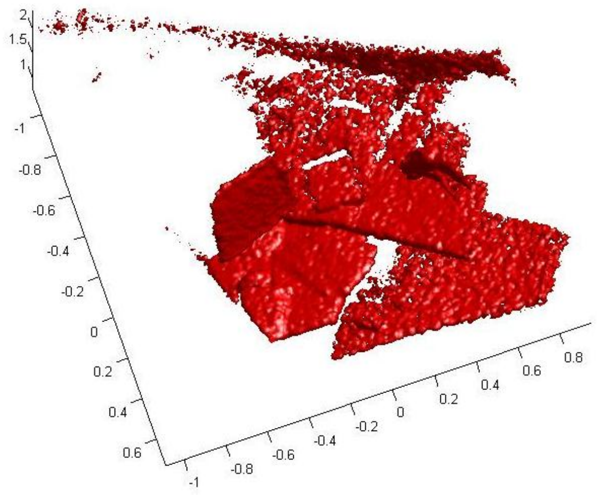
For a sequence of camera motion, (http://www.youtube.com/watch?v=c77Zt7-TZys&feature=youtu.be), the tracking was performed and a visualisation can be viewed here: http://www.youtube.com/watch?v=3hqBgHk6GtE&feature=youtu.be

3D scene reconstruction of a cluttered scene (Office Desk[CVPR], top view):

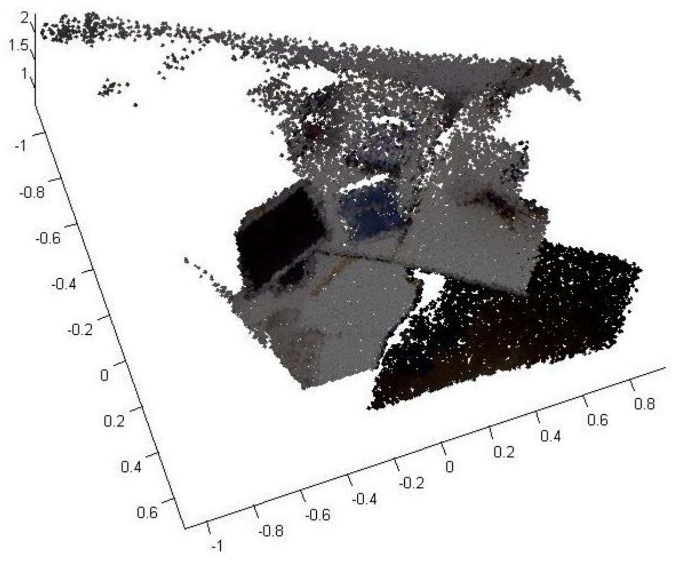
- Point cloud from a single scan:



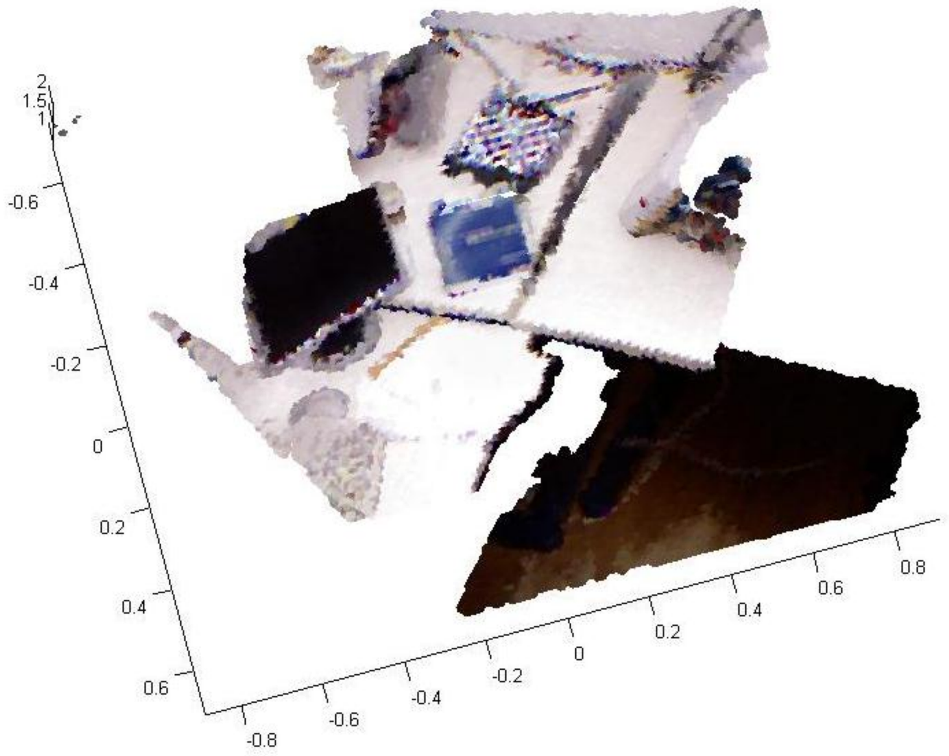
- Dense point cloud obtained after transforming back all the point clouds to ground coordinate system:



- After mapping colour:

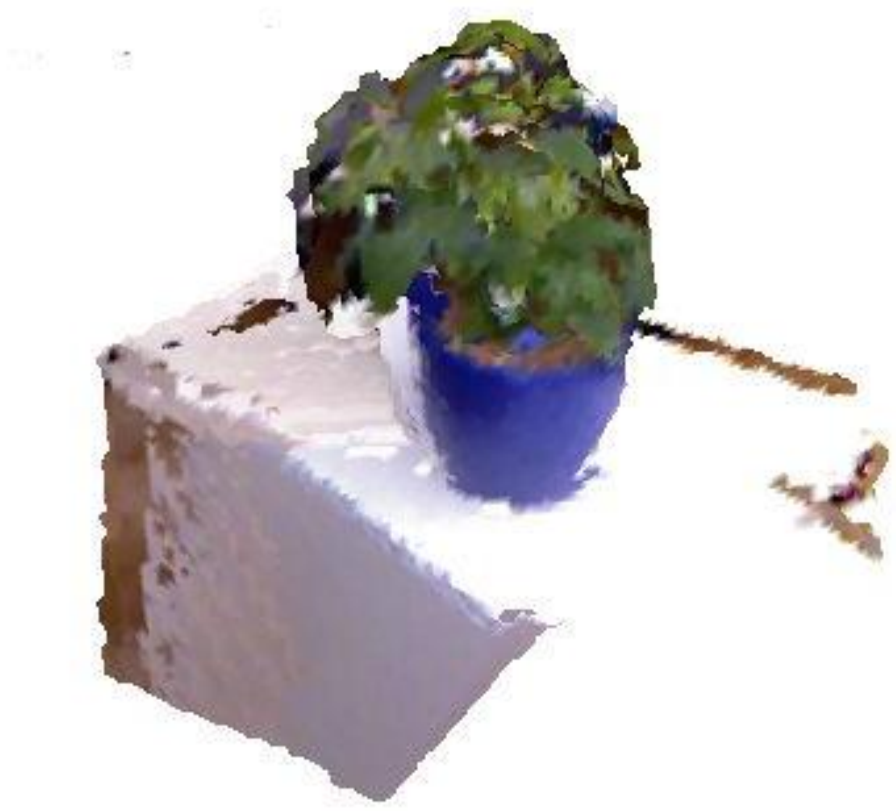


- After scaling the intensities to maximum range:



You can observe the desktop screen, keyboard, mouse and a book though not very clear.

3D reconstruction of an intricate object (Plant [CVPR]):



6. References

1. **Paper : "KinectFusion: Realtime 3D Reconstruction and Interaction Using a Moving Depth Camera".2011**
   Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, Andrew Fitzgibbon.
   [Izadi 11]

2. **Paper : "Efficient Variants of the ICP Algorithm"**
   Szymon Rusinkiewicz and Marc Levoy
   [Rusinkiewicz 01]

3. **Dataset: Computer Vision Group (RGBD SLAM Dataset and Benchmark)**
   Web address : http://cvpr.in.tum.de/data/datasets/rgbd-dataset
   [CVPR]

4. **C++ Library : Libply**
   Web address : http://people.cs.kuleuven.be/~ares.lagae/libply/
   [Libply]

5. **Matlab code for ICP**
   Web Address : http://www.mathworks.in/matlabcentral/fileexchange/27804-iterative-closest-point
   [ICP]

6. **PLY File Format specification**
   http://paulbourke.net/dataformats/ply/
   or
   http://en.wikipedia.org/wiki/PLY_(file_format)
   [PLY]