3-D Object Reconstruction using Hand-Object Interactions

Vaibhav Raheja
Carnegie Mellon University
vaibhavr@andrew.cmu.edu

Rajat Vikram Singh
Carnegie Mellon University
rajats@andrew.cmu.edu

Abstract

One recent research area in today's Vision community has been 3D object reconstruction using RGB-D camera. Such approaches have been widely successful in this domain. However, they suffer from a particular pitfall. These approaches work only for objects that have rich and distinctive geometry or texture features. But, to reconstruct objects without these characteristics is still an open challenge. Existing in-hand scanning systems fail for such symmetric objects in the absence of highly distinct feature. In our project, we use one of the published research[1] in ICCV 2015 that aims to tackle this problem. It uses 3D hand motion for in-hand scanning and builds contacts points from it. Based on it, a 3D model of the featureless object is built.

1. Introduction

The advent of cheap RGB-D cameras has enabled many applications such as 3D object detection, augmented reality, the Internet of things, Human-computer-interaction and the interaction of robots with the real world. Out of these applications we will be focusing on the practice of using RGB-D data to create 3D representation of real world objects. Two ways in which this can be done is by (i). keeping the object stationary and moving the camera around the object and dynamically fusing the information to form a 3D model of the object, (ii). putting the object on a turntable and rotating it while a static camera captures the images and use them to reconstruct the object in 3D. There are many commercial off-the-shelf solutions available to solve this problem such as Fablitec, Skanect1, Shapify, Kinect Fusion which can be used for this application.

Instead of using a turntable, objects can be rotated by hand if we work with a static camera. This is also very convenient for hand-sized objects since it is convenient to setup the experiment. This method is called in-hand scanning. The results for such a method are convincing after multiple improvements to make the process real-time and correcting the loop-closure problem. This method does not use the hand information at all and try to mask the hand information in order to get the object details. However, this method fails for symmetric and relatively featureless objects like toys, mechanical parts or decorative materials etc.

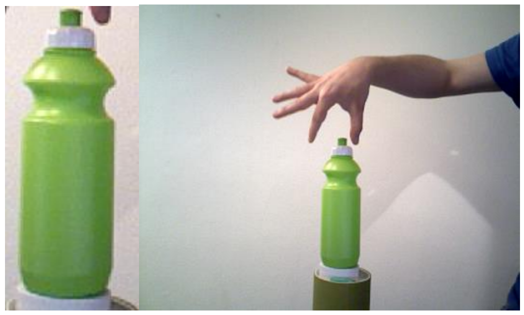
Figure 1. Actual Texture Less Object and Interactions of Hand

In this method, the authors have proposed to use the hand information for in-hand scanning as additional information for reconstructing the texture less objects. Instead of disregarding the hand information, we track the hand between frames and use this information to reconstruct the object.

2. Theory

Various previous papers have described different parts of the problem. We will be discussing some of the approaches that were used by the authors of the original paper and helped us in the understanding of the methods used in the original paper.

2.1. Hand-Motion Capture

One part of the project is to effectively model the hand motion in the frames and use this information for further processing. The authors have used the approach suggested in [3], where a generative-model approach is used to model the hand motion. The method relies on a commercially-scanned 3D model (a mesh defined by vertices and polygons made up of vertices) and manually defining the skeleton structure of the hand into bones, then using the effect of each bone on each vertex to model the shape of the hand. This transformation is given by the following formula:

$$v(\theta)=\Sigma\ \alpha_{v,j}\ T_j(\theta)\ T_j(0)^{-1}\ v(0)$$

where:

$\alpha_{v,j}$ defines the influence of bone j on 3D vertex v,

$T_j(\theta)$ is the transformation of the bone j at a particular pose θ,

$T_j(0)$ and $v(0)$ are the transformation of the bones and the vertices at the starting position,

$v(\theta)$ is the position of the 3D vertices in the given pose.

3. Transformations using Iterative Closest Points Algorithm

The second part of the project focusses on finding the rigid transformations and use it for registering two point-clouds and merging them. From the algorithm, we will get two point-clouds of the object from different frames. The rigid transformation parameters will be calculated using iterative closest point (ICP), using the contact points correspondences in consecutive frames.

The basic steps of ICP are as follows:

i. For each point in the source point cloud, find the closest point in the reference point cloud.
ii. Find the combination of rigid transformations by minimizing a mean squared error objective function which best aligns each source point to the reference point.
iii. Transform the source points and using the generated transformations.
iv. Repeat.

4. Experimental Setup

We are using an RGB video as an input, and we extract 1249 frames from it. In the video, hand interacts with featureless objects by picking them, touching them continuously for some frames, and then leaving contact of the object before repeating this again and again, and rotating the object each time so as to cover its full 2D coverage in the video. Additionally, we are given a file called index.PAIRS that contains the range of frames out of total 1249 frames, where hand is in direct contact with the object. Such contact frames total to 698 out of 1249. These frames are helpful as we will be extracting the contact points computation from only the contact frames. Depth data is given separately for same number of frames. Apart from this, we are using a MESH model to represent 3D hand at each frame. Each mesh contains 10K vertices. .OFF file in the dataset contains the above explained info for each of those 10K vertices. .OFF file is a standard way to represent 3D structures where we have vertices and polygons. Vertices are in the form of 3x1 and they represent three points of the triangle. In addition to this, we have a number of faces that describe the vertex IDs of the faces i.e. the vertices used in building of those faces. An example 3D hand pose in the very first frame is shown in Fig. 2.

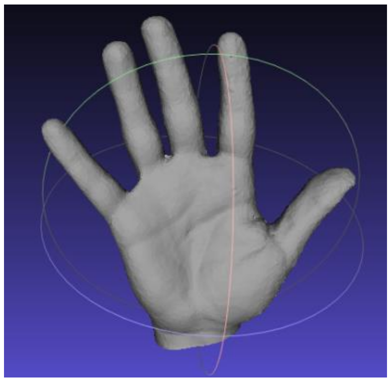

Figure 2. Hand Mesh Model in initial state

Now, we need another set of data to estimate the co-ordinates of those 10K vertices in each frames. In this experiment, a hand is modeled with the help of 20 bones. Each bone has its own weight over each vertex making it a 10K x 20 data. This data is given to us in the form of .SKIN file. Another dataset that lies in the .MOTION file contains the Transformation matrix of all 20 bones for each of these consecutive frames. Transformation Matrix contains 12 elements, and hence, this data set contains 1249 x 12 elements. This data tells us that how each bone moved with respect to itself in each of the frames.

5. Hand Motion Capture

5.1 Hand and Object Extraction

From the RGB-D images, depth values were thresholded to find another RGB-D images that had the relevant noise removed, and finally contained only the object and hand as both of them were at nearly the same depth. Subsequently, we applied Skin Segmentation algorithm to separate the hand data from the object data. Though skin separation algorithm should have worked properly, there were some noise still left. After detecting this hand, it was removed from the object and hand grayscale image, to recover only the object. But the object was recovered with a lot of noise. To remove this noise, we used *bwlabel* function in MATLAB, and the label that carried the maximum mass was retained, and everything else was removed keeping in mind that the object would be a single large entity only. After this operation, object in 2D grayscale image could be recovered perfectly. This is shown in Fig. 3.

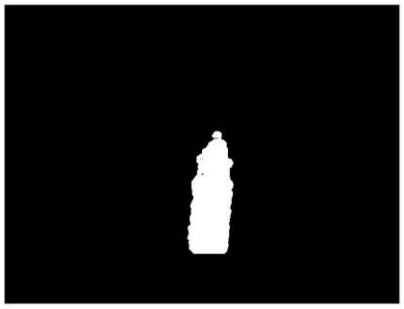

Figure 3. Object Mask Map in 2-D

5.2 Recovery of Object Point Cloud

From this 2D object grayscale image, 3D object was reconstructed using the already available depth data. Hence, now we could reconstruct the 3D object point cloud in PCL format in MATLAB for each frame using the grayscale input frames and the depth data. An object point cloud is shown in Fig. 4.

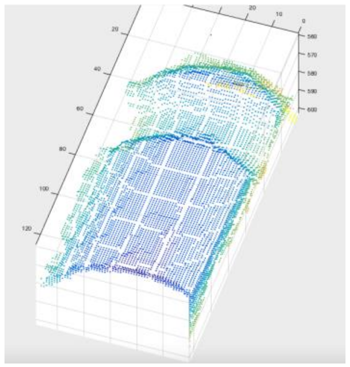

Figure 4. Object Point Cloud

5.3 Creating Hand Point Cloud

With the help of all of the data described above and the using the method of Hand Motion Capture using Linear Skin Blending, we can easily calculate the co-ordinates of each vertex in each frame, after combining the transforming data from bones; weights of each bone on each vertex; and mesh model of the hand. Hence, now we can write a hand mesh model for each of these 1249 frames. With such a mesh model, we can now draw hand pose as shown in Fig. 5. for each of the frames now.

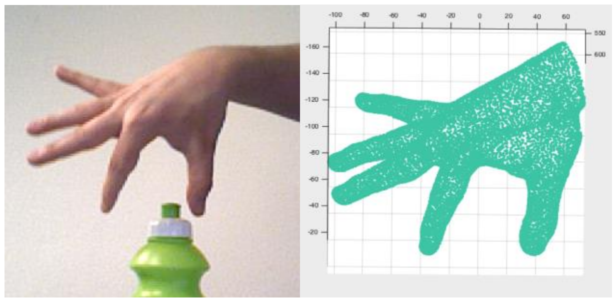

Figure 5. Hand Point Cloud extracted from same frame as shown on the left

6. Object Reconstruction

6.1 Contact Points Computation

Next, we required the contact points between hand and object at each of the frames. To calculate those contact points, first we created a 3D point cloud that contained both the hand point cloud, and the object point cloud at each frame. Furthermore, we calculated the nearest neighbor distance between both of those point cloud, and select the nearest 500 points. But, there is another restriction that we had to impose. Since physically, a touch is possible only with the help of 2 fingers, we back-traced the fingers for all of those 500 points, and finally selected the 2 fingers that gave minimum distances between both of the point clouds, along with the 50 nearest points on each finger. The contact points are shown in red color and can be seen in Fig. 6.

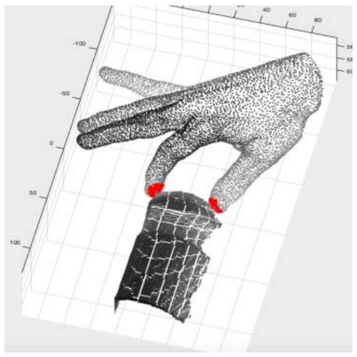

Figure 6. Contact Points Estimation between Hand and Object Points Cloud. Contact Points are labeled in Red.

6.2 Object Reconstruction

Now, with contact points calculated in each frame, we followed those contact points in consecutive frames, and aligned the observed object point cloud for current and previous frames. Afterwards, we used the Iterative Closest Point Algorithm (ICP) to align the current object point cloud with all of the previous frames till that point. To align any two consecutive object point clouds, we used the concept of least minimum square error to finally come up with a transformation matrix that would align them. Finally, aligning every such cloud, we are able to reconstruct the 3-d object.

7. Results, Analysis, and Learnings

As the objective of our project was to reconstruct the 3-d object, various views of the reconstructed object are shown in Fig. 7.

As far as some of the other methods are concerned, the original paper also collects points using the 2-d SIFT key point detection, and the 3-D feature matching. We also tried with 2-D SIFT matching, which was anyway easy using the VLFEAT library, but integrating the same points in object point cloud caused problems for us in MATLAB, and we couldn't integrate them properly given the paucity of the time. So, we proceed with only the contact points extracted from the nearest neighbors of hand and object point clouds. If I had to say something about the most challenging part of the project, it definitely had to be the understanding of the original paper. It took us about 4-5 days before we could even think of starting to implement our own version in MATLAB. Moreover, the paper was not very descriptive of its methods. Translating the paper with its given data was quite difficult. Though, once we had started, it wasn't that tough given the fact that MATLAB has a lot of in-built libraries for point clouds, and those came really handy.

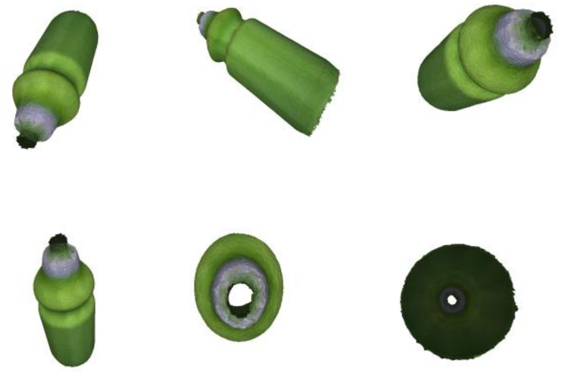
Figure 7. Actual 3D Reconstructions of the texture less object in various views

Additionally, I would say some of the things we tried and failed were trying to incorporate the 3-D features in addition to the contact points. Moreover, it took us a lot of time to estimate the pose of the hand from the given parameters. We were conceptually wrong at one stage, and that ate a lot of our time in implementation. But, finally, we were able to solve it and then, watching the hand pose in each frame was quite satisfying. Another major problem we faced was choosing which contact points for final 3D object reconstruction as in a 3D setup, minimum distance can't be the only imposing conditions for selecting points. Hence, we came up with our approach that we would select 50 contact points, and each of these 50 points should lie on separate fingers, to ensure a physically possible touch.

8. Conclusion

While In-Hand Scanning systems discard information originating from hand, we have implemented the ideas in [1], and have been able to extract the 3-D object cloud based only on the contact correspondences between hand and the object point cloud.

9. Work Distribution

We approached the project with an aim to understand the paper and the problem first. Both of us understood the problem and brainstormed together to find a common understanding. It is no doubt that unless both of us had had regular discussions throughout 4-5 days, understanding this paper would have taken even more. After understanding phase, both of us coded the project almost equally in MATLAB. In terms of the time and effort, it was a group project, and both the team members believe it was an equal effort game that led to understand some of the classic concepts in Computer Vision.

10. References

[1] Tzionas Dmitri, Gall Juergan, "3-D Object Reconstruction from Hand-Object Iteractions", ICCV 2015.

[2] Saran Akansha, Teney Damien, Katani M. Kris., "Hand-Parsing for Fine-Grained Recognition of Human Grasps in Monocular Images", IROS, 2015.

[3] Tzionas Dmitri, et al., "Capturing Hand Motion with an RGB-D sensor fusing a generative model with salient points", Pattern Recognition. Springer International Publishing, 2014. 277-279