6D Pose Estimation using ICP

Sambaran Ghosal, USERNAME : 007, Attempt : Ezra Bridger >>>>>> Luke Skywalker
Department of Electrical and Computer Engineering
UC San Diego
La Jolla, USA
sghosal@ucsd.edu

I. Introduction

An important problem in Robotics and Computer Vision is to determine the pose of objects from a given image. This is useful in many applications such as robotic manipulation where the manipulator needs to know the position of some object to pick up, or in autonomous driving to keep track of where different agents in the scene are and determine a safe trajectory keeping in mind these agents.

In this work, we will implement a method to extract the 6D poses of objects in a given image using Point Clouds and Iterative Closest Point Algorithm.

II. Problem Formulation

Given a set of target point clouds of a given object extracted from training data as $\mathcal{P}train = \{x_i, y_i, z_i\}_{i=1}^{N}$ in the world canonical frame, we want to extract the pose of the same object in a given test set image. Let the test set point clouds of this object be $\mathcal{P}^{test} = \{x_{ti}, y_{ti}, z_{ti}\}_{i=1}^{N_o}$, then objective of the problem is to find $T \in SE(3)$ such that

$$T = \underset{T}{argmin} ||T\mathcal{P}^{train} - \mathcal{P}^{test}||_F^2 \tag{1}$$

This is to be done for all objects in a given test image and for all test images provided to us. In the next section, we will discuss details of how the training point clouds are extracted, and how the ICP is performed.

III. Technical Approach

A. *Extracting point clouds from training data*

Given the training data set, the meta data for each scene contains the objects present in the scene and the corresponding segmented and depth map images. The depth map image is first converted to point clouds using the inverse pinhole camera model as follows

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = K_{cam}^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} Z \tag{2}$$

where Z is the depth value at pixel (u, v). This point clouds is in the camera frame of reference, we need to convert this point cloud to the world canonical frame. We are given the camera extrinsic matrix which is pose of camera wrt world and the object world poses which is the pose of the point cloud in the canonical frame. First to get the point cloud in world frame, we multiply the point cloud with the inverse of the camera extrinsic matrix as follows

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = {}^{world}T_{camera}^{-1} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3}$$

and finally, multiply this with the world pose inverse of the given object which is available to us in the meta data

$$\begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} = {}^{world}T_{pose}^{-1} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{4}$$

This stores the point cloud in the canonical frame in non-rotated non-translated pose which will be easier to use for the ICP registration with the test object point cloud.

These point clouds are stored for each object from each scene in the training data. We also save the object world pose in the different training images.

B. *ICP registration on test set*

Now that we have the partial point clouds of objects from each training image, we visit each test image and for each object in the test image, first we extract the test object point cloud using Eq(2) and Eq(3) which transforms the test object point cloud to the world frame but not in the canonical state. We now iterate through all partial point clouds of this object from the training data that we saved, use the **Open3d** ICP registration function to do the alignment. For the initial transformation guess, we use the world pose of the object in the training data scene that we saved. Then for each training point cloud, the ICP will match it to the target point cloud. We save the pose of the test object as the pose corresponding to the alignment that yields maximum fitness in the Open3D registration.

C. *Hyperparameters*

While experimenting with the validation set, it was observed that most important hyperparameters affecting the performance of the ICP registration were as follows

- Downsampling : It was observed that downsampling both the training and testing partial point clouds affected the performance significantly. On one hand, downsample helps ICP registration avoid local minima by finding non-redundant correspondences between the training and test point clouds and hence positively affects the point cloud. On the other hand, downsampling by a lot negatively affects the performance becasue both point clouds become very sparse and can lead to easy but inaccurate matching. Downsampling also helps reducing the time complexity of the algorithm.

- Threshold : The ICP registration function of Open3D takes in as parameter the threshold. Threshold defines the radius within which the algorithm will look for correspondence. A smaller value means that points closer to each other will be considered as correspondences and if points are further away, there will be no correspondences. If the threshold is large, further away points also are considered for the correspondence. Larger threshold values lead to larger computation time and may also lead to redundant correspondences which negatively affect the performance. Whereas smaller threshold may lead to smaller correspondences, lesser time complexity and avoid local minima, but too less a threshold will again lead to very less correspondences and hence easier but inaccurate matching.
- Initial transformation : The ICP takes in a guess initial transformation to start with. If this guess is good, then we obtain better correspondence between the source and target point clouds which leads to better matching.
- Iterations : ICP max iterations slightly affects the performance. If the initial transformation and correspondences are good, ICP converges very quickly in about 50-100 iterations. For safer limit, we almost kept iterations at around 200.

IV. Results

A. *Validation Set*

We experimented on only 47 images of the validation set because of time constraints, and we varied hyperparameters downsample, threshold. A summary of results obtained using **benchmark.py** for the pose5deg_1cm criteria is shown in Table (I), (II) and (III).

Threshold	Accuracy
0.05	56.17%
0.02	67.23%
0.01	80.85%
0.008	84.26%

TABLE I: Effect of varying threshold for downsample 4

Threshold	Accuracy
0.05	54.47%
0.02	65.63%
0.01	81.28%
0.008	83.40%

TABLE II: Effect of varying threshold for downsample 8

B. *Test Set*

From the above table, we see that we have the highest accuracy using downsample 4 and threshold 0.008.Hence we do the testing using these parameters. As submitted to the internal benchmark (**USERNAME : 007, Attempt : Ezra Bridger** >>>>>> **Luke Skywalker**), we have a test accuracy of 86.67% on the test set and pose5deg_1cm benchmark. Figures(1) - (3) show some succesfull alignments whereas Figures(4) - (5) show some unsuccesfull alignments for the training and testing point clouds.

Threshold	Accuracy
0.05	50.21%
0.02	60.00%
0.01	75.74%
0.008	81.28%

TABLE III: Effect of varying threshold for downsample 16

V. Acknowledgement

I would like to acknowledge Mr Albert Liao for a very productive discussion over last Sunday Dinner where we went through our ideas, exchanged crucial ideas about making testing fast and accurate. Really enjoyed his company. 10/10 the vibe.

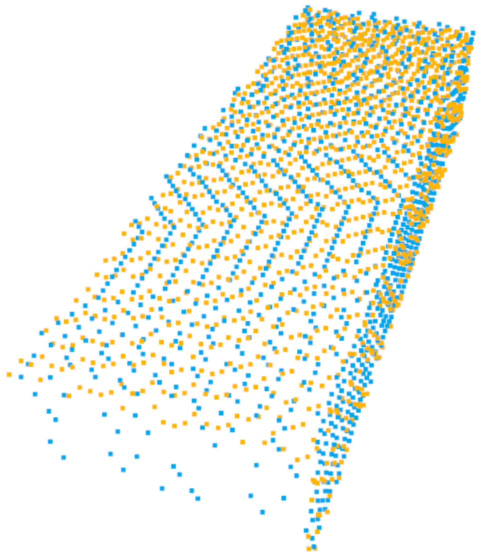

Fig. 1: Success alignment 1

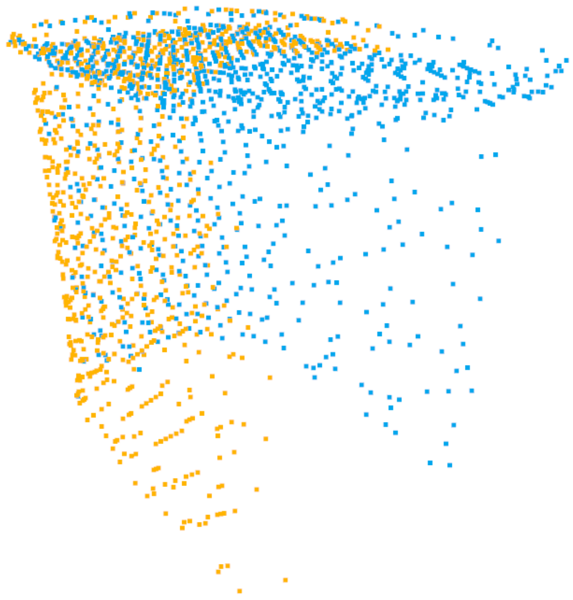

Fig. 3: Success alignment 3

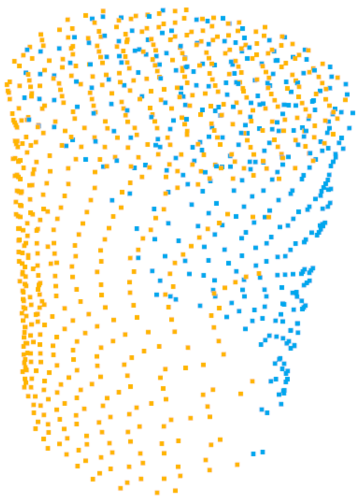

Fig. 2: Success alignment 2

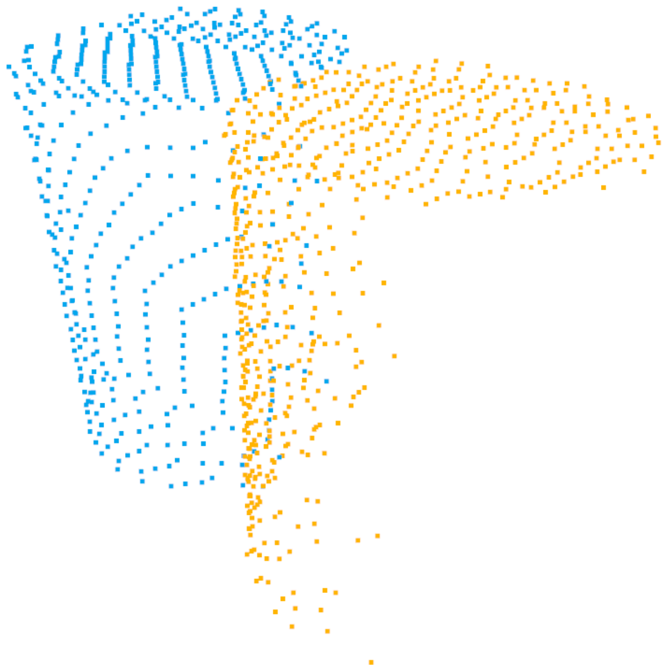

Fig. 4: Unsuccessful alignment 1

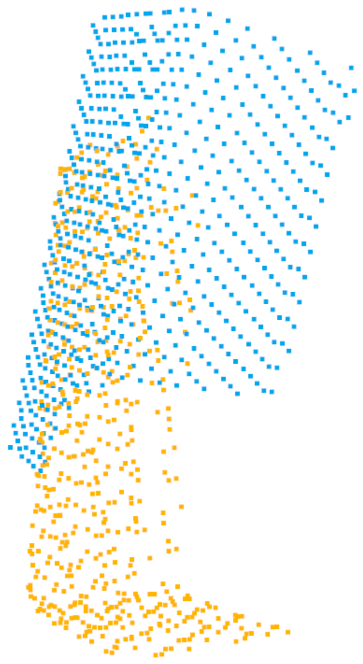

Fig. 5: Unsuccessful alignment 2