3D Object Pose Estimation

May, 2018

Aditya Gupta

Advisor : Prof. Cagdas Onal

Co-Advisor : Selim Ozel

Worcester Polytechnic Institute

Contents

1 Introduction

With the advancement in the field of camera hardware that can process huge amount of complex processes in real-time, the field of robotics have undergone a severe change. Due to this progress of software in machines, many researchers have focused their studies to build smarter and efficient robots. The goal of this project is to improve the sensibility of a prosthetic hand system such that it can 'sense and adapt' based on the object which is in front of it. As of today most of the advanced hand prosthetic are expensive [1] and not very adaptive to environment. This motivates the purpose behind this project, so as to make them more easily accessible and broaden their domain of accessibility. We will use depth camera for the purpose of sensing the data and process the data to know more about the object of interest. Then this information would be used to decide the optimal grasp of the hand. As we can see in Fig [1], is a smart prosthetic hand attached with depth camera (represented with blue box for clear illustrations) from which we obtain the required point cloud data. After acquiring data, the next step consist of segmenting out the object of interest from raw point cloud data, which contains noise and some not-required data. Then next we see two possible approaches to deal with this problem. Approach 1 consist of deep learning techniques which is based on training a network from similar data. Then using the trained network to run inference on. While Approach 2 is more classical, where a similar point cloud as object is given as reference, and similar points in model and point cloud is searched and tracked. Finally, from the information acquired by previous step, the information about the object is extracted and is utilized to decide the correct grasp posture for the hand.

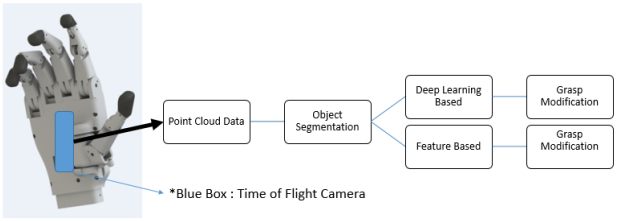

Figure 1: *System Work-flow*

The report is in the following order. Chapter 3 talks about the camera and briefly describes why this particular camera was chosen. Chapter 4 talks about the software used. Chapter 5 talks about the point cloud processing, which basically transform the raw data into the form which can be used by algorithms (deep learning based and feature based to study the data. Chapter 6 and Chapter 7 describes the techniques to extract information about the object, using deep learning and feature based approaches respectively, we will also discuss the experimental results in this section. Finally chapter 8 talks about the conclusion followed by future work.

2 Related Work

Object tracking plays an important role in the filed of robotics. There have been great advancement in this field, earlier approaches were based on calculating the centroid and then tracking it using Kalman Filter, which were robust but were not very consistent in their performance[20]. Other approaches which uses a sample point cloud and align it in real-time such as ICP (Iterative Closest Point), singed distance function (SDF) and their variants [18][19]. Although these approaches are very accurate in real time, but require good initialization and un-cluttered environment, also in case of occlusions they do not perform well [20]. Many variants of ICP has been extended for online pose estimation, one such popular approach is DART (Dense Articulated Real-Time Tracking) framework. To counter the problem of occlusions, Bayesian filter based Particle Filters have been used to estimate the pose of the object. An added advantage with particle filters are, that they can be paralleled, which make them suitable to run on high computational graphical processing units (GPU). They are basically iterative algorithm which updates the estimate of their hypothesis as new data gets available. (Please refer Chapter 7 for detailed explanation). There have been various advancements in the way particle filter is applied for tracking, such as use of Adaptive Particle Filter, which changes number of hypothesis, based on the uncertainty and state space.
From the deep learning point of view, there has not been a significant step towards object tracking (small scale objects, which are graspable by human hand). Mainly due to availability of data. Though there have been huge progress in the field of car and human tracking. Networks such as Faster R-CNN and GOTURN[] are one of the starting deep lean ring networks to solve these type of problems.

3 Hardware

3.1 PMD Picoflexx : Time of Flight Camera

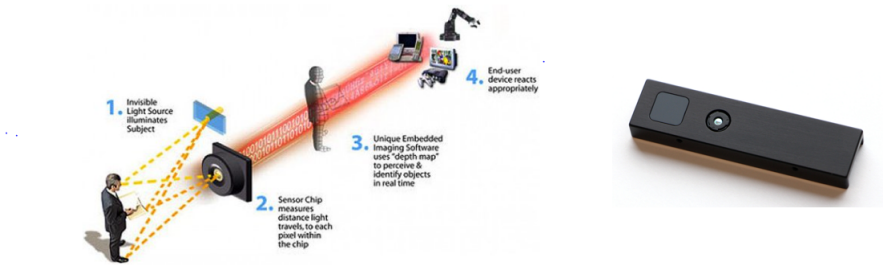

Figure 2: *Depth Perception Work-flow(left). PMD Picolexx (Right)*

Fig 2, shows the camera we will be using for our project, PMD Picoflexx Depth Camera [3]. "A time-of-flight camera (ToF camera) is a range imaging

camera system that resolves distance based on the known speed of light, measuring the time-of-flight of a light signal between the camera and the subject for each point of the image" [2]. In simpler terms, on the basis of time delay of particular light particle to leave and return to camera lens, is the basis to calculate the information to find the location of that particle in the setting under observation.

Next, we will discuss briefly the reason behind choosing this particular camera. To be able to use the depth information of the surrounding with minimum distance possible and small size so as to fit on human size palm, the PMD Picoflexx camera was the most suitable apparatus. There were significant improvement in Picoflexx camera compared to Kinect, which was by-default the first choice for depth cameras. Picoflexx is significantly smaller in size (68mm x 17mm x 7.25 mm) compared to Kinect(33.3cm x 27.4cm x 7.9cm), making it appropriate to fit inside hand. Second, is it's short range cutoff for sending data being 10 cm and for the kinect being 20 cm. These two comparisons are important enough to choose Picoflexx over Kinect. There are other available options like, Intel Depth Camera, Logitech Depth Camera etc, but the size of the Picoflexx camera is most compact making it the most suitable choice for us (at the time of camera discussion, Spring 2017).

There were few dis-advantages while working with the new technology camera, few of them being the high cost and less community support. Kinects software being open source since beginning had established large community base and forums for support. As most of the research related to point cloud was based on Kinect, it took significant amount of time to modify the code and drivers according to the new camera, as its source code is not-open source making community support to minimal. Next we discuss briefly about process to use Picoflexx camera.

3.1.1 ROS Interface of Picoflexx

We will be using Robot Operating System (ROS) to use Picoflex camera. Most of the point cloud related packages were Kinect based for real-time input, based on ROS package called **open-ni**. So it was required to modify the code according to device drivers of Picoflexx camera. Fortunately, research lab at University of Bremen released their code for Picoflexx wrapper for ROS [5], which would be used in the entire project.

3.2 Objects Used

These were the objects used for testing the tracker.

- Object 1, **teddy** is taken as one of the object due to its complex structure, which (as we'll discuss in chapter 7) makes it relatively easier for tracker to find and track. Number of points in reference point cloud : 891
- Object 2, **cup (with handle)** is taken to represent a semi-complex object, and an item which a human actually interact with, in everyday life.

Figure 3: *Object 3 (Left), Object 2 (middle), Object 3 (right)*

Number of point cloud in reference point cloud : 1024

- Object 3, **box** is taken to test the algorithm with simple (hard for algorithm) shaped geometrical objects. Number of points in reference point cloud : 688.

4 Software

4.1 ROS : Robot Operating System

ROS is an open-source, meta-operating system for simultaneously managing multiple process. It is popular in robotics community as it allows rapid integration of complex systems.

As we see in the figure, there are **nodes** which functions as single independent

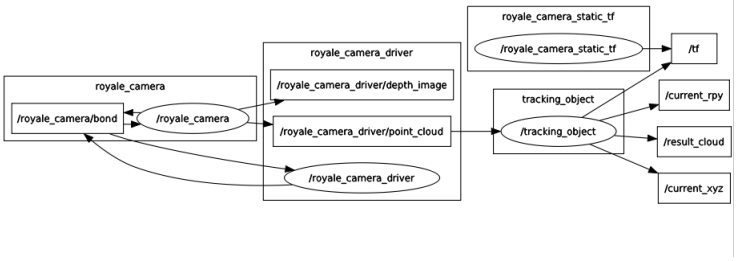

Figure 4: *A running ROS framework displaying nodes for Data Acquisition and Processing.*

component, such as: node for getting point cloud from cameras (royale camera); node for pre-processing the data; node for applying tracking algorithm to that data (tracking object). A node either **publish** data as topic or **subscribe** to data on any other topic. The square boxes in the above figure are nodes, which publishes some data. The lines which connects these boxes represent that way each node subscribe or publish to a particular topic. More more info please see, *www.ros.org*

4.2 PCL : Point Cloud Library

Point Cloud Library [6] is an open source C++ library containing various functionalities to process the point cloud. It has well documented examples to use its API. There is ROS package called *pcl* which let us, import PCL libraries in ROS and use it.

5 Getting Data Ready for Tracking

3D sensors data in the raw form is noisy, therefore data filtering and down-sampling becomes essential steps prior to performing estimation [6].This section will discuss the steps and algorithms to process the raw point cloud data, so that it can be used for further processing.

5.1 Data Acquisition

PMD Picoflexx can be easily connected to computer with USB 2.0 or USB 3.0 connector and does not require any external power source. Basic software to visualize and access data from C++ API is provided by the vendor. Using the ROS wrapper by [5], we can access and visualize the point cloud from Picoflexx in ROS environment(also called RViz). The source code can be accessed by link in [6]

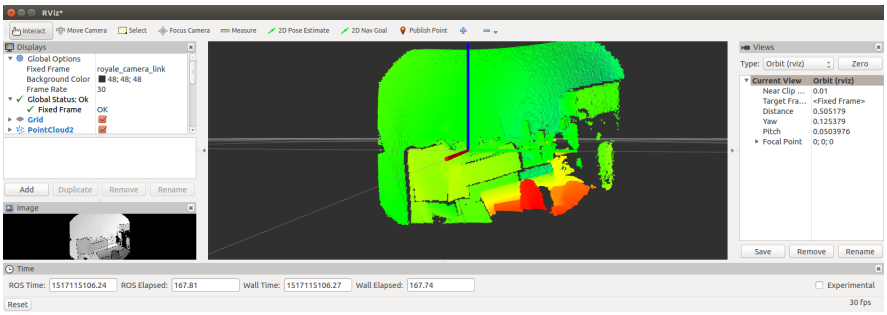

Figure 5: *RViz Screen-shot*

RViz is ROS plugin to visualize the camera output. As we can see in the Fig 3, the RViz plugin can publish the node on which we are calling the point cloud using the C++ API. Camera publish data in different from, grey scale image, RGBD (Red Blue Green Depth) information for each point.

5.2 Down Sampling of Data

The initial point cloud points consist of lot of data (for example : 400000 points) which is more than what is required. Also, this much data would take too much time for processing. A point can be considered a data information acquired

which has the information of RGB value along with depth information. Taking the data as it is to the next step, would be highly expensive and in some cases (depending on the computation power of processor) high lag may persist making it unusable in real time. To overcome this problem we applied Voxelisation technique to down-sample the data. Fig 4, tells us the difference after the point cloud is voxelised, by reducing the points to 40000.

Figure 6: *Point Cloud Raw (left); After Voxelisation (right)*

A Voxel represent value on regular grid in 3D space, and can be visualized as pixel of 3D image. If we imagine voxel as 3D boxes in space over point cloud data, Voxelisation algorithm estimates a single average value for all the points present in that voxel. All the points present will be approximated with their centroid. There may be another approach in which a voxel can be approximated with central value of the that particular voxel, although relatively faster the previous approach giver more accurate results. This is because it biases the point towards a more dense space of points.

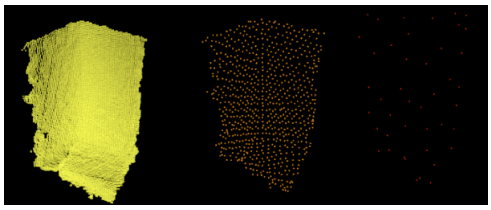

Figure 7: *The left-most (yellow) image represents the original point cloud of milk cartoon with 10000 points; middle (orange) image shows down-sampled image with 1000 points, where voxel size was 0.01 unit. Right-most (red) image shows the down-sampled image with 0.05 unit, with 500 points.*

From Fig 5 we can see that the size of voxel is problem dependent and needs to be chosen after experimentation. Which in our case is 0.01 units for the case of Picoflexx camera. (Where *unit* depends on the camera we are choosing.) Down-sampling adds the advantage of less computational needs, but can lead to loss of required information beyond a point.

We use the PCL version of Voxelisation function, which is highly optimized and compatible with ROS framework. PCL provides **pcl::VoxelGrid** function which can be applied to point cloud in order to down sample the data. [Appendix 1 : showing the code]

5.3 Segmentation

After down-sampling of data, the next step is to filter the data and remove the points which does not come under region of interest. As our final goal is to extract the information for the object, we do not require the point cloud representing tables, room walls or any other surface on which the object resides. We will use technique called RANSAC (Random Sample Consensus) to find out the planar components and delete them.

5.3.1 RANSAC : Random Sample Consensus

The *RANSAC* algorithm proposed by Fishcler and Bolles [9] is general parameter estimation approach. It is an iterative method to estimate parameters of mathematical model from set of inliers, which satisfy properties given as priori. It is assumed that data consist of *inliers*, i.e. whose model can be explained by some mathematical rules. It also assumes that there is a procedure which can decide how optimally does model fits the data. Given the data, RAN-SAC uses a scheme to decide which point will count as inlier and which outlier. [9] summarizes the basic algorithm in the most intuitive manner which is :

1. Model fitted to hypothetical inliers, i.e. all free parameters of model reconstructed from inliers.
2. Remaining data tested against fitted model and, if a point fits well to estimated model, also considered as hypothetical inlier.
3. Estimated model is reasonably good if sufficiently many points have been classified as hypothetical inliers.
4. Model is re-estimated from all hypothetical inliers, as it has only been estimated from the initial set of hypothetical inliers.
5. Model is evaluated by estimating the error of the inliers relative to the model.

The procedure is repeated fixed number of points, each time producing different set of inliers. And if the present iteration exceeds the previous, the model is saved and the process goes on. The advantage of RANSAC lies in the fact that, it works efficiently even when high noise is present. But, we cannot guarantee a solution, which is the trade-off we get.

5.3.2 Plane Segmentation using RANSAC

For a planar segment, minimum of three points (in the point cloud) is required, which constitutes a minimal set. To evaluate the model, the deviation of planes normal is calculated from that of each points normal. And accepted only if the deviation is less than a particular *predefined* angle α. Complete analysis of this topic is beyond the scope of this report please refer [8] for further reading. Talking from the implementation point of view, PCL provides

pcl::SACMODEL_PLANE function which can be applied to point cloud to extract the planar segments. Subsequently, the detected segments can be removed and the remaining point cloud would consist of points of interest.

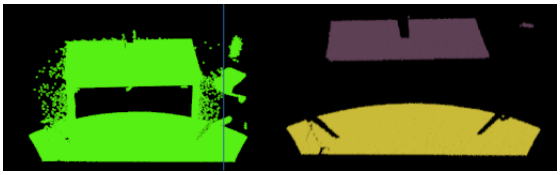

Figure 8: *Initial Point Cloud (left), detected planar segments (right), which can be deleted from the point cloud.*

Please note that, figures 4,5 and 6 are developed from sample point cloud file (.pcd format) and not in real time. This is for better visualization and understanding of the reader. At this stage we have a processed point cloud, having minimal data consisting of object of interests. The next two chapter discusses on the two approaches taken to get the relevant information out of that remaining points. Please note that Chapter 6 is still under progress and therefore does not discuss the result section completely.

6 Approach 1 : Deep Learning Based

In this segment, we will go through our first approach which is based on deep learning. The next three section will go like this: 1) Overview, talks about the details of the proposed solution; 2) Results, talks about the actual outcomes that were achieved and also which were not, finally ; 3)Discussion, talks about findings of the approach.

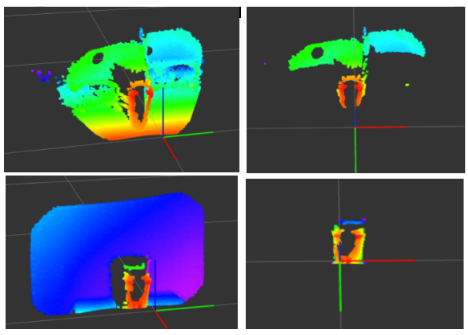

Figure 9: *Comparison for planar segmentation in cluttered and uncluttered environment. Left section shows raw point cloud, right column images show segmented point cloud after Down sampling and Planar Segmentation. As we can observe cluttered environment has sparse point cloud apart from the required point cloud of object.*
Video : `https://youtu.be/-TN0paktmRQ?t=7s`

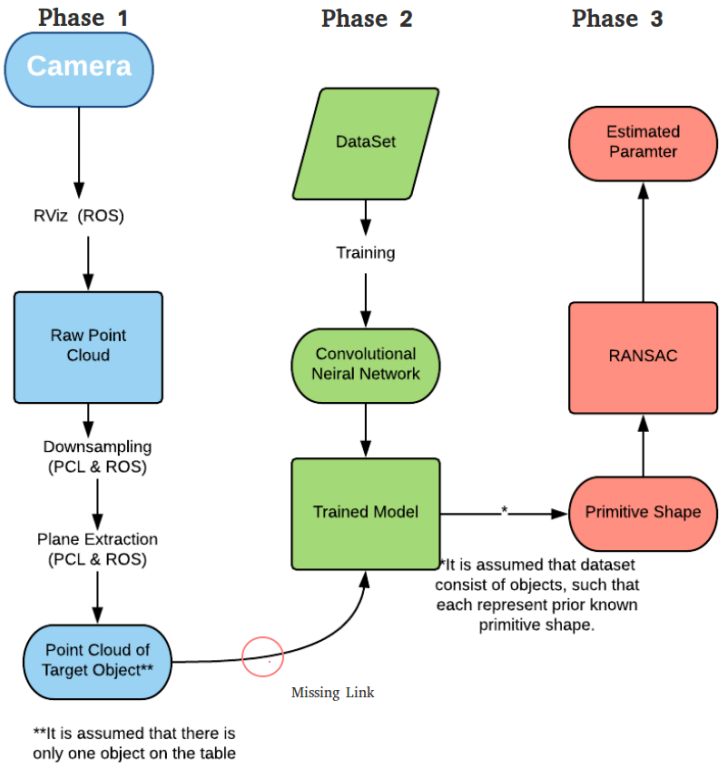

Figure 10: *Flow chart for object parameter estimation using Deep Learning Approach*

6.1 Overview

After obtaining the point cloud from camera, the next step would be to down-sample the data, followed by planar segmentation. Fig 7 shows the result for after these two step. Once we have the point cloud of target object, we pass it on to **trained** Convolutional Neural Network (CNN). As we will see later, the CNN model takes single point cloud as input, which means that if the input point cloud has noise, the results will be in-consistent. Therefore, for simplicity we only consider uncluttered environment in this case. We have discussed the blue segment of Fig 8 in Chapter 4, now next section will talk in detail about the dataset used, training and testing of neural network (green segment in Fig 7).

6.1.1 CNN : Data, Model, Training, Testing

We will now discuss the steps involved from data pre-processing, followed by training and finally testing the model.

1. **Data** : The first step when using neural network is to have sufficient and consistent data. For the purpose of this project we required labelled data with primitive shapes (cube, sphere and cylinder). Due to un-availability of data with labels as primitive shape, we used data provided by ShapeNet community [12]. ShapeNet Data-set provide data for 40 classes ranging from airplane to radio. For the purpose of this project we selected 8 classes : bottle, bowl, cone, cup, flower_pot, lamp, stool and vase (It was assumed that all bottle and cups are cylindrical, rest of the classes were taken to avoid over-fitting). The data comes with separate train and test folder having different samples for each class. For consistency, we randomly chose 1000 samples from training folder and 333 samples from test folder for each class (each file is in **.mat** format).
   We will use code provided by VoxNet [13], for the purpose of data preprocessing. Which converts the raw point cloud in **.pcd** form to required format of **numpyarray**. The code was modified such that it can we used with Keras framework instead of Tensorflow for which it was originally written. (Keras and Tensorflow are python framework for neural networks). This modification was done due to the fact that, Keras provide easier interface to implement neural networks making the process faster. Along with the format conversion, data goes through scale formatting and down-sampling. For the purpose of down-sampling authors in [13] use occupancy grid method, which is voxelisation of point cloud based on weather a particular voxel is occupied or not. A probabilistic function determines the this based on sensor data and previous values. Data-prepossessing plays a crucial role as each sample from every class should be scaled and set free from noises, this is really important at the time of training so as to make unbiased model. Please refer [13] for further reading.
   The idea was, once we have a working system with these 8 labels, we will assume that all bottles and cups are cylindrical, and ignore other primitive shapes in the starting (adding them later). While other classes were present to test the accuracy of the system, without making it bias. Although this is not the correct way to test the model, as the labels are inconsistent with each other. This shortcoming is discussed later in next section.

2. **Model** : We used model provided by [13] called Voxnet for our problem statement, as it was created for similar data (3D point cloud). Before discussing the network, we will briefly talk about the layers and software used.
   Each layer is written in the form *Name(hyperparamter)*.
   Layer 1: *Input Layer*: Fixed input size of $A \times B \times C$,was taken with $A = B = C = 30$. Every input needs to be of same size in input layer, which makes the process of re-scaling important when dealing with data of different type of objects. For example, cup and bottle will have significantly different data points in their raw point cloud, which needs to be

scaled to appropriate size for input layer.
Layer 2: *Convolutional Layer C(f, d, s)* : This has four input, three being related to spatial information and fourth related to feature maps. In total, f number of informed features of the dimension $d \times d \times d \times f$, d being the spatial dimensions and f being feature maps number, s being number of filters. Out of various non-linearities like sigmoid, tanH and ReLU, ReLU was chosen as it overcomes the problem of vanishing gradient problem, and shows better performance with high spatial data [14].
Layer 3: *Pooling Layer P(m)* : To add the linearity to the model, pooling layers were added. This layer also down-samples the data by factor m along the spatial dimension using max pooling type of down-sampling. Which means, taking the maximum value in $m \times m \times m$, d. Out different pooling like, Average pooling, Max pooling and L2-norm, max-pooling was showed the best results [13].
Layer 4 : *Fully Connected Layer F C(n)* : Fully connected layers is added to fuse the local information learned by previous layers and induce the model such that it can learn globally from the input using the previously learned information. From each neuron linear combination of all outputs is learned, Re-LU is used to save for final output layer. The number of output from this layers corresponds to number of classes we used for training, with finally soft-max being used to give probabilistic output for each class.
Architecture : Complete model can be summarized in the form $C(32,5,2)C(32,3,1)P(2)FC(128)FC(K)$, where K is number of classes in the data set.

3. **Training Model** : Training was done for 10 epochs, each one taking approximately 2210 seconds. We used $IntelCorei7-6700HQCPU@2.60GHz8$ with 16 GB Random Access Memory. The final accuracy of the model on testing set was 94.892%. The data was divided into test, train and validation set using Keras. Training results can be seen in fig 9.

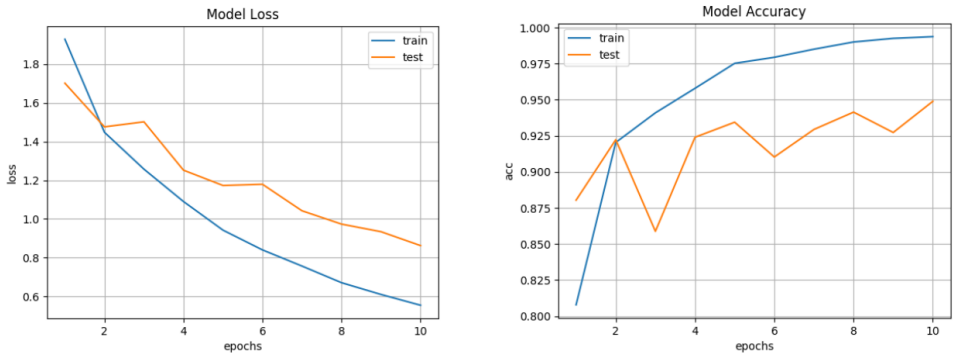

Figure 11: *Training Plots*

4. **Testing Model** : To test the model, ideally, point cloud data from Picoflexx should be taken into account. But the link which let Picoflexx camera data to be sent to CNN model as input was not completed by us. For initial test purpose saved point cloud was given to the model to verify the model. Detailed discussion about the unfinished link in the approach is done in next section. The results for saved point cloud using trained model can be re-created with the code provided. `https://www.dropbox.com/s/br04nf2zbf0km96/code.zip?dl=0`.

5. **Software** : We used *Keras* framework [15] in Python to implement the above discussed framework. Due to easy to use API and strong community support, it was convenient to re-implement the network.

6.1.2 Finding Primitive Shape from Trained Network

According to Fig 8, (assuming we have working interface for sending input from Picoflexx camera to CNN model), the next step was to use RANSAC to find the parameters of the object. According to the plan in Fig 8, once we have the object detected from CNN model, we will also have its primitive shape (as each object would have represented a particular primitive shape). This would have enabled RANSAC to extract out the parameters associated with that primitive shape. For the case of cylinder it would be, (X, Y, Z) coordinate of a point located on the cylinder axis, (X, Y, Z) coordinate of the cylinder's axis direction and cylinder's radius [16].

6.2 Results and Discussion

This section talks about the findings and current status of this approach.

1. The whole approach can be divided into three phases, where Phase 1 was successfully completed and discussed in Chapter 4. The link between Phase 1 and Phase 2, (which means sending the segmented point cloud from Picoflexx camera to trained model) was not completed. This is because, Phase 1 was using ROS as its framework, while Phase 2 was running in non-ROS environment using Keras. One possible workaround would have been to have Keras package in ROS, which was not available at the time of working on this approach. Another approach would have been to use PCL library without ROS framework, and then write a plugin to transfer the processed point cloud from PCL (in C++) to Keras Network (in Python) for which I could not find solution in given time. For Phase 3 it was necessary to complete this link, hence was not achieved.

2. The way to approach the problem was not correct and was based on many assumptions. First of all, un-availability of labelled data (for our case) should have been considered before proceeding further with deep learning. Critically thinking from the implementation point of view before starting the project could have led to different approach and saved us time. For

example, if since starting Camera was used via non-ROS environment, the link between phase 1 and phase 2 could have been bridged more quickly.

The code including READ-ME file can be accessed from the link `https://www.dropbox.com/s/br04nf2zbf0km96/code.zip?dl=0`.

7 Approach 2 : Particle Filter Based Tracking

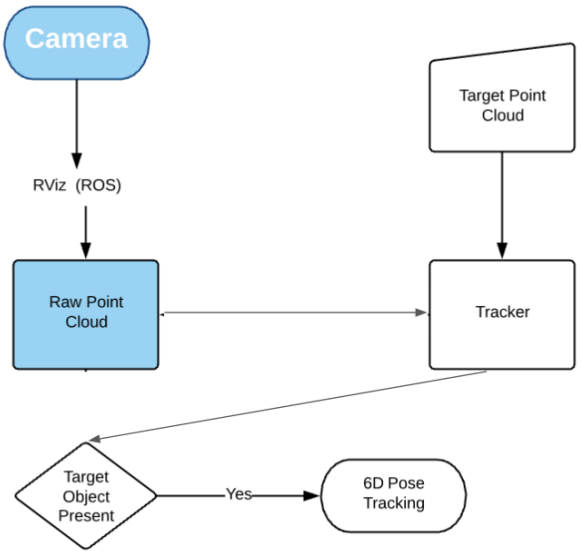

Figure 12: *Approach 2*

7.1 Overview

In this approach[17], we will use Adaptive Particle Filter for 3D Object Tracking using PCL and ROS framework in C++. The whole process is briefly described below.

1. Point Cloud is acquired into ROS as seen in Fig 10 (blue box).
2. To track a particular object, its **.pcd** file needs to be given to particle filter tracker (discussed below) at the execution time of the program.
3. The tracker, which is part of PCL class **pcl::tracking** then predicts a similar object in the point cloud received by the camera.

The approach can be extended for multiple ob jets simultaneously. The system successfully tracks three objects simultaneously in real time. This is a

model based approach, which means we have to provide the point cloud for the object that needs to be tracked. The objects which were used has been described in section 3.2. To test the accuracy of results we change the orientation of object in 6 possible ways: X-axis, Y-axis, Z-axis, rotation along Roll, rotation along Pitch and rotation along Yaw. Further details of the algorithm and experiments are given below.

7.1.1 Adaptive Particle Filter

Particle filter works on the principle of Monte-Carlo algorithms which is used to compute posterior probability of a Markov process. In simpler terms it predicts the next approximation, combining the information from previous prediction and current sensor input. To understand the requirement of the adaptive particle filter, lets first discuss simple particle filter[22]. As discussed above, particle filter estimates the state of system from previous estimate and current sensor input. Let the state to be estimated be x, sensor input be z_t, control input be u_t and belief be $Bel(x_t)$. Particle filter computes the posterior using a set Set_t on n weighted samples distributed based on $Bel(x_t)$:

$$S_t = < x_t^i, w_t^i > | i = 1, ..., n$$

Where x_t and w_t are samples and weights (normalized, i.e. sums up to one) respectively. Then using the procedure SISR (Sequential Importance Sampling with Re-Sampling) it updates the belief of the state estimated. SISR consist of these steps: **Re-Sampling**: Drawing random sample from state space based on weights; **Sampling**: Use control information and previous estimate to calculate *proposal distribution* given by:

$$p(x_t|x_{t-1}, u_{t-1})Bel(x_{t-1})$$

which will then be used in next step. **Importance Sampling**: Weight the sample x_t^j from importance weight and their likelihood. The step which involves updating of weights is linear in time with respect to number of weights that are being updated. Therefore sample size plays crucial role from computational cost point of view. Therefore, adaptive particle filter adapts the number of samples based on the uncertainty. For example, in the beginning there is high un-certainty and therefore more samples are needed, while after some time when tracker finds the object, we need less samples. That is where adaptive particle filter out-performs simple particle filter.

7.2 Experimentation

To see how accurate tracker performs, we moved the object in 6 possible ways (x,y,z,roll,pitch,yaw). For rotation the following setup was created : Arduino Uno was attached to a servo to rotate it in the desired way. for different type of rotations, the setup was fixed accordingly.
For **Roll, Pitch and Yaw** rotations, the object were rotated in the following

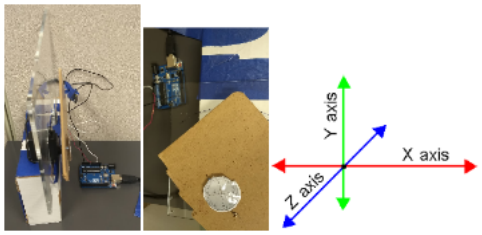

Figure 13: *Setup for Rotation(left, middle); Axis Alignment (right), Y-axis : Vertical Movement w.r.t Table, X-axis : Vertical Movement w.r.t Camera, Z-axis : Horizontal Movement w.r.t Camera*

manner: Phase 1: 0 to 45 degrees; Phase 2: 45 to -45 degrees; Phase 3: -45 to 45 degrees; Phase 4: 45 to 0 degrees. All these phases were completed in 20 seconds period (uniformly).
For **X and Z** linear motion, the object was displaced manually in the following manner: Phase 1: (Reference) to (Reference - 25 cm); Phase 2: (Reference - 25 cm) to (Reference + 25 cm); Phase 3: (Reference +25 cm) to (Reference). All these motions were completed in 40 secs (uniformly).
For **Y** linear motion, the object was displaced manually in the following manner: Phase 1: (Reference) to (Reference - 25 cm); Phase 2: (Reference - 25 cm) to (Reference). All these motions were completed in 10 secs (uniformly).

7.3 Results and Discussion

Simultaneous tracking of three objects was achieved in real time. Also calculate the exact accuracy of the system, three objects were rotated and moved as described in the previous section. Fig 19-50 (after Acknowledgement) shows graph to results for Object 1, 2 and 3 respectively. Each Graph shows the raw tracked value, with refined value after passing it through Low Pass Filter and the Reference Value which it should ideally follow. For roll, pitch and yaw graphs Y axis is in degree and X axis is in time (seconds). For x, y and z graphs Y axis is in distance (meters) and X axis is in time (seconds). There is Root Mean Square Error printed for each case.
As we can see from graphs, tracking worked fine for the case of roll and yaw rotations in all the three cases. It was better in the case of teddy because the structure of point cloud for Teddy is complex, it hence was able to track it better, when compared to simpler point clouds of box and cup where it gets confused. In the case of pitch rotation, tracking failed because, tracker could not differentiate the way point cloud move, as it is not captured by the camera. So if it rotates backward, there is some part of the object which is not visible to the camera, and tracking is now influenced by points which are visible. But due the way arrangement is set up, the pitch movement did not get recognized by the camera. For the case of linear motion, teddy showed better performance in the case of Y-axis and Z-axis movement. Although cup showed better performance in X-axis movement.

It was noted that, that system takes few seconds in the starting to find the the target object (less than 5 seconds). Also, if sufficient points (more than half of the sample point cloud)in the point cloud are visible, the tracking does not get lost even in an occluded environment. The code and procedure for the experiments can be found at :`https://github.com/adityag6994/object_tracking_particle_filter`

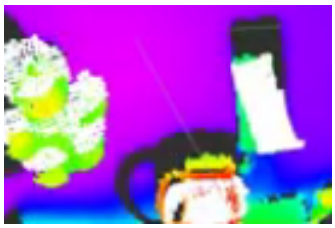

Figure 14: *Screen shot from RViz showing three object being tracked simultaneously (White point cloud is the reference point cloud given to the system, and colored point cloud is the received point cloud coming from camera. Video Link to YouTube :* ***https://www.youtube.com/watch?v=xQwUsMWX4yg***

8 Conclusion and Future Work

The goal of this project is to provide object information using point cloud data such that it can be used to calculate optimal grasp for the object. The first approach which was based on deep learning was partially completed. It was found that a more careful evaluation before proceeding with this approach might have saved some time and could have given more tangible results. This approach would have been more useful if correctly labelled data-set was **already** available. The second approach successfully tracks multiple objects simultaneously in real time. We show with the help of experiments that, a minimum of half of the objects point cloud are required to be successfully matched to the reference point cloud.
With the given fact that particle filters can be paralleled, the next step can to optimize the algorithm such that it is GPU compatible.

9 Acknowledgement

I would like to express my deepest appreciation to **Selim Ozel** and **Prof. Cagdas Onal** for their guidance and support. Also, Soft Robotics Lab, WPI to provide funds to buy the camera.

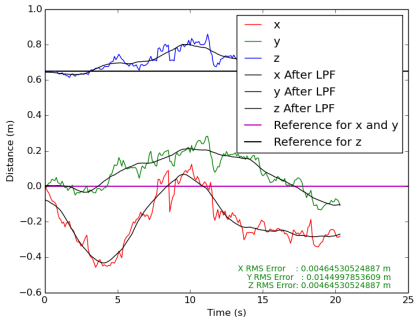

Figure 15: *Object 1: Roll Change in XYZ*

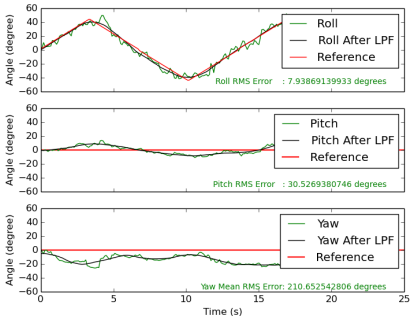

Figure 16: *Object 1: Roll Change in RPY*

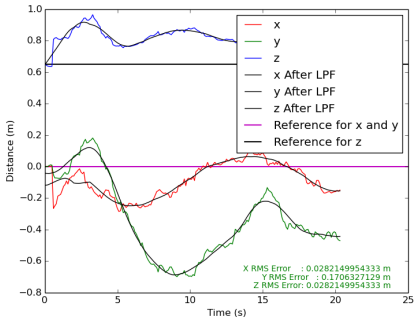

Figure 17: *Object 1: Pitch Change in XYZ*

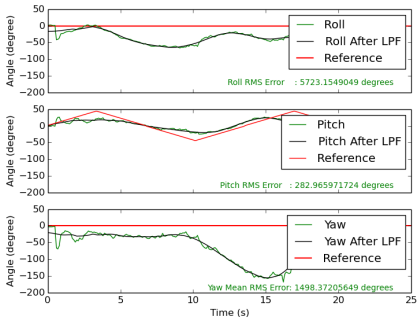

Figure 18: *Object 1: Pitch Change in RPY*

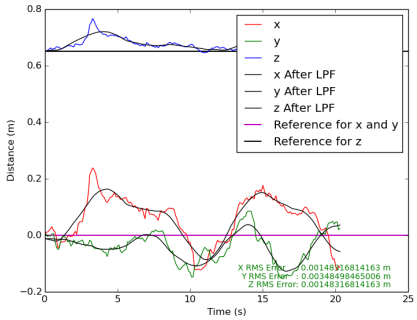

Figure 19: *Object 1: Yaw Change in XYZ*

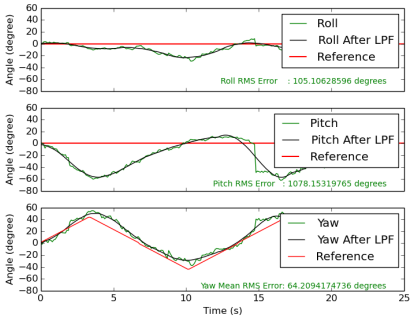

Figure 20: *Object 1: Yaw Change in RPY*

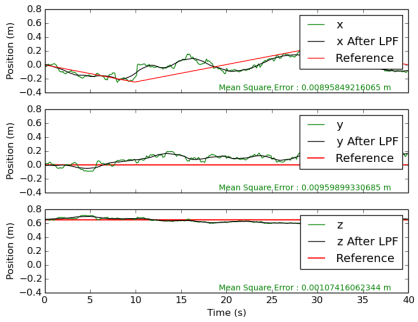

Figure 21: *Object 1: X-axis Change in XYZ*

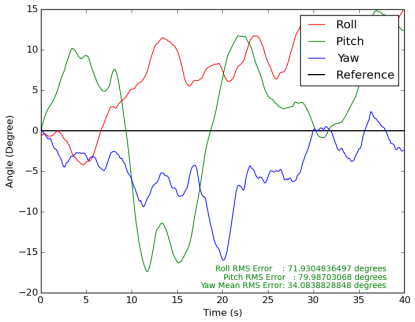

Figure 22: *Object 1: X-axis Change in RPY*

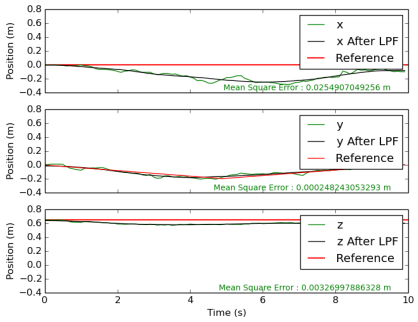

Figure 23: *Object 1: Y-axis Change in XYZ*

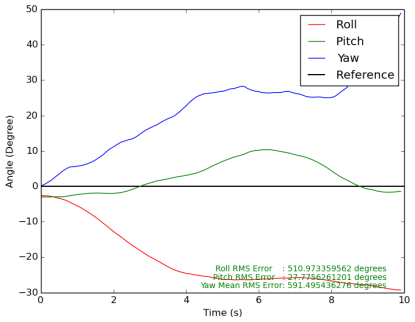

Figure 24: *Object 1: Y-axis Change in RPY*

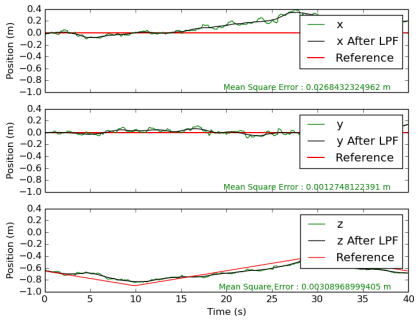

Figure 25: *Object 1: Z-axis Change in XYZ*

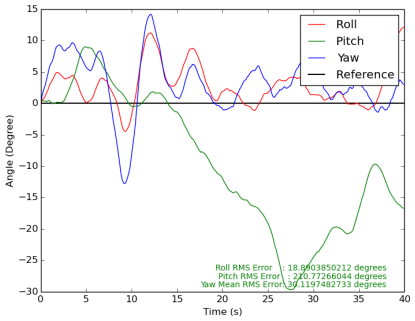

Figure 26: *Object 1: Z-axis Change in RPY*

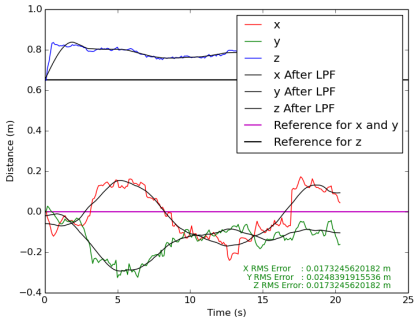

Figure 27: *Object 2: Roll Change in XYZ*

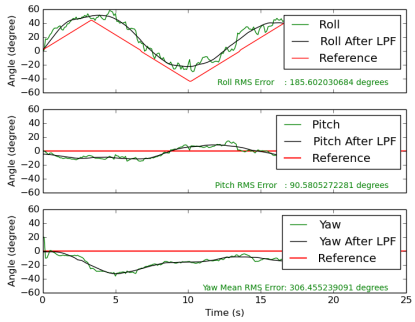

Figure 28: *Object 2: Roll Change in RPY*

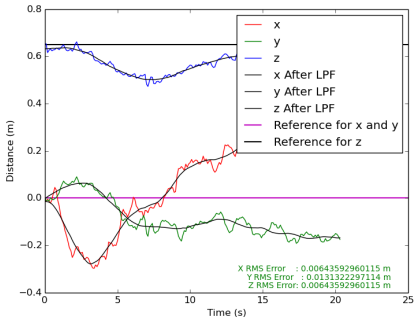

Figure 29: *Object 2: Pitch Change in XYZ*

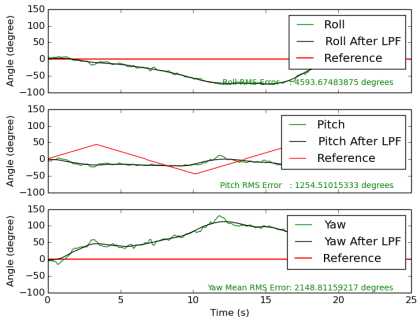

Figure 30: *Object 2: Pitch Change in RPY*

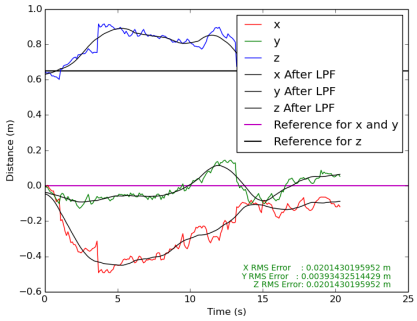

Figure 31: *Object 2: Yaw Change in XYZ*

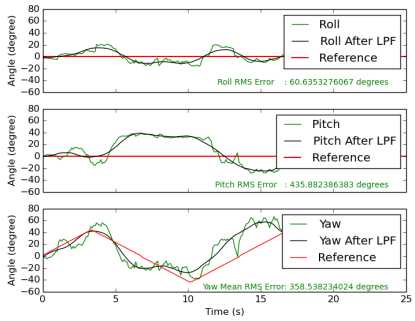

Figure 32: *Object 2: Yaw Change in RPY*

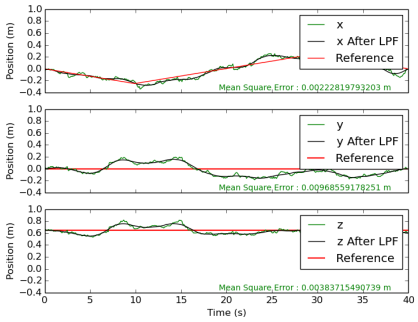

Figure 33: *Object 2: X-axis Change in XYZ*

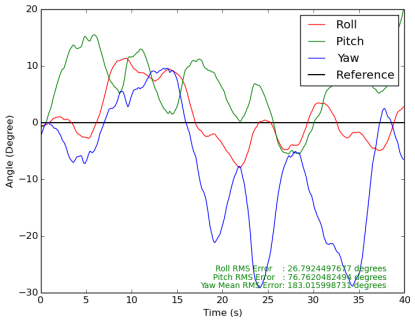

Figure 34: *Object 2: X-axis Change in RPY*

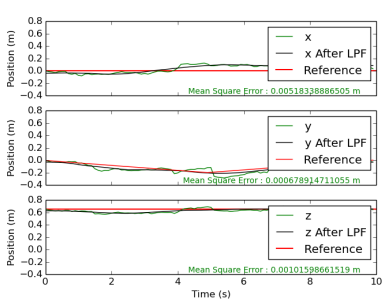

Figure 35: *Object 2: Y-axis Change in XYZ*

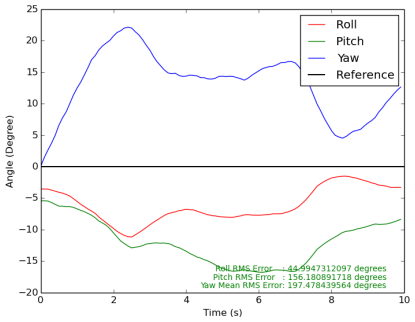

Figure 36: *Object 2: Y-axis Change in RPY*

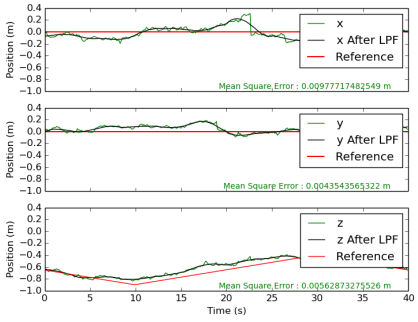

Figure 37: *Object 2: Z-axis Change in XYZ*

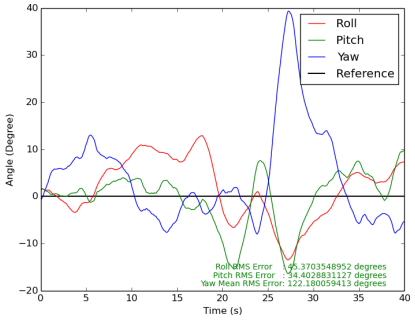

Figure 38: *Object 2: Z-axis Change in RPY*

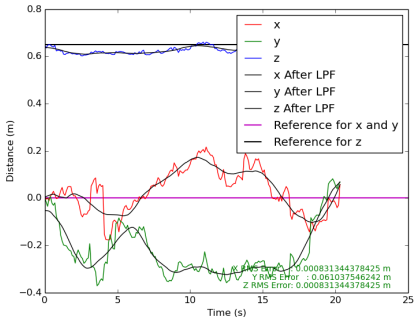

Figure 39: *Object 3: Roll Change in XYZ*

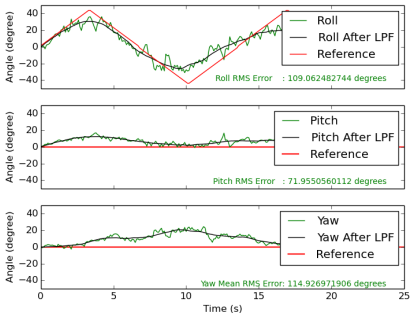

Figure 40: *Object 3: Roll Change in RPY*

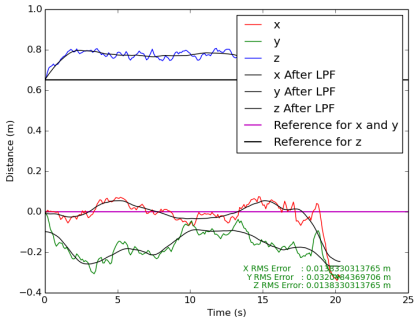

Figure 41: *Object 3: Pitch Change in XYZ*

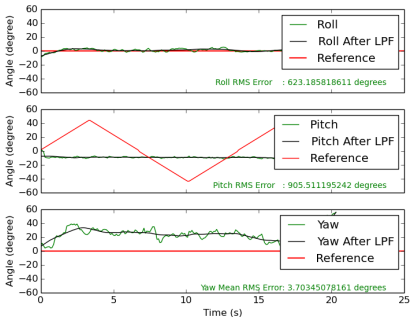

Figure 42: *Object 3: Pitch Change in RPY*

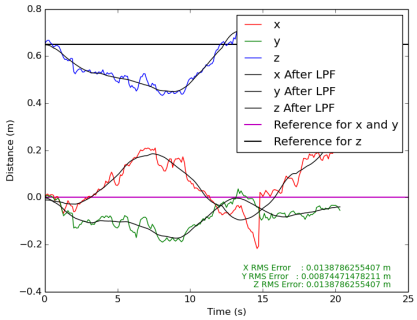

Figure 43: *Object 3: Yaw Change in XYZ*

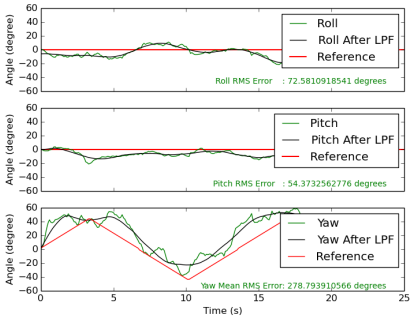

Figure 44: *Object 3: Yaw Change in RPY*

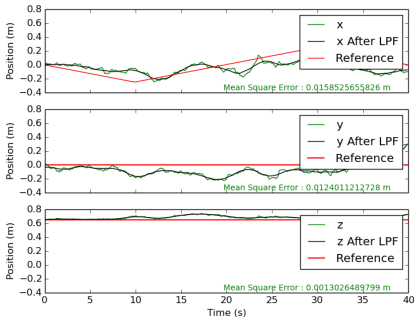

Figure 45: *Object 3: X-axis Change in XYZ*

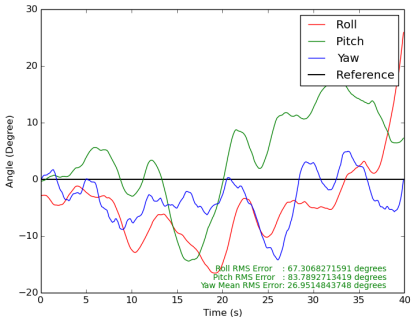

Figure 46: *Object 3: X-axis Change in RPY*

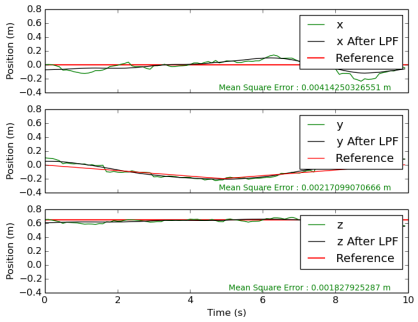

Figure 47: *Object 3: Y-axis Change in XYZ*

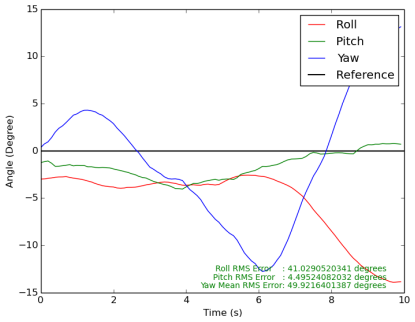

Figure 48: *Object 3: Y-axis Change in RPY*

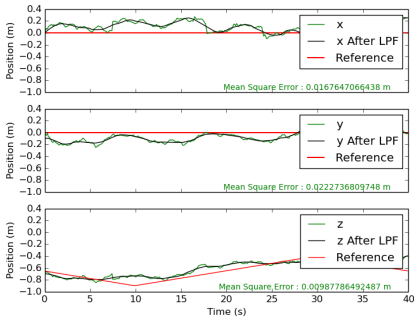

Figure 49: *Object 3: Z-axis Change in XYZ*

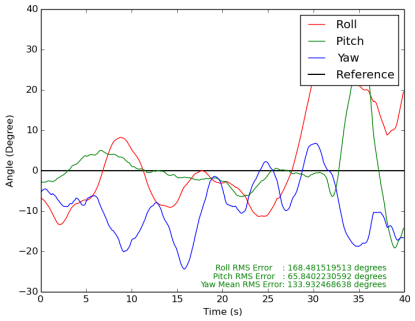

Figure 50: *Object 3: Z-axis Change in RPY*

References

[1] When State-of-the-Art is Second Best. (2015, February 13). Retrieved January 27, 2018, from http://www.pbs.org/wgbh/nova/next/tech/durable-prostheses

[2] Time-of-flight camera. (2018, January 22). Retrieved January 27, 2018, from https://en.wikipedia.org/wiki/Time-of-flight$_camera$

[3] The CamBoard pico family. Retrieved January 27, 2018, from https://pmdtec.com/picofamily/

[4] Powering the world's robots. (n.d.). Retrieved January 27, 2018, from http://www.ros.org/

[5] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., ... Ng, A. Y. (2009, May). ROS: an open-source Robot Operating System. In ICRA workshop on open source software (Vol. 3, No. 3.2, p. 5).

[6] Rusu, R. B., Cousins, S. (2011, May). 3d is here: Point cloud library (pcl). In Robotics and automation (ICRA), 2011 IEEE International Conference on (pp. 1-4). IEEE.

[7] Orts-Escolano, S., Morell, V., Garcia-Rodriguez, J., Cazorla, M. (2013, August). Point cloud data filtering and downsampling using growing neural gas. In Neural Networks (IJCNN), The 2013 International Joint Conference on (pp. 1-8). IEEE.

[8] Schnabel, R., Wahl, R., Klein, R. (2007, June). Efficient RANSAC for point-cloud shape detection. In Computer graphics forum (Vol. 26, No. 2, pp. 214-226). Blackwell Publishing Ltd.

[9] M.A. Fischler and R.C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM , 24(6):381–395, 1981.

[10] Random sample consensus. (2017, November 21). In Wikipedia, The Free Encyclopedia. Retrieved 17:45, January 28, 2018, from https://en.wikipedia.org/w/index.php?title=Random$_sample_consensusoldid$ = 811442233

[11] Derpanis, K. G. (2010). Overview of the RANSAC Algorithm. Image Rochester NY, 4(1), 2-3.

[12] Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., ... Xiao, J. (2015). Shapenet: An information-rich 3d model repository. arXiv preprint arXiv:1512.03012.

[13] Maturana, D., Scherer, S. (2015, September). Voxnet: A 3d convolutional neural network for real-time object recognition. In Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on (pp. 922-928). IEEE.

[14] Krizhevsky, A., Sutskever, I., Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[15] Chollet, F. (2013). Keras deep learning library for python. convnets, recurrent neural networks, and more. runs on theano and tensorflow. GitHub repository.

[16] Documentation. (n.d.). Retrieved January 29, 2018, from http://pointclouds.org/documentation/tutorials/cylinder$_s egmentation.php$

[17] Documentation. (n.d.). Retrieved January 30, 2018, from http://pointclouds.org/documentation/tutorials/tracking.php

[18] Pomerleau, F., Colas, F., Siegwart, R. (2015). A review of point cloud registration algorithms for mobile robotics. Foundations and Trends in Robotics (FnTROB) , 4(1), 1-104.

[19] Bylow, E., Sturm, J., Kerl, C., Kahl, F., Cremers, D. (2013). Real-time camera tracking and 3d reconstruction using signed distance functions. In Robotics: Science and Systems (RSS) Conference 2013 (Vol. 9). Robotics: Science and Systems.

[20] Izatt, G., Mirano, G., Adelson, E., Tedrake, R. (2017). Tracking objects with point clouds from vision and touch. 2017 IEEE International Conference on Robotics and Automation (ICRA). doi:10.1109/icra.2017.7989460

[21] Ren, S., He, K., Girshick, R., and Sun, J. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks"

[22] Izatt, G., Mirano, G., Adelson, E., Tedrake, R. (2017). Tracking objects with point clouds from vision and touch. 2017 IEEE International Conference on Robotics and Automation (ICRA). doi:10.1109/icra.2017.7989460