SUMMER RESEARCH INTERNSHIP REPORT

Computer Vision Lab ,

Indian Institute of Science , Bangalore

Divya K Raman,

EE15B085,

Department of Electrical Engineering,

IIT Madras.

Acknowledgements

I am very grateful to the Department of Electrical Engineering, IIT Madras and my mentor and guide Prof. R. Sarathi (IITM) for allowing and supporting me to take up this internship at IISc Bangalore.

It gives me immense pleasure to express my heartfelt gratitude to Prof. Venu Madhav Govindu, my mentor and guide at IISc Bangalore, who has been very kind to provide me with an opportunity to work in his lab. Prof. Venu Madhav Govindu has been very kind to guide and spend his valuable time through the course of my work here in the midst of his multifarious commitments.

I would like to thank the wonderful people I met at the lab starting from Uttaran for his intellectually stimulating discussions and his immense help in my work here. I would also like to thank Sumit and Mohammadul for their valuable inputs.

Contents

1 Introduction

This report is a document of all the activities and experiments performed at Computer Vision Lab, Indian Institute of Science, Bangalore under the guidance of Dr. Venu Madhav Govindu.

This report contains a tutorial of the iterative closest point(icp) algorithm. It also presents an analysis of the rotation angle and axis results obtained by applying icp on ideal and real datasets. The report further compares how Point Cloud Library's pcl performs in comparison to an implementation in matlab. It also has a simple(not robust) python implementation of the algorithm. Further, it presents a tutorial for two view 3D reconstruction.

Geometry in computer vision is an important sub-field in computer vision which mainly deals with relation between 3D points and their projections onto images. The pinhole camera model is typically used to explain the concepts involved here. The algorithms developed in this field find extensive use in many areas like robotics, computer animation and medical imaging.

2 Iterative Closest Point(ICP) algorithm

2.1 ICP algorithm tutorial

Introduction

Point set registration, aims at finding the spatial transformation capable of aligning two 3D point sets. 2D/3D reconstruction is possible using point set registration which finds extensive use in multiple areas such as robotics, OCR and augmented reality. Multiple algorithms like iterative closest point(ICP), robust point matching(RPM), thin plate spline robust point matching(TPS-RPM) have been proposed to solve the problem of 3D point cloud registration. This tutorial describes in-depth one commonly used algorithm - Iterative Closest Point.

The problem statement

Consider 2 geometrical representations P(data) and X(model) of dimensions p*3 and x*3 respectively which represent the same scene to a good extent. The aim of the algorithm is to find the transformation which when applied to P aligns it to best match the reference. For the algorithm described in this tutorial, the geometric representation is decomposed into a point set. Another assumption made is that the data and the model datasets that are deal with are both rigid bodies and only rigid transformations can be applied.

What is a rigid transformation?

Rigid transformation of a vector space preserves the distance between points. Further, the shape before and after transformation are congruent. Rigid transformations include rotation, translation , or a combination of both. Say a translation T(translation of the origin) and a rotation R is applied on a shape S. Then the shape after the translation becomes S’ = S*R + T. One point to be noted is that R is an orthogonal transformation i.e. $R^T = R^{-1.}$ In proper rigid transformations, det(R) = 1. The determinant is -1 when it represents a reflection.

Overview of ICP algorithm

The model point cloud is kept fixed while the data point cloud is used to iteratively find the transformation which when applied to it aligns both the point clouds to the best possible extent. This algorithm was first proposed by Chen and Medioni, and Besl and McKay.

Input : The two point clouds

Output : The transformation on the data point cloud which will best align it to the model point cloud.

The basic steps in the algorithm is as follows:

1. For each point in the data point cloud, the closest point(Euclidean distance) in the model point cloud is chosen.
2. The combination of rotation and translation which will best align P to X is chosen.
3. The transformation is applied on P.
4. Iterate till a given threshold is reached.

The algorithm in detail

The below description is based on the initial paper on ICP by Besl and McKay.

1 Finding corresponding points

In order to find the translation and rotation needed to align the two point sets, we need to find the corresponding set of points. This can be done by multiple ways such as feature detection. If no information is given, assume that closest points in each point cloud correspond. The algorithm converges if the starting position is close enough so that erroneous pair are not detected as corresponding points.

Given 2 points r1 and r2, the euclidean distance d(r1,r2) = ||r1-r2|| = $(\boldsymbol{\Sigma}(x_{1i}-x_{2i})^2)^{1/2}$.

Given a point r1 and a point set A, the distance between r1 and A is :

$d(r1,A) = \min (d(r1,a_i)\ , \forall i \text{ in } (1, \text{length}(A))$.

Shape P is to be aligned to X. The distance of each point p in P to X is

$d(p,X) = \min (d(p,x_i)\ , \forall i \text{ in } (1, \text{length}(X))$

The closest point operator C and the set of closest points Y is defined as given below:

$Y = C(P,X)$.

This operator C finds the set of closest points in P and X based on the Euclidean distance between pairs of points and the corresponding points set is stored in Y.

Worst Case completes in O(px) time while the average time taken is O(plogx).

2 Finding rotation and translation

The next task is to find the rotation and translation which when applied to P best aligns it to X. The transformation is usually found by a least squares method. The method described in this tutorial is based on [5]. It aims at finding rotation and translation using SVD.

Let the number of corresponding points be N.

Objective function:

Find R and T which minimize $f(R,T) = (1/N)(\sum||x_i - R(p_i) - T||^2, i = 1,2, \ldots, N)$ i.e. we want the two point sets to be as aligned as possible after applying the transformation on P.

2.1 Translation

Assume R is fixed. Denote $f(T) = (1/N)(\sum||x_i - R(p_i) - T||^2, i = 1,2, \ldots, N)$. Optimal T can be got by taking the first derivative of f(T) wrt T. The result obtained is $T = X' - R(P')$. X',P' denote the respective center of masses. In this case, since we take unweighted points, the center of mass is just $\sum x_i/N$. It changes to $\sum x_i w_i / \sum w_i$ in the case of weighted points where w_i denotes the corresponding weights.

2.2 Rotation

$f(R,T) = (1/N)(\sum||x_i - R(p_i) - T||^2, i = 1,2, \ldots.., N)$.

At this stage, we have T in terms of R. Substituting it,

$f(R,T) = (1/N)(\sum||x_i - R(p_i) - X' - R(P')||^2, i = 1,2, \ldots.., N)$.

Let $a_i = (p_i - P')$,$b_i = x_i - X'$.

Then, $R = \arg\min \sum||b_i - R(a_i) ||^2, i = 1,2, \ldots.., N$. This is equivalent to saying that we want to find the rotation that will minimize the difference in alignment between point clouds A and B where the translation is zero.

$||b_i - R(a_i) ||^2 = b_i^T b_i + a_i^T a_i - 2 b_i^T R a_i$. Note that $RR^T = 1$(rotation matrix is orthogonal). The summation of this expression is what we need to minimize while choosing the value of R. This is equivalent to saying :

$R = \arg\min \sum - 2 b_i^T R a_i = \arg\max \sum b_i^T R a_i$.

Now, $\sum b_i^T R a_i = tr(B^T RA)$, where tr denotes the trace of the matrix. Let r be the dimension of R. Then B is of dimension r*N with $b_i$as its columns and A is of dimension r*N with $a_i$as its columns. Therefore, we need to find R so that the trace of $B^T RA$ is maximized.

Now, $tr(BA) = tr(AB)$ provided the dimensions are compatible. $tr(B^T RA) = tr(RAB^T)$.

$S = AB^T$is a r*r covariance matrix. Let's take the singular value decomposition of S.

$S = U\Sigma V^T$. Let's now substitute this into the expression we are trying to maximize.

$tr (RAB^T) = tr (RS) = tr (RU\Sigma V^T) = tr (\Sigma V^T RU)$. V,R,U are all orthogonal matrices => $M = V^T RU$ is also orthogonal. This implies that $m_i m_i^T=1$ for each M's column m_i. This forces each entry of M to be lesser than or equal to 1.

We are now left to find the maximum value of $tr(\Sigma M)$. Σ is a diagonal matrix with non negative entries(from SVD). Let its entries be denoted by σ_i.

Then tr(ΣM) = $\Sigma(\sigma_i m_{ii})$ for i ranging from 1 to r which is lesser than $\Sigma\sigma_i$, i = 1,2,....,r. Therefore the trace is maximized. if $m_{ii} = 1$. Since M is orthogonal, M is the identity matrix I.

$I = M = V^T RU \Rightarrow R = VU^T$.

2.3 Transforming P and checking

Having found R and T, we now apply the rotation and translation on the point set P. The mean square error in the kth iteration is found as:

$d_k = (1/N) \sum||p_{ik} - x_i||^2$. Terminate the iteration when the change in the mean squared error $d_k - d_{k-1}$ falls below a preset threshold.

3 Convergence Theorem

Key ideas :

i) Least squares registration reduces the average distance between corresponding points during each iteration.

ii)The closest point determination reduces the distance for each point individually which also reduces the average distance.

Theorem : The ICP algorithm always converges monotonically to the local minimum with respect to the mean square distance objective function.

Proof :

$d_k = (1/N) \sum||p_{ik} - x_{ik}||^2$ is the alignment error.

The correspondence error is $e_k = (1/N) \sum||p_{i(k+1)} - x_i||^2$ where the transformation obtained in the kth iteration is applied on p_k to get p_{k+1} . Now , d_k is always lesser than e_k.

Let us apply the closest point operator for the next iteration.

$Y_{k+1} = C(X,P_k)$.

Let’s say Y stores the closest points in X corresponding to P. Then,

$||y_{i(k+1)} - p_{i(k+1)}|| < ||y_{i(k)} - p_{i(k+1)}||$

=> $e_{k+1} < d_k$(Equality applies after convergence).

Therefore , $0 < d_{k+1} < e_{k+1} < d_k < e_k$. Equality applies in case of convergence.

References:

1. Besl, P. J., & McKay, N. D. (1992, April). Method for registration of 3-D shapes. In Robotics-DL tentative (pp. 586-606). International Society for Optics and Photonics.

2. Slides by Ronen Gvili : www.cs.tau.ac.il/~dcor/Graphics/adv-slides/ICP.ppt

3. Note on Least-Squares Rigid Motion Using SVD by Olga Sorkine-Hornung and Michael Rabinovich, Department of Computer Science, ETH Zurich

4. https://en.wikipedia.org/wiki/Point_set_registration

5. https://en.wikipedia.org/wiki/Rigid_transformation

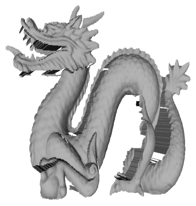
6. https://en.wikipedia.org/wiki/Iterative_closest_point

2.2 Data models

Two datasets have been used the analysis of the algorithm.

1. Stanford Dragon
   This is a standard dataset acquired from the Stanford Computer Graphics Laboratory website - https://graphics.stanford.edu/data/3Dscanrep/.
   This dataset consists of 15 point clouds corresponding to the depth scans of the dragon taken at 24 degree intervals thus covering the whole 360 degree view of the dragon. Considering this dataset to be ideal, it has been used without applying any pre-filtering techniques on the point clouds.

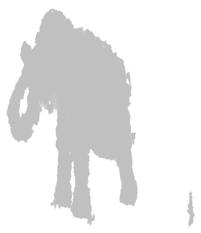
   3D visualization of the dragon from one view is given below:



2. Elephant
   Microsoft's Kinect camera available at the lab has been used to capture the depth scans of the elephant model available at lab. 72 depth scans (at 5 degree intervals) have been acquired by rotating the turn-table on which the elephant has been placed. An RGB image of one view of the elephant is given below :

We apply bilateral depth filtering to these point clouds which is an edge preserving filtering method.

3D visualization of the elephant from one view is given below:



2.3 Rotation angle error analysis

This chapter contains an analysis of the angle of rotation got on using ICP to register the point clouds. The analysis has been done on both the Stanford dragon point clouds and the elephant point clouds. The code for analysis has been written in matlab. The icp code used for this experiment is the code by Martin Kjer and Jakob Wilm, Technical University of Denmark, 2012 written in matlab.

Using the ICP to register multiple point clouds :

Given to point clouds p and p ,

[R,T] = icp(q,p) gives the rotation and translation that has to be applied on p2 to align it with p1.

q = R*p + T.

Reconstruction of a complete model using multiple 3D point clouds requires us to transform and bring all the point clouds to a single frame of reference and stitch them together. Lets name our point clouds p1 , p2 , p3 ....p72. Let’s consider the frame of reference for our reconstruction to be the frame to which p1 corresponds. Thus, all other point clouds , namely p2,p3 ....p72, need to be brought into the frame of p1 and stitched together.

[R21,t21] = icp(p1,p2) gives the transformation that needs to be applied on p2 to align it with p1.

[R32,t32] = icp(p2,p3) gives the transformation that needs to be applied on p3 to align it with p2.

We now need to find the transformation from p3 to p1.

[R31,T31] = icp(p1,p3) will directly give the transformation that needs to be applied on p3 to align it with p1. But the results may not be correct in this case. The large transformation that needs to be applied between p3 and p1 may need lead to erroneous correspondences in the correspondence matching step of the iterative closest point algorithm and thus may lead to incorrect results. One point worthy noting here is that icp works accurately only on point clouds which represent the same scene to good extent.

To find the transformation from p3 to p1 , we find the accumulative transformation.

p1 = R21*p2 + T21

p2 = R32*p3 + T32

Therefore, p1 = R21 *R32 *p3 + R21 * T32 + T21.

The net rotation is therefore R21 * R32. The net translation is R21 * T32 + T21.

In the case of a turn-table generated sequence , we are interested only in the rotation and the net rotation as the translation in all cases and hence the net translation is ideally supposed to be zero.

Therefore the net rotation from ith point cloud to the first point cloud can be generalized as R21 * R32 * ....Ri(i-1). The matlab code of the algorithm is given below :

```
  V1 = load('Vert/Scan16.mat');

V1 = V1.V;

accumR = eye(3);

fileID = fopen('elephant_accumulated.txt','a');

for i=17:89

V2 = load(['Vert/Scan' num2str(i) '.mat']);

V2 = V2.V;

[R,T] = original_icp(V1,V2);

%V1 = R*V2

%V2 = R1*V3

%V1 = R*R1*V3
```

```
        %accumR = R*R1

        accumR = accumR * R;%Accumulating the rotation

        r = vrrotmat2vec(accumR);

        format = ['Scan ' num2str(i) 'to Scan 16 Angle: %f degrees Axis: %f %f
%f \n'];

        fprintf(fileID,format,radtodeg(r(4)),r(1),r(2),r(3));

        A = [R,T];

        save(['elephant_icp/Scan' num2str(i) 'toScan' num2str(i-1)],'A');

        V1 = V2;

        end
```

Error Analysis :

The ideal accumalated rotation from scan 73 to scan 1 is 360 degrees about the y axis. This corresponds to an identity rotation matrix. The net error rotation matrix is therefore :

$R_{accum}R_{error}$ = I. Therefore , $R_{error} = R^{-1}_{accum}$. This is the accumulated error. Thus , the error in each icp step is $(R_{error})^{1/72}$.

The matlab code for this is as given below :

```
        errorR = inv(accumR);

        %74 scans give errorR => each scan gives errorR^1/74.

        individualError = real(mpower(errorR, 1/74));

        r = vrrotmat2vec(individualError);

        rad2deg(r(4))

        Result : r = 0.1166 0.9040 0.4114 0.0248

        error in angle : 1.4227 degrees.
```

The axis has a deviation of 25.84 degrees (arccos(0.9)) from the y axis.

Results :

Elephant :

The rotation angle and the axis between (i+1)th view and ith view has been acquired and analyzed. The angle rotation is around 3.5 - 4 degrees in most cases while the expected angle is 5 degrees. Increasing the number of iterations in icp to see if the convergence is better yields the same results.

The corresponding accumulated rotation angle and axis has also been acquired. The expected angle of rotation from Scan 73 to Scan 1 (Here Scan 89 to Scan 16) is 360 degrees. The angle obtained here is about 255 degrees. The average error in the angle of rotation is 1.4227 degrees which is 28.5 % of the expected angle 5 degrees. The axis has a deviation of 25.84 degrees (arccos(0.9)) from the y axis.

Dragon :

The rotation angle and the axis between (i+1)th view and ith view has been acquired and analysed. The angle rotation is around 20 - 24 degrees in most cases while the expected angle is 24 degrees.

The corresponding accumulated rotation angle and axis has also been acquired. The expected angle of rotation from the last scan to the first scan taken in order is 360 degrees. The angle obtained here is about 316 degrees. The average error in the angle of rotation is 2.9283 degrees which is 12.2 % of the expected angle 24 degrees.

We see that the results are better in case of an ideal dataset. An error percentage of about 12 degrees in the ideal case indicates the need for a better icp algorithm for more accurate results.

Sources of error:

- Turn table was manually rotated to acquire the depth scans which can lead to errors in both the rotation angle and axis(may not be exactly 5 degrees about y axis), rotation about smaller angles while capturing the depth scans can lead to better correspondences between 2 successive point clouds.

- Turn table needs to be kept sturdy so that axis isn’t affected.

2.4 Analysis on axis of rotation

Errors in the set up and while performing the experiment can affect the results obtained on using icp on the point clouds in order to align and stitch them together. In an ideal setup, the axis of rotation remains the same throughout the experiment. This may however not be true in many real cases. We attempt to analyze how the axis of rotation changes as various depth scans are taken. We also compute the ’average axis’ and see how much each axis deviates from this average.

COORDINATE SYSTEMS

Coordinate systems are not fixed. They can be chosen as per the needs of the set up and experiment performed. We use the right handed 3D Cartesian coordinate system centered at the camera’s center (can be object centered too). The camera is mounted on a plain surface and is at almost the same level as the turn table. The camera’s mount are taken care to not be in an inclined position which will change the coordinate system. The turn table is also maintained in a similar position. The turn table’s surface is the x-z plane in this case and the y axis is perpendicular and pointing upwards. If this setup is perfect, the axis of rotation should be the y axis corresponding to the direction vector [0 1 0]. However, this is not true in many cases due to the turn table’s possible jiggle while rotating.

AVERAGE AXIS

Successive point clouds are taken pairwise and the transformation that aligns one to the other is computer using icp. The axis and angle of rotation is obtained using the rotation matrix. The axis of rotation is of interest here. Let’s call them n_1 , n_2..... n_{72} where each is a column vector of size 3*1. These are basically unit vectors and are points on the unit sphere. The average axis is also a point in the unit sphere. Simply taking the mean of the axis will not give us the average axis as its norm won’t be one. A central tendency attempts to find the center of the probability distribution.

$C(n_{avg}) = \sum d^2(n_i, n_{avg})$ is the cost function we are trying to minimize. Let us define this cost function to be the angle between the two unit vectors. Then,

$C(n_{avg}) = \sum\theta^2(n_i ,n_{avg}) = \sum(\cos^{-1}(n_i .n_{avg}))^2$.

$\cos^{-1}(n_i .n_{avg}) = \theta$ (inverse cos of dot product)

$\Rightarrow \cos\theta = n_i .n_{avg}$

$\Rightarrow 1 - \sin^2\theta = 1 - (n_i .n_{avg})^2$.

We therefore need to maximize $\sum(n_i .n_{avg})^2$.

$(n_i .n_{avg})^2 = (n_i .n_{avg})(n_i .n_{avg})^T = n_{avg} . n_i.n_i^T.n_{avg}$

$n_i.n_i^T$ is a 3*3 square matrix of rank 1. $\sum n_i.n_i^T$ is generally full rank if $i > 2$. We now take the SVD of $\sum n_i.n_i^T = U\sum V^T$. $V = U$ here because $\sum n_i.n_i^T$ is a symmetric matrix. The value of n_{avg} is the first column of U.

The matlab code for the same is given below :

```
sum = zeros(3,3);

axes = zeros(72,3);

for i=17:89%1 to 72 ideally

a = load(['Scan' num2str(i) 'toScan' num2str(i-1) '.mat']);

RT = a.A;

R = [RT(1,1) RT(1,2) RT(1,3);RT(2,1) RT(2,2) RT(2,3);RT(3,1) RT(3,2) RT(3,3)];

r = vrrotmat2vec(R);

ni = [r(1);r(2);r(3)];

axes(i-16,:) = transpose(ni);

niT = transpose(ni);

temp = ni*niT;

sum = sum+temp;

end

[U Sigma V] = svd(sum);

axis = U(:,1);%first column of U.
```

Result :

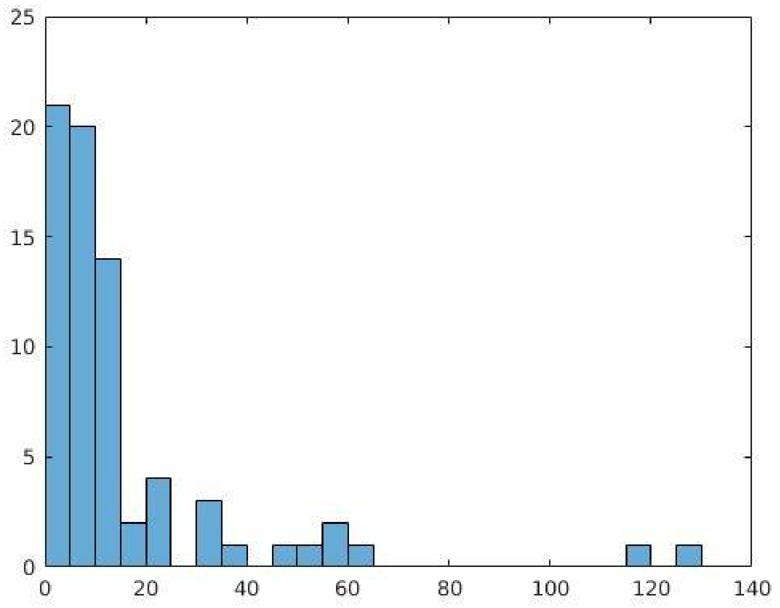
n_{avg} = [-0.0433 0.9963 0.0749]^T which is an axis 4.9616 degrees away from the y axis.

Verification

We now wish to find how accurate our result is. We find the angle between each axis and the central axis and make a histogram plot. The matlab code for the same is given below :

```
theta = zeros(72,1);

%Verifying the accuracy of the estimated ’average axis’.

for i=1:72

dotp = (axes(i,1)*axis(1,1)) + (axes(i,2)*axis(2,1)) + (axes(i,3)*axis(3,1))

prod = norm(axes(i,:))*norm(transpose(axis))

cosf = dotp/prod;

theta(i,1) = radtodeg(acos(cosf));

end

h = histogram(theta);

h.BinWidth = 5;
```

The plot is given below :



We see that the angle of deviation from the central axis in most cases is below 15 degrees. Ideally , the deviation should be zero. The average axis tells us the average tilt of the camera(/turn table) in our set-up assuming that the icp algorithm used gives us perfect results. Perfect estimation of the average axis is quite difficult as the errors can come not just from the set-up but also the icp algorithm itself.

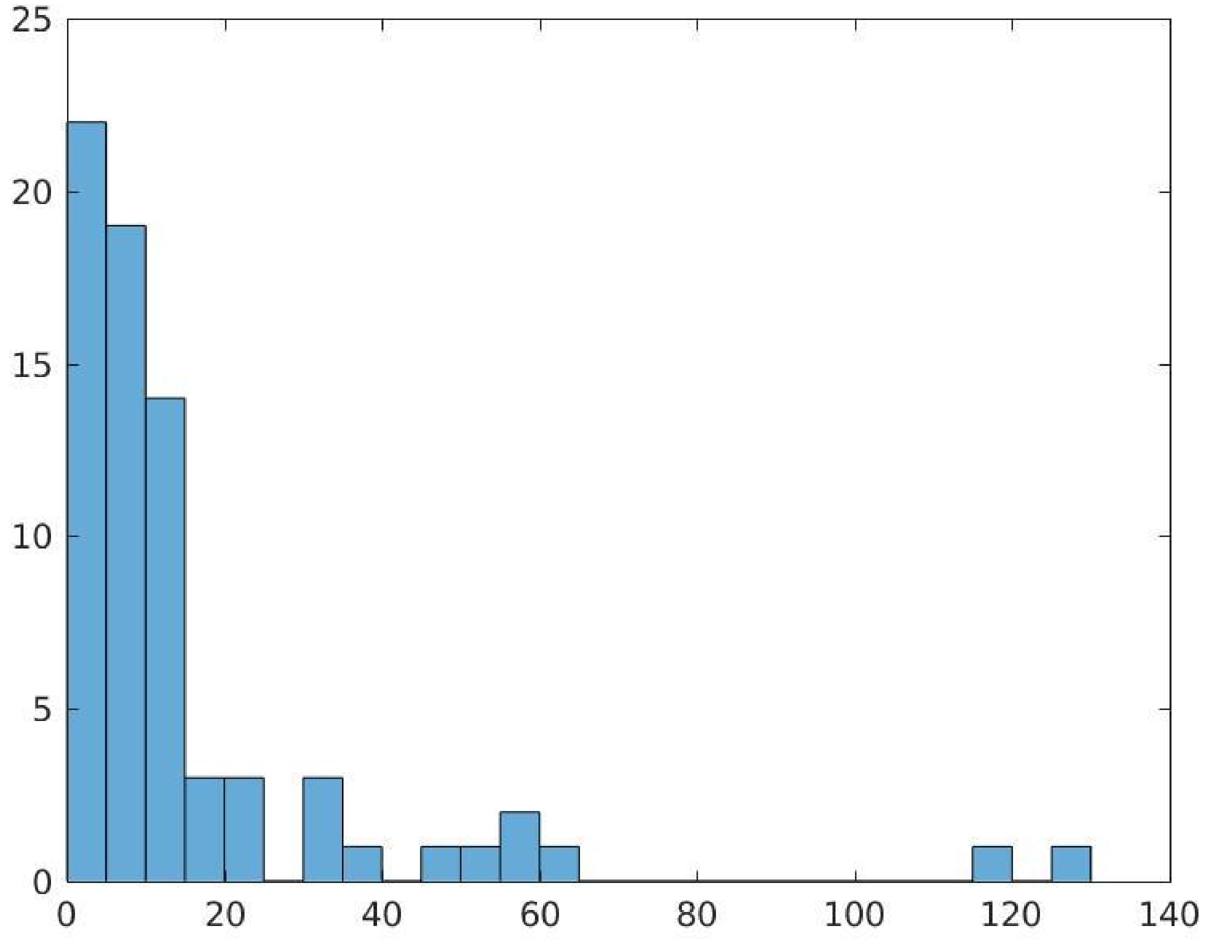
Iterative weighted least squares

We see that there are cases where the deviation angle is more than 100 degrees. These large deviations have a tendency to affect the ’average axis’. We thus find the average axis by using the iterative weighted method. Here , in the first iteration , all cases are assigned equal weights of 1. We then assign appropriate weights to each observation and iteratively compute the average axis. The iteration stops when convergence specified by appropriate parameters is reached. The matlab code for the same is given below :

```
%iterative weighted least squares algorithm

temp = zeros(72,1);

for i=1:5

average = radtodeg(acos(axis(2,1)))

temp = (theta-average);
```

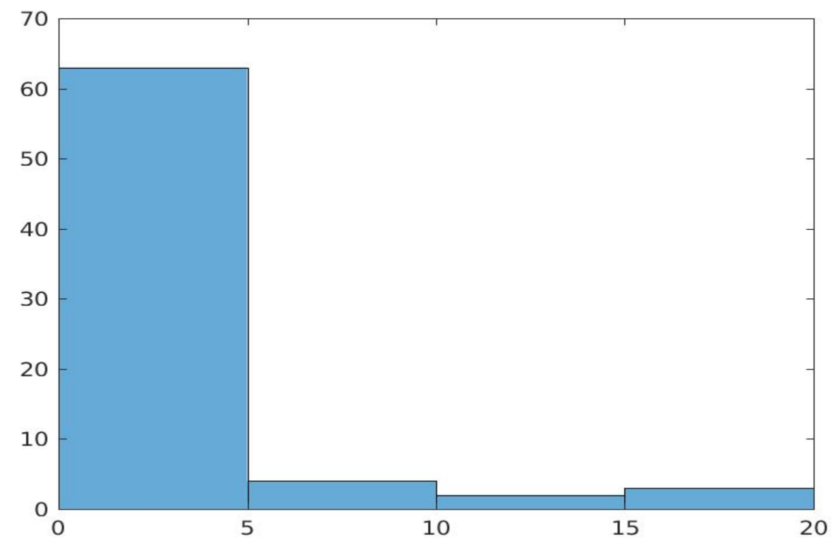
```
vari = 0;

for j=1:72

vari = vari+(temp(j,1)*temp(j,1));

end

vari = vari/72;

weights = normpdf(theta,average,sqrt(vari));

sumax = zeros(3,3);

for j=1:72

ni = axes(j,:);

niT = transpose(ni);

tempo = niT*ni;

tempo = tempo*weights(j,1);

sumax = sumax+tempo;

end

[U sigma V] = svd(sumax);

axis = V(:,1)%first column of V

errortheta = zeros(72,1);

for j=1:72

dotp = (axes(j,1)*axis(1,1)) + (axes(j,2)*axis(2,1)) + (axes(j,3)*axis(3,1));

prod = norm(axes(j,:))*norm(transpose(axis));

cosf = dotp/prod;

errortheta(j,1) = radtodeg(acos(cosf));

end

h = histogram(errortheta);

h.BinWidth = 5;

end
```

The weights are assigned using the normal probability distribution where the mean and variance needs to be calculated appropriately. Convergence is reached in a maximum of 5 iterations in most cases. The histogram distribution is given below :



The average axis is 4.7261 degrees away from the y axis.

Next, the same analysis is performed on the ideal Stanford dragon dataset. The average axis got in this case is exactly the y axis as expected ([-0.0099 1.0000 -0.0005]). The histogram plot for the deviation from the average axis is as given below :



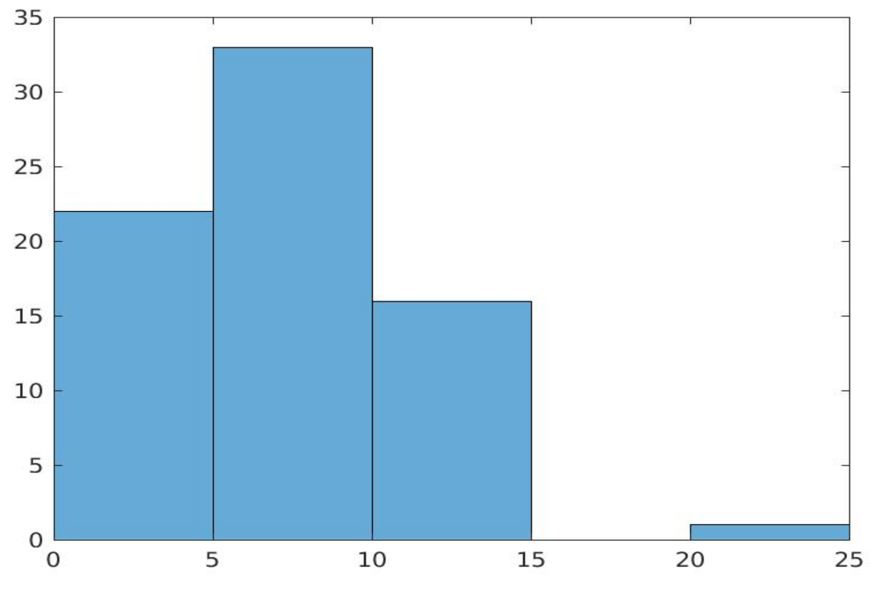
The histogram plot shows that the deviation is below 5 degrees in most cases. Also, the maximum deviation does not exceed 20 degrees which again proves that our estimation is pretty good.

2.5 PCL’s ICP algorithm

In this sub-chapter , we analyze the angle of rotation and the axis of rotation by applying Point cloud library's ICP algorithm on the elephant point clouds. We also try to improve the robustness of the algorithm by using reciprocal correspondences for finding correspondences. Reciprocal correspondences determines the correspondences from the source point cloud to the target point cloud and and from the target point cloud to the source point cloud and takes the intersection of both. We also improve the convergence by setting a better threshold. The code for the same is given below :

```
#include <iostream>
#include <pcl/io/pcd_io.h>
#include <pcl/point_types.h>
#include <pcl/registration/icp.h>
int
 main (int argc, char** argv)
{
  pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_in (new
pcl::PointCloud<pcl::PointXYZ>);
  pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_out (new
pcl::PointCloud<pcl::PointXYZ>);

if (pcl::io::loadPCDFile<pcl::PointXYZ> ("cloud17.pcd", *cloud_in) == -1) //*
load the file
  {
    PCL_ERROR ("Couldn't read file \n");
    return (-1);
  }
int i;
std::stringstream ss;
for(i=18;i<90;i++){
           ss << "cloud" << i << ".pcd";
            pcl::io::loadPCDFile<pcl::PointXYZ>(ss.str(), *cloud_out);
            ss.str(std::string());
  pcl::IterativeClosestPoint<pcl::PointXYZ, pcl::PointXYZ> icp;
  icp.setInputCloud(cloud_in);
  icp.setInputTarget(cloud_out);
icp.setTransformationEpsilon (1e-8);
icp.setUseReciprocalCorrespondences(true);
  pcl::PointCloud<pcl::PointXYZ> Final;
  icp.align(Final);
  std::cout << "has converged:" << icp.hasConverged() << " score: " <<
  icp.getFitnessScore() << std::endl;
  std::cout << icp.getFinalTransformation() << std::endl;
*cloud_in = *cloud_out;

}
return (0);
}
```

Analysis :
The rotation angle is 4-5 degrees in most cases. The angle between the last and the first scan taken in order is 316.2 degrees. The error angle per pair is 0.6 degrees and the average angle is 4.39 degrees.
The average axis is 3.03 degrees away from the y axis. The histogram plot for the deviation from the average axis is given below :



The deviation from the average axis is below 10 degrees in most cases and the maximum deviation is 25 degrees. Comparing the results obtained by using pcl's icp implementation and the implementation written in matlab , we find that the transformation results are much better and closer to the expected results when pcl's implementation is used. This indicates that point cloud library's icp implementation gives more accurate results.

2.6 A python code for ICP algorithm

A part of the implementation of the ICP algorithm based on the tutorial in 2.1 is given below:

```
#importing modules
import numpy as np
import cv2
import scipy as sp
import scipy.io as sio
from scipy.spatial import KDTree
from numpy import matlib
from numpy import linalg

#Loading vertices
V1 = sio.loadmat('Scan29.mat')
V2 = sio.loadmat('Scan30.mat')
V1 = V1['V']#Loadmat returns a dictionary.
V2 = V2['V']#3*n array

V1 = np.transpose(V1)
V2 = np.transpose(V2)#Transpose to make it n*3 array
temp = V2
corres = np.zeros((len(V1),3))#V2's points corresponding to those in V1 at
the same index

#V1 is fixed. Transformation which will align V2 i.e. temp to V1 needs to be
found

#temporary rotation and translation
T = np.zeros((1,3))
R = np.ones((3,3))

#Kd tree nearest neighbour algorithm to find correspondences
for k in range(0,niter):
        tree = KDTree(temp) #array temp is not copied but modified

        point = V1[i]
        a = tree.query(point)#1 nearest neighbour from V2's Kdtree structure
        nearestNeighbourIndex = a[1]
        corres[i] = tree.data[nearestNeighbourIndex]

        #finding R and T using svd

        #finding centroids and subtracting each point from its corresponding
centroid
        length = len(V1)#=len(temp)
        Ac = np.sum(V1[:][:],axis=0)
        Ac = Ac/length
        Bc = np.sum(corres[:][:],axis=0)
        Bc = Bc/length

        Ac = matlib.repmat(Ac,length,1)
        Bc = matlib.repmat(Bc,length,1)
        A = V1-Ac#fixed point cloud
        B = corres-Bc
        C = np.dot((np.transpose(B)),A)
        U,s,V = linalg.svd(C)
        R = np.dot(np.transpose(V),np.transpose(U))
        T = Ac – np.transpose(np.dot(R,np.transpose(Bc)))

        print R , T
```

This implementation uses a kd tree to find the nearest neighbour and find correspondences. The transformation matrix has been found using the svd method. The rms error between the corresponding points after transformation can be used as the parameter to determine the convergence. Using the error parameter, we can iteratively find R and T to align one point cloud perfectly to another. This code can also be made robust for better accuracy.

3 Two view 3D reconstruction

The problem of 3D reconstruction of an object from its 2D images taken from multiple views has seen tremendous evolution over the years. This problem involves finding the 3D positions of all the points based on their projections into the multiple image frames. A basic subset of this problem would be to find the 3D projections of all corresponding points given two 2D images which sufficiently represent the same scene which is where we start.

1 Convert images to gray scale

- Gray scale images are easier to process and have the intensity sufficient for most image processing tasks.
- Luminance is more important than chrominance.

2 Finding correspondences

This step involves detecting the corners, edges and other features in the two images and matching them. Harris corner detector algorithm is one commonly used algorithm.

Corner detection aim : Find patches/windows that generate a large variation when moved around.

Corner detectors : Harris corner detector, Shi-Tomasi corner detector.

The movement of a window can be translated mathematically as :

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u,y+v) - I(x,y)]^2$$

E = difference between the original and the moved window.
u = window's displacement in the x direction
v = window's displacement in the y direction
w(x, y) is the window at position (x, y). This acts like a mask ensuring that only the desired window is used.
I = intensity of the image at a position (x, y)
I(x+u, y+v) = intensity of the moved window
Windows that produce a large E value correspond to corner locations. This leads us to maximizing the term

$$\sum_{x,y}[I(x+u,y+v)-I(x,y)]^2$$

Using Taylor's series,

$$E(u,v) \approx \sum_{x,y}[I(x,y)+uI_x+vI_y-I(x,y)]^2$$

$$E(u,v) \approx \sum_{x,y} u^2{I_x}^2+2uvI_xI_y+v^2{I_y}^2$$

$$E(u,v) \approx [u \quad v]\left(\sum\begin{bmatrix}{I_x}^2 & I_xI_y \\ I_xI_y & {I_y}^2\end{bmatrix}\right)\begin{bmatrix}u \\ v\end{bmatrix}$$

$$M = \sum w(x,y)\begin{bmatrix}{I_x}^2 & I_xI_y \\ I_xI_y & {I_y}^2\end{bmatrix}$$ which leads to

$$E(u,v) \approx [u \quad v]\, M \begin{bmatrix}u \\ v\end{bmatrix}$$

Eigenvalues of the matrix can help determine the suitability of a window. A score, R, is calculated for each window: All windows that have a score R greater than a certain value are corners. They are good tracking points.

The Shi-Tomasi corner detector works quite well even when the Harris corner detector fails.

$$R = min(\lambda_1, \lambda_2)$$

is the score used.

To find correspondences, cross correlation or matched filtering is done. Correlation based algorithms produce a dense set of correspondences while feature based matching produces a sparse set of correspondences. Zero mean templates and intensity normalization techniques are used to avoid intensity ambiguities. The sum of squared differences (block matching) is claimed to be a better technique than correlation technique.

Pairs of correspondences can thus be found.

3 Estimating the essential matrix

We need to find the camera orientation used to capture one image in relation to the camera orientation used to capture the other image in order to find the 3D projection of corresponding points.

Epipolar geometry :

Consider a 3D point w. Consider 2 cameras in different orientations capturing the scene. Consider points x1 and x2 corresponding to the projection of w on camera plane 1 and camera plane 2. The point w must lie on the line joining optical center 1 and x1. The point w must also lie on the line joining optical center 2 and x2.The intersection of these 2 lines gives the 3D point w. x2 must lie on the projection of this ray on camera plane 2 which is referred to as an epipolar line. This leads to something known as the epipolar constraint : For any point in the first image, the corresponding point in the second image is constrained to lie on a line. The particular line on which it is constrained to lie depends on the extrinsic and intrinsic camera parameters. Rays in 3D converge at the optical center of the camera. Therefore, the epipolar lines also converge at a point know as the epipole. Epipole 2 is the projection of optical center of camera 1 into camera plane 2 and vice versa. Epipole 1, epipole 2, optical center 1 and optical center 2 lie on a line.

Fundamental matrix and essential matrix :

Fundamental matrix and essential matrix algebraically relate corresponding points between two images and represent the epipolar geometry.

Let the world coordinate system be centered on the first camera. This implies that the extrinsic camera parameters of the first camera are {I,0}. The second camera is in position $\{\Omega,\tau\}$. We assume that the cameras are normalised so that the intrinsic parameters $\Lambda_1 = \Lambda_2 = I$.

In homogeneous coordinates,

$\lambda_1 x'_1 = [I,0]w'$

$\lambda_2 x'_2 = [\Omega,\tau]w'$, where w is the 3D point, x_1 and $x_2$are its 2D projections into the 2 camera planes.

Simplifying the two equations,

$\lambda_1 x'_1 = w$

$\lambda_2 x'_2 = \Omega w + \tau$

Substituting one into the other,

$\lambda_2 x'_2 = \lambda_1 \Omega\, x'_1 + \tau$.

This is the constraint between the corresponding points between the 2 images.

Taking the cross product of both sides with τ,

$\lambda_2 \tau * x'_2 = \lambda_1 \tau * \Omega x'_1$. Taking the inner product of both sides with x'_2,

$x'^T_2 \tau * \Omega x'_1 = 0$ where the scaling factors $\lambda_1$and λ_2 are not present.

The cross product τ * Ω can be represented by a matrix E called the essential matrix. Therefore,

$x'^T_2 E x'_1 = 0$.

The fundamental matrix plays the role of the essential matrix when normalised intrinsic parameters cannot be used. The equations for the projection of 3D points into camera planes then become

$\lambda_1 x'_1 = \Lambda_1[I,0]w'$

$\lambda_2 x'_2 = \Lambda_2[\Omega,\tau]w'$

$\Rightarrow x'^T_2 \Lambda_2^{-T} E \Lambda_1^{-1} x'_1 = 0$

or $x'^T_2 F x'_1 = 0$. where $F = \Lambda_2^{-T} E \Lambda_1^{-1}$ or $E = \Lambda_2^T F \Lambda_1$.

The elements of the fundamental matrix are ambiguous upto scale => degrees of freedom of fundamental matrix = 8. Thus , a minimum of 8 pairs of points are needed to determine the matrix. The fundamental matrix is usually computed using robust estimation procedures such as RANSAC.

4 Estimating the camera pose and 3D reconstruction

The next step would be to extract the extrinsic parameters of the camera from the essential matrix which will give us the rotation and translation between the 2 cameras.

$E = \tau * \Omega$.

=> $E = \tau_X \Omega$ where the cross product $\tau *$ is expressed as τ_X.

Recovering rotation and translation from the essential matrix is known as the relative orientation problem.

$$W = \begin{matrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{matrix}$$ is defined.

Singular value decomposition of E gives us $E = ULV^T$.

Then, we choose

$\tau_X = ULWU^T$. Translation vector +/- τ is recovered.

$\Omega = UWV^T, UW^TV^T$.

Magnitude of the translation vector is set to unity by convention. This leads us to 4 possible solutions for rotation and translation. The correct solution is the one in which the 3D points are in front of both the cameras. 3D points corresponding to pairs of 2D corresponding points can be found by triangulation.

The matlab implementation of a simple 3D reconstruction from a pair of images based on the method described above and matlab's documentation on structure from motion is given below :

```
%Reading in images

I1 = imread('Dinosaur/dino1.png');
```

```
I2 = imread('Dinosaur/dino2.png');

figure

imshowpair(I1, I2);

title('Original Images');

%camera parameters

K = transpose([525 0 -360;0 525 -288;0 0 1]);

cameraParams = cameraParameters('IntrinsicMatrix',K);

%%Finding correspondences

% Detect feature points

imagePoints1 = detectMinEigenFeatures(rgb2gray(I1), 'MinQuality', 0.1);

imagePoints2 = detectMinEigenFeatures(rgb2gray(I2), 'MinQuality', 0.1);

% Create the point tracker

tracker = vision.PointTracker('MaxBidirectionalError', 1, 'NumPyramidLevels',
5);

% Initialize the point tracker

imagePoints1 = imagePoints1.Location;

initialize(tracker, imagePoints1, I1);

% Track the points

[imagePoints2, validIdx] = step(tracker, I2);

matchedPoints1 = imagePoints1(validIdx, :);

matchedPoints2 = imagePoints2(validIdx, :);

% Visualize correspondences

figure

showMatchedFeatures(I1, I2, matchedPoints1, matchedPoints2);

title('Tracked Features');
```

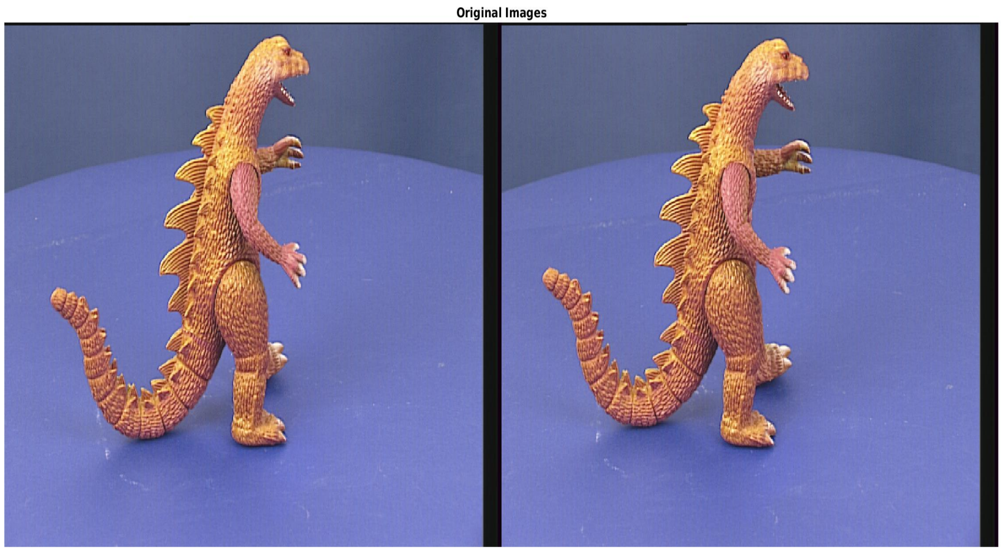
```
%Fundamental matrix computation

[fMatrix, epipolarInliers] = estimateFundamentalMatrix(matchedPoints1,
matchedPoints2, 'Method', 'MSAC');

 % Find epipolar inliers

inlierPoints1 = matchedPoints1(epipolarInliers, :);

inlierPoints2 = matchedPoints2(epipolarInliers, :);

%Rotation and translation estimation

[R, t] = cameraPose(fMatrix, cameraParams, inlierPoints1, inlierPoints2);

%Camera matrices

camMatrix1 = cameraMatrix(cameraParams, eye(3), [0 0 0]);

camMatrix2 = cameraMatrix(cameraParams, R', -t*R');

%% More Points for better reconstruction

% Detect dense feature points

imagePoints1 = detectMinEigenFeatures(rgb2gray(I1), 'MinQuality', 0.01);

% Create the point tracker

tracker = vision.PointTracker('MaxBidirectionalError', 1, 'NumPyramidLevels',
5);

% Initialize the point tracker

imagePoints1 = imagePoints1.Location;

initialize(tracker, imagePoints1, I1);

% Track the points

[imagePoints2, validIdx] = step(tracker, I2);

matchedPoints1 = imagePoints1(validIdx, :);

matchedPoints2 = imagePoints2(validIdx, :);

% Compute the 3-D points
```

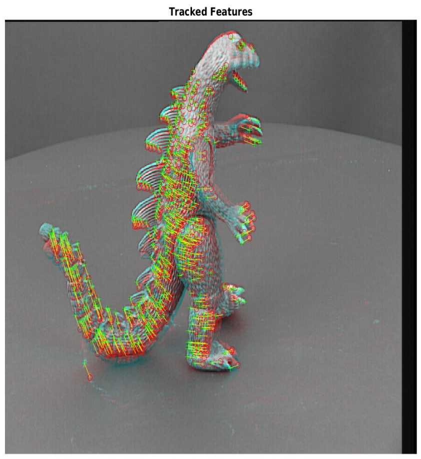
points3D = triangulate(matchedPoints1, matchedPoints2, camMatrix1, camMatrix2);

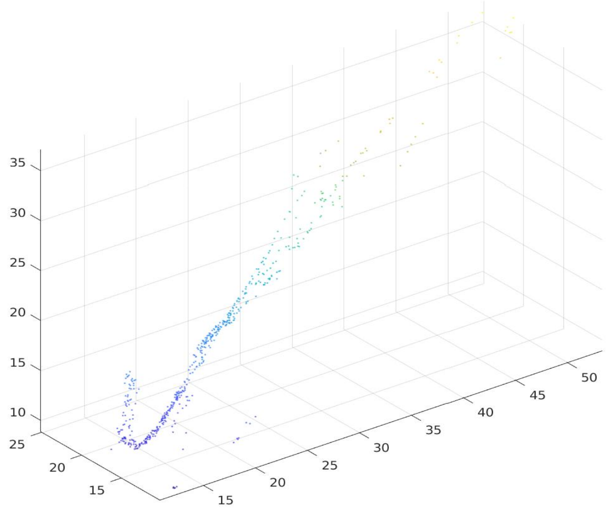
%Visualising the reconstruction

ptCloud = pointCloud(points3D); pcshow(ptCloud);

Reconstruction results :

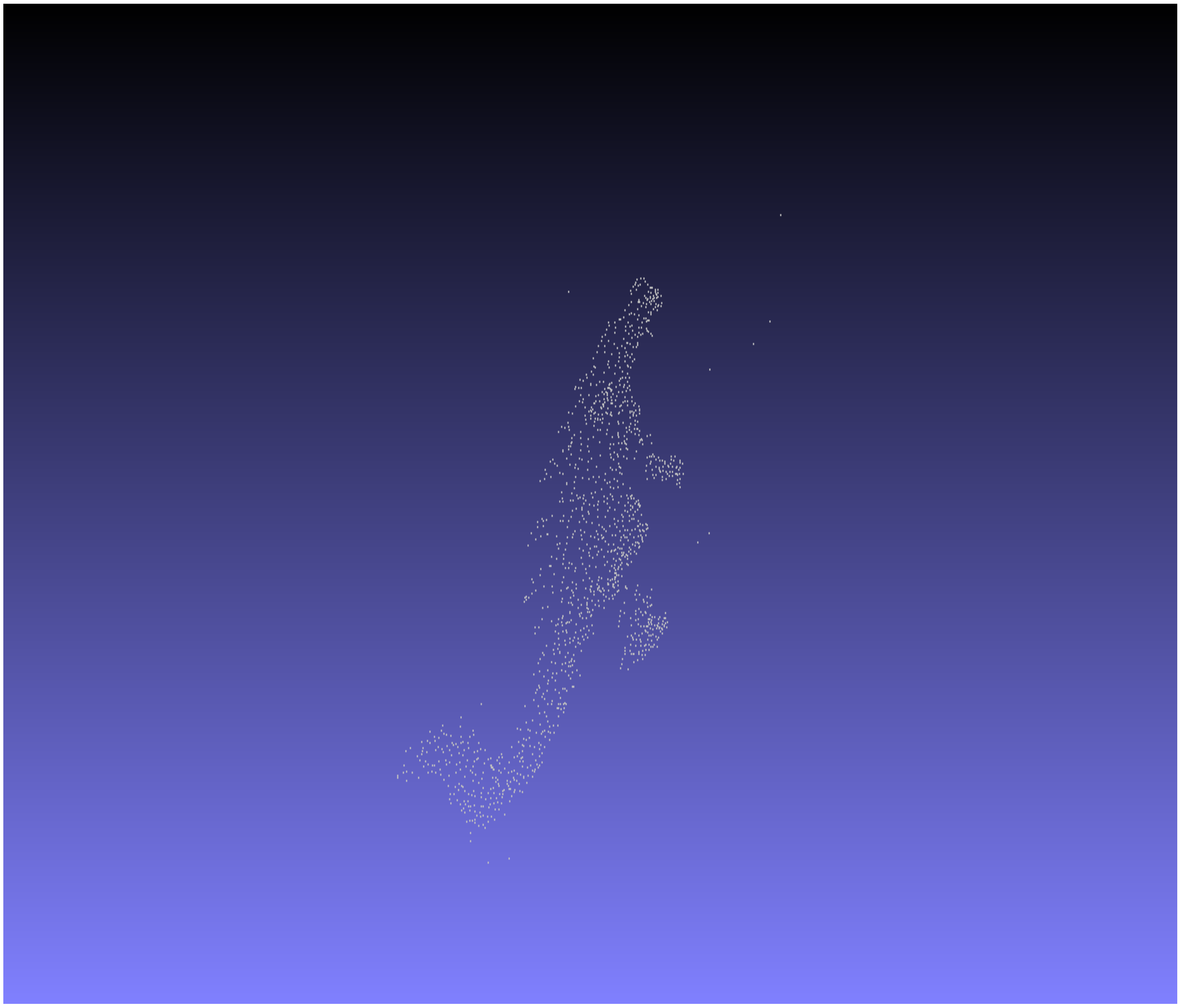






The first image on the top shows the original pair of images which differ by 10 degree rotation. The second image shows the corresponding points between the 2 images. We see that a good number of points have been detected on the body and tail of the dinosaur while very few points have been detected in other parts of the image. This is probably due to the presence of better features(sharper corners capable of producing a large variation when the corresponding window is moved around) in certain parts of the image. The third image shows a visualization of the 3D plot. The tail and the body of the dinosaur can be clearly detected in it. The head is not easily identifiable as the point cloud is sparse there.

A denser reconstruction of the dinosaur using the same pair of images is given below :



In the denser reconstruction, the whole dinosaur is clearly visible. The number of points used in this dense reconstruction is 1380.

A short note on multi-view reconstruction :

Let's say we have multiple images of an object taken from various angles spread over the 360 degree scale. 3D reconstruction of the object from these images is possible if sufficient number of images taken from different views is given. One option for this would be to do a pairwise dense 3D reconstruction and then align all the point clouds using icp and stitch them together to obtain the entire 3D object. Incremental SfM and global SfM are the most commonly adopted methods for 3D reconstruction from multiple images. The reconstruction is refined using a bundle adjustment method.

The dinosaur images have been got from
http://www.robots.ox.ac.uk/~vgg/data/data-mview.html

References :

1. Book - Computer Vision : Models,Learning and inference , Simon J.D. Prince

2. Book – Multiple View Geometry in Computer Vision , Richard Hartley and Andrew Zisserman

3. Paper - Schonberger, J. L., & Frahm, J. M. (2016). Structure-from-motion revisited. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 4104-4113).

4. http://aishack.in/tutorials/harris-corner-detector/

4 Summary of results, limitations and future work

1. Iterative Closest Point

Analysis of rotation angle and axis using an implementation of the iterative closest point algorithm in matlab and point clouds library clearly proves that pcl's implementation yields far better results in terms of both speed and accuracy. In the latter case, the error in rotation angle is 0.6 degrees while the error is 1.4 degrees when the implementation in matlab is used. Ideal datasets yield better results than real datasets. Since this is a double edged problem, ideal datasets help in determining the accuracy of the icp algorithm. Noisy datasets tend to impact the results of real datasets hence good filtering algorithms are necessary. In case of turn-table sequences, the turn-table can jiggle while rotating which too can impact the results. The 'average axis' has been computed using the iterative weighted least squares method and analyzed. Techniques to eliminate outliers will help yield more accurate results.

2. Two view 3D reconstruction

A tutorial on 3D reconstruction using two images has been presented. Better feature matching techniques to generate large number of accurate correspondences will help in achieving a denser reconstruction.