CS6790 Course Project

Divya K Raman,EE15B085 and Sharath Girish,EE15B058

14 May 2018

Paper: Building Rome in a day.
Authors: Sameer Agarwal, Noah Snavely, Ian Simon, Steven M. Seitz, Richard Szeliski

1 Introduction

This paper deals with reconstructing 3D scenes from extremely large collections of photographs from sources such as internet photo sharing sites. These images are unstructured and are not taken from a single camera which makes the reconstruction process hard. The paper uses a collection of parallel distributed matching and reconstruction algorithms designed to minimize serialization bottlenecks. Their results demonstrate that it is possible to reconstruct cities consisting of 150K images in less than a day on a cluster with 500 compute cores.

As a part of this project, we test the code provided by the authors on various data-sets to see how it works and analyze the results. We have additionally done a very basic and simple matlab implementation which works on structured images taken from a simple camera.

2 Analysis of results- bundler toolkit

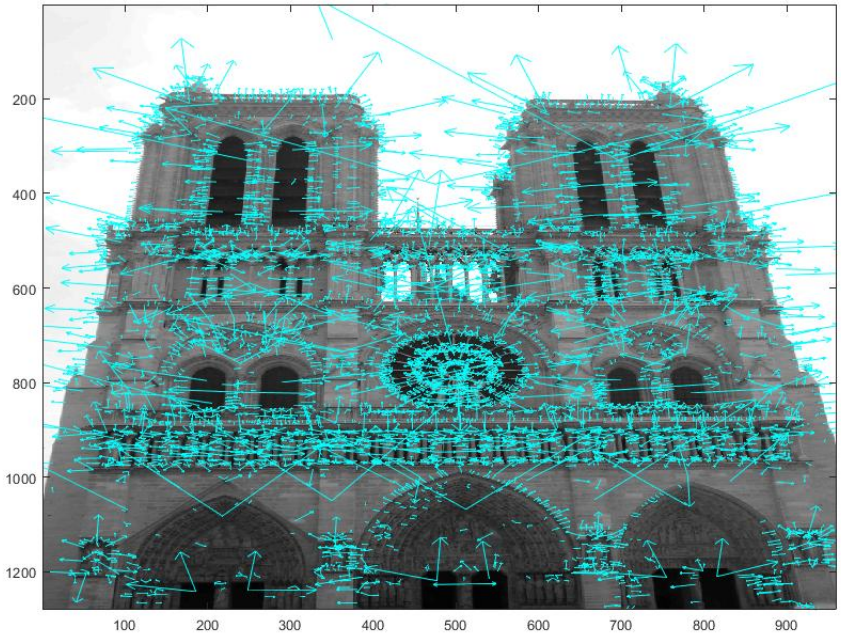
Source: http://www.cs.cornell.edu/ snavely/bundler/
Brief overview of what the code does:

Using the exif files(if available for a particular image), focal length information is extracted. This list of images is stored using the script 'extract_focal.pl'. David Lowe's SIFT Keypoint detector is used to get SIFT feature descriptors from each image. Feature matching is done using kd tree wherein SIFT features of one image are inserted in a kd tree and those of another image are used as query.Nearest neighbour search using kd tree takes O(NlogN) time and hence is quite fast. After the matching, tracks are generated, inconsistent tracks are discarded. Then, sparse bundle adjustment is done to get the camera poses and reconstruct the 3D world points.

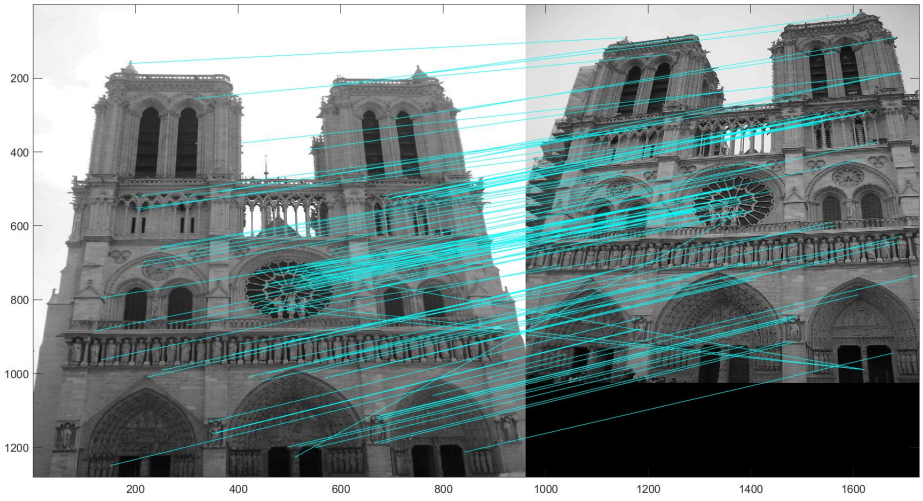
Bundler is basically a SfM system for reconstruction from unordered internet collections. It contains routines for linear algebra techniques(SVD, cholesky decomposition, QR decomposition, eigen values, etc); estimating homographies, fundamental matrix, camera parameters,etc; bundle adjustment package and many other functions required for 3D geometry. All these functions are written in C. Bundler outputs a bundle file called 'bundle_¡n¿.out' containing the current state of the scene(camera poses, rotations, translations, etc.) after each set of images has been registered (n = the number of currently registered cameras). After all possible images have been registered, Bundler outputs a final file named 'bundle.out'. In addition, a "ply" file containing the reconstructed cameras and points is written after each round which can be visualised in meshlab or even in matlab .

2.1 Notre Dame

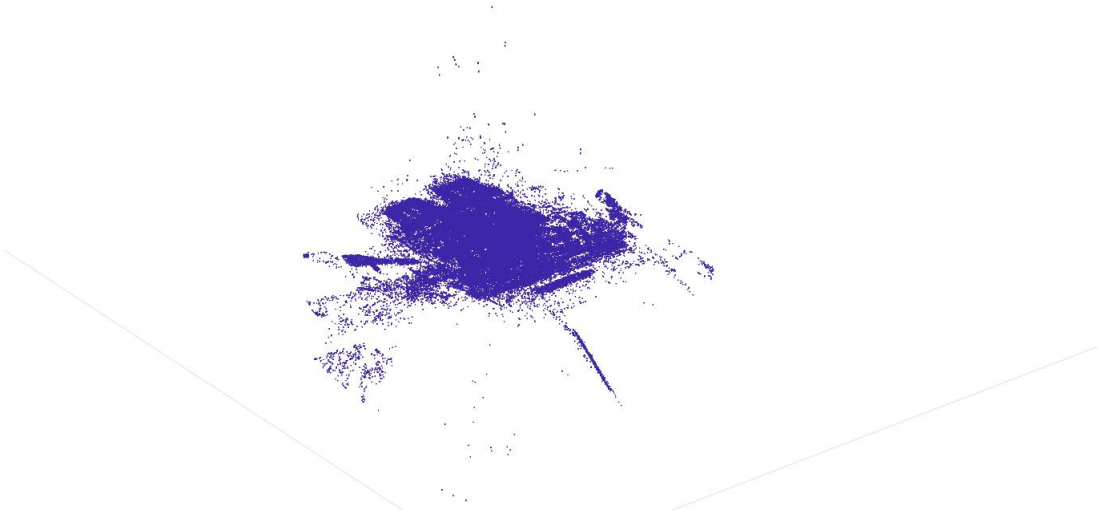
This dataset had 96 images. The reconstruction took around 2 hours.
SIFT features detected for one of the images:
6069 keypoints found:



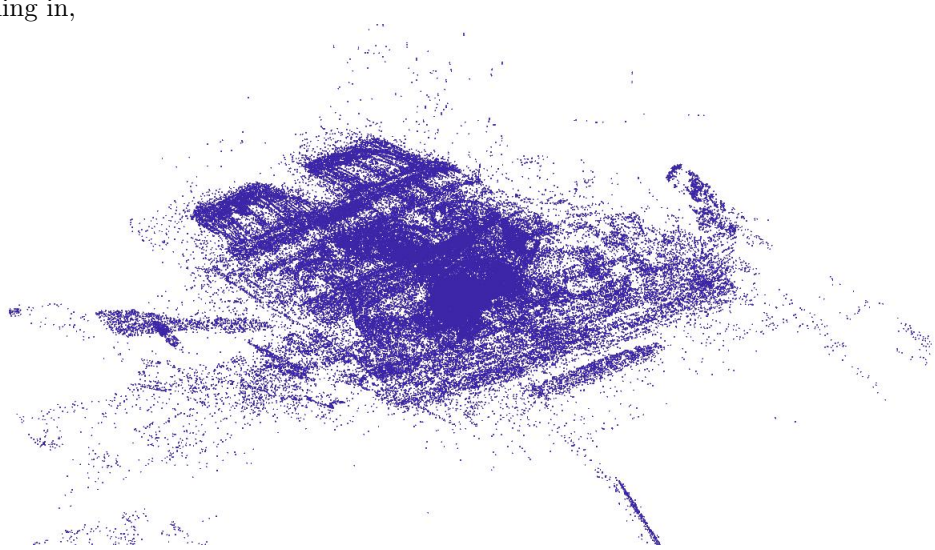
Feature matching between 2 images using SIFT:
6069,6005 keypoints respectively in each image, 115 matches



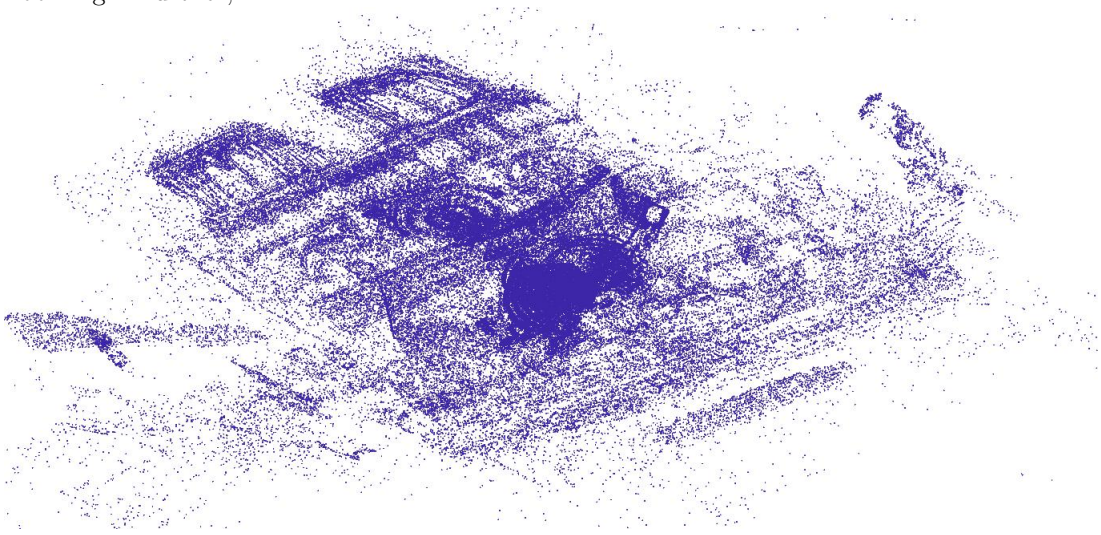
Final 3D reconstruction of Notre Dame:



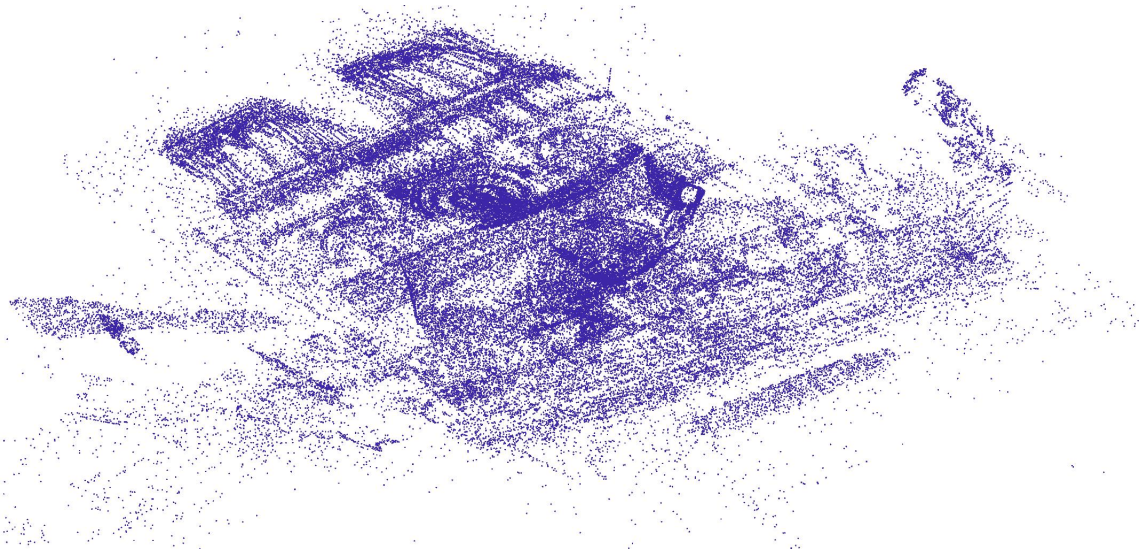
Zooming in,



Zooming in further,



The below image shows the reconstructed scene after the code runs on 46 images.



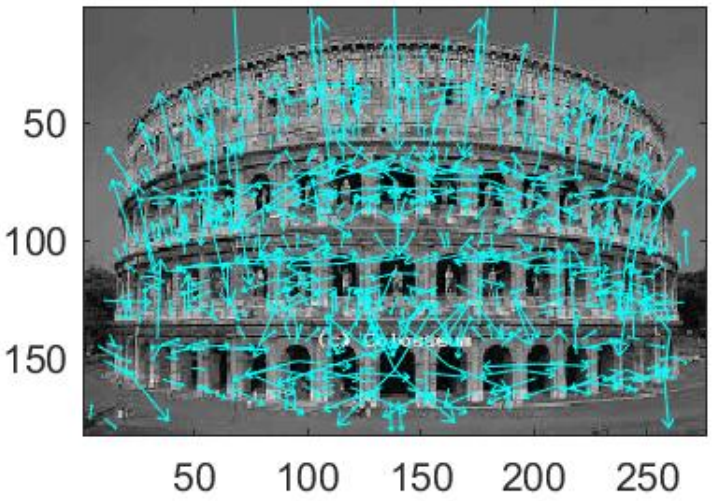
The reconstruction with 96 images is denser than the reconstruction with 46 images. Due to good feature matches and a good number of images taken from a wide view, the reconstruction is pretty good.

2.2 Colosseum of Rome

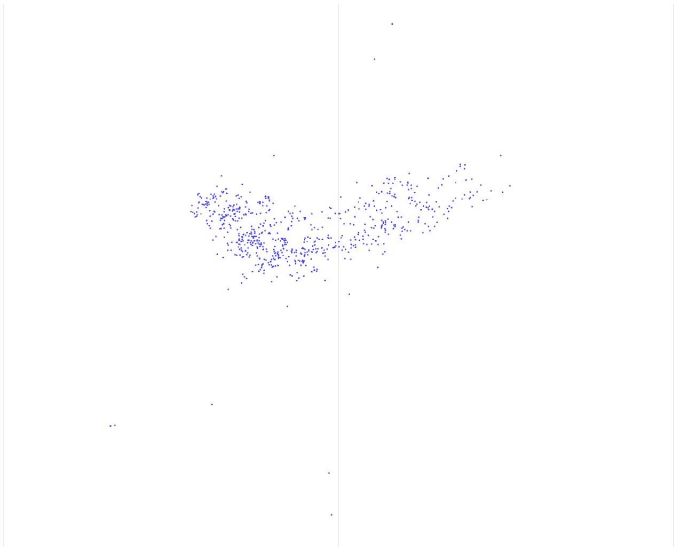
This was run on 80 images.

SIFT features detected for one of the images:

665 keypoints found:



3D reconstruction:



The reconstruction is very sparse though the number of images is about 80. The overall structure of the colosseum has been reconstructed. The fine details are missing which can be reconstructed by better feature matching and a better dataset. Images were very similar to each other and hence a lot of new points weren't triangulated leading to a sparse reconstruction. For dense reconstruction, we need a lot of images taken from various different angles.

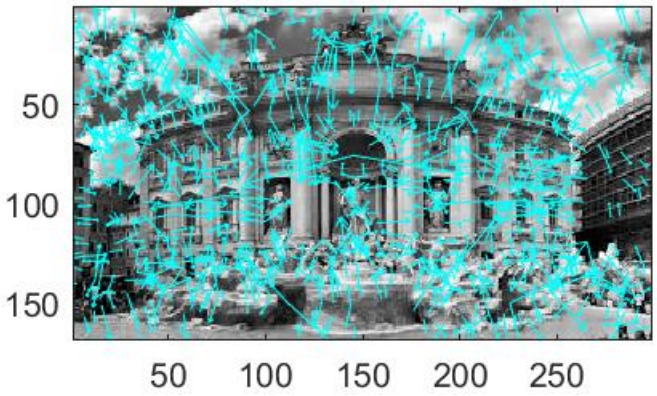
Also, running SIFT on colosseum returns very less feature keypoints and number of feature matches between two images obtained is very less. All this makes the reconstruction very sparse.

For denser points, the authors recommend us to use a software package called PMVS2 written by Yasutaka Furukawa which runs dense multi-view stereo. The pipeline is to run Bundler to get camera parameters, use the provided Bundle2PMVS program to convert the results into PMVS2 input, then run PMVS2.

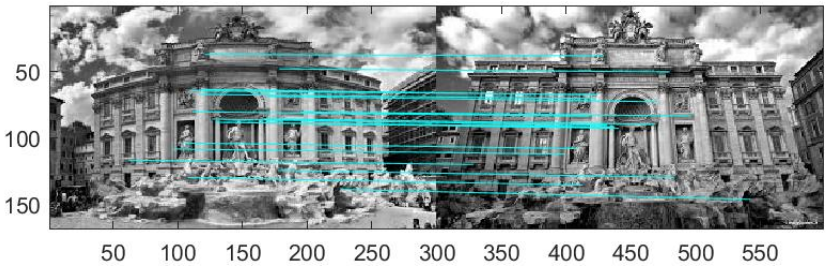
2.3 Trevi Fountain

The dataset used had 19 images. These images were manually handpicked from the internet. Not many distinct yet similar looking images could be found and hence the dataset is very small.

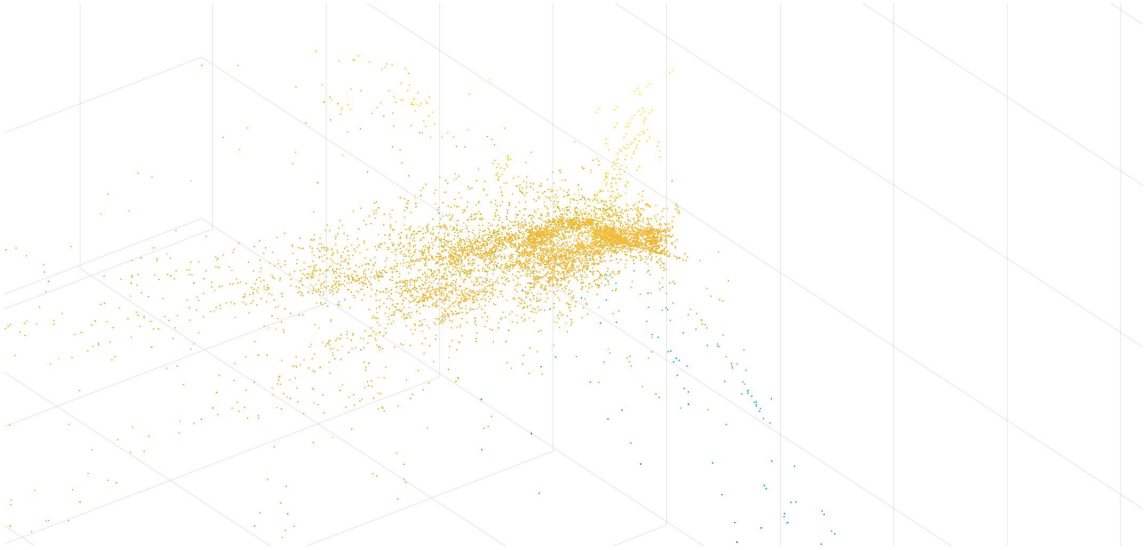
SIFT features detected for one of the images:
932 keypoints found:



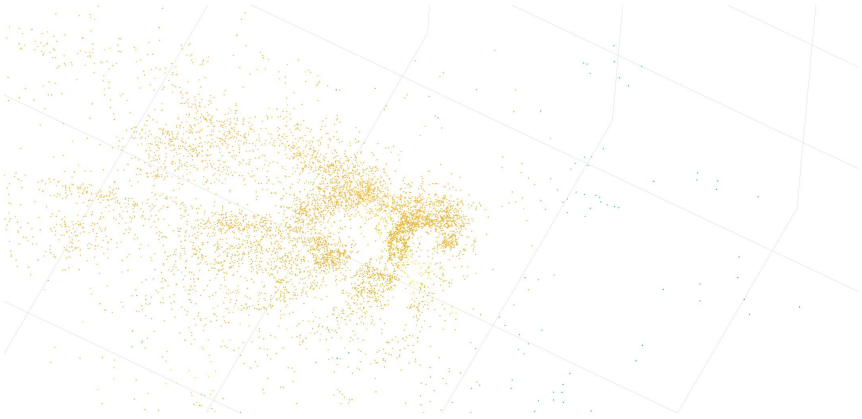
Feature matching between 2 images using SIFT:
932,868 keypoints respectively in each image



Number of matches found is very less.
3D reconstruction of Trevi fountain:



Zooming in and rotating,



Reconstruction is sparse. More images will lead to a denser reconstruction. The overall structure seems to have been reconstructed. The finer details are missing.

Analysis and Interpretation:

More images lead to denser and better reconstruction. Also, better feature matching leads to a better and denser reconstruction, wherein the finer details also get reconstructed. This code lets us use images taken from different cameras and different angles to reconstruct entire cities. The authors used multiple nodes and GPUs run the code on millions of images to reconstruct entire cities.

SIFT performs decently well even on images that are quite far apart. It is rotation and scale invariant and is hence better than many other descriptors.SIFT is based on the Histogram of Gradients i.e. the gradients of each pixel in the patch need to be computed which takes time. The use of kd tree for matching also makes the algorithm computationally efficient.

Sparse bundle adjustment is done to find the camera poses and 3D world points. The algorithm's optimisation problem is solved using the Levenberg-Marquardt Algorithm.

Visualisation of 3D point clouds can be done using Meshlab or matlab(using pcread an pcshow functions).

Notre Dame gives the best results. This dataset was provided by the authors and had a huge number of distinct yet similar images. Feature matches were very good and a dense 3D reconstruction was obtained. Feature matches in colosseum and trevi fountain needs to be improved. Better datasets need to be collected for trevi fountain and colosseum for a good 3D reconstruction.

3. Basic Matlab implementation

We attempted a basic and simple matlab implementation of 3D reconstruction from multiple views. It works only on a pair of images taken from a single camera.

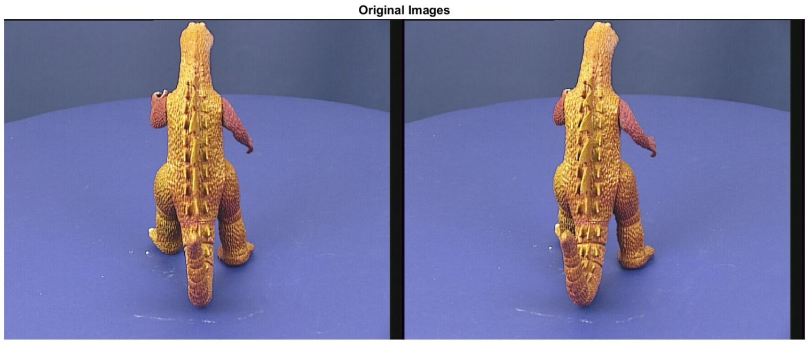
The code:

```
%Matlab code for 3D reconstruction given a pair of views.
%This code is an adaptation of the StructurefromMotion code by matlab.

I1 = imread('Datasets/Dino/dino30.jpg');
I2 = imread('Datasets/Dino/dino31.jpg');
figure
imshowpair(I1, I2, 'montage');
title('Original Images');

%camera parameters
K = transpose([525 0 -320;0 525 -240;0 0 1]);
cameraParams = cameraParameters('IntrinsicMatrix',K);


%%Finding correspondences
% Detect feature points
imagePoints1 = detectMinEigenFeatures(rgb2gray(I1), 'MinQuality', 0.01);
imagePoints2 = detectMinEigenFeatures(rgb2gray(I2), '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, :);

% Visualize correspondences
figure
showMatchedFeatures(I1, I2, matchedPoints1, matchedPoints2);
title('Tracked Features');

%Fundamental matrix computation
[fMatrix, epipolarInliers] = estimateFundamentalMatrix(matchedPoints1, matchedPoints2, 'Meth
```

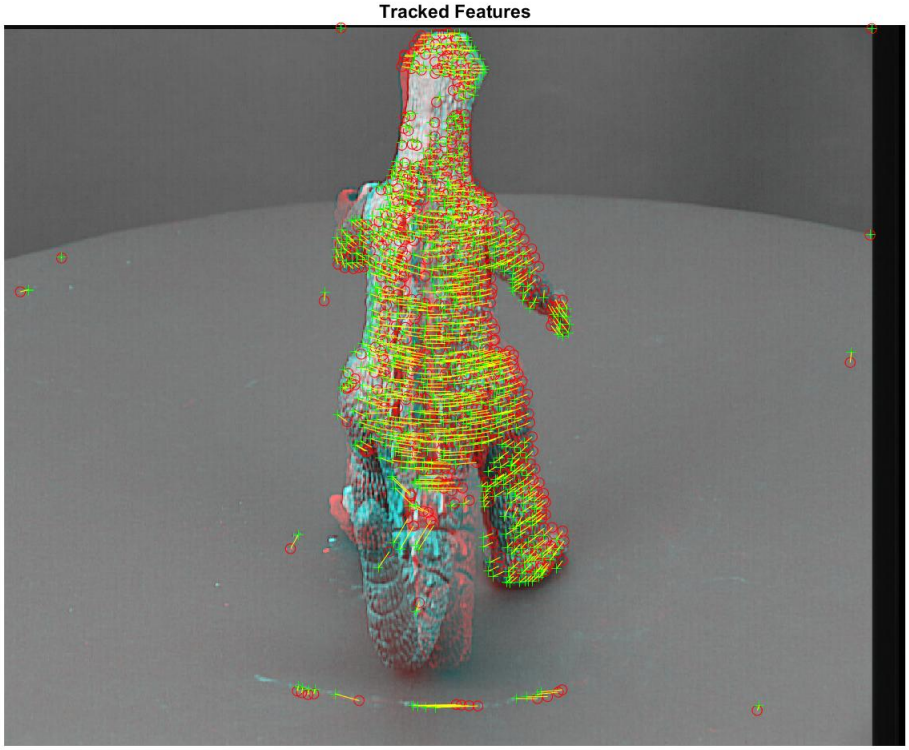
```
% Find epipolar inliers
inlierPoints1 = matchedPoints1(epipolarInliers, :);
inlierPoints2 = matchedPoints2(epipolarInliers, :);
figure
showMatchedFeatures(I1,I2,inlierPoints1,inlierPoints2);
title('Inlier Points');

%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);%change minquality a

% 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);
```

The results on a pair of dinosaur images(turn table sequence, captured by rotating the turn table in 10 degree steps):

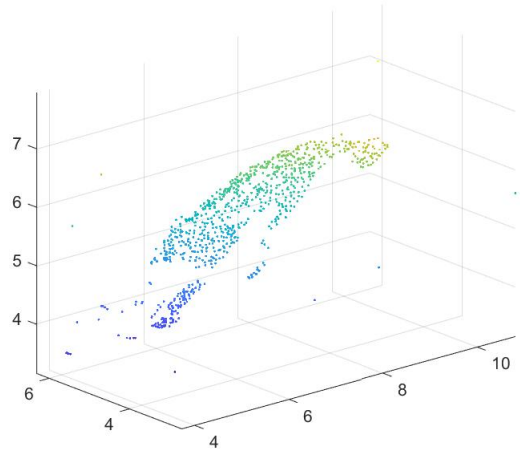
Original images:



Feature matches:



3D reconstruction:



Here, feature matches were found using detectMinEigenFeatures function in matlab. The detectMinEigenFeatures function uses the minimum eigenvalue algorithm developed by Shi and Tomasi to find feature points. Features are then matched, the fundamental matrix is computed. This fundamental matrix and the intrinsic parameters of the camera are used to find the relative pose between the 2 cameras which is used to triangulate points and reconstruct the scene.

The ideal angle of rotation between any two consecutive pairs of images is expected to be 10 degrees. Also, we expect zero translation as this is a turn table sequence.

Adding another view and running bundle adjustment to refine and get accurate camera poses reports an error. Furthermore, this code works only on structured images taken from a single camera. Also, the intrinsic camera parameters need to be explicitly passed to the code. These are the aspects we need to work on to make the code better.