CMSC660 Scientific Computing

Final Project

Affine Structure from Motion

Angjoo Kanazawa

December 19th, 2011

1 Introduction

Over the last few decades, advances in Scientific Computing has impacted almost every area of Science and Engineering. Especially in Computer Science, it has become a necessary and critical tool for anyone involved in high-level research. Computer Vision is one of the quintessential examples of research areas that heavily build upon methods studied in Scientific Computing. Numerical methods are very prevalent in Computer Vision where the primary interest lies in the analysis and understanding of images which are represented in numerical matrices. This project explores applications of computational algorithms explored in Scientific Computing via tackling the problem of 3D reconstruction of an object from a stream of images.

The 3D reconstruction problem consists of a series of challenges starting from developing the camera model and feature representation of the image, tracking such features over the image sequences , and finally reconstructing the 3D geometry from the tracked points. The application of algorithms explored in Scientific Computing is ubiquitous in all of these steps.

The main challenge of recovering 3D geometry of objects and camera motion simultaneously from a set of tracked points is referred to as the Structure from Motion (SfM) problem. The solution to the problem has a wide range of application in 3D modeling, virtual and augmented reality models in computer graphics, camera calibration and many more. It is a well studied problem with various approaches; this project focuses on the *factorization* method proposed by Tomasi and Kanade [7] under the orthographic camera projection model. The algorithm provides a numerically stable closed from optimal solution via the singular-value decomposition technique under certain conditions. [1, p. 435]

This project follows the Project 4 of Derek Hoiem's CS 543/ECE 549 course at the University of Illinois at Urbana-Champaign: project description. Section 2 reviews the basics of camera projection models, notations and the problem statement. Section 3 presents the SfM pipeline in detail. Section 4 touches upon methods to further refine the result and we conclude in Section 5. All intermediate results for the experiment is discussed in appropriate sections.

2 Background

2.1 Orthographic Projection

A camera model projects world point onto an 2D image plane. An affine camera, often used for its simplicity, preserves up to affine transformation of world points to image points. Basically it is a linear mapping of world points followed by a translation, where the points can rotate, scale, and translate but parallelism is preserved in the projection, i.e. parallel lines remain parallel.[5, p.38]. An *orthographic camera model* is a specific type of affine camera where the world points are projected in parallel onto the image plane and the depth information

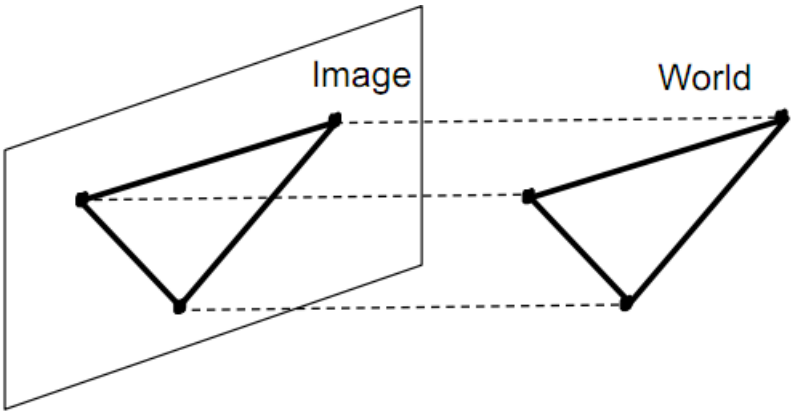

Figure 1: Orthographic Projection

of the world Z, is simply ignored. Mathematically,

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} R \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} + \boldsymbol{t}$$

$$\boldsymbol{x} = M\boldsymbol{X} + \boldsymbol{t}$$

Where $M = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} R \in \mathbf{R}^{2\times 3}$ is referred to as the projection matrix, and R is a 3 by 3 matrix representing the affine motion (rotation, translation, or both) of the camera and t is the displacement vector [1, p.172]. The rows of M, $\boldsymbol{i}, \boldsymbol{j}$, are orthonormal unit vectors corresponding to the x and y-axis of the image plane respectively. In orthography, the location of the camera is $\boldsymbol{k} = \boldsymbol{i} \times \boldsymbol{j}$. In another words M encapsulates the motion of the camera under orthography.

2.2 Notations and Assumptions

A "world point" refers to a point in the 3D coordinate system, $\boldsymbol{X} = (X, Y, Z)^T$. An "image point" refers to a point projected onto an image plane in the 2D coordinate system, $\boldsymbol{x} = (x, y)^T$.

$I(x, y)$ denotes the pixel intensity value of image I, and $I(x, y, t)$ denotes the pixel value of image I at time t. ∇I is the image gradient $\begin{pmatrix} I_x & I_y \end{pmatrix}$ and H is the image hessian $\begin{pmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{pmatrix}$.

In this project, all projections are orthographic 2.1 and the camera motion is affine. The images are assumed to have a mean-zero Gaussian noise. The world projected on the image sequences is rigid and the pixel intensities of images over sequences are constant i.e. all frames were taken under a static environment with no brightness changes. Also for the experiment we assume that there is no occlusion.

2.3 Problem Statement

Given F frames of sequential images (videos), obtain a trajectory of P image points for all F: $\{x_{fp} = (u_{fp}, v_{fp})^T | f = 1, \ldots, F, p = 1, \ldots, P\}$. Then solve for X_p, the world coordinate of all P points from the observations s.t. the distance between the real world points and the approximated world points is minimized.

For the experiments, the hotel image stream used in the original Tomasi and Kanade paper was used. Frames 1, 15, 30, and 45 are in figure 2.3.

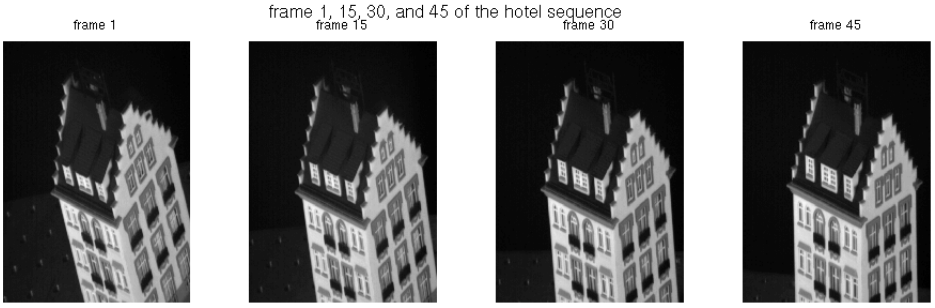

Figure 2: frames 1, 15, 30, and 45 of the hotel image stream

3 Structure for Motion

There are three main components for 3D reconstruction of a scene from an image stream. First we need to select a subset of "interesting" pixels from the original frame. These points are then tracked through out the rest of the sequence. This set of point correspondence across all frames is then fed into the factorization algorithm. Appropriate keypoint selection and accurate feature tracking are critical for a successful 3D reconstruction.

3.1 Keypoint Selection

It's neither computationally possible nor optimal to solve the world coordinate for every pixel of image sequences. Many of the features in an image are repetative and undistinctive that it is not helpful to have arbitrary many point correspondences. Instead, we need to select keypoints in the image that are reliable, salient, and meaningful.

Harris corner detector is an optimal feature selection for the tracker that is used for this project. (It's optimality is discussed in section 3.2). The idea is that if a pixel is "interestin", we should easily recognize it by looking through a small window, where shifting a window in any direction would give a large change in intensity. i.e. we accept a point $\boldsymbol{x}$ if SSD of the displacement of the window by some distance $\boldsymbol{d} = (u, v)^T$, is large in all direction:

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

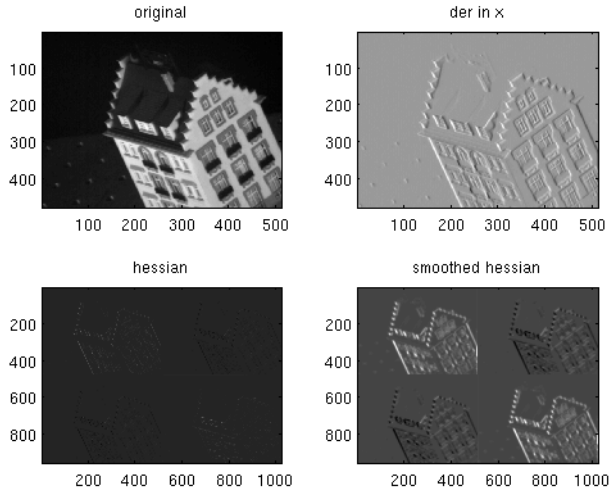

Figure 3: Plot of the image gradients of the first frame

Assuming $\boldsymbol{d}$ is small, the taylor series expansion of I is

$$I(x+u, y+v) \approx I(x,y) + I_x u + I_y v + \mathcal{O}(d^T d)$$

where $I_x = \frac{\partial I}{\partial x}$, $I_y = \frac{\partial I}{\partial y}$.

Then,

$$\begin{aligned} E(u,v) &= \sum_{(x,y)\in W} [I(x+u, y+v) - I(x,y)]^2 \\ &= \sum_{(x,y)\in W} [I(x,y) + I_x u + I_y v - I(x,y)]^2 \\ &= \sum_{(x,y)\in W} (\begin{pmatrix} I_x & I_y \end{pmatrix} \cdot \begin{pmatrix} u \\ v \end{pmatrix})^2 \\ &= \begin{pmatrix} u & v \end{pmatrix} \begin{pmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} \\ &= d^T H d. \end{aligned}$$

Since the two eigenvalues of H ,λ_1, λ_2, denotes the amount of change in the direction of its corresponding eigenvector, we accept a window if

$$min(\lambda_1, \lambda_2) > \tau,$$

where τ is predefined threshold [4]. Figure 3 illustrates the components of the Hessian of frame one.

Using the harris detector, 518 points were found on frame 1 of the hotel sequence 4.

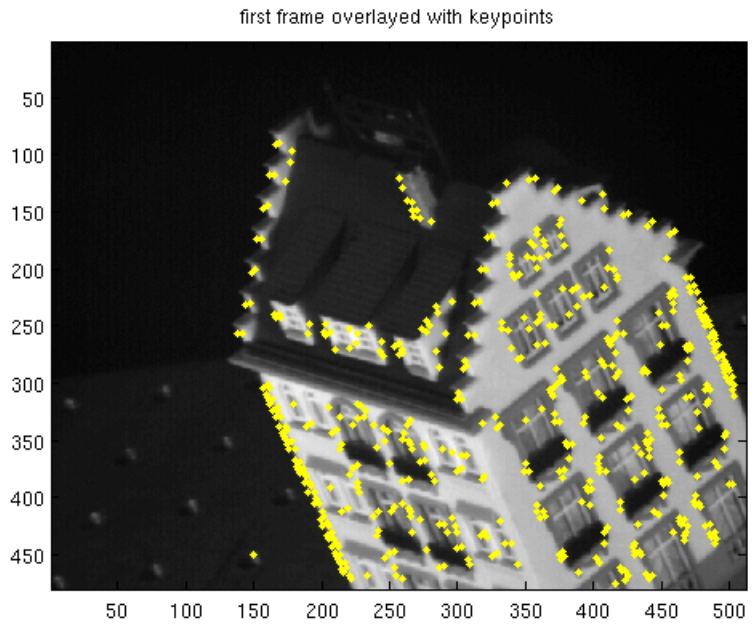

Figure 4: First 518 keypoints found on frame 1

3.2 Feature Tracking

Now using the detected keypoint, we track these points over all frames using the Kanade-Lucas-Tomasi (KLT) Tracker [6]. The basic idea of the KLT tracker is that although image intensities change as camera moves, images taken at near time are strongly related to each other because they tend to refer to the same scene given the assumption of local brightness constancy. This means that under an ideal static environment, image at time $t+1$ can be obtained by moving every pixel in the at time image t by a suitable amount [4].

3.2.1 Estimating Camera Motion

This displacement can be a translation, an affine mapping, or the combination of both. Here we only consider translation because the world is rigid. Mathematically, given $\boldsymbol{x}$ at frame t, we want to find a displacement vector, $\boldsymbol{d} = (u, v)^T$, that minimizes the dissimilarity

$$I(\boldsymbol{x} + \boldsymbol{d}, t + 1) - I(\boldsymbol{x}, t) = 0 \tag{1}$$

The taylor expansion of

$$I(\boldsymbol{x} + \boldsymbol{d}, t + 1) = I(x + u, y + v, t + 1) \approx I(x, y, t) + I_x u + I_y v + I_t + \mathcal{O}(d^T d).$$

Where I_t is the temporal gradient (for time) $I_t(\boldsymbol{x}) = I(\boldsymbol{x}, t) - I(\boldsymbol{x} + \boldsymbol{d}, t + 1)$.

Substituting that back to (1), we obtain a system of linear equation:

$$\begin{aligned} 0 &\approx I(x,y,t) + I_x u + I_y v + I_t - I(x,y,t) \\ &= \nabla I(\boldsymbol{x}) \cdot \boldsymbol{d} + I_t(\boldsymbol{x}) \end{aligned}$$

But here we have 2 unknowns and only one constraint. In fact even with the brightness constancy assumption, it is difficult to track a single point unless the point is extremely distinctive [6]. Therefore we consider minimizing the dissimilarity within a small window (typically around 15 x 15) and obtain an overdetermined linear system of equations:

$$\sum_{\boldsymbol{x} \in W} \nabla I(\boldsymbol{x}) \cdot \boldsymbol{d} = -\sum_{\boldsymbol{x} \in W} I_t$$

$$A\boldsymbol{d} = -\boldsymbol{t}$$

Using the normal equations, we solve this least linear squares problem:

$$A^T A \boldsymbol{d} = -A^T \boldsymbol{t} \tag{2}$$

$$\begin{pmatrix} I_x(x_1) & \dots & I_x(x_{|W|}) \\ \vdots & & \vdots \\ I_y(x_1) & \dots & I_y(x_{|W|}) \end{pmatrix} \begin{pmatrix} I_x(x_1) & \dots & I_y(x_1) \\ \vdots & & \vdots \\ I_x(x_{|W|}) & \dots & I_y(x_{|W|}) \end{pmatrix} \boldsymbol{d} = - \begin{pmatrix} I_x(x_1) & \dots & I_y(x_1) \\ \vdots & & \vdots \\ I_x(x_{|W|}) & \dots & I_y(x_{|W|}) \end{pmatrix} \begin{pmatrix} I_t(x_1) \\ \vdots \\ I_t(x_{|w|}) \end{pmatrix} \tag{3}$$

$$\begin{pmatrix} \sum_{\boldsymbol{x} \in W} I_x^2 & \sum_{\boldsymbol{x} \in W} I_x I_y \\ \sum_{\boldsymbol{x} \in W} I_y^2 & \sum_{\boldsymbol{x} \in W} I_y I_x \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = - \begin{pmatrix} \sum_{\boldsymbol{x} \in W} I_x I_t \\ \sum_{\boldsymbol{x} \in W} I_y I_t \end{pmatrix} \tag{4}$$

$$Z\boldsymbol{d} = \boldsymbol{e} \tag{5}$$

Since solving for $\boldsymbol{d}$ by the method above requires us to know I_t, which requires an initial displacement $\boldsymbol{d}_0$, so we solve for the vector iteratively. Namely, we start with $\boldsymbol{d}_0 = (0,0)^T$ to compute $I_{t_0} = I(\boldsymbol{x}, t+1) - I(\boldsymbol{x}, t)$, then solve for $\boldsymbol{d}_1$ using I_{t_0}. We iterate this until $||\boldsymbol{d}_k - \boldsymbol{d}_{k+1}|| \leq \delta$, δ some threshold.

Here, points that go out of the frame at one point in the sequence are discarded, but there are methods to alleviate for points that disappear and comeback through out the sequence [3]. Figure 5 is the tracked path of 30 random keypoints over the entire sequence. The magenta start is the position of points at frame 1.

3.2.2 Numerical Stability

Can we solve (5) reliably? It seems like in practice, it's not unusual to solve for $\boldsymbol{d}$ simply by taking the pseudo-inverse of Z, which is known to be atrocious in the Scientific Computing community. But there is a reason why here it is okay to be less careful about solving for $\boldsymbol{d}$ and it leads to the discussion of the optimality of our keypoint selector.

For (5) to be stable, w need Z to be well-conditioned. The condition number of Z is $\kappa(Z) = ||A||||A^{-1}||$. Here, Z is symmetric and positive definite (because it's the Hessian), so the condition number is also the ratio of the smallest to largest eigenvalue. Here it's $\frac{\lambda_1}{\lambda_2}$. Note that our criteria for choosing our keypoint

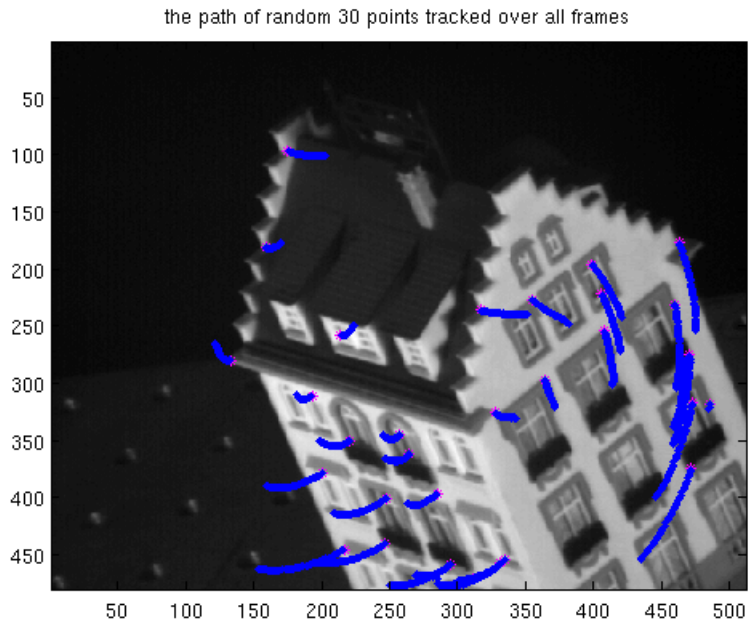

Figure 5: Plot of the random 30 points tracked over all frames

was $\min(\lambda_1, \lambda_2) > \tau$. So we are assured that the smaller eigenvalue is sufficiently large. Since we're dealing with images with a maximum possible pixel value, we are also assured that the larger eigenvalue will not be arbitrarily large. So with this keypoint selection, we're guaranteed to have a well conditioned Z. This is why in practice it is okay to use the pseudo-inverse of Z. In the experiment, the average condition number was around 2. This means in MATLAB solving for d using the `mldivide` operator is more than sufficient.

3.3 The Factorization Method

3.3.1 Preprocessing

As discussed in 2.1, orthographic camera model projects the world points onto the image plane as a linear mapping followed by a translation: $\boldsymbol{x} = M\boldsymbol{X}+t$. Our goal is to esimate the camera motion M_f and t_f for each frame and the world points, X_p, of all tracked image points, so that the distance of the estimated image point from these parameters and the measured image point is close. i.e. our minimization problem is:

$$\min_{M_f, t_f, X_p} \sum_f \sum_p ||\boldsymbol{x}_{fp} - (M_f \boldsymbol{X}_p + \boldsymbol{t}_f)||^2 \tag{6}$$

Taking the derivative of (6) with respect to t_f and setting it to 0 gives us $t_f = \bar{x}_f - M_f \bar{X}_p$, where $\bar{x}_f = \frac{1}{P}\sum_p x_{fp}$ and $\bar{X}_p = \frac{1}{P} X_p$ i.e. the mean of all points in image f. But the origin of the world point is arbitrary so we can just

set that to 0, which gives us $t_f = \bar{x}_f$. This also agrees with the fact that affine camera maps the centroid of 3D world points to the centroid of the projected image [1, p438]. This means that if we center all image points in fth frame, $t_f = 0$ and we can remove the translation term from the model. Now our objective is refined to:

$$\min_{M_f, t_f, X_p} \sum_f \sum_p ||\boldsymbol{x}_{fp} - M_f \boldsymbol{X}_p||^2 \tag{7}$$

3.3.2 Fundamentals

The key idea of the factorization algorithm is that image *measurements* can be decomposed into the product of two separate factors: *motion* and *shape*.

Using the set of P tracked corresponding points, we can represent the image sequence by a $2F \times P$ *measurement matrix* , where we stack the first and the second component of $\boldsymbol{x}$ for all frames:

$$W = \begin{pmatrix} x_{11} & \cdots & x_{1P} \\ \vdots & & \vdots \\ x_{F1} & & x_{FP} \\ y_{11} & \cdots & y_{1P} \\ \vdots & & \vdots \\ y_{F1} & & y_{FP} \end{pmatrix}$$

$\boldsymbol{x}_{ij} = (x_{ij}, y_{ij})^T$ is the j-th point in the i-th frame.

Recall that camera motion under orthography is

$$M_f = \begin{pmatrix} i_f \\ j_f \end{pmatrix}$$

where $i_f, j_f \in \mathbf{R}^3$, a pair of orthonormal unit vectors corresponding to the x, y-axis of the image plane respectively. These vectors over F frames are collected into a *motion matrix* $M \in \mathbf{R}^{2F \times 3}$

$$M = \begin{pmatrix} i_1^T \\ \vdots \\ i_F^T \\ j_1^T \\ \vdots \\ j_F^T \end{pmatrix}$$

We let $s_p = \boldsymbol{X}_p = (X_p, Y_p, Z_p)^T$ be the 3D coordinates of feature p in the fixed world point with the same origin. These vectors are collected into a *shape matrix* $S \in \mathbf{R}^{3 \times P}$ s.t. $S = \begin{pmatrix} s_1 & \cdots & s_p \end{pmatrix}^T$. Using this notation, for a single frame we get

$$\begin{pmatrix} x_{fp} \\ y_{fp} \end{pmatrix} = \begin{pmatrix} i_f^T \\ j_f^T \end{pmatrix} \begin{pmatrix} X_p \\ Y_p \\ Z_p \end{pmatrix}$$

$$w_{fp} = M_f S_p$$

So for all frames, we have the equation

$$W = MS.$$

Our goal is to estimate W by $\hat{M}$ and $\hat{S}$, where $\hat{W} = \hat{M}\hat{S}$. Now our objective can be re-written as a least squares problem:

$$\min_{M,S} ||W - \hat{M}\hat{S}||^2 \tag{8}$$

[3].

3.3.3 Rank Theorem

Note that W is the product of a $2F$ by 3 motion matrix and 3 by P shape matrix, therefore under an ideal noise free environment, W is at most rank 3 [7]. Assuming $2F \geq P$, we can achieve the least squares approximation by factoring W by SVD:

$$W = U\Sigma V^T$$

Because of the rank theorem, the diagonal of Σ has at most three nonzero largest singular values in the first three entries. So the closest approximation of W, even with noise, is obtained by taking the three greatest singular values of W with the corresponding left and right eigenvectors. Namely,

$$\begin{aligned} W &\approx U_{2F\times 3}\Sigma_{3\times 3}V^T_{3\times P} \\ &= \begin{pmatrix} u_{1,1} & u_{1,2} & u_{1,3} \\ \vdots & & \vdots \\ \vdots & & \vdots \\ u_{2F,1} & u_{2F,2} & u_{2F,3} \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{pmatrix} \begin{pmatrix} v_{1,1} & \cdots & \cdots & v_{1,p} \\ v_{2,1} & \cdots & \cdots & v_{2,p} \\ v_{3,1} & \cdots & \cdots & v_{3,p} \end{pmatrix} \end{aligned}$$

By using SVD, the factorization algorithm achieves numerical stability and we are guaranteed to converge to the global minimum of (8) [2].

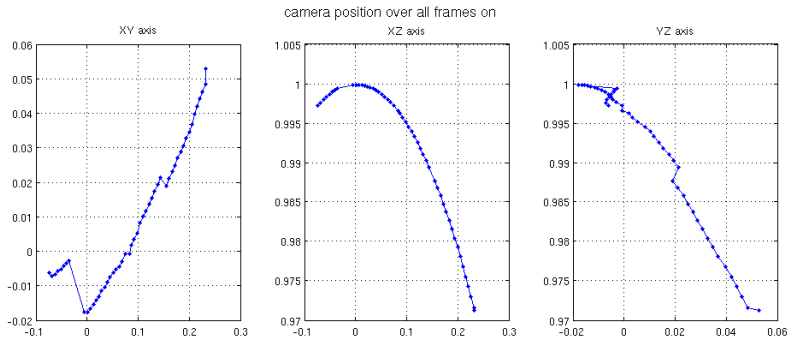

Figure 6: Plot of camera path from each dimension

4 Eliminating Affine Ambiguity

One possible solution to estimate $\hat{M}$ and $\hat{S}$ is to choose $\hat{M} = U_{2F\times 3}\Sigma_{3\times 3}^{1/2}$ and $\hat{S} = \Sigma_{3\times 3}^{1/2}V_{3\times P}^T$, or $\hat{M} = U_{2F\times 3}$ and $\hat{S} = \Sigma_{3\times 3}V_{3\times P}$, because either way we get $\hat{W} = \hat{M}\hat{S} = U_{2F\times 3}\Sigma_{3\times 3}V_{3\times P}^T$

The approximation of $\hat{M}$ and $\hat{S}$ has such ambiguities, since an arbitrary 3 by 3 invertible matrix Q may be inserted in the decomposition as $\hat{W} = (\hat{M}Q)(Q^{-1}\hat{S})$, i.e. the approximation is unique only up to an affine transformation. We can upgrade this affine approximation to a metric approximation by imposing the metric information obtained from the motion matrix M to solve for Q.

Let

$$L = QQ^T = \begin{pmatrix} l_1 & l_2 & l_3 \\ l_2 & l_4 & l_5 \\ l_3 & l_5 & l_6 \end{pmatrix}.$$

Recall that $\hat{M}$ is made of orthonormal unit vectors where the first F entries of $\hat{M}$, $\hat{i}_f \in \mathbf{R}^3$, is orthogonal to the corresponding second F entries of M, $\hat{j}_f \in \mathbf{R}^3$. Since $M = \hat{M}Q$, the corresponding rows of $\hat{M}$ must satisfy equations

$$\begin{aligned} \hat{i}_f^T L \hat{i}_f &= 1 \\ n\hat{j}_f^T L \hat{j}_f &= 1 \\ \hat{i}_f^T L \hat{j}_f &= 0. \end{aligned}$$

Stacking up elements of L into a vector and re-writing the appropriate terms we get an over-determined system of linear equations $G\boldsymbol{l} = \boldsymbol{c}$, where

$$G = \begin{pmatrix} f(\boldsymbol{i}_1, \boldsymbol{i}_1) \\ \vdots \\ f(\boldsymbol{i}_F, \boldsymbol{i}_F) \\ \vdots \\ f(\boldsymbol{j}_1, \boldsymbol{j}_1) \\ \vdots \\ f(\boldsymbol{j}_F, \boldsymbol{j}_F) \\ \vdots \\ f(\boldsymbol{i}_1, \boldsymbol{j}_1) \\ \vdots \\ f(\boldsymbol{i}_F, \boldsymbol{j}_F) \end{pmatrix} \in \mathbf{R}^{3F\times 6}$$

and $\boldsymbol{c}$ is a vector whose first $2F$ entries are 1 and the last F entries are 0, and $f(\boldsymbol{i}_f, \boldsymbol{j}_f) = [i_{f1}j_{f1}, i_{f1}j_{f2} + i_{f2}j_{f1}, i_{f1}j_{f3} + i_{f3}j_{f1}, i_{f2}j_{f2}, i_{f2}j_{f3} + i_{f3}j_{f2}, i_{f3}j_{f3}]$.

We can solve for $\boldsymbol{l}$ using any least squares problem solver, then employ cholesky decomposition to obtain Q [3]. For the experiment, solutions obtained by the `mldivide` operator and SVD were identical.

Finally we have $M = \hat{M}Q$, and $S = Q^{-1}\hat{S}$, where S contains the 3D world point of the P tracked points and M has the position of the camera over F frames.

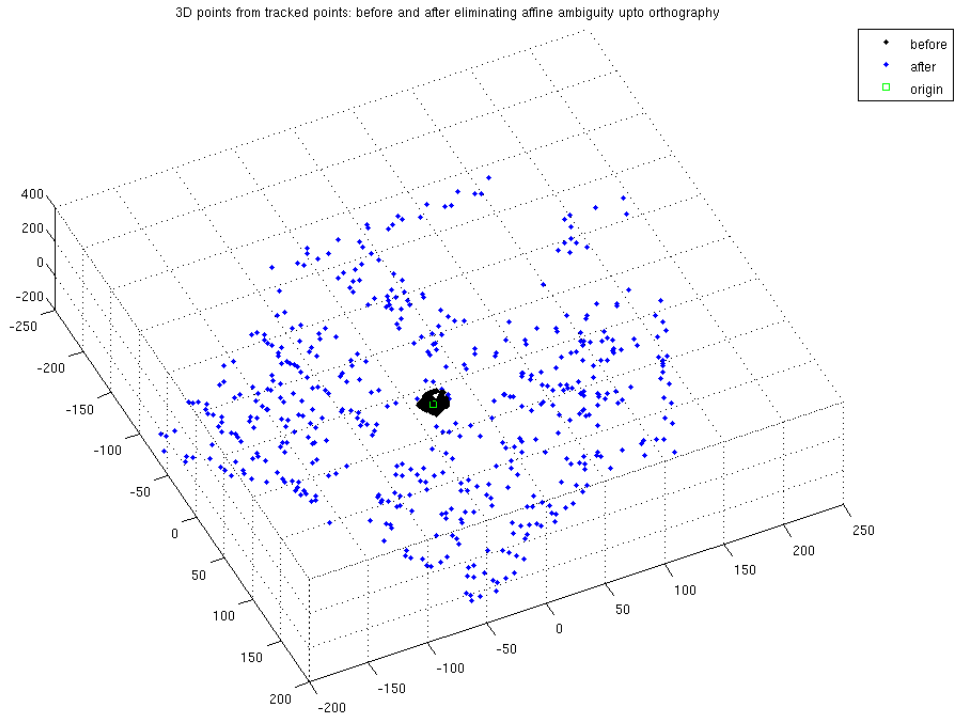

Figure 7: 3D world points before and after removing the affine ambiguity

Figure 7,8, and 9 are the reconstructed 3D world points before and after removing the affine ambiguity in different view points. The camera position in each frame is given by the cross product of M_f's rows $\boldsymbol{i}_f \times \boldsymbol{j}_f$. The 3D path of camera movement from each dimension is plotted on 6.

5 Conclusion

The main challenge for Structure from Motion lies not in creating the model of the system but in accurately estimating the parameters of the model. Theoretically, an appropriate camera model's parameters are well-defined up to certain ambiguities. However, the model is non-linear and has high sensitivity to parameter variations which makes the problem numerically ill-conditioned or computationally intensive in direct methods [2].

The main contribution of the factorization method is that under orthography, the relationship between the measurement, the motion and the shape matrix has a simple expression regardless of the shape or the camera movement [7]. The algorithm is also numerically stable and the convergence to minimize the norm squared error of the approximation of W is guaranteed because of SVD. Even when the number of F grows, there are methods to estimate W efficiently

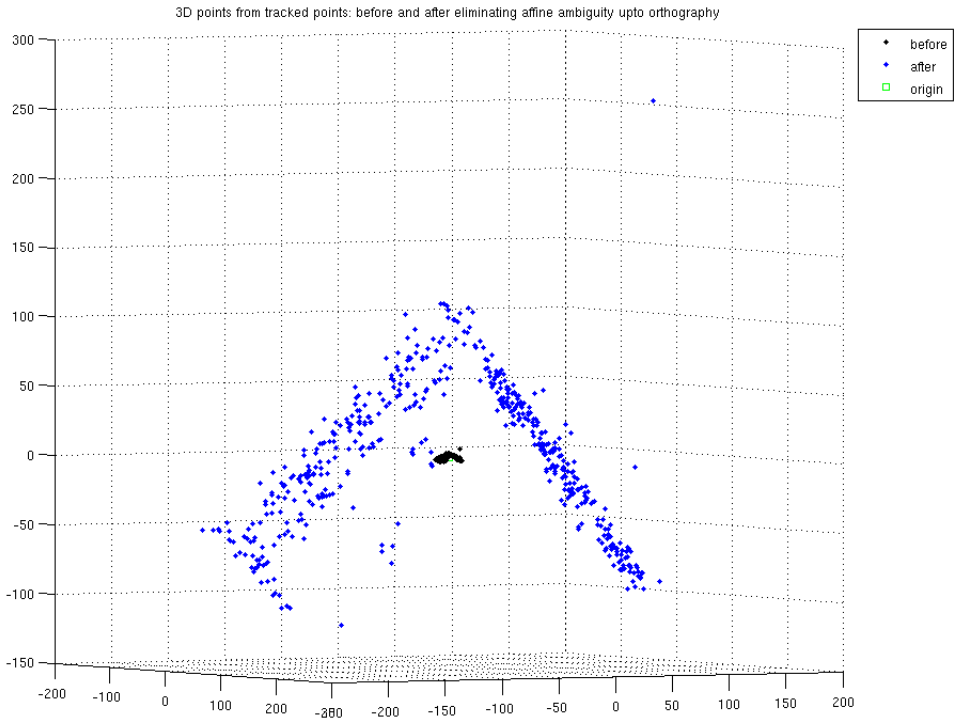

Figure 8: 3D world points before and after removing the affine ambiguity

because the full SVD of W is not necessary. The factorization method presented in this project assumes orthographic projection but the algorithm has been further extended to work with most camera models as well[2].

Every step discussed above was implemented in MATLAB (appendix A). As proposed the method gives accurate reconstruction of the world points.

A Source Code

do_SfM.m The driver.

```
function do_SfM(config_file)
%%%%%%%%%%
% CMSC660 Fall'11 Final Project: Affine Structure from Motion(SfM)
% doSfM.m
% Driver script to do affine SfM, to run, do:
% do_SfM('config'); where 'config' refers to the config.m in this directory
%
% Angjoo Kanazawa 11/23/'11
%%%%%%%%%%

%% Step 1: get initial keypoints
```

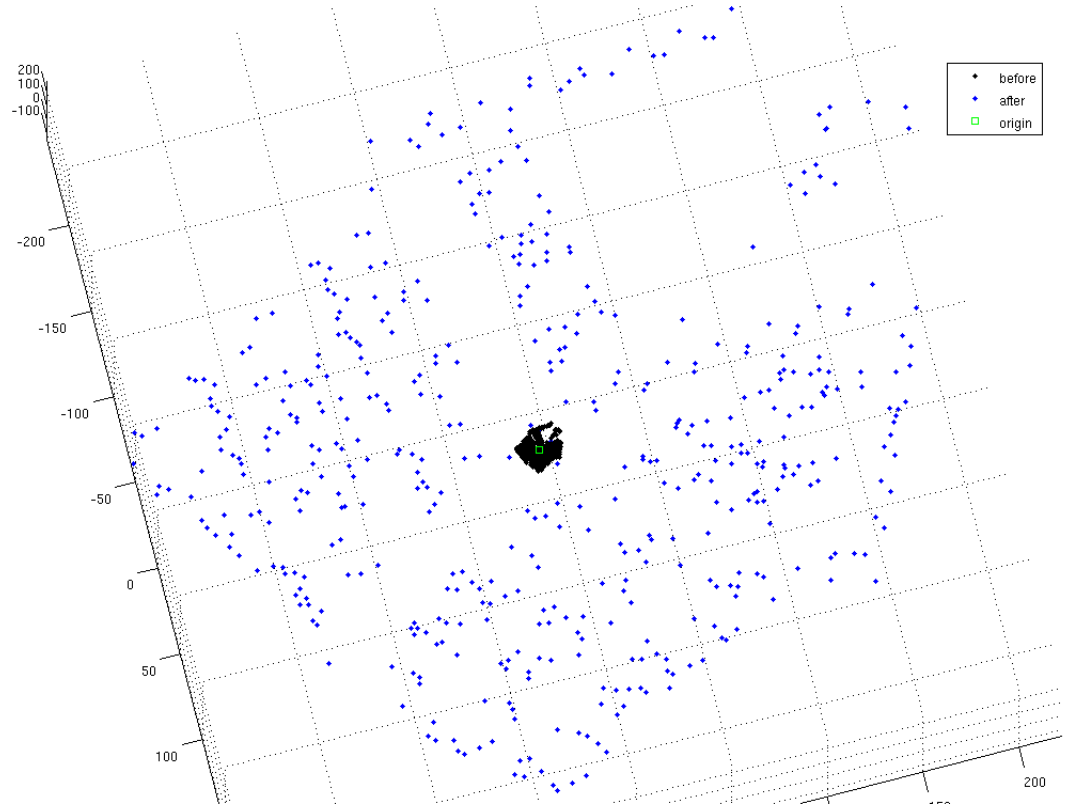

Figure 9: 3D world points before and after removing the affine ambiguity

```
[keyXs, keyYs] = do_getKeypoints(config_file);

%% Step 2: track features

[trackedXs, trackedYs] = do_trackFeatures(config_file);

%% Step 3: Affine Structure for Motion via Factorization

[M S] = do_factorization(config_file);
```

do_getKeypoints.m Code for section 3.1

```
function [keyXs, keyYs] = do_getKeypoints(config_file)
%%%%%%%%%%
% getKeypoints.m
% script to get the key points from im using Harris corner detector
% INPUT - im: image to get the keypoint
%         tau: threshold to do non-maxima supression
%
% OUTPUT - keyXs, keyYs: keypoints found in the initial
% frame. saves them in the file specified by config.m
```

```
%
% parts of script referenced from: http://www.csse.uwa.edu.au/~pk/research/matlabfns/
%
% Angjoo Kanazawa 11/23/'11
%%%%%%%%%%

%% Evaluate the global configuration file and load parameters
eval(config_file);

imFiles  = getImageSet(IMAGE_DIR); % gets cell array of frames (img files)
F = length(imFiles); % number of frames
fprintf('getting intial keypoints from %s\n', imFiles{1});

im = double(imread(imFiles{1}));

if strcmp(Feature.method, 'harris')
    % compute image derivatives in x and y using Peter Kovesi's
    % accurate derivative
    [Ix Iy] = derivative5(im, 'x', 'y');
    % old way
    % dx = [ -1 0 1 ; -1 0 1 ; -1 0 1];
    % dy = dx';
    % Ix = imfilter(im, dx, 'same');
    % Iy = imfilter(im, dy, 'same');

    % compute components of H
    Ix2 = Ix.^2;
    Iy2 = Iy.^2;
    IxIy = Ix.*Iy ;

    % smooth H using gaussian filter
    filt = fspecial('gaussian', 6*Feature.sigma, Feature.sigma);
    Ix2sm = imfilter(Ix2, filt, 'same');
    Iy2sm = imfilter(Ix2, filt, 'same');
    IxIysm = imfilter(IxIy, filt, 'same');

    % display plot
    % sfigure; subplot(2,2,1); imagesc(im); colormap('gray'); title('original');
    % subplot(2,2,2); imagesc(Ix); colormap('gray');title('der in x');
    % subplot(2,2,3); imagesc([Ix2 IxIy; IxIy Iy2]);colormap('gray'); title('hessian');
    % subplot(2,2,4); imagesc([Ix2sm IxIysm; IxIysm Iy2sm]);colormap('gray'); title('smoothed hessia

    % compute the corner response matrix = det(H) - a*trace(H)^2
    M = Ix2sm.*Iy2sm - IxIy.^2 - Feature.alpha*(Ix2sm + Iy2sm).^2;

    % perform non-maxima supression over Feature.radius window size

    % Make mask to exclude points within Feature.radius of the image boundary.
    bordermask = zeros(size(im));
    bordermask(Feature.radius+1:end-Feature.radius, Feature.radius+1:end-Feature.radius) = 1;
    % dilate image
    M_sup = ordfilt2(M, Feature.radius^2, ones(Feature.radius));
    % find points that's still there in dilated image & stronger than Feature.tau
    corner = (M==M_sup) & M>Feature.tau & bordermask;
```

```
    % plot
    if VERBOSE
        sfigure; subplot(221); imagesc(M); title('corner response M');
        subplot(222); imagesc(M_sup); title('max dilated');
        colormap('gray');
        subplot(223); imagesc(M> Feature.tau); title(['corner response > thresh']);
        colormap('gray');
        subplot(224); imagesc(corner); title('corner response max suppressed');
        colormap('gray');
    end

    [keyXs, keyYs] = find(corner); % get the r, c index of key points

else %do SIFT
   keyXs = [];
   keyYs = [];
end

if VERBOSE
    sfigure;
    imagesc(im); colormap('gray'); hold on;
    plot(keyYs, keyXs, 'y.');
    title(['first frame overlayed with keypoints']);
end

save(keypoints_f, 'keyXs', 'keyYs');
```

do_trackFeatures.m Code for section 3.2

```
function [trackedXs, trackedYs] = do_trackFeatures(config_file)
%%%%%%%%%%
% do_trackFeatures.m
% Top level file to track freatures from the key points obtained in
% do_getKeypoints.m
% OUTPUT - trackedXs, trackedYs: the tracked points. Saves the
% result in the file specified by config.m
%
% DESCRIPTION
% for each keypoint at frame f, I(x,y,f), we want to compute
% expected translation in the next frame I(x', y', f+1)
%
% Angjoo Kanazawa 12/16/'11
%%%%%%%%%%

%% Evaluate the global configuration file and load parameters
eval(config_file);

% load the data computed in do_getKeypoints.m
load(keypoints_f, 'keyXs', 'keyYs');

% gets cell array of image file names (frames)
imFiles  = getImageSet(IMAGE_DIR);
F = length(imFiles);
```

```
P =  numel(keyXs);
if ~exist(tracked_pts_f);
    trackedXs = zeros(F, P);
    trackedYs = zeros(F, P);
    trackedXs(1, :) = keyXs; trackedYs(1, :) = keyYs;
    for i=2:F
        [trackedXs(i,:) trackedYs(i,:)] = predictTranslationAll(trackedXs(i-1, :), trackedYs(i-1, :)
                                                        imread(imFiles{i-1}), imread(imFiles{i}));
    end
    % remove nans i.e. points that went out of frame
    outFrame = find(isnan(trackedXs(end, :)));
    trackedXs(:, outFrame) = [];
    trackedYs(:, outFrame) = [];
    P = P - numel(outFrame);
    save(tracked_pts_f, 'trackedXs', 'trackedYs', 'P');
else
    load(tracked_pts_f);
end


%% Draw the path of random 30 tracked points over all frames
if VERBOSE
    pts = randperm(P);
    pts = pts(1:30);
    sfigure; imagesc(imread(imFiles{1})); colormap('gray'); hold on;
    plot(trackedYs(1, pts), trackedXs(1,pts),'m*');
    for f=2:F
        plot(trackedXs(f, pts), trackedYs(f,pts), 'b.');
    end
    title('the path of random 30 points tracked over all frames');
end
```

do_predictTranslationALl.m

```
function [newXs newYs] = predictTranslationAll(startXs, startYs,im0,im1);
%%%%%%%%%
% implementation of the KLT tracker introduced in:
% Carlo Tomasi and Takeo Kanade. Detection and Tracking of Point Features. Carnegie Mellon Universit
% predictTranslationAll.m
% script to get new X, Y, locations in im1 for all startXs and
% startYs in im0
%
% Computes the gradients here, calls predictTranslation.m to get
% predict eahc keypoint independently
%
% Angjoo Kanazawa 11/23/'11
%%%%%%%%%

if ~isa(im0, 'double')  im0 = double(im0); end
if ~isa(im1, 'double')  im1 = double(im1); end

% Using the brightness constancy assumption, we expect the pixel
% intensity of location x, y at frame f is same as the pixel
% instensiy of location x'=x+u, y'=y+v at frame f+1
% i.e. I(x,y,f) = I(x', y', f+1), where u and v are displacement of
```

```
% pixels in the next frame.

% With an additional constraint that this must be true within w by w window, this amounts to solving
% -I_t(ps) = grad I(ps)[u; v] => Ax = b
% where A = grad I(ps), b = -I_t(ps), x = [u;v]
% - ps are all points in the w by w window
% - I_t is the temporal gradient: I(x'y', f+1) - I(x,y,f)

%% Step 1 compute the gradient of im0

[Ix Iy] = derivative5(im0, 'x', 'y');
numPoints = length(startXs);
newXs = zeros(numPoints, 1); newYs =  zeros(numPoints, 1);
for i=1:numPoints
    fprintf('.');
    if ~isnan(startXs(i)) || ~isnan(startYs(i))
        [newX newY] = predictTranslation(startXs(i), startYs(i), Ix, Iy, ...
                                    im0, im1);
    else
        newX = nan; newY= nan;
    end
    newXs(i) = newX;
    newYs(i) = newY;
    end
end
```

predictTranslation.m

```
function [newX newY] = predictTranslation(startX, startY,Ix, Iy,im0,im1);
%%%%%%%%%%
% implementation of the KLT tracker introduced in:
% Carlo Tomasi and Takeo Kanade. Detection and Tracking of Point Features. Carnegie Mellon Universit
% predictTranslation.m
% For a single X Y location, use Ix and Iy, im0 and im1 to compute
% the new location X', Y' iteratively using Newton-Raphson style iteration
%
% Angjoo Kanazawa 11/23/'11
%%%%%%%%%%

% Using the brightness constancy assumption, we expect the pixel
% intensity of location x, y at frame f is same as the pixel
% instensiy of location x'=x+u, y'=y+v at frame f+1
% i.e. I(x,y,f) = I(x', y', f+1), where u and v are displacement of
% pixels in the next frame.

% With an additional constraint that this must be true within w by w window, this amounts to solving
% -I_t(ps) = grad I(ps)[u; v] => Ax = b
% where A = grad I(ps), b = -I_t(ps), x = [u;v]
% - ps are all points in the w by w window
% - I_t is the temporal gradient: I(x'y', f+1) - I(x,y,f)

%% Step 1 compute the gradient of im0

WINDOW = 15;
% ignore points that are outside or close to the border (within 3)
```

```
radius = 3;
bordermask = zeros(size(im0));
bordermask(radius+1:end-radius, radius+1:end-radius) = 1;

% all points in the window x window
% make A: [sum_w Ix*Ix sum_w Ix*Iy; sum_w Ix*Iy sum_w Iy*Iy]

% get the indices of the window grid we want to look at
[x_w, y_w] = meshgrid(startX-WINDOW:startX+WINDOW, startY-WINDOW:startY+WINDOW);
Img1_w = interp2(im0, x_w, y_w);
Img2_w = interp2(im1, x_w, y_w);
Ix_w = interp2(Ix, x_w, y_w);
Ix_w = Ix_w(~isnan(Ix_w));
Iy_w = interp2(Iy, x_w, y_w);
Iy_w = Iy_w(~isnan(Iy_w));

Ixy_w = interp2(Iy.*Ix, x_w, y_w);
Ixy_w = Ixy_w(~isnan(Ixy_w));

A = [sum(Ix_w.^2) sum(Ixy_w); sum(Ixy_w) sum(Iy_w.^2)];

%% get It = I(x', y', t+1) - (x,y,t+1)
% iteratively so the first one is, (x0', y0') = (x,y)

diff = norm(Img2_w- Img1_w);
dx = 100;
uv = [0;0]; %(x',y') starts at (x0,y0)
itr = 0;
maxItr = 30;
while dx > 0.01 & itr < maxItr
    Img2_w_new = interp2(im1, x_w+uv(1), y_w+uv(2));
    It = Img2_w_new - Img1_w;
    % remove if point/window (X+uv) moves out of frame
    if length(find(isnan(It)))>0
        uv = [NaN; NaN];
        break;
    end
    % calculate b = - [sum_w IxIt sum_w IyIt]
    b = - [sum(Ix_w.*It(:)); sum(Iy_w.*It(:))];
    % estimate (u,v)
    uv_new = A\b;
    % update displacement
    uv = uv+uv_new;
    itr = itr + 1;
    diffNew =norm(Img2_w - Img2_w_new);
    if diff < diffNew
        break;
    end
    dx = abs(diff - diffNew);
    diff = diffNew;
end
if isnan(uv)
    newX = nan; newY=nan;
else
```

```
    %maybe not needed but check if this computed uv sets key point
    %out of frame
    x_int = interp2(im1, startX+uv(1), startY+uv(2));
    if isnan(x_int)
        newX = nan; newY=nan;
        fprintf('\nout of frame!\n');
    else
        newX = startX+uv(1); newY = startY+uv(2);
    end
end
```

do_factorization.m Code for section 3.3.

```
function [M S] = do_factorization(config_file, Xs, Ys)
%%%%%%%%%%
% do_factorization.m
% Using tracked points, implement affine structure from motion
% procedure described in
% "Shape and Motion from Image Streams under Orthography: a
% Factorization Method" 1992 by Tomasi and Kanade.
%
% INPUT - Xs, Ys (optional): tracked 2D points from sequences in
% format F x P, where F is the number of frames and P is the
% number of points tracked. If not supplied will load from the file specified in config.m
%
% OUTPUT - M: 2*F by 3 Motion matrix (Camera movements)
%        - S: 3 by P Shape matrix (3D world coordinates)
%
% ALGORITHM
%   1. represent the input as a 2F x P measurement matrix W
%   2. Compute SVD of W = USV'
%   3. Define M' = U_3(S_3)^(1/2), S' = (S_3)^(1/2)V'_3 (U_3 means the
%   first 3 x 3 block, where M' and S' are liner transformations of
%   the actual M and S
%   4. Compute Q by imposing the metric constraints i.e. let L = QQ'
%   and solve Gl = c for l, use cholseky to recover Q
%   5. Compute M and S using M', S', and Q
%
% Angjoo Kanazawa 12/14/'11
%%%%%%%%%%

%% Evaluate the global configuration file and load parameters
eval(config_file);

if nargin == 1
    data = load(tracked_pts_f);
    Xs = data.trackedXs; Ys = data.trackedYs;
end

[F P] = size(Xs);

%%% 0. to eliminate translation, center all points. i.e. subtract
%% mean of each row
Xs = bsxfun(@minus, Xs, mean(Xs, 2));
```

```
Ys = bsxfun(@minus, Ys, mean(Ys, 2));

%%% 1. compute W
W = [Xs; Ys];

%%% 2. SVD of W
[U D V] = svd(W);

%%% 3. make M', S'
Mhat = U(:, 1:3)*sqrt(D(1:3, 1:3));
Shat = sqrt(D(1:3, 1:3))*V(:, 1:3)';

%%% 4. Compute Q, impose the metric constraints
Is = Mhat(1:F, :);
Js = Mhat(F+1:end, :);


gfun = @(a, b)[ a(1)*b(1), a(1)*b(2)+a(2)*b(1), a(1)*b(3)+a(3)*b(1), ...
              a(2)*b(2), a(2)*b(3)+a(3)*b(2), a(3)*b(3)] ;
G = zeros(3*F, 6);
for f = 1:3*F
    if f <= F
        G(f, :) = gfun(Is(f,:), Is(f,:));
    elseif f <= 2*F
        %        fprintf('do j(%d) ', mod(f, F+1)+1);
        G(f, :) = gfun(Js(mod(f, F+1)+1, :), Js(mod(f, F+1)+1, :));
    else
        %        fprintf('\tdo i,j(%d)', mod(f, 2*F));
        G(f, :) = gfun(Is(mod(f, 2*F),:), Js(mod(f, 2*F),:));
    end
end

c = [ones(2*F, 1); zeros(F, 1)];

% solve Gl = c (do it by svd later)
l = G\c;

% could be a programatic way, but hey we "see" 3D or 2D
L = [l(1) l(2) l(3);...
     l(2) l(4) l(5);...
     l(3) l(5) l(6)] ;

Q = chol(L); % finally!

%fprintf('check %g\n', all(all(L = Q'*Q)));

%%% 5. get M and S
M = Mhat*Q;
S = inv(Q)*Shat;

if VERBOSE
    %% plot of 3D points
    sfigure;
    plot3(Shat(1, :), Shat(2,:), Shat(3,:),'k.'); hold on;
```

```
    plot3(S(1, :), S(2,:), S(3,:),'b.');
    plot3(0,0,0,'gs');
    grid on;
    title(['3D points from tracked points: before and after eliminating ' ...
           'affine ambiguity upto orthography']);
    legend('before enforcing metric constraints',['after enforcing metric ' ...
                        'constraints', 'origin']);
    %% plot of the predicted 3D path of the cameras
    %The camera position for each frame is given by the cross product
    %kf = if  jf. For consistent results, normalize all kf to be unit
    %vectors. Give three plots, one for each dimension of kf.

    camera_pos = zeros(F, 3);
    for f = 1:F
        kf = cross(M(f,:), M(f+F, :));
        camera_pos(f,:) = kf/norm(kf); % in unit norm
    end

    % save this plot in 3 axis.......
    %sfigure; plot3(camera_pos(:, 1), camera_pos(:, 2), camera_pos(:, 3),'.-');
    sfigure; plot3(camera_pos(:, 1), camera_pos(:, 2), [1:F], '.-');
    grid on; zlabel('frames');
    title('camera position over frame on XY axis');
    sfigure; plot(camera_pos(:, 1), camera_pos(:, 3), [1:F]);
    grid on; zlabel('frames');
    title('camera position over frame on XZ axis');
    sfigure; plot(camera_pos(:, 2), camera_pos(:, 3));
    grid on; zlabel('frames');
    title('camera position over frame on YZ axis');
    % triangulate..?
    keyboard
    X = S(1, :);
    Y = S(2, :);
    Z = S(3, :);
    tri = delaunay(X,Y);
    trimesh(tri, X,Y,Z);
end
```

References

[1] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, New York, NY, USA, 2 edition, 2003.

[2] Takeo Kanade and Daniel D. Morris. Factorization methods for structure from motion. *Philosophical Transactions of the Royal Society of London, Series A*, 356(1):1153–1173, 1998.

[3] Toshihiko Morita and Takeo Kanade. A sequential factorization method for recovering shape and motion from image streams. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:858–867, 1997.

[4] Jianbo Shi and C. Tomasi. Good features to track. *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on*, pages 593–600, June 1994.

[5] Richard Szelski. *Computer Vision: Algorithms and Applications*. Springer, 2010.

[6] Carlo Tomasi and Takeo Kanade. Detection and tracking of point features. Technical report, International Journal of Computer Vision, 1991.

[7] Carlo Tomasi and Takeo Kanade. Shape and motion from image streams under orthography: a factorization method. *Int. J. Comput. Vision*, 9:137–154, November 1992.