Computer Vision
Lab Exercise 4
Structure from Motion

This week, the lab introduces structure from motion - inferring 3D shape from 2D images. Points matches across frames are saved into a matrix and the Tomasi-Kanade factorization is used for performing SfM (Structure from Motion).

1 Finding point correspondences

For finding point correspondences between multiple images (Not necessarily from the same camera) there are two possible options that we will consider in this course.

1. Extracting SIFT descriptors and matching these descriptors at key-point locations followed by a *chaining* step. This will be discussed in Lab Assignment 6.

2. Using Optical Flow to estimate the apparent motion vectors: v_x, v_y of image pixels or regions from one frame to the next. The theoretical details of Optical Flow computations will be treated in a coming lecture and are not essential for successfully completing this lab assignment.

For this lab session, we provide a script that implements the Lucas-Kanade Optical Flow-based tracker: <LKtracker.m>.
Note: You are not expected to do any work with this script for optical flow based tracking. If you are interested, you can have a look at the implementation, as extra information for the Optical Flow lecture.

For this lab we will provide:

- The point correspondences obtained from the Lucas-Kanade Optical Flow-based tracker as mat files ('Xpoints' & 'Ypoints'). You can use these to perform the Structure-from-motion assignment.
- We also provide **measurement_matrix.txt** point correspondences obtained using SIFT matches and chaining as we will do in Lab 6.

2 Structure from Motion

Using the feature points from the provided measurement matrix or the provided Xpoints and Ypoints mat file containing Optical Flow tracked points, as input, perform the affine structure from motion procedure.

The following steps will guide you through factorization for structure from motion :

1. After you load the point correspondences, normalize them by subtracting the centroid (mean).

$$\left(\mathbf{x}_j - \frac{1}{n} \sum_{k=1}^{n} \mathbf{x}_k \right)$$

   - **Hint** : Mean should be taken across different views of each point. So, you need the mean of each row.

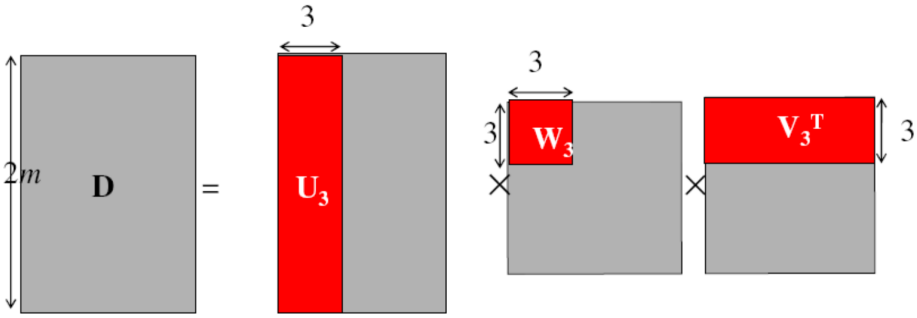
2. The matrix of measurements, D needs to be decomposed into two matrices: M and S.

   Apply SVD to the $2m \times n$ loaded points matrix to express it as $D = U * W * V'$.

$$\mathbf{D} = \begin{bmatrix} \hat{\mathbf{x}}_{11} & \hat{\mathbf{x}}_{12} & \cdots & \hat{\mathbf{x}}_{1n} \\ \hat{\mathbf{x}}_{21} & \hat{\mathbf{x}}_{22} & \cdots & \hat{\mathbf{x}}_{2n} \\ & & \ddots & \\ \hat{\mathbf{x}}_{m1} & \hat{\mathbf{x}}_{m2} & \cdots & \hat{\mathbf{x}}_{mn} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \\ \vdots \\ \mathbf{A}_m \end{bmatrix} \begin{bmatrix} \mathbf{X}_1 & \mathbf{X}_2 & \cdots & \mathbf{X}_n \end{bmatrix}$$

points ($3 \times n$)

Cameras ($2m \times 3$)




```
1       [... ,... ,... ] = svd(X);
```

The matrix D must have rank 3. Keep only the top 3 columns/rows such that:

```
1       % The matrix of measurements must have rank 3.
2       % Keep only the corresponding top 3 rows/columns ...
            from the SVD decompositions.
3       U = ...
4       W = ...
5       V = ...
```

- U is a $2m \times 3$ matrix,
- W is a 3×3 matrix,
- V is a $n \times 3$ matrix.

3. From U, W, V compute the matrices M and S. A possible decomposition is $M = U * W^{\frac{1}{2}}$ and $S = W^{\frac{1}{2}} * V^T$

```
% Compute M and S: One possible decomposition is ...
    M = U W^{1/2} and S = W^{1/2} V'
M = ...
S = ...
save('M','M')
```

4. This decomposition is not unique, to resolve the affine ambiguity, we use Least Squares and the Cholesky decomposition. For this we:

   - We define a matrix L such that $A_i L A_i = Id$, where A_i are the x and y coordinates of the motion matrix M, and Id is the identity matrix.

```
% We define the starting value for L, L0 as: ...
    A1 L0 A1' = Id
A1 = M(1:2, :);
L0 = pinv(A1' * A1);

% We solve L by iterating through all images ...
    and finding L one which minimizes ...
    Ai*L*Ai' = Id, for all i.
% LSQNONLIN solves non-linear least squares ...
    problems. Please check the Matlab ...
    documentation.
L = lsqnonlin(@residuals, L0);
```

   We use the Matlab *lsqnonlin* function which minimizes a given function (In our case the function handler: *@residuals*) starting from an initial solution guess, in our case L_0.

   We then need to define the function <residuals.m> to be optimized by the *lsqnonlin*: $A_i L A_i = Id$.

```
% Compute the residuals
for i = 1:size(M, 1)/2 % Loop over the ...
    cameras: rows are ordered as x1,y1, ...
    x2,y2, ..

```

```
        % Define the x and y projections for the ...
            current camera:
        Ai = M(i*2-1 : i*2, : );

        % Definite the function to be minimized: ...
            Ai L Ai' - Id = 0
        diff_i = ...

        diff(i,:) = diff_i(:);
    end
```

- We decompose L using the Cholesky decomposition into $L = CC^T$.
- We redefine M and S as: $M = MC$ and $S = C^{-1}S$.

```
    % Recover C from L by Cholesky decomposition.
    C = chol(L,'lower');

    % Update M and S with the corresponding C ...
        form: M = MC and S = C^{-1}S.
    M = ...
    S = ...
```

5. With the now estimated 'S' matrix, you can use *plot3* to visualize the points. The rows of S are the 3D coordinates (x, y and z) vectors needed as the argument to *plot3*.

6. Perform these steps using the given points in the **measurement_matrix-.txt** and the points obtained from the Optical Flow tracker (Xpoints and Ypoints matrices). Compare the obtained results.