# Robotics:Perception Assignment 4
# Structure from Motion

## 1 Introduction

In this programming assignment, we will implement key elements of structure from motion except for bundle adjustment. Structure from motion takes a set of images and compute camera poses and 3D point cloud, simultaneously. It has been widely used in computer vision, graphics, robotics, and even medical imaging. Photo tourism (`http://phototour.cs.washington.edu/`) and motion capture from body-mounted cameras (`http://drp.disneyresearch.com/projects/mocap/`) are successful applications of structure from motion.

(a) INPUT: Images


(b) OUTPUT: 3D reconstruction

Figure 1: (a) Given the input images of Levine Hall at University of Pennsylvania, (b) reconstruct 3D point cloud and camera poses.

Figure 1 illustrates a set of input images and 3D reconstruction of camera poses and scene points. Points in the ground plane and two facades are reconstructed, which forms orthogonal angles. Our task is to implement the key elements in the structure from motion: to estimate fundamental matrix and essential matrix, point triangulation, camera localization via PnP, and nonlinear refinement.

We provide camera intrinsic parameter, $\mathbf{K}$, and 2D point correspondences between three images (noisy image correspondences are already filtered by RANSAC). We also provide visualization tools: 3D camera and point visualization and image reprojection visualization. These visualizations help to debug the each function.

# Robotics:Perception Assignment 4
# Structure from Motion

## 2 Technical Details

The MATLAB script **run_sfm.m** will be the main script to run this assignment. It loads **data.mat** that includes 2D correspondences, $\mathbf{x}_1 \leftrightarrow \mathbf{x}_2 \leftrightarrow \mathbf{x}_3$, camera intrinsic parameter, $\mathbf{K}$, and three images, $\mathcal{I}_1$, $\mathcal{I}_2$, and $\mathcal{I}_3$. The correspondences are established by nearest neighbor search in SIFT local descriptors filtered by fundamental matrix based RANSAC. Based on the correspondences, we will reconstruct 3D point cloud, $\mathbf{X} \in \mathbb{R}^{n\times 3}$, where $n$ is the number of the 3D points, and two relative camera poses, $(\mathbf{R}_2, \mathbf{C}_2)$ and $(\mathbf{R}_3, \mathbf{C}_3)$, with respect to the first camera, $(\mathbf{R}_1 = \mathbf{I}_3, \mathbf{C}_1 = \mathbf{0}_3)$.

The script **run_sfm.m** sequentially calls **EstimateFundamentalMatrix**, **EssentialMatrixFromFundamentalMatrix**, **LinearTriangulation**, **LinearPnP**, and **NonlinearTriangulation**.

### 2.1 Fundamental Matrix Estimation

Given $N \geq 8$ correspondences between two images, $\mathbf{x}_1 \leftrightarrow \mathbf{x}_2$, implement the following function that linearly estimates a fundamental matrix, $\mathbf{F}$, such that $\mathbf{x}_2^\mathsf{T}\mathbf{F}\mathbf{x}_1 = 0$:

```
F = EstimateFundamentalMatrix(x1, x2)
```

(INPUT) `x1` and `x2`: $N\times 2$ matrices whose row represents a correspondence.

(OUTPUT) `F`: $3\times 3$ matrix with rank 2.

The fundamental matrix can be estimated by solving linear least squares ($\mathbf{A}\mathbf{x} = \mathbf{0}$). Because of noise on correspondences, the estimated fundamental matrix can be rank 3. The last singular value of the estimated fundamental matrix must be set to zero to enforce the rank 2 constraint. **NOTE**: normalize the fundamental matrix such that $||\mathbf{F}|| = 1$.

### 2.2 Essential Matrix Estimation

Given the fundamental matrix computed by Section 2.1, estimate $\mathbf{E} = \mathbf{K}^\mathsf{T}\mathbf{F}\mathbf{K}$:

```
E = EssentialMatrixFromFundamentalMatrix(F, K)
```

(INPUT) `K`: $3\times 3$ camera intrinsic parameter

(INPUT) `F`: fundamental matrix

(OUTPUT) `E`: $3\times 3$ essential matrix with singular values (1,1,0).

An essential matrix can be extracted from a fundamental matrix given camera intrinsic parameter, $\mathbf{K}$. Due to noise in the intrinsic parameters, the singular values of the

essential matrix are not necessarily (1,1,0). The essential matrix can be corrected by reconstructing it with (1,1,0) singular values, i.e., $\mathbf{E} = \mathbf{U} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \mathbf{V}^\mathsf{T}$. **NOTE**: normalize the essential matrix such that $||\mathbf{E}|| = 1$.

## 2.3 Linear Triangulation

We will provide the relative camera transform between image 1 ($\mathbf{R}_1 = \mathbf{I}_3, \mathbf{C}_1 = \mathbf{0}_3$) and image 2 ($\mathbf{C}_2, \mathbf{R}_2$). Given the relative transformation and point correspondences, $\mathbf{x}_1 \leftrightarrow \mathbf{x}_2$, triangulate 3D points using linear least squares:

```
X = LinearTriangulation(K, C2, R2, x1, x2)
```

(INPUT) `C1` and `R1`: the first camera pose

(INPUT) `C2` and `R2`: the second camera pose

(INPUT) `x1` and `x2`: two $N \times 2$ matrices whose row represents correspondence between the first and second images where $N$ is the number of correspondences.

(OUTPUT) `X`: $N \times 3$ matrix whose row represents 3D triangulated point.

## 2.4 Linear Camera Pose Estimation

We will register the image 3 into the reconstructed coordinate system using 2D-3D correspondences, $\mathbf{x} \leftrightarrow \mathbf{X}$. The camera pose estimation uses linear least squares to compute the camera projection matrix, $\mathbf{P} \in \mathbb{R}^{3\times4}$, and factors out camera intrinsic parameter: $\mathbf{K}^{-1}\mathbf{P} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}$ where $\mathbf{t}$ is camera translation in the camera coordinate system, i.e., $\mathbf{C} = -\mathbf{R}^\mathsf{T}\mathbf{t}$. $\mathbf{R}$ must be cleaned up such that it is orthogonal matrix with the determinant 1 and $\mathbf{t}$ must be rescaled:

$$\mathbf{R} = \mathbf{U}\mathbf{D}\mathbf{V}^\mathsf{T} \tag{1}$$

$$\begin{cases} \mathbf{R}_\mathrm{c} = \mathbf{U}\mathbf{V}^\mathsf{T}, \quad \mathbf{t}_\mathrm{c} = \mathbf{t}/\mathbf{D}_{1,1} & \text{if } \det(\mathbf{U}\mathbf{V}^\mathsf{T}) = 1 \\ \mathbf{R}_\mathrm{c} = -\mathbf{U}\mathbf{V}^\mathsf{T}, \quad \mathbf{t}_\mathrm{c} = -\mathbf{t}/\mathbf{D}_{1,1} & \text{if } \det(\mathbf{U}\mathbf{V}^\mathsf{T}) = -1 \end{cases} \tag{2}$$

where $\mathbf{D}_{1,1}$ is the first singular value of the $\mathbf{R}$. $\mathbf{R}_\mathrm{c}$ and $\mathbf{t}_\mathrm{c}$ are the cleaned-up rotation and translation for the camera.

**IMPORTANT NOTE**: While theoretically you can use the $x$ directly when solving for the $P$ matrix then use the $K$ matrix to correct the error, this is more numeically unstable, and thus it is better to calibrate the $x$ values before the computation of $P$ then extract $R$ and $t$ directly. This means using instead of $x$, using the values $x_c = K^{-1}x$.

```
[C R] = LinearPnP(X, x, K)
```

(INPUT) `X` and `x`: $N \times 3$ and $N \times 2$ matrices whose row represents correspondences between 3D and 2D points, respectively.

(INPUT) `K`: intrinsic parameter

(OUTPUT) `C` and `R`: camera pose $(\mathbf{C}, \mathbf{R})$.

## 2.5 Nonlinear Triangulation

Given three camera poses and linearly triangulated points, $\mathbf{X}$, refine the locations of the 3D points that minimizes reprojection error:

```
X = NonlinearTriangulation(K, C2, R2, C3, R3, x1, x2, x3, X0)
```

(INPUT) `C2` and `R2`: the second camera pose

(INPUT) `C3` and `R3`: the third camera pose

(INPUT) `x1`, `x2`, and `x3`: $N \times 2$ matrices whose row represents correspondence between the first, second, and third images where $N$ is the number of correspondences.

(INPUT and OUTPUT) `X`: $N \times 3$ matrix whose row represents 3D triangulated point.

We will minimize reprojection error:

$$\underset{\mathbf{X}}{\text{minimize}} \quad \sum_{j=1}^{3} \left(\tilde{x}_i - u_i/w_i\right)^2 + \left(\tilde{y}_i - v_i/w_i\right)^2, \tag{3}$$

where

$$\begin{bmatrix} u_i \\ v_i \\ w_i \end{bmatrix} = \mathbf{K}\mathbf{R}_i \left(\mathbf{X} - \mathbf{C}_i\right). \tag{4}$$

$\mathbf{C}_i$ and $\mathbf{R}_i$ are the $i^{\text{th}}$ camera optical center and orientation. We can solve this nonlinear optimization by finding $\Delta\mathbf{X}$:

$$\Delta\mathbf{X} = \left(\mathbf{J}^\mathsf{T}\mathbf{J}\right)^{-1}\mathbf{J}^\mathsf{T}\left(\mathbf{b} - \mathbf{f}(\mathbf{X})\right) \tag{5}$$

$$\mathbf{b} = \begin{bmatrix} \tilde{x}_1 & \tilde{y}_1 & \tilde{x}_2 & \tilde{y}_2 & \tilde{x}_3 & \tilde{y}_3 \end{bmatrix}^\mathsf{T} \tag{6}$$

$$\mathbf{f}(\mathbf{X}) = \begin{bmatrix} u_1/w_1 & v_1/w_1 & u_2/w_2 & v_2/w_2 & u_3/w_3 & v_3/w_3 \end{bmatrix}^\mathsf{T} \tag{7}$$

where $\mathbf{J}$ is the Jacobian for $\mathbf{X}$:

$$\mathbf{J} = \left[ \begin{array}{ccc} \frac{\partial \mathbf{f}_1}{\partial \mathbf{X}}^\mathsf{T} & \frac{\partial \mathbf{f}_2}{\partial \mathbf{X}}^\mathsf{T} & \frac{\partial \mathbf{f}_3}{\partial \mathbf{X}}^\mathsf{T} \end{array} \right]^\mathsf{T} \tag{8}$$

$$\frac{\partial \mathbf{f}_i}{\partial \mathbf{X}} = \left[ \begin{array}{c} \frac{w_i \partial u_i / \partial \mathbf{X} - u_i \partial w_i / \partial \mathbf{X}}{w^2} \\ \frac{w_i \partial v_i / \partial \mathbf{X} - v_i \partial w_i / \partial \mathbf{X}}{w^2} \end{array} \right] \tag{9}$$

$$\frac{\partial u_i}{\partial \mathbf{X}} = \left[ \begin{array}{ccc} fr_{11} + p_x r_{31} & fr_{12} + p_x r_{32} & fr_{13} + p_x r_{33} \end{array} \right] \tag{10}$$

$$\frac{\partial v_i}{\partial \mathbf{X}} = \left[ \begin{array}{ccc} fr_{21} + p_y r_{31} & fr_{22} + p_y r_{32} & fr_{23} + p_y r_{33} \end{array} \right] \tag{11}$$

$$\frac{\partial w_i}{\partial \mathbf{X}} = \left[ \begin{array}{ccc} r_{31} & r_{32} & r_{33} \end{array} \right]. \tag{12}$$

where $\mathbf{K} = \left[ \begin{array}{ccc} f & & p_x \\ & f & p_y \\ 0 & 0 & 1 \end{array} \right]$ and $\mathbf{R} = \left[ \begin{array}{ccc} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{array} \right]$.
You will iteratively update $\mathbf{X}_{\text{new}} = \mathbf{X}_{\text{old}} + \Delta\mathbf{X}$ to find the local minimum.

# 3 Visualizing Results

We provide two visualization tools: **DisplayCorrespondence** and **Display3D**. These functions display reprojection error onto an image and 3D camera and points, respectively.

# 4 Submitting

To submit your results, run the **submission** script, which will test your functions. This script will generate a mat file called RoboticsPerceptionWeek4Submission.mat. Upload this file onto the assignment page, and you should receive your score immediately.