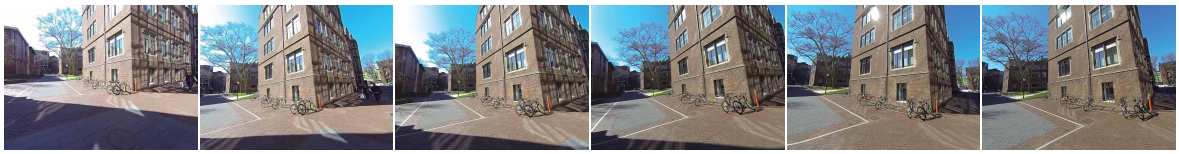
Project 2: Structure from Motion

CIS 580, Machine Perception, Spring 2015
Preliminary report due: 2015.04.20. 11:59AM
Final Due: 2015.05.04. 11:59AM

This project aims to reconstruct a 3D point cloud and camera poses of 6 images as shown in Figure 1. Your task is to implement the full pipeline of structure from motion including two view reconstruction, triangulation, PnP, and bundle adjustment. For nonlinear optimization parts, you are free to choose an optimizer such as built-in functions in MATLAB or Sparse Bundle Adjustment package (http://users.ics.forth.gr/∼lourakis/sba/). Input images are taken by a GoPro Hero 3 camera (Black Edition) and fisheye lens distortion is corrected. We also provide correspondences between all possible pairs of images, i.e, $\mathcal{I}_i \leftrightarrow \mathcal{I}_j$ for $\forall i, j$ where $\mathcal{I}_i$ is the i^{th} image. In Figure 1(b), 6 cameras and 1459 points are reconstructed in 3D.



(a) INPUT: Images

(b) OUTPUT: 3D reconstruction

Figure 1: (a) Given 6 images of space in front of Levine Hall, (b) reconstruct 3D point cloud and camera poses.

Data repository: `cis.upenn.edu/~cis580/Spring2015/Projects/proj2/SfMProjectData.zip`
Submission: Until the preliminary report due, submit one page document to Canvas which includes visualization of at least one step, e.g. visualization of matches after outlier rejection via RANSAC. Until the final due, submit your complete code to turnin
Collaboration: It's an individual or a pair project.

Data We provide 6 undistorted images, calibration data, and matching data. The image resolution is 1280×960 and the intrinsic parameter, $\mathbf{K}$, is specified in the `calibration.txt` file.

(*Matching file name*) The data are stored in 5 files—`matching1.txt`, `matching2.txt`, `matching3.txt`, `matching4.txt`, and `matching5.txt`. `matching3.txt` contains matching between the third image and the fourth, fifth, and sixth images, i.e., $\mathcal{I}_3 \leftrightarrow \mathcal{I}_4$, $\mathcal{I}_3 \leftrightarrow \mathcal{I}_5$, and $\mathcal{I}_3 \leftrightarrow \mathcal{I}_6$. Therefore, `matching6.txt` does not exist because it is the matching by itself.

(*Matching file format*) Each matching file is formatted as follows for the i^{th} matching file:

nFeatures: (the number of feature points of the i^{th} image—each following row specifies matches across images given a feature location in the i^{th} image.)
Each row: (the number of matches for the j^{th} feature) (R) (G) (B) (u_{ij}) (v_{ij}) (image id) (u) (v) (image id) (u) (v) ...

An Example of `matching1.txt`

nFeatures: 2002
3 137 128 105 454.740000 392.370000 2 308.570000 500.320000 4 447.580000 479.360000
2 137 128 105 454.740000 392.370000 4 447.580000 479.360000

Algorithm The full pipeline of structure from motion is shown in Algorithm 1. You will program this full pipeline guided by the functions described in following sections. This pseudocode does not include data management, e.g., converting matches obtained from the data files to feature points.

Algorithm 1 Structure from Motion

```
for all possible pair of images do
    [x1 x2] = GetInliersRANSAC(x1, x2);                ▷ Reject outlier correspondences.
end for
F = EstimateFundamentalMatrix(x1, x2);                 ▷ Use the first two images.
E = EssentialMatrixFromFundamentalMatrix(F, K);
[Cset Rset] = ExtractCameraPose(E);
for i = 1 :  4 do
    Xset{i} = LinearTriangulation(K, zeros(3,1), eye(3), Cset{i}, Rset{i}, x1,
x2);
end for
[C R] = DisambiguateCameraPose(Cset, Rset, Xset);      ▷ Check the cheirality condition.
X = NonlinearTriangulation(K, zeros(3,1), eye(3), C, R, x1, x2, X0));
Cset← {C}, Rset← {R}
for i = 3 : I do                  ▷ Register camera and add 3D points for the rest of images
    [Cnew Rnew] = PnPRANSAC(X, x, K);                  ▷ Register the i^th image.
    [Cnew Rnew] = NonlinearPnP(X, x, K, Cnew, Rnew);
    Cset ← Cset ∪ Cnew
    Rset ← Rset ∪ Rnew
    Xnew = LinearTriangulation(K, C0, R0, Cnew, Rnew, x1, x2);
    Xnew = NonlinearTriangulation(K, C0, R0, Cnew, Rnew, x1, x2, X0);   ▷ Add 3D
points.
    X ← X ∪ Xnew
    V = BuildVisibilityMatrix(traj);                   ▷ Get visibility matrix.
    [Cset Rset X] = BundleAdjustment(Cset, Rset, X, K, traj, V);        ▷ Bundle
adjustment.
end for
```

1 Matching

In this section, you will refine matches provided by the matching data files by rejecting outlier matches based on fundamental matrix.

1.1 Fundamental Matrix Estimation

Goal 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`: 3×3 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.

1.2 Match Outlier Rejection via RANSAC

Goal Given N correspondences between two images ($N > 8$), $\mathbf{x}_1 \leftrightarrow \mathbf{x}_2$, implement the following function that estimates inlier correspondences using fundamental matrix based RANSAC:

```
[y1 y2 idx] = GetInliersRANSAC(x1, x2)
```

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

(OUTPUT) `y1` and `y2`: $N_i \times 2$ matrices whose row represents an inlier correspondence where N_i is the number of inliers.

(OUTPUT) `idx`: $N \times 1$ vector that indicates ID of inlier `y1`.

A pseudo code the RANSAC is shown in Algorithm 2.

Algorithm 2 GetInliersRANSAC

$n \leftarrow 0$
for $i = 1 : M$ **do**
  Choose 8 correspondences, $\hat{\mathbf{x}}_1$ and $\hat{\mathbf{x}}_2$, randomly
  `F = EstimateFundamentalMatrix(`$\hat{\mathbf{x}}_1$`,` $\hat{\mathbf{x}}_2$`)`
  $\mathcal{S} \leftarrow \emptyset$
  for $j = 1 : N$ **do**
    if $|\mathbf{x}_{2j}^\mathsf{T}\mathbf{F}\mathbf{x}_{1j}| < \epsilon$ **then**
      $\mathcal{S} \leftarrow \mathcal{S} \cup \{j\}$
    end if
  end for
  if $n < |\mathcal{S}|$ **then**
    $n \leftarrow |\mathcal{S}|$
    $\mathcal{S}_{in} \leftarrow \mathcal{S}$
  end if
end for

2 Relative Camera Pose Estimation

In this section, you will initialize relative camera pose between the first and second images using an essential matrix, i.e., $(\mathbf{0}, \mathbf{I}_{3\times3})$ and $(\mathbf{C}, \mathbf{R})$.

2.1 Essential Matrix Estimation

Goal Given $\mathbf{F}$, estimate $\mathbf{E} = \mathbf{K}^\mathsf{T}\mathbf{F}\mathbf{K}$:

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

(INPUT) `K`: 3×3 camera intrinsic parameter

(INPUT) `F`: fundamental matrix

(OUTPUT) `E`: 3×3 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., $\mathbf{E} = \mathbf{U}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}\mathbf{V}^\mathsf{T}$.

2.2 Camera Pose Extraction

Goal Given $\mathbf{E}$, enumerate four camera pose configurations, $(\mathbf{C}_1, \mathbf{R}_1)$, $(\mathbf{C}_2, \mathbf{R}_2)$, $(\mathbf{C}_3, \mathbf{R}_3)$, and $(\mathbf{C}_4, \mathbf{R}_4)$ where $\mathbf{C} \in \mathbb{R}^3$ is the camera center and $\mathbf{R} \in SO(3)$ is the rotation matrix, i.e., $\mathbf{P} = \mathbf{K}\mathbf{R}\begin{bmatrix} \mathbf{I}_{3\times3} & -\mathbf{C} \end{bmatrix}$:

```
[Cset Rset] = ExtractCameraPose(E)
```

(INPUT) `E`: essential matrix

(OUTPUT) `Cset` and `Rset`: four configurations of camera centers and rotations, i.e., `Cset{i}`=$\mathbf{C}_i$ and `Rset{i}`=$\mathbf{R}_i$.

There are four camera pose configurations given an essential matrix. Let $\mathbf{E} = \mathbf{U}\mathbf{D}\mathbf{V}^\mathsf{T}$ and $\mathbf{W} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$. The four configurations are enumerated below:

1. $\mathbf{C}_1 = \mathbf{U}(:,3)$ and $\mathbf{R}_1 = \mathbf{U}\mathbf{W}\mathbf{V}^\mathsf{T}$
2. $\mathbf{C}_2 = -\mathbf{U}(:,3)$ and $\mathbf{R}_2 = \mathbf{U}\mathbf{W}\mathbf{V}^\mathsf{T}$
3. $\mathbf{C}_3 = \mathbf{U}(:,3)$ and $\mathbf{R}_3 = \mathbf{U}\mathbf{W}^\mathsf{T}\mathbf{V}^\mathsf{T}$
4. $\mathbf{C}_4 = -\mathbf{U}(:,3)$ and $\mathbf{R}_4 = \mathbf{U}\mathbf{W}^\mathsf{T}\mathbf{V}^\mathsf{T}$.

Note that the determinant of a rotation matrix is one. If $\det(\mathbf{R}) = -1$, the camera pose must be corrected, i.e., $\mathbf{C} \leftarrow -\mathbf{C}$ and $\mathbf{R} \leftarrow -\mathbf{R}$.

3 Triangulation

In this section, you will triangulate 3D points given two camera poses followed by nonlinear optimization. This triangulation also allows you to disambiguate four camera pose configuration obtained from the essential matrix.

3.1 Linear Triangulation

Goal Given two camera poses, $(\mathbf{C}_1, \mathbf{R}_1)$ and $(\mathbf{C}_2, \mathbf{R}_2)$, and correspondences $\mathbf{x}_1 \leftrightarrow \mathbf{x}_2$, triangulate 3D points using linear least squares:

```
X = LinearTriangulation(K, C1, R1, 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.

3.2 Camera Pose Disambiguation

Goal Given four camera pose configuration and their triangulated points, find the unique camera pose by checking the *cheirality* condition—the reconstructed points must be in front of the cameras:

```
[C R] = DisambiguateCameraPose(Cset, Rset, Xset)
```

(INPUT) `Cset` and `Rset`: four configurations of camera centers and rotations

(INPUT) `Xset`: four sets of triangulated points from four camera pose configurations

(OUTPUT) `C` and `R`: the correct camera pose

The sign of the Z element in the camera coordinate system indicates the location of the 3D point with respect to the camera, i.e., a 3D point $\mathbf{X}$ is in front of a camera if $(\mathbf{C}, \mathbf{R})$ if $\mathbf{r}_3^\mathsf{T}(\mathbf{X} - \mathbf{C}) > 0$ where $\mathbf{r}_3$ is the third row of $\mathbf{R}$. Not all triangulated points satisfy this condition due to the presence of correspondence noise. The best camera configuration, $(\mathbf{C}, \mathbf{R}, \mathbf{X})$ is the one that produces the maximum number of points satisfying the cheirality condition.

3.3 Nonlinear Triangulation

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

```
X = NonlinearTriangulation(K, C1, R1, C2, R2, x1, x2, X0)
```

(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.

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

The linear triangulation minimizes algebraic error. Reprojection error that is geometrically meaningful error is computed by measuring error between measurement and projected 3D point:

$$\underset{\mathbf{X}}{\text{minimize}} \quad \left(u - \frac{\mathbf{P_1^\mathsf{T}}\tilde{\mathbf{X}}}{\mathbf{P_3^\mathsf{T}}\tilde{\mathbf{X}}}\right)^2 + \left(v - \frac{\mathbf{P_2^\mathsf{T}}\tilde{\mathbf{X}}}{\mathbf{P_3^\mathsf{T}}\tilde{\mathbf{X}}}\right)^2,$$

where $\widetilde{\mathbf{X}}$ is the homogeneous representation of $\mathbf{X}$. $\mathbf{P}_i^\mathsf{T}$ is each row of camera projection matrix, $\mathbf{P}$. This minimization is highly nonlinear because of the divisions. The initial guess of the solution, `X0`, estimated via the linear triangulation is needed to minimize the cost function. This minimization can be solved using a nonlinear optimization toolbox such as `fminunc` or `lsqnonlin` in MATLAB.

4 Perspective-n-Point

In this section, you will register a new image given 3D-2D correspondences, i.e., $\mathbf{X} \leftrightarrow \mathbf{x}$ followed by nonlinear optimization.

4.1 Linear Camera Pose Estimation

Goal Given 2D-3D correspondences, $\mathbf{X} \leftrightarrow \mathbf{x}$, and the intrinsic parameter, $\mathbf{K}$, estimate a camera pose using linear least squares:

```
[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 $(\mathbf{C}, \mathbf{R})$.

2D points can be normalized by the intrinsic parameter to isolate camera parameters, $(\mathbf{C}, \mathbf{R})$, i.e., $\mathbf{K}^{-1}\mathbf{x}$. A linear least squares system that relates the 3D and 2D points can be solved for $(\mathbf{t}, \mathbf{R})$ where $\mathbf{t} = -\mathbf{R}^\mathsf{T}\mathbf{C}$. Since the linear least square solve does not enforce orthogonality of the rotation matrix, $\mathbf{R} \in SO(3)$, the rotation matrix must be corrected by $\mathbf{R} \leftarrow \mathbf{U}\mathbf{V}^\mathsf{T}$ where $\mathbf{R} = \mathbf{U}\mathbf{D}\mathbf{V}^\mathsf{T}$. If the corrected rotation has -1 determined, $\mathbf{R} \leftarrow -\mathbf{R}$. This linear PnP requires at least 6 correspondences.

4.2 PnP RANSAC

Goal Given $N \geq 6$ 3D-2D correspondences, $\mathbf{X} \leftrightarrow \mathbf{x}$, implement the following function that estimates camera pose $(\mathbf{C}, \mathbf{R})$ via RANSAC:

```
[C R] = PnPRANSAC(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 $(\mathbf{C}, \mathbf{R})$.

A pseudo code the RANSAC is shown in Algorithm 3.

4.3 Nonlinear PnP

Goal Given 3D-2D correspondences, $\mathbf{X} \leftrightarrow \mathbf{x}$, and linearly estimated camera pose, $(\mathbf{C}, \mathbf{R})$, refine the camera pose that minimizes reprojection error:

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

(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

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

Algorithm 3 PnPRANSAC

$n \leftarrow 0$
for $i = 1 : M$ **do**
  Choose 6 correspondences, $\hat{\mathbf{X}}$ and $\hat{\mathbf{x}}$, randomly
  `[C R] = LinearPnP(`$\hat{\mathbf{X}}$`, `$\hat{\mathbf{x}}$`, K)`
  $\mathcal{S} \leftarrow \emptyset$
  for $j = 1 : N$ **do**
    $e = \left(u - \frac{\mathbf{P_1^\intercal \tilde{X}}}{\mathbf{P_3^\intercal \widetilde{X}}}\right)^2 + \left(v - \frac{\mathbf{P_2^\intercal \widetilde{X}}}{\mathbf{P_3^\intercal \widetilde{X}}}\right)^2$ ▷ Measure reprojection error.
    if $e < \epsilon_r$ **then**
      $\mathcal{S} \leftarrow \mathcal{S} \cup \{j\}$
    end if
  end for
  if $n < |\mathcal{S}|$ **then**
    $n \leftarrow |\mathcal{S}|$
    $\mathcal{S}_{in} \leftarrow \mathcal{S}$
  end if
end for

The linear PnP minimizes algebraic error. Reprojection error that is geometrically meaningful error is computed by measuring error between measurement and projected 3D point:

$$\underset{\mathbf{C},\mathbf{R}}{\text{minimize}} \quad \sum_{j=1}^{J} \left(\left(u_j - \frac{\mathbf{P_1^\intercal \tilde{X}_j}}{\mathbf{P_3^\intercal \widetilde{X}_j}} \right)^2 + \left(v_j - \frac{\mathbf{P_2^\intercal \widetilde{X}_j}}{\mathbf{P_3^\intercal \tilde{X}_j}} \right)^2 \right),$$

where $\widetilde{\mathbf{X}}$ is the homogeneous representation of $\mathbf{X}$. $\mathbf{P}_i^\intercal$ is each row of camera projection matrix, $\mathbf{P}$ which is computed by $\mathbf{KR}\left[\begin{array}{cc} \mathbf{I}_{3\times 3} & -\mathbf{C} \end{array}\right]$. A compact representation of the rotation matrix using quaternion is a better choice to enforce orthogonality of the rotation matrix, $\mathbf{R} = \mathbf{R}(\mathbf{q})$ where $\mathbf{q}$ is four dimensional quaternion, i.e.,

$$\underset{\mathbf{C},\mathbf{q}}{\text{minimize}} \quad \sum_{j=1}^{J} \left(\left(u_j - \frac{\mathbf{P_1^\intercal \tilde{X}_j}}{\mathbf{P_3^\intercal \widetilde{X}_j}} \right)^2 + \left(v_j - \frac{\mathbf{P_2^\intercal \widetilde{X}_j}}{\mathbf{P_3^\intercal \tilde{X}_j}} \right)^2 \right), .$$

This minimization is highly nonlinear because of the divisions and quaternion parameterization. The initial guess of the solution, (`C0`, `R0`), estimated via the linear PnP is needed to minimize the cost function. This minimization can be solved using a nonlinear optimization toolbox such as `fminunc` or `lsqnonlin` in MATLAB.

5 Bundle Adjustment

In this section, you will refine all camera poses and 3D points together initialized by previous reconstruction by minimizing reprojection error.

5.1 Visibility Matrix

Goal The relationship between a camera and point, construct a $I \times J$ binary matrix, $mathbf{V}$ where $\mathbf{V}_{ij}$ is one if the j^{th} point is visible from the i^{th} camera and zero otherwise:

```
V = BuildVisibilityMatrix(traj)
```

5.2 Bundle Adjustment

Goal Given initialized camera poses and 3D points, refine them by minimizing reprojection error:

```
[Cset Rset Xset] = BundleAdjustment(Cset, Rset, X, K, traj, V)
```

(INPUT) `X`: reconstructed 3D points.
(INPUT) `K`: intrinsic parameter
(INPUT) `traj`: a set of 2D trajectories
(INPUT) `V`: visibility matrix
(INPUT and OUTPUT) `C` and `R`: camera pose $(\mathbf{C}, \mathbf{R})$.

The bundle adjustment refines camera poses and 3D points simultaneously by minimizing the following reprojection error over $\{\mathbf{C}_i\}_{i=1}^{I}$, $\{\mathbf{q}_i\}_{i=1}^{I}$, and $\{\mathbf{X}_j\}_{j=1}^{J}$:

$$\underset{\{\mathbf{C}_i,\mathbf{q}_i\}_{i=1}^{I},\{\mathbf{X}\}_{j=1}^{J}}{\text{minimize}} \quad \sum_{i=1}^{I}\sum_{j=1}^{J}\mathbf{V}_{ij}\left(\left(u_{ij}-\frac{\mathbf{P_{i1}^{\mathsf{T}}\tilde{X}_j}}{\mathbf{P_{i3}^{\mathsf{T}}\tilde{X}_j}}\right)^2+\left(v_{ij}-\frac{\mathbf{P_{i2}^{\mathsf{T}}\tilde{X}_j}}{\mathbf{P_{i3}^{\mathsf{T}}\tilde{X}_j}}\right)^2\right).$$

This minimization can be solved using a nonlinear optimization toolbox such as `fminunc` and `lsqnonlin` in MATLAB but will be extremely slow due to a number of parameters. The Sparse Bundle Adjustment toolbox (http://users.ics.forth.gr/∼lourakis/sba/) is designed to solve such optimization by exploiting sparsity of visibility matrix, $\mathbf{V}$. Note that a small number of entries in $\mathbf{V}$ are one because a 3D point is visible from a small subset of images. Using the sparse bundle adjustment package is not trivial but it is much faster than MATLAB built-in optimizers. If you are going to use the package, please follow the instructions:

1. Download the package from the website and compile the mex function in the matlab folder (See README.txt).
2. Refer to `SBA_example/sba_wrapper.m` in our data folder that shows an example of using the package. You need to modify the code to set up camera poses and 3D points.
3. Fill `SBA_exmple/projection.m` function that reprojects a 3D point to an image.

6 Putting All Things Together

Write a program that run the full pipeline of structure from motion based on Algorithm 1 and compare your result with an off-the-shelf structure from motion software, `VisualSfM` (http://ccwu.me/vsfm/).