Project: Camera Rectification and Structure from Motion

CIS 580, Machine Perception, Spring 2018

April 18, 2018

In this project, you will learn how to estimate the relative poses of two cameras and compute a point cloud from their correspondences.
There are total two milestones:

1. In Milestone 1, given two images, you are supposed to estimate the relative pose of them based on feature correspondences, using essential and fundamental matrices. Then you need to initialize a 3D point cloud environment using linear triangulation method.

2. In Milestone 2, given a third camera frame, you will complete the structure-from-motion pipeline, using Bundle Adjustment to refine all estimated states, including all camera poses and 3D point cloud.

Instructions. Submit your report and complete code to canvas. Note that this is an **individual** assignment.

Contents

0 Introduction **2**

1 Milestone 1: Camera Rectification and 3D Point Cloud Initialization **4**
- 1.1 Matching . . . 4
  - 1.1.1 Fundamental Matrix Estimation . . . 4
  - 1.1.2 Match Outlier Rejection via RANSAC . . . 4
- 1.2 Relative Camera Pose Estimation . . . 5
  - 1.2.1 Essential Matrix Estimation . . . 5
  - 1.2.2 Camera Pose Extraction . . . 5
- 1.3 Triangulation . . . 6
  - 1.3.1 Linear Triangulation . . . 6
  - 1.3.2 Camera Pose Disambiguation . . . 6

2 Implementation Tips **7**
- 2.1 Data Preprocessing . . . 7
- 2.2 Error Accumulation Issue . . . 7

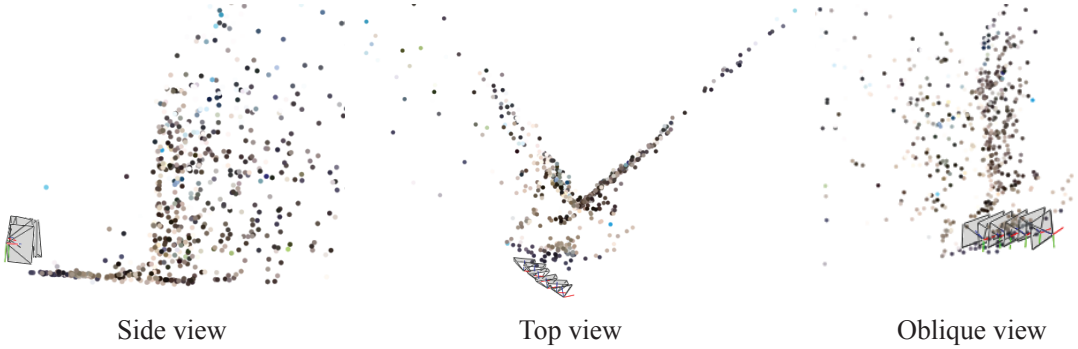
3 Submission **8**
- 3.1 Milestone 1 (Due Wednesday, April 25, 11:59 PM) . . . 8
- 3.2 Milestone 2 (Due Sunday, May 6, 11:59 PM) . . . 8

4 FAQs **9**

0 Introduction

This project aims to reconstruct a 3D point cloud and camera poses of more than 2 images as shown in Figure 1 (e.g. 6). 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 The dataset we provide includes at least two undistorted images, calibration data, and matching data. The image resolution is 1280×960 and the intrinsic parameter, **K**, is specified in the `calibration.txt` file.

(*Matching file format*) The matching data is stored in a .txt file—e.g. `matchingi.txt` contains matching between the **i**th image and the **i+1** th, **i+2** th, ..., **N** th images (suppose we have total N images), i.e., `matching3.txt` stores correpsondenecs $\mathcal{I}_3 \leftrightarrow \mathcal{I}_4$, $\mathcal{I}_3 \leftrightarrow \mathcal{I}_5$, and $\mathcal{I}_3 \leftrightarrow \mathcal{I}_6$. Therefore, `matchingN.txt` does not exist because it is the matching by itself.

Note There exist outliers such that you need to implement some clean up function such as RANSAC.

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

1 Milestone 1: Camera Rectification and 3D Point Cloud Initialization

In the milestone one, you are given data for the **First Two** camera frames, your task is to estimate the relative pose of these two views and then, initialize a 3D point cloud environment.

1.1 Matching

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

1.1.1 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 $\mathbf{x}_2^\top \mathbf{F} \mathbf{x}_1 = 0$:

```
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.

1.1.2 Match Outlier Rejection via RANSAC

Goal Given N correspondences between two images ($N \geq 8$), $\mathbf{x}_1 \leftrightarrow \mathbf{x}_2$, 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 1.

Algorithm 1 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

1.2 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 $(\mathbf{C}, \mathbf{R})$.

1.2.1 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., $\mathbf{E} = \mathbf{U}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}\mathbf{V}^\mathsf{T}$.

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

1.3 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.

1.3.1 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.

1.3.2 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 X0] = 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

(OUTPUT) `X0`: the 3D triangulated points from 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 $\mathbf{r}_3(\mathbf{X} - \mathbf{C}) > 0$ 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.

2 Implementation Tips

In this section, we will cover some tips of the practical implementation that might do a favor to your project.

2.1 Data Preprocessing

All raw observations are in a txt format, it's highly recommended to store all features information in several large **search tables**.

Suppose there are **N** camera frames, and given **N-1** matching data files. Note that each observation point obtained by those data files is unique in the 3D world. Now you can create a large matrix to store position information of all observations, e.g. matrix **Mu** stores the coordinate u of observations in all frames.

For example, in the above case, the matrix Mu has shape $M \times N$, where M represents the number of all observations and N is the number of camera frames. Therefore, Mu(i, j) represents the u coordinate of the **ith** feature in the **jth** frame. In addition, if the **ith** feature is not captured by the **jth** frame, you can set some invalid value for Mu(i, j) like 0 or negatives.

Similarly, you can create any type of search matrix you want to help the project. Also, the above instance of data prepocessing is just a simple demo and recommendation, you are free to use any other approach to handle the raw data.

2.2 Error Accumulation Issue

Actually, the Structure From Motion process will accumulate errors/drifts for sure no matter how good you optimize those states. Therefore, here we share some tips, trying to avoid a large accumulated error.

1. **Clean up bad estimated points**
   It's possible that you will get some weird points that locate behind your camera plane or whose positions are pretty far away in your scaled 3D environment. For those poor-qualified points, you can try to stop optimizing their states in the Bundle Adjustment (BA).

2. **Local Optimization**
   It's a trade-off issue when you determine how many keyframes or 3D points can be included into the Bundle Adjustment optimization. More specifically,

   - **Global BA** will take all previous estimated states into a consideration, which can reduce relative large estimated errors. However, the global optimization will introduce a larger error drift, and it also needs much of time for single one iteration step search.
   - **Local BA** only considers some adjacent keyframes states (e.g. a keyframe island of 3 frames), it can truly speed up the optimization steps and reduce the accumulated errors. However, it's difficult to handle the global states.

   In this project, since we only require to implement three frames BA, it might not be a big issue. However, if you are interested in including more frames, you can follow these local optimization tips that can make your implementation efficient.

3 Submission

Each submission below should be a **.zip** file and submit via **Canvas**.

3.1 Milestone 1 (Due Wednesday, April 25, 11:59 PM)

Finish the Milestone 1, submit all required codes, include a README.md file to clarify how to execute the algorithm. Also, write a report contains the following things:

- Visualization of feature matching after outlier rejection.
- Visualization of 3D points cloud.
- Visualization of camera configuration.

3.2 Milestone 2 (Due Sunday, May 6, 11:59 PM)

Finish the Milestone 2, submit your complete code, include a README.md file to clarify how to execute the algorithm. Also, write a report contains the following things:

- Visualization of the all camera pose and 3D points cloud.
- Analysis of the bundle adjustment implementation.

4 FAQs

Q. Where does $X0$ come from? Looking at pg.2, Algorithm 1, line 11, in project2, it's not clear what X0 is in the method `NonlinearTriangulation(K, zeros(3,1), eye(3), C, R, x1, x2, X0);`
A. It's an initial estimate of $X0$ from your linear triangulation.

Q. I have a question about the matching data. I can't find matching between (image1 and image5), (image1 and image6). They are suppose to be in the matching1.txt, however I can't find any point in the matching1.txt which shows matching to the image5 or image6. The same problem also occurs in matching2.txt. I can't find matching between (image2 and image5), (image2 and image6).
A. It is okay to have no matches between 1 and 5 or 6 as long as there is other image that can link them together for instance, image 3.

Q. In the non linear triangulation formula provided in the assignment,
1. is P matrix the same as $K[R\ t]$?
2. Are we supposed to use P of camera 1 or P of camera 2? or both?
3. Are we supposed to minimize error of all 3D points at once or we can iterate through the points minimizing error one by one?

A. 1. $P = KR[I_{3x3}\ -C]$.
2. Both; you'll compute the reprojection errors for each camera separately then add them together so that each point has a reprojection error from camera 1 + reprojection error from camera 2.
3. You need to iterate through the points minimizing error one by one.

Q. In 2.2 Linear Camera Pose Estimation, I want to make sure whether $t = -R^T C$ or $t = -RC$ Because in slide, it said $t = -RC$, but the description of this project said $t = -R^T C$ And since $P = KR[I_{3x3}\ -C]$, I thought $t = -RC$ makes more sense to me.
A. You are right. $t = -RC$. The definition can be different depending on the books.

Q. Where is the world origin? Is there a single image whose camera pose is declared as the world origin as well?
A. Yes. Usually set any one camera coordinates as the world coordinates. In this project, it's the first camera.

Q. linearPnP - I am confused about the outputs for this section. We use SVD to obtain a P from which you can extract R and C. I believe I understand that the R from this SVD is not necessarily orthogonal and one must perform SVD again to force orthogonality, but I am confused on whether we return the C OR t where $t = -RC$ where R is the orthogonality-enforced R and C is from our original computation of P. Is this correct or am I misunderstanding?
A. You are right. If you extract R from P after multiplying inv(K), R is not necessarily orthogonal because of noise. You can run SVD on R to refine it and then you can extract C with the refined R. This is not ideal but it will give you a good initialization for your nonlinear PnP.

Q. BuildVisibilityMatrix - what is `traj`? How is it structured?
A. It's a collection of the set of 2D points which are projections of the same 3D points. So, if there are N 3D points (size(X, 1) = N), the traj can be `traj = cell(N, 1)` and `traj{i}` = (projected 2d points of X_i on each images). Can be more elaborated.

Q. What is the good number of iterations in RANSAC?
A. M is the number of iterations. In theory, you would like to try M which is fairly high.

In practice, you can use a large number such as 500-1000.

You can validate the iteration number by checking the number of inliers, i.e., the number of inliers must be majority of your set.

Q. What's the meaning of $r_3(X - C) > 0$ and $[0\ 0\ 1]X \geq 0$?
A. The reconstructed point must be in front of your two cameras. Therefore, the two equations must hold for each X, i.e., $X_3 > 0$ (first camera) and $r_3(X - C) > 0$ (second camera).

Q. How do we decide which 4 potential (R,C) to pick? Just count on how many points have met the inequality and pick the most?
A. Yes. Count the number of points that satisfy the cheirality condition. Also please visualize your camera and points in 3D to check whether your reconstruction makes sense.

Q. NonlinearTriangulation Conceptual Problem - What is the point of using X as the parameter we optimize over for this?

I would have thought that we are trying to optimize the poses of the second camera to match it's transformation from the first camera through the 3D points. But that's not what optimizing over X does.
A. In NonlinearTriangulation, the goal is to "refine" 3D points X given camera poses, measurements (x1 and x2), and initial guess of X (obtained from linear triangulation). Since it's non-linear optimization, we need initial X.