16-720A Computer Vision: Assignment 4
3D Reconstruction

Instructor: John Galeotti, Kris M. Kitani
TAs: Leonid Keselman, Tanya Marwah,
Shashank Tripathi, Vibha Nasery, Jiayuan Li

Total Points: 100
Extra Credit Points: 30

Instructions

1. **Integrity and collaboration:** If you work as a group, include the names of your collaborators in your write up. Code should **NOT** be shared or copied. Please **DO NOT** use external code unless permitted. Plagiarism is strongly prohibited and may lead to failure of this course.
2. **Start early!**
3. **Piazza:** If you have any questions, please look at Piazza first. We've created folders for every question, and a sticky thread at the top for each question. Please go through all the previous posts tagged in the appropriate comment before submitting a new question and adding its tag in the sticky.
4. **Write-up:** Please note that we **DO NOT** accept handwritten scans for your write-up in this assignment. Please type your answers to theory questions and discussions for experiments electronically.
5. **Code:** Please stick to the function prototypes mentioned in the handout. This makes verifying code easier for the TAs.
6. **Submission:** Please include all results in your writeup pdf submitted on Gradescope. For code submission, create a zip file, `<andrew-id>.zip`, composed your Python files. Please make sure to remove any temporary files you've generated. Your final upload should have the files arranged in this layout.
   - `<AndrewId>.zip`
     - `<AndrewId>/`
       - `python/`
         - `q2.py`

- q3.py
- q4.py
- q5.py
- run_q2.py
- run_q3.py
- run_q4.py
- run_q5.py

1 Theory (25 points)

Before implementing our own 3D reconstruction, let's take a look at some simple theory questions that may arise. The answers to the below questions should be relatively short, consisting of a few lines of math and text (maybe a diagram if it helps your understanding).

Q1.1 (5 points)

Suppose two cameras fixate on a point P (see Figure 1) in space such that their principal axes intersect at that point. Show that if the image coordinates are normalized so that the coordinate origin $(0, 0)$ coincides with the principal point, the $\mathbf{F}_{33}$ element of the fundamental matrix is zero.

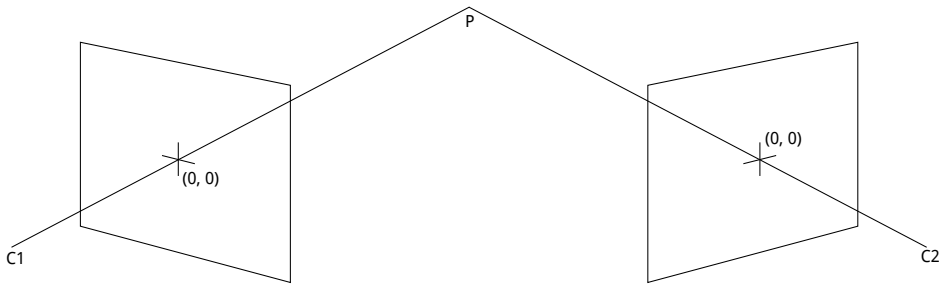

Figure 1: Figure for Q1.1. $C1$ and $C2$ are the optical centers. The principal axes intersect at point P.

Q1.2 (5 points)

Consider the case of two cameras viewing an object such that the second camera differs from the first by a *pure translation* that is parallel to the x-axis. Show that the epipolar lines in the two cameras are also parallel to the x-axis. Backup your argument with relevant equations.

Q1.3 (5 points)

Suppose we have an inertial sensor which gives us the accurate positions (R_i and t_i, where R is the rotation matrix and t is corresponding translation vector) of the robot at time i. What will be the effective rotation (R_{rel}) and translation (t_{rel}) between two frames at different time stamps? Suppose the camera intrinsics (K) are known, express the essential matrix (E) and the fundamental matrix (F) in terms of K, R_{rel} and t_{rel}.

Q1.4 (10 points)

Suppose that a camera views an object and its reflection in a plane mirror. Show that this situation is equivalent to having two images of the object which are related by a skew-symmetric fundamental matrix. You may assume that the object is flat, meaning that all points on the object are of equal distance to the mirror (*Hint:* draw the relevant vectors to understand the relationships between the camera, the object and its reflected image.)

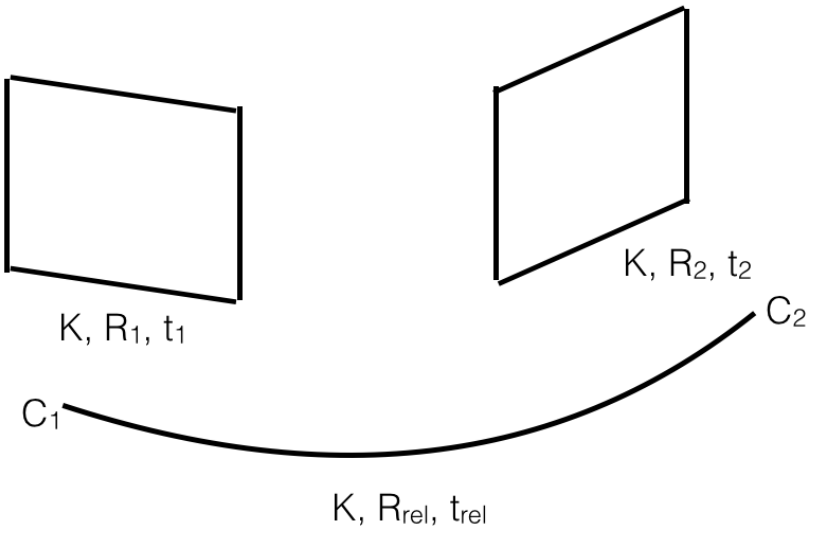

Figure 2: Figure for Q1.3. $C1$ and $C2$ are the optical centers. The rotation and the translation is obtained using inertial sensors. R_{rel} and t_{rel} are the relative rotation and translation between two frames.

2 Fundamental Matrix estimation (20 points)

In this section you will explore different methods of estimating the fundamental matrix given a pair of images. In the `data/` directory, you will find two images (see Figure 3) from the Middlebury multi-view dataset[1], which is used to evaluate the performance of modern 3D reconstruction algorithms.

2.1 The Eight point algorithm

The 8-point algorithm (discussed in class, and outlined in Section 10.1 of Forsyth & Ponce) is arguably the simplest method for estimating the fundamental matrix. For this section, you may manually select point correspondences in an image pair. However, it is recommended that you use the provided correspondences in `data/some_corresp.mat`.

Q 2.1 (5 pts)

Submit a function with the following signature for this portion of the assignment:

```
function F = eightpoint(pts1, pts2, M);
```

where `pts1` and `pts2` are $N \times 2$ matrices corresponding to the (x, y) coordinates of the N points in the first and second image repectively. `M` is a scale parameter.

[1] `http://vision.middlebury.edu/mview/data/`

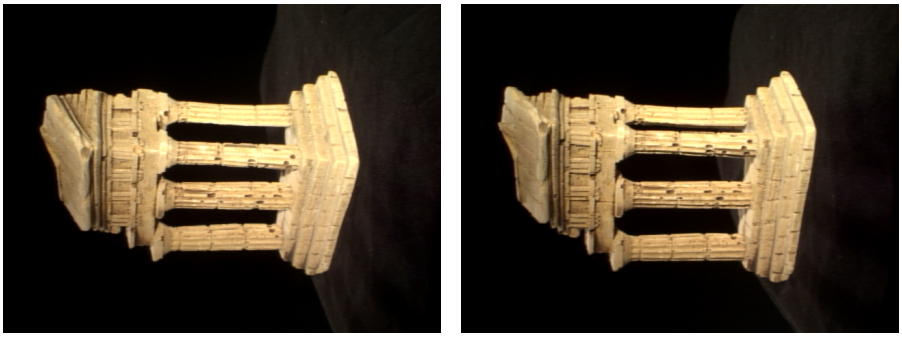

Figure 3: `Temple` images for this assignment

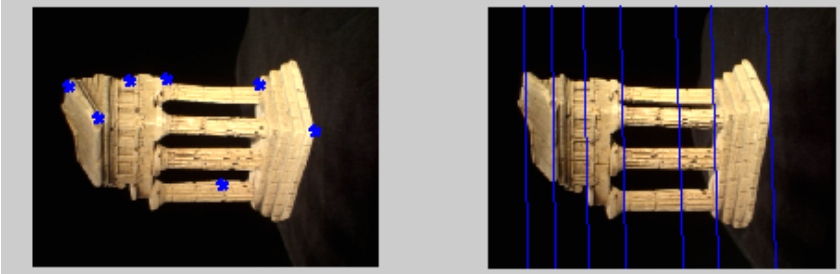

Figure 4: Visualizing epipolar lines

Do NOT use any built-in methods for performing this estimation (except for `svd`), you must do it yourself!

- You should scale the data as was discussed in class, by dividing each coordinate by M (the maximum of the image's width and height). After computing $\mathbf{F}$, you will have to "unscale" the fundamental matrix.
  Hint: If $x_{normalized} = Tx$, then $F_{unnormalized} = T^T \mathbf{F} T$.
  You must enforce the singularity condition of the $\mathbf{F}$ before unscaling.
- Remember that the x-coordinate of a point in the image is its column entry, and y-coordinate is the row entry. Also note that eight-point is just a figurative name, it just means that you need at least 8 points; your algorithm should use an over-determined system ($N > 8$ points). But the final answer should only be rank 2
- To visualize the correctness of your estimated $\mathbf{F}$, use the supplied function `plot_epipolar_lines` (takes in F, the two images, pts1 and the indices). This lets you visualize the epipolar line in the other image corresponding to points in the first image. (Figure 4).
- **Output:** Save your matrix `F`, scale `M`, 2D points `pts1` and `pts2` to the file `q2_1.mat`.
  In your write up: Write your recovered $\mathbf{F}$ and include an image of some

example output of `plot_epipolar_lines`.

2.2 The Seven Point Algorithm

Since the fundamental matrix only has seven degrees of freedom, it is possible to calculate **F** using only seven point correspondences. This requires solving a polynomial equation. In the section, you will implement the seven-point algorithm (described in class, and outlined in Section 15.6 of Forsyth and Ponce).

Q 2.2 (15 pts)

Select 7 points among the provided correspondences, and use these points to recover a fundamental matrix **F**. The function should have the signature:

```
function F = sevenpoint(pts1, pts2, M)
```

where `pts1` and `pts2` are 7×2 matrices containing the correspondences and `M` is the normalizer (use the maximum of the images' height and width), and `F` is a cell array of length either 1 or 3 containing Fundamental matrix/matrices. Use `M` to normalize the point values between $[0, 1]$ and remember to "unnormalize" your computed **F** afterwards.

- Use `plot_epipolar_lines` to visualize **F** and pick the correct one.
- **Output:** Save your matrix `F`, scale `M`, 2D points `pts1` and `pts2` to the file `q2_2.mat`.
  In your write up: Write your recovered **F** and print an output of `plot_epipolar_lines`. Also, include an image of some example output of `plot_epipolar_lines` using the seven point algorithm.

3 Metric Reconstruction (20 points)

You will compute the camera matrices and triangulate the 2D points to obtain the 3D scene structure. To obtain the Euclidean scene structure, first convert the fundamental matrix $\mathbf{F}$ to an essential matrix $\mathbf{E}$. Examine the lecture notes and the textbook to find out how to do this when the internal camera calibration matrices $\mathbf{K}_1$ and $\mathbf{K}_2$ are known; these are provided in `data/intrinsics.mat`. Do NOT use any built-in methods for performing this estimation (except for svd), you must do it yourself!

Q 3.1 (2 points)

Write a function to compute the essential matrix $\mathbf{E}$ given $\mathbf{F}$ and $\mathbf{K}_1$ and $\mathbf{K}_2$ with the signature:

```
function E = essentialMatrix(F, K1, K2)
```

In your write up: Write your estimated $\mathbf{E}$ using $\mathbf{F}$ from the eight-point algorithm.

Given an essential matrix, it is possible to retrieve the projective camera matrices $\mathbf{M}_1$ and $\mathbf{M}_2$ from it. Assuming $\mathbf{M}_1$ is fixed at $[\mathbf{I}, 0]$, $\mathbf{M}_2$ can be retrieved up to a scale and four-fold rotation ambiguity. For details on recovering $\mathbf{M}_2$, see section 7.2 in Szeliski. We have provided you with the function `camera2` to recover the four possible $\mathbf{M}_2$ matrices given $\mathbf{E}$ and $\mathbf{K}_2$.

Note: The `M1` and `M2` here are projection matrix of the form: $M_1 = \big[I|0\big]$ and $M_2 = \big[R|t\big]$.

Q 3.2 (13 points)

Using the above, write a function to triangulate a set of 2D coordinates in the image to a set of 3D points with the signature:

```
function [P, err] = triangulate(C1, p1, C2, p2)
```

where `p1` and `p2` are the $N \times 2$ matrices with the 2D image coordinates and `P` is an $N \times 3$ matrix with the corresponding 3D points per row. `C1` and `C2` are the 3×4 camera matrices. Remember that you will need to multiply the given intrinsics matrices with your solution for the canonical camera matrices to obtain the final camera matrices. Various methods exist for triangulation - probably the most familiar for you is based on least squares (see Szeliski Chapter 7 if you want to learn about other methods):

For each point i, we want to solve for 3D coordinates $P_i = \big[x_i, y_i, z_i\big]^T$, such that when they are projected back to the two images, they are close to the original 2D points. To project the 3D coordinates back to 2D images, we first write P_i in homogeneous coordinates, and compute $\mathbf{C}_1 P_i$ and $\mathbf{C}_2 P_i$ to obtain the 2D homogeneous coordinates projected to camera 1 and camera 2, respectively.

For each point i, we can write this problem in the following form:

$$\mathbf{A}_i P_i = 0,$$

where $\mathbf{A}_i$ is a 4×4 matrix, and P_i is a 4×1 vector of the 3D coordinates in the homogeneous form. Then, you can obtain the homogeneous least-squares solution (discussed in class) to solve for each P_i.

In your write up: Write down the expression for the matrix $\mathbf{A}_i$.

Once you have implemented triangulation, check the performance by looking at the reprojection error:

$$\texttt{err} = \sum_i \|p_{1i}, \widehat{p_{1i}}\|^2 + \|p_{2i}, \widehat{p_{2i}}\|^2$$

where $\widehat{p_{1i}} = Proj(\mathbf{C}_1, P_i)$ and $\widehat{p_{2i}} = Proj(\mathbf{C}_2, P_i)$.

Note: The `C1` and `C2` here are projection matrix of the form: $\mathbf{C}_1 = \mathbf{K}_1\mathbf{M}_1 = \mathbf{K}_1 \begin{bmatrix} I|0 \end{bmatrix}$ and $\mathbf{C}_2 = \mathbf{K}_2\mathbf{M}_2 = \mathbf{K}_2 \begin{bmatrix} R|t \end{bmatrix}$.

Q 3.3 (5 points)

Obtain the correct `M2` from `M2s` by testing the four solutions through triangulations. Use the correspondences from `data/some_corresp.mat`. Save the correct `M2`, the corresponding `C2`, 2D points `p1`, `p2`, and 3D points `P` to `q3_3.mat`. In your writeup, include the sum of squared errors for both projections, and your selected C2 matrix.

4 3D Visualization (10 points)

You will now create a 3D visualization of the temple images. By treating our two images as a stereo-pair, we can triangulate corresponding points in each image, and render their 3D locations.

Q 4.1 (5 points)

Implement a function with the signature:

```
function [x2, y2] = epipolarCorrespondence(im1, im2, F, x1, y1)
```

This function takes in the x and y coordinates of a pixel on `im1` and your fundamental matrix $\mathbf{F}$, and returns the coordinates of the pixel on `im2` which correspond to the input point. The match is obtained by computing the similarity of a small window around the (x_1, y_1) coordinates in `im1` to various windows around possible matches in the `im2` and returning the closest.

Instead of searching for the matching point at every possible location in `im2`, we can use $\mathbf{F}$ and simply search over the set of pixels that lie along the epipolar line (recall that the epipolar line passes through a single point in `im2` which corresponds to the point (x_1, y_1) in `im1`).

There are various possible ways to compute the window similarity. For this assignment, simple methods such as the Euclidean or Manhattan distances between the intensity of the pixels should suffice. See Szeliski Chapter 11, on stereo matching, for a brief overview of these and other methods.

Implementation hints:

- Experiment with various window sizes.
- It may help to use a Gaussian weighting of the window, so that the center has greater influence than the periphery.
- Since the two images only differ by a small amount, it might be beneficial to consider matches for which the distance from (x_1, y_1) to (x_2, y_2) is small.

To help you test your `epipolarCorrespondence`, we have included:

```
function [coordsIM1, coordsIM2] = plot_matched_points(im1, im2, F)
```

See Figure 5 for a reference visualization. Note that your output doesn't have to exactly match Figure 5.

It's not necessary for your matcher to get *every* possible point right, but it should get easy points (such as those with distinctive, corner-like windows). It should also be good enough to render an intelligible representation in the next question.

In you write up: Include a screenshot of `plot_matched_points` with some detected correspondences OR include a screenshot of the matched correspondences (ground truth and computed).

Output: Save the matrix `F`, points `pts1` and `pts2` which you used to generate the screenshot to the file `q2_6.mat`.

Q 4.2 (5 points)

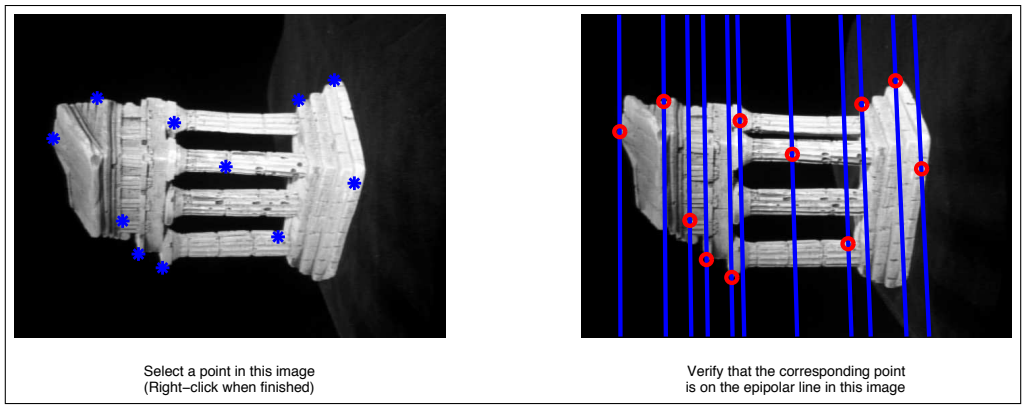

Figure 5: Corresponding points

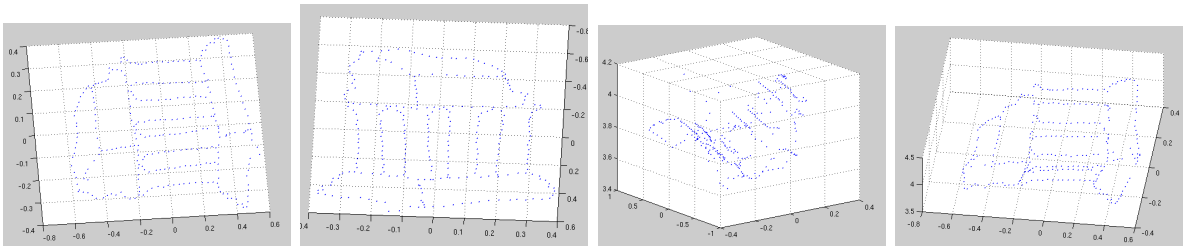

Figure 6: An example point cloud

Included in this homework is a file `data/templeCoords.mat` which contains 288 hand-selected points from `im1` saved in the variables `x1` and `y1`.

Now, we can determine the 3D location of these point correspondences using the `triangulate` function. These 3D point locations can then plotted using the PYTHON function scatter. Write a function `visualize`, which loads the necessary files from `../data/` to generate the 3D reconstruction using scatter. The resulting figure can be rotated for better visualization.

In your submission/writeup: Take a few screenshots of the 3D visualization so that the outline of the temple is clearly visible, and include them with your homework submission.

Output: Again, save the matrix `F`, matrices `M1`, `M2`, `C1`, `C2` which you used to generate the screenshots to the file `q4_2.mat`.

Q 4.3 (Extra credit: 5 points)

Perform dense stereo matching (that is, compute a correspondence for every x,y image pair in the image) and triangulation. In your writeup, include the resulting point-cloud.

5 Bundle Adjustment (25 points)

Q 5.1 (5 points)

In some real world applications, manually determining correspondences is infeasible and often there will be noisy correspondences. Fortunately, the RANSAC method seen in class can be applied to the problem of fundamental matrix estimation.

Implement the above algorithm with the signature:

```
function [F,inliers] = ransacF(pts1, pts2, M)
```

where `M` is defined in the same way as in section 2.

We have provided some noisy correspondences in `some_corresp_noisy.mat` in which around 75% of the points are inliers. Compare the result of RANSAC with the result of the eightpoint when ran on the noisy correspondences. Briefly explain the error metrics you used, how you decided which points were inliers and any other optimization you may have made.

Q 5.2 (15 points)

So far we have independently solved for camera matrix, $\mathbf{M}_j$ and 3D projections, P_i. In bundle adjustment, we will jointly optimize the reprojection error with respect to the points P_i and the camera matrix $\mathbf{M}_j$.

$$err = \sum_{ij} \|p_{ij} - Proj(\mathbf{C}_j, P_i)\|^2,$$

where $\mathbf{C}_j = \mathbf{K}_j\mathbf{M}_j$, same as in Q3.2.

For this homework we are going to only look at optimizing the extrinsic matrix. To do this we will be parametrizing the rotation matrix $\mathbf{R}$ using Rodrigues formula to produce vector $\mathbf{r} \in \mathbb{R}^3$. Write a function that converts a Rodrigues vector $\mathbf{r}$ to a rotation matrix $\mathbf{R}$

```
function [R] = rodrigues(r)
```

as well as the inverse function that converts a rotation matrix $\mathbf{R}$ to a Rodrigues vector $\mathbf{r}$

```
function [r] = invRodrigues(R)
```

Q 5.3 (5 points)

Using this parameterization, write an optimization function

```
function [residuals] = rodriguesResidual(K1, M1, p1, K2, p2, x)
```

where x is the flattened concatenation of $\mathbf{P}$, $\mathbf{r}_2$, and $\mathbf{t}_2$:

```
x = [P(:); r2; t2]
```

$\mathbf{P}$ are the 3D points; $\mathbf{r}_2$ and $\mathbf{t}_2$ are the rotation (in the Rodrigues vector form) and translation vectors associated with the projection matrix $\mathbf{M}_2$.The `residuals` are the difference between original image projections and estimated projections (the square of 2-norm of this vector corresponds to the error we computed in Q3.2):

Use this error function and Python's nonlinear least square optimizer scipy.optimize.minimize to write a function to optimize for the best extrinsic matrix and 3D points using the inlier correspondences from `some_corresp_noisy.mat` and the RANSAC estimate of the extrinsics and 3D points as an initialization.

```
function [M2, P] = bundleAdjustment(K1, M1, p1, K2, M2_init, p2,
                                    P_init)
```

In your write up, include an image of the original 3D points and the optimized points as well as the reprojection error with your initial $\mathbf{M}_2$ and $\mathbf{P}$, and with the optimized matrices.

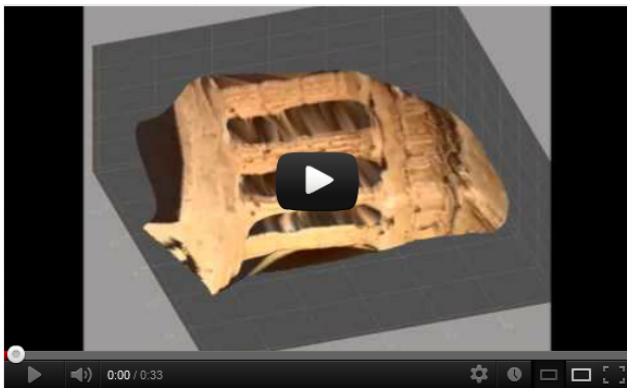

Figure 7: Video of Dense 3D reconstruction is found at https://youtu.be/WOAqndrcDIw

6 Extra Credit (30 points)

Q 6.1 (5 points)

The sparse 3D points from last item are a rather crude visualization of the 3D scene. There are many more ways that interesting 3D visualizations can be generated, including meshes, dense 3D point clouds, depth maps, and so on. Feel free to use third party code; it is not necessary to include it in your submission, just include figures and a brief writeup explaining how they were obtained (and maybe a video!) See Figure 7 For example, here is for an example visualization from a student (click the captionthumbnail!). Your score for this question will depend on the quality of your visualization.

Q 6.2 (20 points)

Implement a file/function called `extraCredit` which loads at least 6 images from the TempleRing dataset `http://vision.middlebury.edu/mview/data/data/templeRing.zip`, and performs feature detection, matching, camera pose estimation, and triangulation for the statute image sequence. Include the transformation matrices in your write-up, including a description of how you kept scale constant throughout your tracking sequence. Jointly bundle adjust the entire result (your x vector should contain all P's (make sure to not duplicate points!), r's and t's). The intrinsics are fixed and identical to K1 in the included intrinsics.mat file.

To keep track of scale, you should keep track of which features are consistent across which observations. The bundle adjustment optimization will then scale them appropriately.