CIS 580, Machine Perception, Spring 2019
Homework 4
Due: Tuesday March 26 2019, 5pm

- This is an individual homework and worth 100 points
- You must submit your solutions on Gradescope, the entry code is 96JGNN. We recommend that you use LaTeX, but we will accept scanned solutions as well.
- You must submit code to the autograder through turnin on a CETS run server.
- Start early! If you get stuck, please post your questions on Piazza or come to office hours!

1 Short questions

1. Consider the pan-tilt unit (PTU) shown in figure 1. If $P_w, P_c \in \mathcal{R}^3$ describe the same physical point, respectively in world and camera coordinates, what are the rotation $R \in \mathcal{SO}(3)$ and the translation $T \in \mathbb{R}^3$ such that $\boxed{P_w = RP_c + T}$? Notice that the camera center is not at the center of the pan rotation of α: this is why in practice it is hard to simulate a pure camera rotation with a PTU.

2. Suppose we know the camera always moves (rotation and translation) in a plane parallel to the image plane. Show that:

   (a) The essential matrix $E = \widehat{T}R$ is of special form

   $$E = \begin{bmatrix} 0 & 0 & a \\ 0 & 0 & b \\ c & d & 0 \end{bmatrix}, \quad a, b, c, d \in \mathbb{R}$$

   (b) Without using the SVD-based decomposition, find a solution to (R, T) in terms of (a, b, c, d).

3. Prove that for $R \in \mathcal{SO}(3)$ and $a, b \in \mathbb{R}^3$, $R(a \times b) = (Ra) \times (Rb)$.

4. **Distance to a line in $\mathbb{P}^2$** Prove that for a line in $\mathbb{P}^2$ defined as $\{\tilde{x} \in \mathbb{P}^2 | \tilde{x}^T l = 0\}$, the squared distance of a point $x \in \mathbb{R}^2$ to the line in the projection plane is the following:

   $$d^2 = \frac{(\tilde{x}^T l)^2}{\|\widehat{e_3} l\|^2},$$

   where $\tilde{x} = [x\ 1]^T$, $e_3 = [0\ 0\ 1]$ and $\widehat{e_3}$ is the hat matrix for e_3.

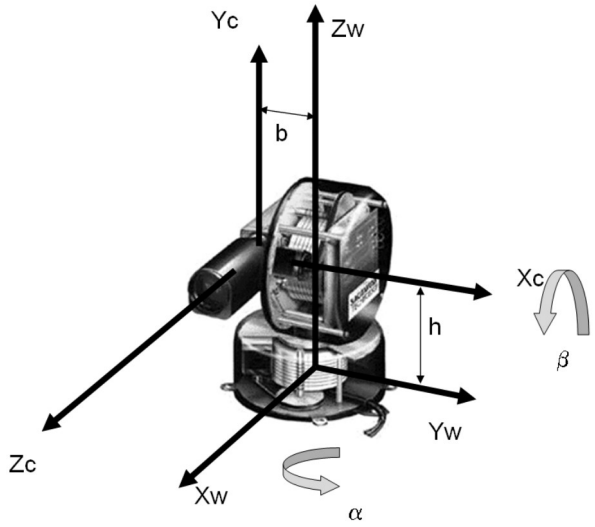

Figure 1: A pan-tilt unit.

2 3D Reconstruction from two 2D images

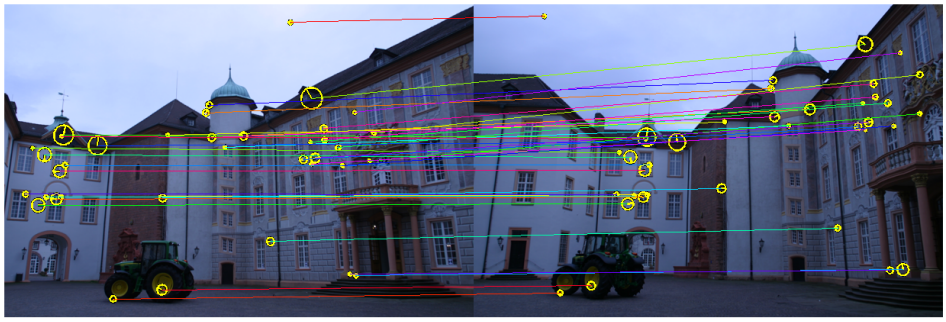

Figure 2: The two images we will use in this problem. A few SIFT matches are shown on top of the images.

Your friend Satsok took some pictures of a castle he visited last summer, and he would love to turn them into a 3D miniature model of the castle. Having heard about your freshly learned computer vision skills, he gives you two pictures (as shown in figure 2) and challenges you to find out where he was standing when he took them and what the 3D scene looks like. He gives you the following details:

- He took the two pictures with no zoom and he turned the auto-focus off: the two views share the same matrix of intrinsics $K = \begin{bmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{bmatrix}$.
- He doesn't know f, u_0, v_0 but is willing to help you estimate them (part 2.1).

- You need to recover the 3D transformation (R, T) between the two views, such that $P_1, P_2 \in \mathbb{R}^3$ describe the same scene point in frame 1 and 2, and $\boxed{P_2 = RP_1 + T}$.

We will use the following notations:

- The notation $\mathbf{x} = (x, y, 1)$ (homogeneous metric coordinates) will be used for calibrated projections:

  $\boxed{P_1 = \lambda_1 \mathbf{x_1}, \quad P_2 = \lambda_2 \mathbf{x_2}}$

- The notation $\mathbf{u} = (u, v, 1)$ (homogeneous pixel coordinates) will be used for uncalibrated projections:

  $\boxed{\mathbf{u_1} \sim K\mathbf{x_1}, \quad \mathbf{u_2} \sim K\mathbf{x_2}}$

  $\boxed{\mathbf{u} \sim K\mathbf{x} \;\Leftrightarrow\; u = fx + u_0,\; v = fy + v_0}$

- epi($\mathbf{x_1}$) is the epipolar line in camera 2 corresponding to $\mathbf{x_1}$.

2.1 Calibration using vanishing points

Your friend doesn't know the intrinsics of the camera, but you know how to recover them: you ask him to take pictures of a big calibration cube as shown in figure 3, using the same settings as when he took the castle pictures. You then run your favorite corner detector on the images he gives you: the result of this process is stored for you in `calib_images.mat`, load it and take a look at the calibration images (`calib_images{i}` for $i = 1, \ldots, 4$).

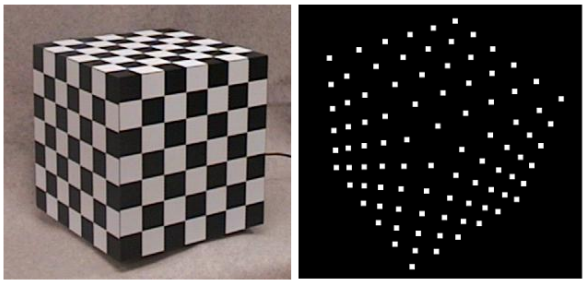

Figure 3: Calibration box and detections of corner points.

1. You are going to use three orthogonal vanishing points $\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_2}$ to estimate the focal length f and projection center (u_0, v_0) as summarized in the lecture notes. You can use the notation $\mathbf{v_i} = (v_{ix}, v_{iy}, v_{iz})$ for the homogeneous coordinates of the vanishing points (in the implementation you can normalize the coordinates so that $v_{iz} = 1$)

   (a) Explain why both the estimation of f and (u_0, v_0) from the vanishing points would fail if the faces of the box (hence the vanishing points) were not orthogonal.

(b) Why is `calib_images{4}` (shown to the right in figure 3) not the best choice of a calibration image in terms of numerical stability?

(c) For two different calibration images, will the vanishing points be roughly the same? Will f and (u_0, v_0) be roughly the same?

2. **Focal length** Show that $s = \frac{1}{f^2}$ is the solution of a linear system $\mathbf{a}s = \mathbf{b}$ where you need to express the 3 coefficients of the vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^3$ as a function of the coordinates of $\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3}$. In practice, because of noise, the coordinates of b are not exactly the ones of a all multiplied by the same value: you can solve the least-squares problem $\min_{s\in\mathbb{R}} \|\mathbf{a}s - \mathbf{b}\|$ by calling `s = a\b` in Matlab.

3. **Projection center in the image** Recall that (u_0, v_0) is the orthocenter of $\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_2}$. By using the fact that the orthogonality $\mathbf{x} \perp \mathbf{y}$ translates as the equation $\mathbf{x}^T\mathbf{y} = 0$, show that (u_0, v_0) is the solution of a least-squares problem. You can use the SVD decomposition method to solve it.

4. Complete the function `vanishingCalibration.m` that takes a calibration image as an input, lets you click on lines to estimate the vanishing points, and uses the positions of the vanishing points to estimate and return the calibration matrix K. For the calibration images `calib_images` you should obtain something around $f = 552$ and (u_0, v_o) should be at the center of the image. These are the same parameters as the ones used for the pictures of the castle.

5. If an image is scaled by a factor α (`imresize(im,alpha)` for instance in Matlab), what is the new intrinsic matrix K_{scaled} compatible with the scaled image compared to the original K? This can be useful in case you want to resize the images that your friend gives you and still know what intrinsics to use.

2.2 Estimation of the essential matrix

Now that you know the camera intrinsics $f = 550$ and $(u_0, v_0) = (307.5, 205)$, you can attempt to reconstruct the scene from the SIFT matchings in the two images (figure 2). The SIFT descriptors are simply computed and matched using the VLFeat toolbox[1], which you can download and extract to a folder of your choice, and load in Matlab with the following command (use \ instead of / on a Windows machine):

```
run('[path-to-vlfeat]/toolbox/vl_setup.m');
```

That's it! You should now be able to use function calls in `reconstruct_script.m`. Although all the calls to VLFeat for SIFT extraction and matching are already done for you, you might want to take a quick look at the beautiful and simple tutorial located at http://www.vlfeat.org/overview/sift.html. Start playing a bit with the first lines of `reconstruct_script.m` to see what kind of inputs you will be using in your functions. Also, quickly read the whole script to understand the overall pipeline.

[1] www.vlfeat.org

1. **Least-squares estimation** The script `reconstruct_script.m` extracts the positions of the matching SIFT descriptors and feeds them to the function `estimateEmatrix.m`: complete the function so that it takes two $N \times 2$ matrices of matching points `X1,X2` as input and estimates E using the SVD decomposition method as in the 8-point algorithm described in the lecture notes and the "E-matrix" handout. Don't forget to project the E you obtain onto the space of essential matrices by redecomposing `[U,S,V] = svd(E)` and returning $U\text{diag}(1,1,0)V^T$.

2. **RANSAC estimation** The estimation of E can be made much more robust by selecting pairs of points that reach a common agreement: in this section you will implement a basic RANSAC algorithm to eliminate outliers (spurious matchings) and obtain a better estimate of E.

   (a) One iteration of the algorithm uses the code from the previous question to estimate the essential matrix E based on a random set of 8 correspondences, and then evaluates how good E is for the other pairs $(\mathbf{x_1}, \mathbf{x_2})$. If E is perfect, for all pairs $(\mathbf{x_1}, \mathbf{x_2})$, $\mathbf{x_2}$ lies exactly on the epipolar $\text{epi}(\mathbf{x_1})$ computed from E and $\mathbf{x_1}$. Therefore, we will use the distances to the epipolars as an error measure. Show that the distance of a matching point to the epipolar is the following:

$$d(\mathbf{x_2}, \text{epi}(\mathbf{x_1}))^2 = \frac{(\mathbf{x_2}^T E \mathbf{x_1})^2}{\|\widehat{\mathbf{e_3}} E \mathbf{x_1}\|^2}.$$

   (b) Complete the function `estimateEmatrixRANSAC.m` that takes a set of matching points and estimates `E` using RANSAC. The algorithm steps are the following:

   - Pick a random set of 8 pairs (you can use `randperm`), estimate E using them and compute the individual residuals for all the other pairs $(\mathbf{x_1}, \mathbf{x_2})$: $d(\mathbf{x_2}, \text{epi}(\mathbf{x_1}))^2 + d(\mathbf{x_1}, \text{epi}(\mathbf{x_2}))^2$.
   - Count how many residuals are lower than $\varepsilon = 10^{-5}$ (consensus set), and if this count is the largest so far, store the current estimate of `E` as the best estimate so far,
   - Iterate as many times as needed according to the probability of failure (500 iterations should be enough)

   (c) *Extra-credit (4pts)*: Use vectorized Matlab code to compute the residuals without `for` loops.

3. **Drawing the epipolar lines** You can now use the essential matrix to plot epipolar lines in the images, as shown in figure 4. Note that E defines epipolars for *calibrated* points $(\mathbf{x_1}, \mathbf{x_2})$. For the uncalibrated points, we showed in class there is a perfectly identical relationship by substituting the **fundamental matrix** F to E:

$$\mathbf{x_2}^T E \mathbf{x_1} = \mathbf{u_2}^T \underbrace{K^{-T} E K^{-1}}_{F} \mathbf{u_1} = 0 \Leftrightarrow \mathbf{u_1} \in \text{epi}_F(\mathbf{u_2})$$

   Complete the function `drawEpipolarLines.m` that draws the epipolar lines for a set of matching image points (uncalibrated pixel coordinates). Note that you just need to fill in the equations, the "drawing" part is already done for you.

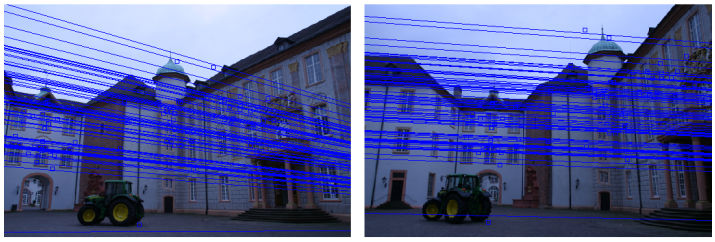

Figure 4: Some epipolar lines in both images: for a given $\mathbf{x_1}$, the matching point $\mathbf{x_2}$ should be as close as possible to epi($\mathbf{x_1}$) and vice versa.

2.3 Pose recovery and 3D reconstruction

It is now time to recover the transformation R, T between the two cameras. For a given estimate of E, remember that there are two possible solutions for $(\widehat{T}, R)$ (twisted pair ambiguity):

$$(\widehat{T}_1, R_1) = \left(UR_Z\left(\frac{\pi}{2}\right)\Sigma U^T, UR_Z\left(\frac{\pi}{2}\right)^T V^T \right)$$
$$(\widehat{T}_2, R_2) = \left(UR_Z\left(-\frac{\pi}{2}\right)\Sigma U^T, UR_Z\left(-\frac{\pi}{2}\right)^T V^T \right)$$

where (U, Σ, V) is the SVD decomposition of E and $R_Z\left(\frac{\pi}{2}\right) = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$.

1. Show that the third column of U has associated hat matrix $UR_Z\left(+\frac{\pi}{2}\right)U'$. Also show that the opposite of the third colum of U has associated hat matrix $UR_Z\left(-\frac{\pi}{2}\right)U'$. Therefore, in the two solutions above, you can output T instead of its hat matrix and simply use the third column of U and its opposite for T_1 and T_2.

2. Show that the sign switch between $R_Z\left(+\frac{\pi}{2}\right)$ and $R_Z\left(-\frac{\pi}{2}\right)$ inside the decompositions of the solutions is equivalent to a left-multiplication of the solutions by $R_T(\pi)$ where T is the third column of U.

3. Complete the function `poseCandidatesFromE.m` that takes E as an input and returns four possible solutions for (R, T):

   - the twisted pair for E,
   - *and* the twisted pair for $-E$. This is because if we find E that satisfies all the epipolar constraints ${\mathbf{x_1}}^T E\mathbf{x_2} = 0$ (part 2.2), the opposite matrix $-E$ also verifies the constraint since the right-hand side is zero, and we don't know a priori whether E or $-E$ is the essential matrix we are looking for.

4. **Triangulation** Given a candidate (R_i, T_i), we can now attempt to reconstruct all the points from the pairs (x_1, x_2), and check whether (R_i, T_i) is the correct transformation (we need to pick one out of the four candidates): we will pick the candidate (R_i, T_i) that has the highest number of reconstructed points *in front of both camera*. Consider one of the matchings (x_1, x_2): we can reconstruct the 3D point by computing the intersection of the two rays coming from camera 1 and 2:

$$\lambda_2 \mathbf{x_2} = \lambda_1 R \mathbf{x_1} + T,$$

In practice, the equality doesn't hold (the two rays don't intersect perfectly) and you need to do a least square estimation of λ_1, λ_2.

For a given pair $(\mathbf{x_1}, \mathbf{x_2})$, formulate the linear system such that $\begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix}$ is its solution. Complete the part of `reconstruct3D.m` that loops through the four transformation candidates (T, R) and all pairs $(\mathbf{x_1}, \mathbf{x_2})$, and solves for the depths λ_1, λ_2.

The selection of the right (T, R) out of the four possibilities is already coded for you but you should take a look at it and make sure you understand it: we simply count for each candidate (T_i, R_i) how many points are in front of both cameras (namely both $\lambda_1, \lambda_2 > 0$) and keep the transformation that achieves the maximum number.

5. You can now run the whole pipeline in `reconstruct_script.m` to estimate the transformation (T, R) between the two cameras and the relative positions of the 3D points. Figure 5 shows an example of output from `plotReconstruction.m`, shown in the (X, Z) plane (we're on top of the scene, looking down at it like a street map). Note that all depths and the translation (baseline) are up to a scale.

6. Complete the function `showReprojections.m` to compare image points in a given camera to the reprojection of images points from the other camera. All you need to do in this function is apply the camera projection model (and be careful about the expression of the 3D transform from 1 to 2 and from 2 to 1).

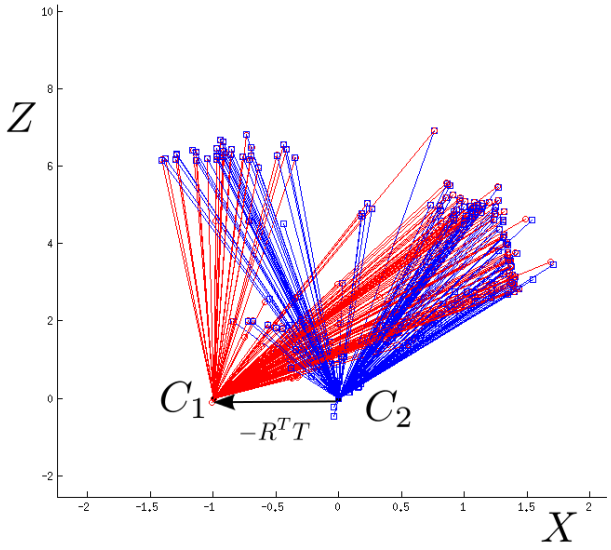

Figure 5: Top view of the the scene, coordinate are expressed in the frame of camera 2 (notice its center is at $0, 0$).