Computer Vision Assignment 3

Saurabh Vaishampayan
svaishampaya@student.ethz.ch

November 2021

1 Problem 1: Camera calibration

This problem involves calibration of camera given a set of corresponding 2D-3D points. Throughout the problem the camera model is assumed to be a linear projection model. **We will not be able to calibrate the effects of radial distortion in this method, hence this part of the full camera model taught in the lecture is not accounted in this method.** We now go over the steps in the implementation and state our results, conclusions and answers to the questions.

1.1 Data Normalisation

Both the sets of 3D and 2D points are normalised by centering them about their respective means and scaling them by inverse of standard deviation. This is necessary because the constraint matrix we have looks like the following (for 1 correspondence):

$$\mathbf{A} = \begin{bmatrix} \mathbf{0^T} & -1 \times \mathbf{X_i^T} & y_i \mathbf{X_i^T} \\ 1 \times \mathbf{X_i^T} & \mathbf{0^T} & -x_i \mathbf{X_i^T} \end{bmatrix} \tag{1}$$

Here the 2D coordinate is $x = [x_i, y_i, 1]^T$ and the corresponding 3D coordinate is $\mathbf{X_i}$. Upon inspection one can see that there are $O(1), O(X), O(x \times X), O(x)$ entries in the matrix, where X represents typical size of 3D vector elements and x represents the size of typical 2D elements. The values of the matrix may range from $<< 1$ to $>> 1$. This will result in an ill conditioned matrix where there are some very small terms, some $O(1)$ terms and some very large terms. Due to finite precision this may result in a solution that is far off from the true solution. Hence we normalise both the vectors $\mathbf{x}, \mathbf{X}$ so that $O(X), O(x), O(x \times X) \sim 1$ resulting in a better conditioned matrix with values of $O(1)$. We do this by:

$$\hat{\mathbf{x}} = \mathbf{T_{2D}} \mathbf{x} \tag{2}$$

$$\hat{\mathbf{X}} = \mathbf{T_{3D}} \mathbf{X} \tag{3}$$

The matrices $\mathbf{T_{3D}}, \mathbf{T_{3D}}$ were saved as instructed and will be used later to compute the actual projection matrix.

1.2 DLT

Our task is to find a projection matrix $\mathbf{P}$ such that $||\mathbf{x} \times \mathbf{PX}||$ is minimized for the given dataset.

$$\mathbf{P} = \begin{bmatrix} \mathbf{P_1^T} \\ \mathbf{P_2^T} \\ \mathbf{P_3^T} \end{bmatrix} \tag{4}$$

The condition $\mathbf{x} \times \mathbf{PX} = 0$ results in 2 independent constraints for each correspondence (it gives 3 equations but one of them is a linear combination of other 2).
In the case of multiple points we have the equation

$$\mathbf{Ap} = \mathbf{0}$$

Here

$$\mathbf{A_{2i:2i+1}} = \begin{bmatrix} \mathbf{0^T} & -1 \times \mathbf{X_i^T} & y_i\mathbf{X_i^T} \\ 1 \times \mathbf{X_i^T} & \mathbf{0^T} & -x_i\mathbf{X_i^T} \end{bmatrix} \tag{5}$$

Here the 2D coordinate is $x = [x_i, y_i, 1]^T$ and the corresponding 3D coordinate is $\mathbf{X_i}$ for the i^{th} point.

$$\mathbf{p} = \begin{bmatrix} \mathbf{P_1} \\ \mathbf{P_2} \\ \mathbf{P_3} \end{bmatrix} \tag{6}$$

We assembled the entries of $\mathbf{p}$ as rows of $\mathbf{P}$ and will take care of this while reassembling. When we have more constraints than unknowns we have to solve the overdetermined problem and do so by minimizing the norm of $\mathbf{Ap}$, which is already implemented using SVD.

1.3 Reprojection error

We get an initial estimate of $\mathbf{P}$ from DLT. However it is obtained by minimizing the norm of the cross product of 2D coordinate and projected coordinate. **This is an algebraic error minimization procedure and does not have any physical meaning. Infact for understanding this drawback one can use the following intuition**:
Given a 2D point $\mathbf{x}$, the vectors $\mathbf{PX}$ that give the same error function form a family of cones, i.e. every vector forming a fixed angle with $\mathbf{x}$ will give the same error value (norm of cross product), while infact they correspond to very different 2D homogenous points. Hence we need to use an error metric that takes into account the physical understanding.
The error objective function

$$\mathbf{P} = \text{argmin}||\mathbf{x} - \mathbf{PX}|| \tag{7}$$

This takes into account the error in the projected 2D space, i.e. we want the projection matrix such that it minimzes the distance between observed 2D points and projected 2D points. For each point we need to first normalize $\mathbf{PX}$ first to make sure that we stay in the 2D manifold of error in reprojected distance. Unlike the earlier error, now the family of $\mathbf{PX}$ vectors for a given $\mathbf{x}$ for constant error not only lie on a circle as opposed to a cone, but the circle also is in the plane of 2D camera points. Hence as opposed to the earlier part where points on the cone could correspond to very different projected points, here we are guaranteed to have the projected 2D points within a fixed radius of observed 2D point. **This is the difference between algebraic error and geometric error.**
Regarding the error measure $\mathbf{e} = \mathbf{x} \times \mathbf{PX}$:

- First of all this is a vector metric, we need to make it a scalar first. One option would just be taking the norm.
- One can scale $\mathbf{P}$ and reduce the error, so one must put in a constraint to make sure $\mathbf{P}$ is non zero.
- After doing this, the problem becomes exactly equivalent to DLT. DLT uses algebraic error and is not as good as using reprojection error as described earlier.
- Hence the error measure is not a good choice.

The reported reprojection error changes as following during the optimization:

- Reprojection error before optimization : 0.0006316426059796234
- Reprojection error after optimization: 0.0006253538899291181

The error increases slightly in the middle iterations. This is because the update step size is not small enough and it takes it to a larger function value in the energy landscape. It could also be that the function is nonconvex and multimodal in the neighbourhood. The reprojection error before the start of nonlinear optimization is lower than after the first iteration, although it decreases to a lower value after iterations. This is because we imposed that the last component is close to 1 and probably this sudden imposition of constraint causes a large enough change in $\mathbf{P}$ to increase the reprojection error.

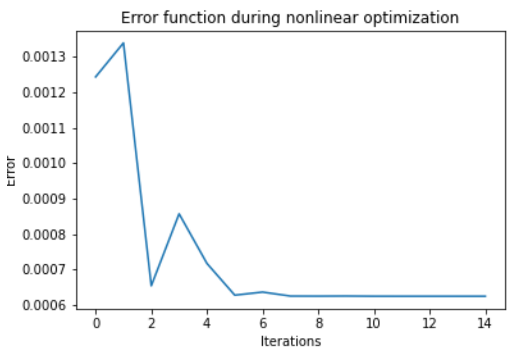

Figure 1: Reprojection error during iterations

1.4 Renormalising Projection matrix

We wanted to solve

$$\mathbf{x} = \mathbf{P}\mathbf{X}$$

We had to normalise the points for numerical stability

$$\hat{\mathbf{x}} = \mathbf{T_{2D}}\mathbf{x}$$

$$\hat{\mathbf{X}} = \mathbf{T_{3D}}\mathbf{X}$$

We then solved

$$\hat{\mathbf{x}} = \hat{\mathbf{P}}\hat{\mathbf{X}} \tag{8}$$

Using equations (2) and (3)

$$\mathbf{T_{2D}}\mathbf{x} = \hat{\mathbf{P}}\mathbf{T_{3D}}\mathbf{X}$$

Hence,

$$\mathbf{P} = \mathbf{T_{2D}^{-1}}\hat{\mathbf{P}}\mathbf{T_{3D}} \tag{9}$$

1.5 Decomposing Projection matrix

- Intrinsics:

$$K = \begin{bmatrix} 2.713e+03 & 3.313e+00 & 1.481e+03 \\ 0.000e+00 & 2.710e+03 & 9.654e+02 \\ 0.000e+00 & 0.000e+00 & 1.000e+00 \end{bmatrix} \tag{10}$$

- Rotation of camera

$$R = \begin{bmatrix} -0.774 & 0.63 & -0.007 \\ 0.309 & 0.369 & -0.877 \\ -0.552 & -0.681 & -0.481 \end{bmatrix} \tag{11}$$

- Translation parameter:

$$t = [0.0470.0543.441]^T \tag{12}$$

1.6 Plots

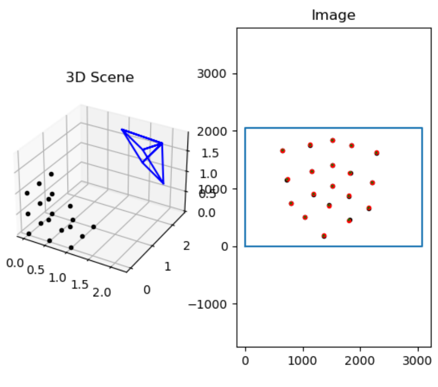

Figure 2: Camera Projections

1.7 Conclusions

From the plot one can see that the estimated camera position is in line with the expected camera position. The reprojected points and original 2D points also overlap almost exactly. The estimated intrinsics matrix has positive diagonals indicating a real physical camera. In our report we also explored the differences between algebraic error and geometric error and also gave an example for physical intuition for the difference between the two. We also plotted the variation of reprojection error during the optimization. We also commented on the sudden jump in error when going from DLT to nonlinear reprojection error minimization (1st iteration).

2 Structure from motion

The approach we follow in the code (filling in the blank sections) is as below. We explain the subparts in detail in later sections:

1. Choose an initial pair of images. We got better results starting from image pairs (5,6) and hence changed it from the given image pair (3,4)
2. Calculate essential matrix from the given point. We used the notation given in the lecture and calculated the Essential matrix wrt point in image 1 giving the line in image 2 (reverse of the notation in code, both are equivalent upto a transpose). This is implemented in EstimateEssentialMatrix function.
3. First we needed to fill in the TriangulatePoints function. This triangulates a 3D point given its image in the 2 cameras. To remove the points that end up behind either/or cameras, we simply reproject the triangulated point back to the 2 cameras. If the reprojected point has the last entry of its honogenous 2D coordinate less than 0, then it has ended up behind the camera and needs to be removed.
4. Decompose the Essential matrix, which results in 4 possible relative poses. The relative pose which gives the maximum number of triangulated points in front of the camera is chosen from the 4 possible relative poses. This is implemented in EstimateEssentialMatrix function and a filtering step in main() of sfm.py
5. Choose the first camera from the initialisation pair (camera 5) as reference (centre at origin) and calculate the projection matrices for both the cameras.

6. Now that one has the 3D coordinates for the triangulated points, one can do absolute pose estimation for a new camera image and obtain projection matrices for each camera view. This is performed in EstimateImagePose function.

7. Now that we have the Projection matrices for each camera, we can triangulate new 3D points (not present in the initial image pair). We just have to run a loop and use the same TriangulatePoints3D function for each new image pair. This is implemented in functions Triangulate image and UpdateReconstructionState

8. After all valid 3D points are obtained after triangulation the final scene is plotted. In the following sections, we just elaborate more on the parts we were asked to implement in code and mention results and conclusions.

2.1 Essential Matrix Estimation and derivation

First we come to the normalized image plane by doing:

$$\mathbf{x_{1,normalized}} = \mathbf{K^{-1}}\mathbf{x_{1,unnormalized}} \tag{13}$$

$$\mathbf{x_{2,normalized}} = \mathbf{K^{-1}}\mathbf{x_{2,unnormalized}} \tag{14}$$

From now on we refer to $\mathbf{x_{1,normalized}}, \mathbf{x_{1,normalized}}$ as simply $\mathbf{x_1}, \mathbf{x_2}$ for ease of notation. The constraint equation for essential matrix $\mathbf{E}$ we use is given below:

$$\mathbf{x_2^T E x_1} = \mathbf{0} \tag{15}$$

Here $\mathbf{x_1}$ are the normalized image coordinates for the point in the first image and $\mathbf{x_2}$ are the image coordinates for the second image. $\mathbf{Ex_1}$ gives the equation of line in the second image plane. This is using the pedagogy from the lecture and is an inversion of the one given in code. We just used this ordering for better interpretation, and the 2 approaches are equivalent upto a transpose. It is important to state the ordering we used clearly, because this assumption means that the parameters derived from Essential Matrix (Relative Rotation and Translation) are with Camera 1 assumed as the reference. We state a short version of derivation below, making use of the cycling properties of trace of matrix.

$$\mathbf{x_2^T E x_1} = 0$$

$$Tr(\mathbf{x_2^T E x_1}) = 0$$

We know that $Tr(ABC) = Tr(BCA) = Tr(CAB)$.

$$Tr(\mathbf{E x_1 x_2^T}) = 0$$

Writing the essential matrix as follows:

$$\mathbf{E} = \begin{bmatrix} \mathbf{E_1^T} \\ \mathbf{E_2^T} \\ \mathbf{E_3^T} \end{bmatrix} \tag{16}$$

Also $\mathbf{x_1 x_2^T}$ when evaluated becomes

$$\mathbf{x_1 x_2^T} = \begin{bmatrix} x_1x_2 & x_1y_2 & x_1 \\ y_1x_2 & y_1y_2 & y_1 \\ x_2 & y_2 & 1 \end{bmatrix} = [\mathbf{C_1 C_2 C_3}] \tag{17}$$

Since Trace is sum of diagonals, we have:

$$\mathbf{E_1^T C_1} + \mathbf{E_2^T C_2} + \mathbf{E_3^T C_3} = 0 = \mathbf{C_1^T E_1} + \mathbf{C_2^T E_2} + \mathbf{C_3^T E_3} \tag{18}$$

Writing this we get the derivation for the constraint equations for 1 correspondence as:

$$\mathbf{A}\mathbf{e} = \mathbf{0} \tag{19}$$

Here,

$$\mathbf{A} = [x_2x_1, x_2y_1, x_2, y_2x_1, y_2y_1, y_2, x_1, y_1, 1] \tag{20}$$

$$\mathbf{e} = \begin{bmatrix} \mathbf{E_1} \\ \mathbf{E_2} \\ \mathbf{E_3} \end{bmatrix} \tag{21}$$

For multiple points we just make the constraint matrix by stacking the rows for different correspondences. n correspondences give us a $N \times 9$ matrix A.

$$A_i = [x_2^i x_1^i, x_2^i y_1^i, x_2^i, y_2^i x_1^i, y_2^i y_1^i, y_2^i, x_1^i, y_1^i, 1] \tag{22}$$

In the code we just had to build the constraint matrix and reshape $\mathbf{e}$ to get $\hat{\mathbf{E}}$. After getting the constraint matrix by DLT, we need to impose rank-2 constraint on $\hat{\mathbf{E}}$. We do this by taking SVD of $\hat{\mathbf{E}}$

$$\hat{\mathbf{E}} = \mathbf{U\Sigma V^H} \tag{23}$$

$$\hat{\mathbf{E_{new}}} = \mathbf{U\Sigma_{new}V^H} \tag{24}$$

$$\mathbf{\Sigma_{new}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{25}$$

$\hat{\mathbf{E_{new}}}$ is our final answer. Our code passes without the assertion error of $\mathbf{x_2^T E x_1} < 0.01$ which means our estimate is good.

2.2 Filtering step for point triangulation

We are given the code which implements point triangulation already. The function which does this requires the absolute orientations of the two cameras, and this is discussed in next section. For now, assume that we have the absolute orientations (location of camera centre and Rotation parameter for the camera) for the 2 cameras. The code given computes the 3D point from the image coordinates of the same point on the two cameras.
Let this point be denoted by $\mathbf{X}$. We assume that $\mathbf{X}$ is normalized (last component of vector is 1). If not, first normalize and then proceed.
We are asked to check if the reconstructed point is a valid point or not. To do this we reproject the point to the two cameras by the following:

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = \mathbf{P_1X} \tag{26}$$

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \mathbf{P_2X} \tag{27}$$

Here $\mathbf{P_1}, \mathbf{P_2}$ are the projection matrices for the two cameras. We note the following:
Although $[x_1, y_1, z_1]$ and $[-x_1, -y_1, -z_1]$, where x_1, y_1, z_1 are positive, represent the same point in homogenous coordinates, the second is obtained when a point behind the camera is projected. This is not a valid scenario physically, and hence we must discard the points which have a negative 3^{rd} component in the 3-vector for the point. But since we have 2 cameras we get the following condition:
If $(\mathbf{P_1X})_3 > 0$ and $(\mathbf{P_2X})_3 > 0$ then $\mathbf{X}$ is a valid 3D point. Here $()_3$ refers to the third component of the 3-vector.

2.3 Finding the correct decomposition of the Essential Matrix

Note that in the notation we followed, the first camera from the initial camera pair is taken as the reference and the essential matrix is calculated for the point x_2 to lie on the line formed by Ex_1.
The essential matrix can be decomposed as relative rotation and translation (in that order) of the second camera w.r.t first camera. However, this gives 4 possible pairs of (R, t) which give the same essential matrix. We now need to select which one of the 4 is the best combination for the scene given to us.
Note that the scene is upto reference change. Hence we just give the first camera (Image 5) the canonical coordinates as absolute coordinates and for the second camera (image 6) the absolute coordinates are given by the above mentioned relative rotations and translations w.r.t canonical coordinates. Mathematically this means,

$$\mathbf{P_1} = [\mathbf{I}|\mathbf{0}]; \quad \mathbf{P_2} = [\mathbf{R}|\mathbf{t}] \tag{28}$$

Here $\mathbf{P_1}, \mathbf{P_2}$ are respective projection matrices for cameras 1 and 2. Note that we still have not chosen which of the 4 combinations of rotations and translations is the correct one. The TraingulatePoints() function requires that the 2 cameras have absolute poses estimated, so we just assign the 2 cameras all of the 4 possible relative poses (which determines the absolute poses in each case, because Camera 1 has canonical coordinates). With this, we calculate the TriangulatePoints() function, which returns a set of 3D triangulated points. We select the combination which gives the maximum length for the 3D points vector returned (i.e. the maximum points ended up in front of the camera).
This gives us the best choice for the relative pose. We set Camera 1 (Image 5) to have canonical coordinates now finally and Camera 2(Image 6) to have the coordinates given by best combination of rotation and translation obtained from the decomposition. Now we have a fixed reference system and we estimate the absolute poses for the other cameras and create the 3D scene in the next sections.

2.4 Absolute Pose estimation

This section was already implemented and given to us.

2.5 Map Extension

2.5.1 TriangulateImage()

In this function, we just implemented a for loop for point triangulation with pairs of cameras. We are given a reference image set and a new image. The new image gives us new points and combining this new image with each of the images in the reference image set gives us a set of new triangulated 3D points. Hence in the for loop we just take each image in the reference image set and perform point triangulation using this image and the new image. We save the 2D-3D correspondences for the reference images as well as the new image in a dictionary.

2.5.2 UpdateReconstructionState()

In the previous function, we get 2D-3D correspondences for all images in the reference image set plus the new image. However these correspondences point to array locations in a new array. The information from the new array must be merged with the existing set of 3D triangulated points. We just append the new array to the existing 3D triangulated points array. However, we now also need to update the keypoints for 2D-3D correspondences in the reference image set. Since we just appended the new 3Dpoints array to the old array, each of the correspondence keypoints for the reference will just need to be offseted by the length of the old array.

2.6 Plots

We get the 3D scene for the pose. We have included one perspective of the 3D scene as a visualisation below. One can immediately see that the patch of points correspond to the

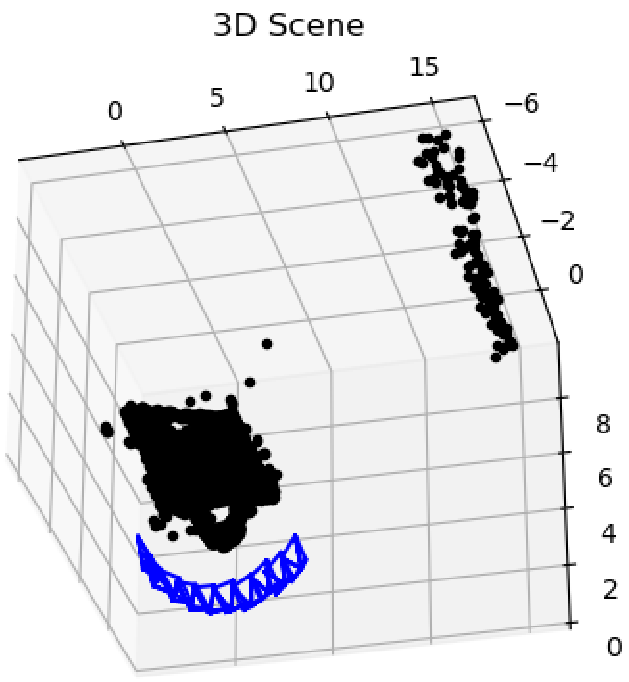

Figure 3: One perspective of the reconstructed 3D scene

fountain. Upon playing around with the perspective, we see that the fountain seems to be reconstructed pretty well. However, there is a vertical patch of points on the right side(of this image). These are actually not outliers. These points are obtained by triangulation of correspondences from Image 8 and 9. When one plots images 8 and 9, one can see that only they both capture a part of the building on the right side of the fountain. So these are indeed legititmate points and not an error in our code. When Images 8 and 9 are removed, we get the reconstruction which only includes the fountain.

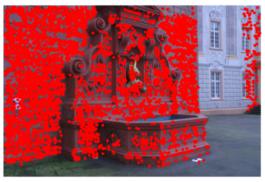

Figure 4: Image 8

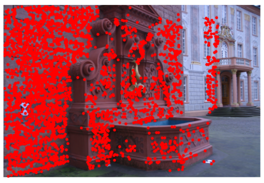

Figure 5: Image 9