TTIC 31040: Introduction to Computer Vision
Winter 2022

Problem Set #3

Out: Feb 6th, 2022

Due: Friday Feb 18th, 11:59pm

Instructions

How and what to submit? Please submit your solutions electronically via Canvas. Put your answer to qualitative discussions in a PDF file (typeset or handwritten/photographed or scanned) named `writeup.pdf`. Then delete the data folder, and submit the entire hw3 directory as a single tarball (not zip please) with the command

```
tar cvf hw3.tar hw3/
```

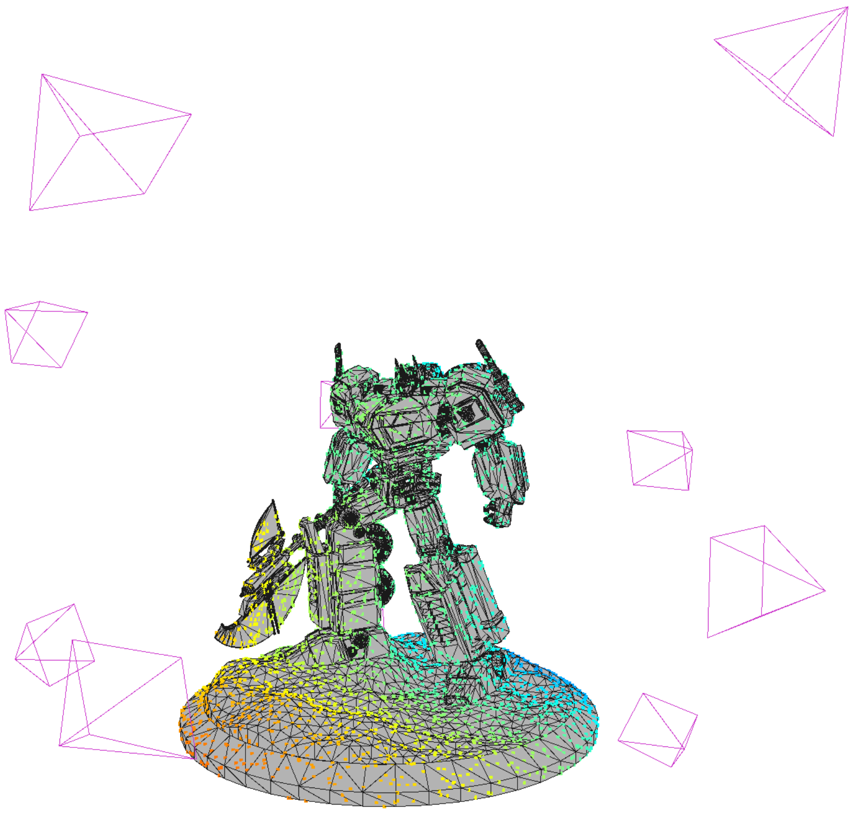
Late submissions There will be a penalty of **25 points** for any solution submitted within **48 hours** past the deadline. No submissions will be accepted past then. Please email instructors for special circumstances.

What is the required level of detail? When asked to derive something, please clearly state the assumptions, if any, and strive for balance: justify any non-obvious steps, but try to avoid superfluous explanations.

When asked to plot something, please include the figure image in the directory. If multiple entities appear on a plot, make sure that they are clearly distinguishable (by color or style of lines and markers) and references in a legend or titles. When asked to provide a brief explanation or description, try to make your answers concise, but do not omit anything you believe is important. If there is a mathematical answer, provide it precisely (and accompany by only succinct words, if appropriate).

When submitting code, please make sure it's reasonably documented, runs and produces all the requested results. If discussion is required/warranted, you can include it in the PDF writeup.

Collaboration policy Collaboration is allowed and encouraged, as long as you (1) write your own solution entirely on your own, (2) specify names of student(s) you collaborated with in your writeup.



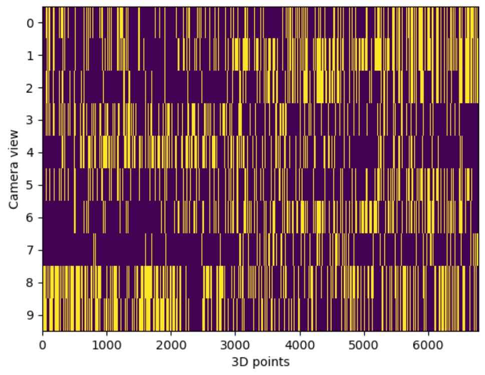


Figure 1: Our sparse reconstruction setup: we have 10 cameras looking at a robot statue. The colored keypoints are projected to each view subject to the visibility constraints as shown in the bottom figure. We will reconstruct the robot from these image projections.

Software Setup

Apart from the Open3D introduced earlier in hw2, we will in addition use PyTorch for automatic differentiation during bundle adjustment, and LieTorch to support manifold optimization of Lie Group elements (in our case $SE(3)$).

Please install PyTorch with

```
conda install pytorch -c pytorch
```

and LieTorch with

```
pip install git+https://github.com/w-hc/lietorch
```

Note that this step involves code compilation on your local machine, and might take a few minutes.

1 Prelude

1.1 Procrustes Analysis

We have seen the use of Singular Value Decomposition several times by now. Here is another classic problem that is relevant to Structure from Motion and vision in general. We will use it later in two different contexts.

Problem 1 [10 points]

Given matrices $A, B \in \mathbb{R}^{m \times n}$, we are looking for an orthogonal matrix $X \in \mathbb{R}^{n \times n}$ that best aligns B to A. Specifically, we are looking for

$$\min_{X \in U(n)} ||A - BX||_F$$

where $|| \cdot ||_F$ denotes Frobenius norm. Find the expression for optimal X by applying SVD and prove it is optimal. You may use the properties that $||Q||_F = \mathrm{Tr}(Q^T Q)$ and that $\mathrm{Tr}(PQ) = \mathrm{Tr}(QP)$ and also that trace is a linear mapping.

End of problem 1

In the case $n = 3$, we can think of A and B as two point clouds, each with m points, and we are looking for a rotation that fits B to A.

Note that a rotation matrix is in the Special Orthogonal Group $SO(3)$. For X to be a rotation, we need both $X^T X = I$ and $\det(X) = 1$. If the SVD solution above returns an orthogonal matrix X with $\det(X) = -1$, then it is not a valid rotation since it will change the handedness / chirality of B. We will encounter this problem when we recover t and R from the essential matrix.

1.2 SfM Data

In this problem set we will work under a synthetic setup to reconstruct a statue of "Optimus Prime". We sample points from the surface of the mesh, and project them to 10 camera viewpoints subject to occlusion and visibility constraints. This is illustrated in Fig 1.

We will do a sparse reconstruction of the statue from these 10 images, assuming known intrinsics and keypoint correspondences. First we introduce the format of the data.

- `optimus_prime.obj`: a mesh of the statue for you to play with.
- `camera_intrinsics.json`: image size and FoV.
- `camera_poses.npy`: the 10 camera poses. array of shape $(10, 4, 4)$.
- `multiview_visibility.npy`: boolean array of shape $(10, 6782)$. Each column corresponds to a common keypoint observed by 10 cameras, and in which view they are visible. Each column is often referred to in SfM literature as a "track".
- `pts_3d.npy` array of shape $(6782, 3)$. The 3D points sampled from the mesh surface. We will use their projections as proxies for keypoint locations. In real setting keypoints are extracted and matched using something like SIFT descriptor.
- `view_0.png` up to `view_9.png`. These are the 10 projected images of the statue. Open3D has problem loading mesh texture, and this is the best we can do for now.

We will do two-view reconstruction using view 1 and view 5. Multiview, incremental reconstruction is left as a bonus problem.

Problem 2 [5 points]
Complete the routine `common_visible_points`, which uses multiview visibility matrix to create a mask indexing into the 3D points common to view 1 and view 5. Execute `engine.q2`, which visualizes the 2 cameras, the visible 3D points, and the mesh together. Pick your favorite viewpoint and take a screen shot that includes all the objects. Save it to `setup.png`. Note that the back of the mesh has no keypoints chosen since they are not visible to camera 1 and 5.

End of problem 2

2 Epipolar Geometry

Again for clarification, we define the **pose** of a camera as the its location and local x, y, z coordinates expressed in the world frame. It is a 4×4 matrix. The extrinsic / world2cam matrix is the inverse of the camera pose. Please refer to hw2 "how not to get lost during spatial transforms" for more details.

Problem 3 [5 points]
Complete the routine `t_and_R_from_pose_pair`. Given two camera poses, it computes the t and R that transforms 3D points from the local coordinates at pose1 to local coordinates at pose2.

Complete the inverse routine `pose_pair_from_t_and_R`. Given t and R as defined above, it returns two compatible camera poses, with the first pose set to identity since absolute orientation is ambiguous. This routine will be handy when you generate the 4 chirality pose pairs.

End of problem 3

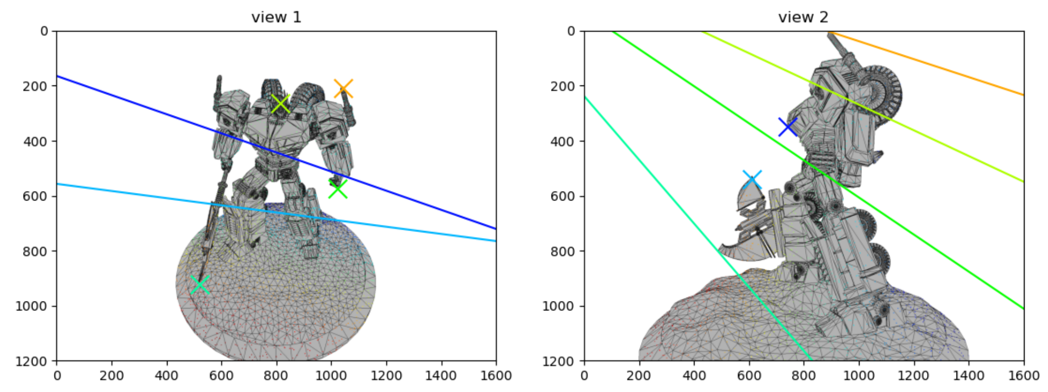


Figure 2: Selected points and their epipolar lines. We've chosen the sharp tips of prominent objects to demonstrate the epipolar geometry.

Problem 4 [10 points]
Complete the routine `essential_from_t_and_R`. Given t and R, it computes essential matrix E.

Complete the routine `F_from_K_and_E`. Given essential matrix E and intrinsic matrix K, it returns the fundamental matrix F. Complete the inverse routine, `E_from_K_and_F`.

In `engine.q4`, fill in the expression for the location of epipoles on image 1 and image 2 respectively. The assertions will check that the two epipoles are in the left and right null space of the fundamental matrix.

Now execute `engine.q4`. This will fire up an interactive plot. Left-click on one view and the corresponding epipolar lines will appear on the other image. Right-click to clear all drawings.

Click on a few iconic points in both views, and save the figure as `epipolar.png`. We have attached a reference plot in Fig 2.

The code for this interactive visualization widget is provided to you. It might help with your project and you can extend it.

End of problem 4

3 Triangulation

Problem 5 [10 points]
Complete the routine `triangulate`. Given two camera projection matrices (the product of camera intrinsics and extrinsic matrix $P = KE$) and two sets of corresponding pixel locations from two views, it returns a set of 3D points.

End of problem 5

4 Eight-point Algorithm

Problem 6 [10 points]
Complete the routine `eight_point_algorithm`. Given two sets of corresponding pixel locations from two views, it returns the fundamental matrix.

We are solving a linear system in the form $\mathbf{Ax} = \mathbf{0}$ with the constraint that $||\mathbf{x}|| = 1$. Our starter code will print out the condition number of the constraint matrix $\mathbf{A}$. In general the bigger the condition number the less tolerant the system is to noisy inputs.

You will gain a one point bonus by implementing it in a vectorized way e.g. using `einsum`.

Complete the routine `enforce_rank_2`. It enforces that the fundamental matrix is of rank 2.

End of problem 6

Problem 7 [10 points]
Complete the routine `normalized_eight_point_algorithm`. It first transforms the (x, y) pixel coordinates to be within $[-1, 1]$, and then pass them as arguments to `eight_point_algorithm` as a subroutine. The returned $\hat{F}$ is then post-processed to be the intended fundamental matrix.

Modern linear algebra libraries re-scale each rows of a linear system $\mathbf{Ax} = \mathbf{b}$ before solving it to improve numerical conditioning. This is called "equilibration". Disucss why this automatic step is **not** helpful in the case of 8-point algorithm. This is what makes the seemingly trivial preprocessing in normalized 8-point algorithm so critical, and no linear algebra software can help us unless we explicitly intervene.

You can read Richard Hartley's paper **In defense of eight-point algorithm** here.

End of problem 7

5 Chirality Disambiguation

Now that we have computed the fundamental and essential matrix from pixel correspondences, we will proceed to recover t and R.

We know that $\lambda E = [t]_\times R$. After recovering t up-to-scale from the left-null space of E, we can find R by solving the procrustes problem $\operatorname{argmin}_{R \mid R^T R = I} ||E - [t]_\times R||_F$.

Problem 8 [5 points]
Complete the routine `t_and_R_from_essential`. Since E is of rank 2, its last singular value is 0, and the product of $U\Sigma V^T$ is invariant if the last singular vector of U (or equivalently the last singular vector of V) is multiplied by -1. This gives us two candidate for rotations R_1 and R_2. Note that in order to be valid rotations we need to do $\hat{R} = R \det(R)$ so that the determinant is 1. Scaling by $+1$ / -1 is allowed because essential matrix is defined up to arbitrary scale. Together with the sign multiplicity on the translation, we have 4 candidate pairs $(t, R_1), (t, R_2), (-t, R_1), (-t, R_2)$.

End of problem 8

Problem 9 [10 points]
Out of the 4 hypotheses, only 1 pair of t and R will place all the 3D points in front of both cameras. Complete the routine `disambiguate_four_chirality_by_triangulation` . Use the `triangulation` routine earlier and count how many 3D points are in front of both cameras. Recall our world axis convention. Camera looks at negative z-axis.

End of problem 9

Problem 10 [10 points]
Put the things we have so far together in a two-view SfM pipeline. Complete `engine.sfm_pipeline` following the hints.

Execute `engine.q10`. This runs our two-view SfM pipeline without bundle adjustment. It will plot the pose and point configurations of the 4 hypotheses. Put them into one image and save it to `four.png`. We have attached a reference plot in Fig 4. Make sure that the assertion at the end passes.

End of problem 10

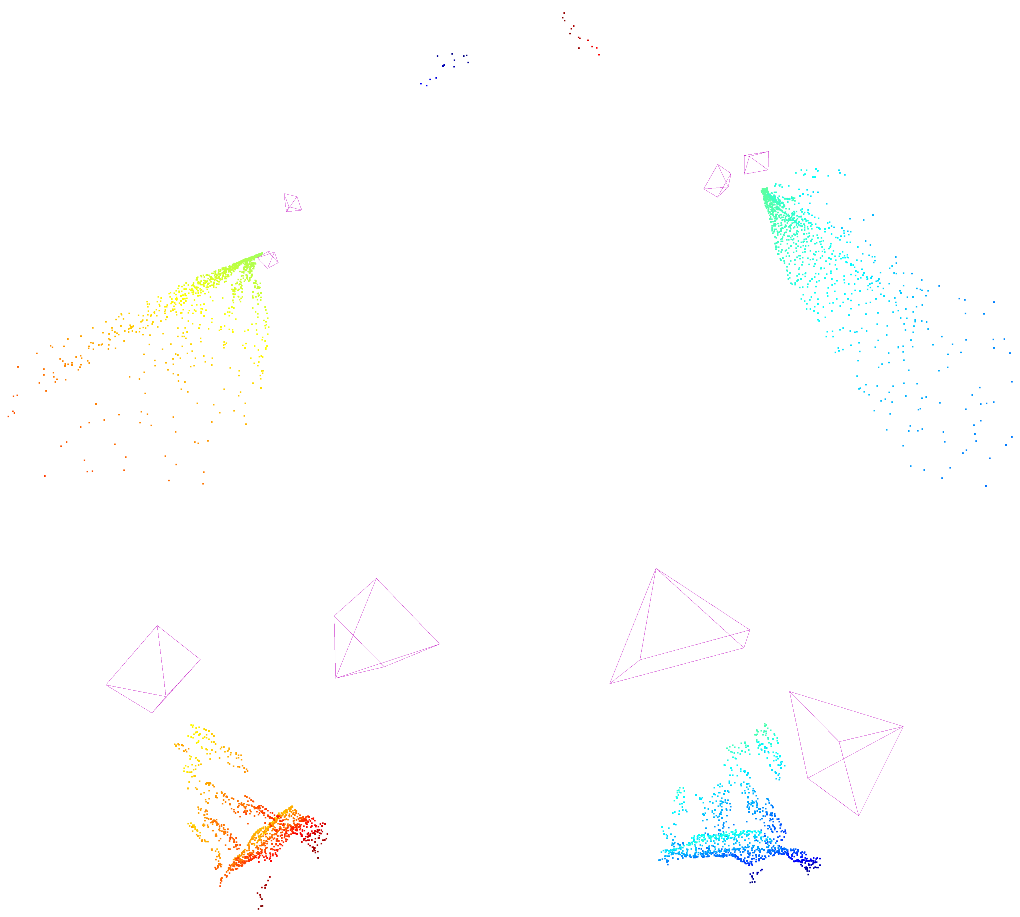


Figure 3: The four configurations we extract from essential matrix. The first two have cameras pointing at opposite directions. The third one flips both cameras. Only the last one provides the correct configuration.

Note that near the end we called the function `align_B_to_A` to align our predicted point cloud with the ground truth point cloud. It again solves a Procrustes problem. This is necessary since the 3D reconstruction is up to scale and orientation ambiguity. You can try to plot the predicted points directly against the mesh, and observe that it's smaller than Optimus Prime' left foot. Such is our giant.

6 Bundle Adjustment

Bundle adjustment is about updating 3D point locations and camera poses to minimize pixel reprojection error. Specfically,

$$\min_{\mathbf{X},\mathbf{P}} \sum_j \sum_i ||x_{ij} - \Pi(X_j, P_i)||$$

where Π denotes camera projection.

We will use PyTorch for its autodiff feature to compute derivatives. Traditionally non-linear optimization problems are often solved with variants of Gauss-Newton methods (such as Levenberg-Marquardt). This requires a high quality sparse linear algebra library. For simplicity we will use first-order gradient-descent based methods. In our testing we observed that the popular Adam optimizer designed for deep learning optimization works much better than simple SGD variants. We will use Adam.

Rotation matrices are restricted to be on a manifold in $\mathbb{R}^9$. You can count that there are 6 global non-linear polynomials restricting a rot matirx (3 for unit norm of each row; 3 for pairwise orthogonality; det = 1 is another polynomial, whose derivative is in the row-span of the Jacobian of the previous 6; hence not counted). Thus there are only 3 local degrees of freedom.

Also rotations are often conveniently represented using unit-norm "quaternions". It's good to know.

Conventional gradient updates will produce invalid rotation matrices. We use Zachary Teed's LieTorch for this its support of Lie Group optimization. If you want to read more, please consult this wonderful **document**. It has accompanying **YouTube videos** to help build an intuition of what Lie Algebra does.

Problem 11 [**0 points**]
Read the routine `bundle_adjustment`. Note how the poses are converted back and forth.

End of problem 11

Problem 12 [**15 points**]
Execute `engine.q12`. This will introduce noise to pixel keypoints and also use bundle adjustment to improve the reprojection error. Report the condition number, initial and final bundle-adjusted reprojection error of the following items:

- 8-point algorithm
- normalized 8-point algorithm.
- Randomly flipping 2% of the keypoint correpondences.

If you want, visualize the final reconstructed statue after alignment in case of severe noise, esp. corredpondence flipping noise.

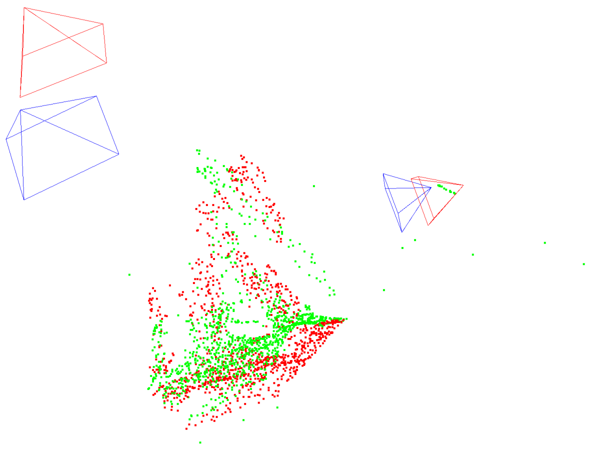


Figure 4: Reconstruction after bundle-adjustment in the presence of 2% correspondence flipping and $N(0, 1)$ keypoint noise. After BA there remains a 14.4 pixel reprojection error. The ground truth point/camera in Red. Predicted point/camera in green and blue.

End of problem 12

7 Mulitview SfM Integration

Problem 12 [20 bonus points]
Our setup provides 10 images from different camera poses. In this bonus section, feel free to integrate the other 8 images into the pipeline. You can use PnP to register the other views, and do incremental bundle adjustment. This problem is open-ended and please report your findings.

End of problem 12