CS 498: Assignment 2: Multi-view Geometry

Due by 11:59pm Friday, March 4th 2022

February 11, 2022

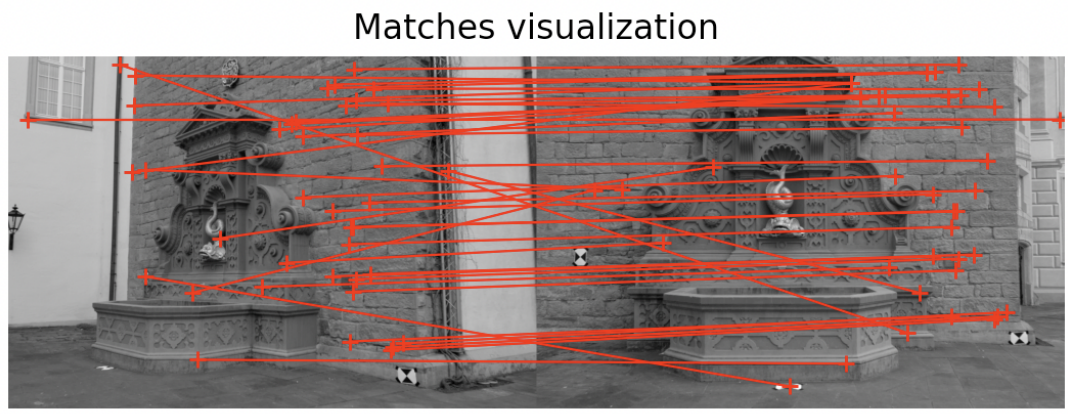
Submission

In this assignment you will code your own structure from motion and stereo matching algorithm, to convert the 2D observations to 3D structures. The starter code consists of 4 python files you will modify along with a folder of some images and saved data. Please put together a single PDF with your answers and figures for each problem, and submit it to Gradescope (Course Code: JBXJVZ). If your code produces figures they should go in this pdf, along with a description of what you did to solve the problem. We recommend you add your answers to the latex template files we provided. For code submission, make sure you use the provided ".py" files with your modification and the dumped ".npz" file. The graders will check both your PDF submission and your code submission if needed.

Keypoint Matching

Question 1 (Putative Matches)[2 pts]: Here you will be modifying "correspondence.py". Your task is to select putative matches between detected SIFT keypoints found in two images. Detecting keypoints will be done for you by cv2. The matches should be selected based on the Euclidean distance between the pairwise descriptors. In your implementation, make sure to filter your keypoint matches using Lowe's ratio test (link). For this question, you should implement your solution without using third-party functions e.g., cv2 knnmatch, which can significantly trivialize the problem. Meanwhile, please avoid solutions using for-loop which slows down the calculations. **Hint**: To avoid using for-loop, check out `scipy.spatial.distance.cdist (X,Y,'sqeuclidean')`.

Question 1 Answer: The output of the SIFT algorithm is keypoints as well as descriptions. The purpose of these descriptions to give a vector representation of each pixel (each vector 128 long). To select putative matches, we want to match the pixel vectors between two images with the smallest distance (thus having the highest similarity). Now the issue with this approach is that if the closest match and the second closest match are pretty similar to each other, then that indicates that the pair may not be a strong fit. This is where we can take advantage of the Lowes Ratio Test to filter out out putative matches only if the the closest match is twice or more similar as the second closest match.



Fundamental Matrix Estimation

Question 2 (Eight-point Estimation) [3 pts]: Here you will be modifying "fundamental.py". For this question, your task is to implement the unnormalized eight-point algorithm (covered in class lecture) and normalized eight-point algorithm (link) to find out the fundamental matrix between two cameras. We provide code to compute the quality of your matrices based on the average geometric distance, i.e. the distance between each projected keypoint from one image to its corresponding epipolar line in the other image. Please report this distance in your pdf.

Question 2 Answer: The purpose of the eight point algorithm is given eight good matches between images, we can accurately compute a unique fundamental matrix. The reason we need the fundamental matrix is then we have a clear mapping between points in two images. The second part of the eight point algorithm is that we can actually normalize the input points before doing SVD. This is done by centering our X, Y at the origin and then scaling it such that the average distance from the origin is square root of 2. By scaling it to this constant we are ensuring that our X and Y components are scaled to 1 unit length. To satisfy the coplanarity restriction, we have to do an extra step in forcing our fundamental matrix to be of Rank 2. Lastly, if we had normalized our points we have to denormalize them back after the fundmental matrix was calculated. By doing this process we go an average geometric distance of 0.029 with our un-normalized points and then a slightly improved 0.018 when normalized.

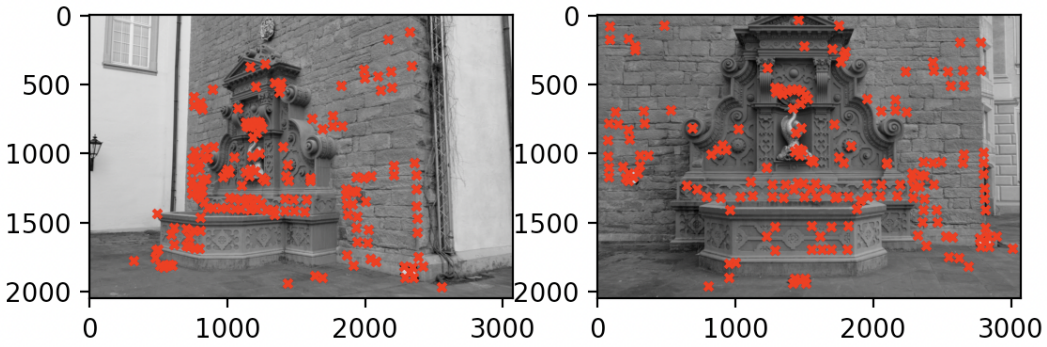
Question 3 (RANSAC) [3 pts]: Here you will be modifying "fundamental.py". Your task is to implement RANSAC to find the fundamental matrix between two cameras. Please report the average geometric distance based on your estimated fundamental matrix, given 1, 100, and 10000 iterations of RANSAC. Please also visualize the inliers with your best estimated fundamental matrix in your solution for both images (we provide a visualization function). In your PDF, please also explain why we do not perform SVD or do a least-square over all the matched key points.

Question 3 Answer: The purpose of RANSAC is to iteratively take random samples from our set of good matches and then estimate the fundamental matrix (with normalization for best performance). Once we have this matrix we can then calculate the number of inliers, which is the number of points when projected to the second image, are within a set threshold distance away from their epipolar line. We then keep the fundamental matrix that had maximized the number of inliers. We then see if increasing the number of iterations will improve our average distance from the epipolar lines. The reason we don't just do SVD or Least Squares over all of our data is because we are assuming that there are outliers in our

dataset and any model that includes them will be inherently biased. The purpose of RANSAC is to take the minimum number of points we need to get a unique solution and randomly sample many times to converge on our best solution that includes as many points as possible while ignoring outliers.

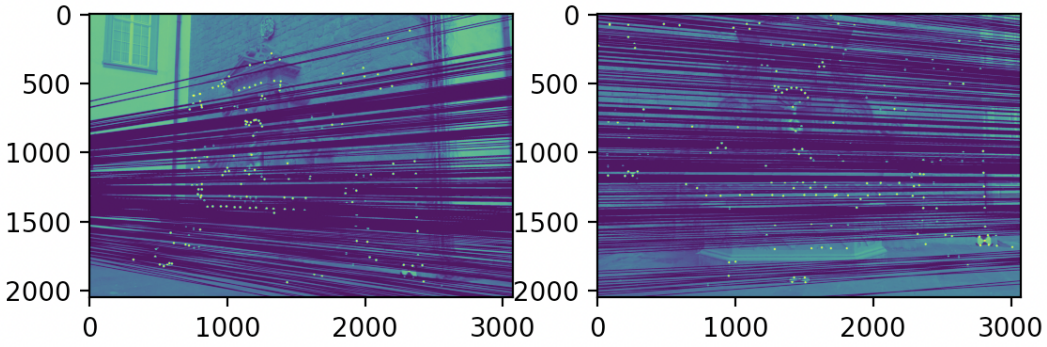
My results also showed that increasing the number of iterations did give a better average geometric error:

- 1 Iterations: 0.366
- 100 Iterations: 0.016
- 1000 Iterations: 0.0079



Question 4 (Epipolar Line Visualization) [1 pt]: Here you will be modifying "fundamental.py". Please visualize the epipolar line for both images for your estimated F in Q2 and Q3. Here we do not provide you visualization code. To draw on images, cv2.line, cv2.circle are useful for plotting lines and circles. Check our Lecture 4, Epipolar Geometry, to learn more about equation of epipolar line. This link also gives a thorough review of epipolar geometry.

Question 4 Answer:

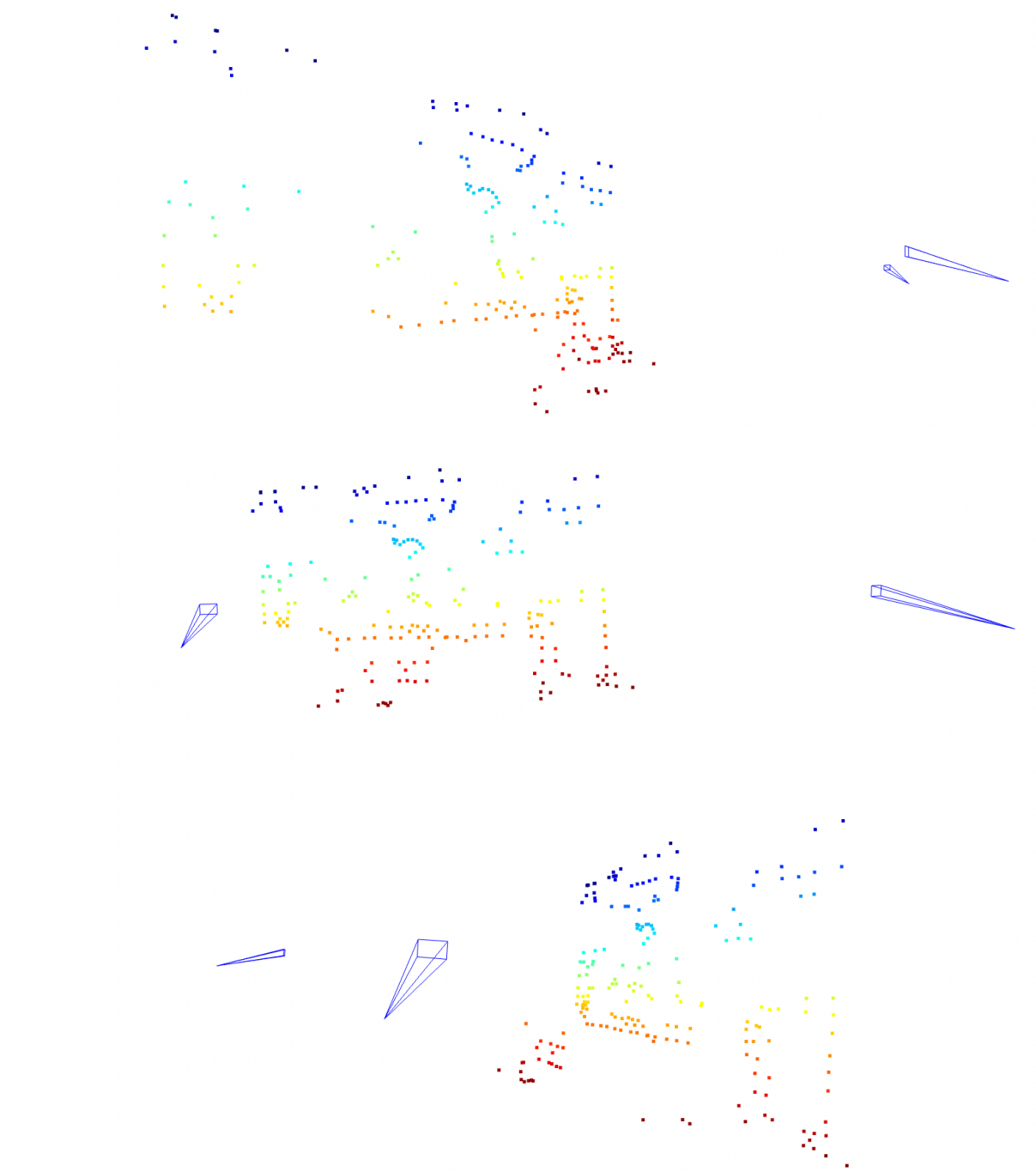


Unprojection

After solving relative poses between our two cameras, now we will move from 2d to 3d space.

Question 5 (Triangulation) [3 pts]: Here you will be modifying "triangulation.py". You are given keypoint matching between two images, together with the camera intrinsic and extrinsic matrix. Your task is to perform triangulation to restore the 3D coordinates of the key points. In your PDF, please visualize the 3d points and camera poses in 3D from three different viewing perspectives.

Question 5 Answer: The purpose of triangulation is to be able to estimate the 3d point in space based on an input stereo images and their respective projection matricies.



Stereo Estimation

Now given two camera poses, we could get the relative camera pose and a sparse set of 3D point based on sparse keypoint matching. Could we go further and recover the dense geometry? The answer is yes. In question 6, you will reason a dense point cloud from a pair of images taken at the different views.

Question 6 (Stereo) [3 pts + 1 bonus pt]: Given two images $(\mathbf{I}_1, \mathbf{I}_2)$ with ground-truth intrinsic matrices $\mathbf{K}_1, \mathbf{K}_2$, extrinsic matrices $\mathbf{R}_1, \mathbf{t}_1, \mathbf{R}_2, \mathbf{t}_1$. Our goal is to recover a dense correspondence between the two images using stereo matching.

Computing the dense matching is computationally expensive. We down-sample both images by s times to reduce the computational burden. Could you describe what will be the updated intrinsic matrices? Please report the intrinsics on your write-up and fill in the code to update $\mathbf{K}_1, \mathbf{K}_2$.

We do notice that the pair of images are not perfectly fronto-parallel. To make matching easier, we will rectify the pair of images such that its epipolar lines are always horizontal. To achieve this, we simply call `cv2.StereoRectify` function. The updated projection matrices are also returned. Please go through the code and understand what each step does. You do not need to write any code here.

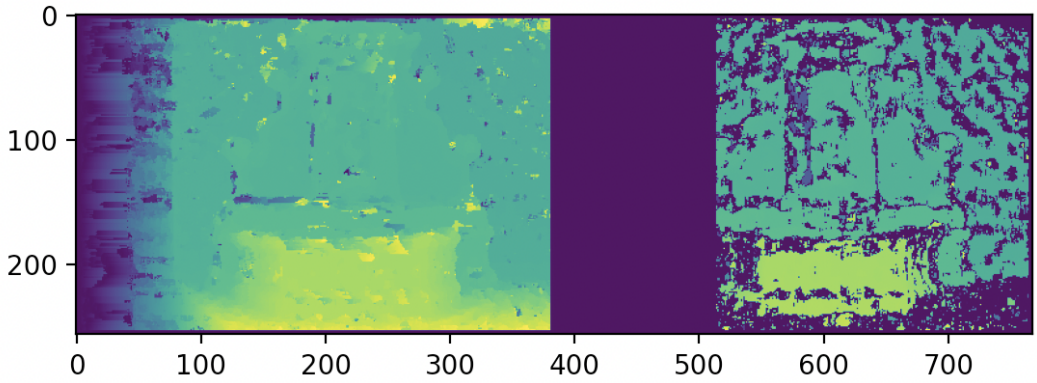
Finally, we will do stereo matching. Specifically, given a pixel at (i, j) on the left image, we will compute a winner-take-all disparity value that minimizes the sum of square distance (SSD) between a pair of left and right image patches, centered at (i, j) and $(i, j - d)$:

$$d[i, j] = \arg\min_d \sum_{m=-w}^{w} \sum_{n=-w}^{w} (I_1[i + m, j + n] - I_2[i + m, j + n - d])^2$$

where $w * 2 + 1$ is the block size and w is an integer. Please do this in a sliding window manner to get the dense disparity map. Please visualize the disparity map you get and include it in your report.

Do you think the disparity you get is satisfactory? You could compare your results against stereo matching results implemented by an off-the-shelf method `cv2.stereoBM` (we already provide the code). Visualize both methods side-by-side and provide insights into further improving your results. Given disparity map estimation, could you get the dense point cloud? Try to get the point cloud and visualize your point cloud. (Bonus 1 pt)

Question 6 Answer: Our goal in this question is to recover our dense correspondences between two images. We do this by selecting a patch of some kernel size on our first image, and then sliding this patch leftward on our second image. At each step we will be calculating our sum of square distances, and the pair of image patches that minimize this difference will be our match. We then store at that pixel value the number of pixel offsets we had to do to get to that patch, also called our disparity. By doing this we can calculate disparities for all of our points and get an understanding of the depth in our image.

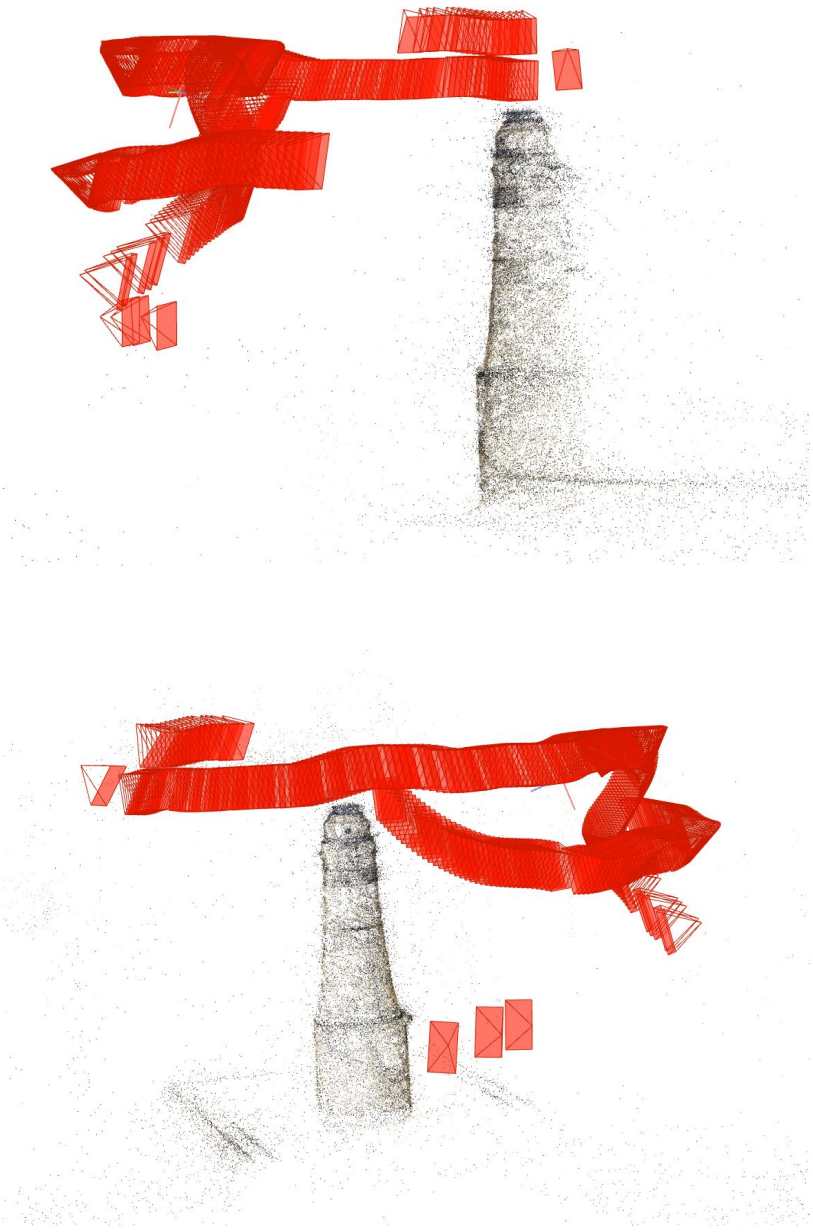


Question 7 (Structure-from-Motion) [bonus 3 pts]: We believe you have a thorough understanding of multi-view geometry now. This bonus question will expand the two-view structure-from-motion and stereo estimation problem into multiple views. You have two directions to win the three bonus points: 1) Write your own mini bundle adjustment solver and run it on the fountain dataset. You could leverage Ceres, g2o, GTSAM libraries as your non-linear least square solver, but you need to derive the Jacobian of the energy terms and write your own code to initialize camera poses and 3D points, based on two-view geometry; 2) Collect your own data and utilize Colmap (https://demuc.de/colmap/) to run the full MVS pipeline. This includes correspondences matching, initialization, bundle adjustment, multi-view stereo matching, and depth fusion. Visualize your camera trajectories, sparse point cloud from SFM, and the final dense point cloud from multi-view stereo. Try your best to get the most creative and visually appealing results. We will choose the top solutions in each direction and announce them in the class.

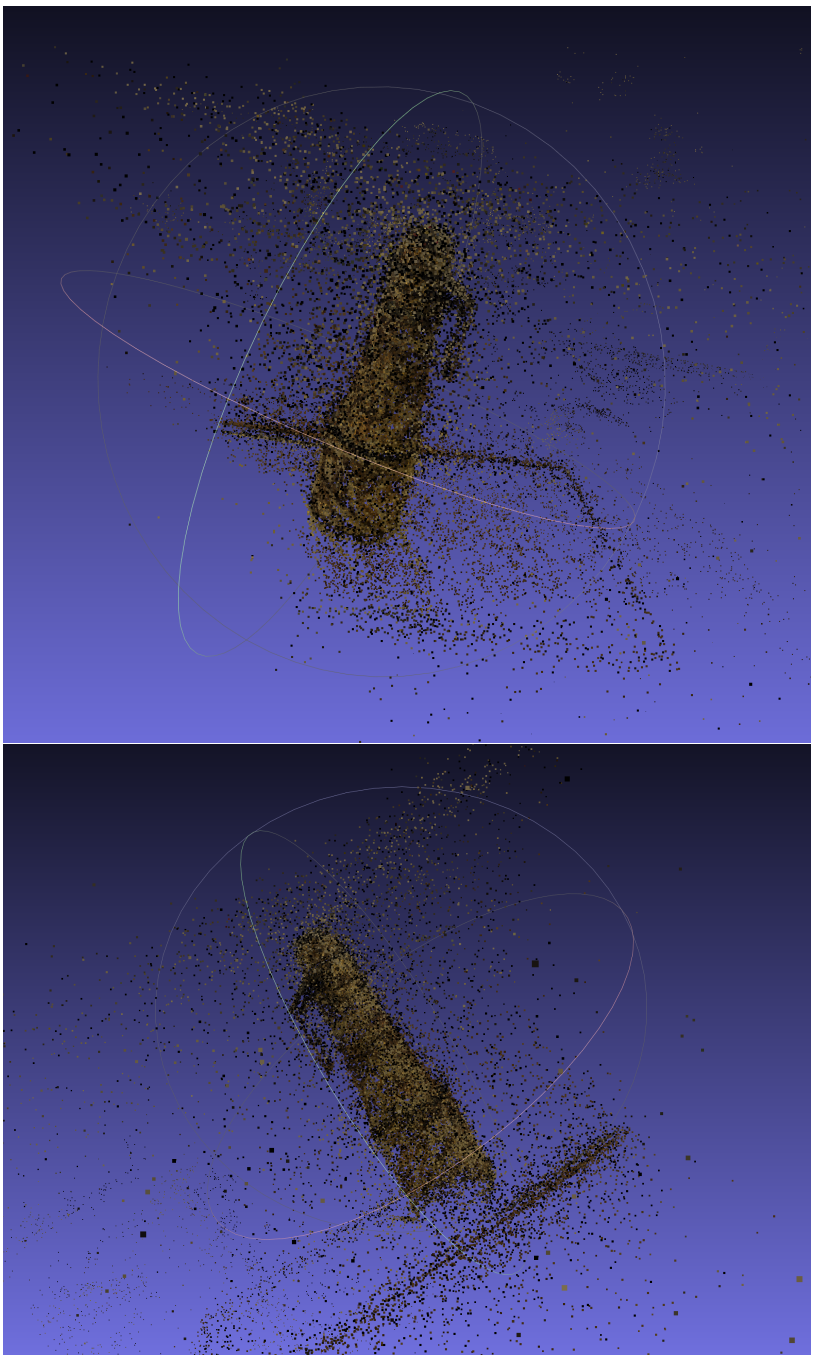
Question 7 Answer: I decided to do the ColMap question as I was curious about how we can take a series of images to stitch together a 3D projection. The object I decided to do was my 600mm F4 Lens that I use for all of my wildlife photography. The reason for this is most of the examples I saw online were images of heterogeneous textures (such as architecture), and I wanted to see how the algorithm would respond to something with a repeated pattern and a busy background (not an ideal scenario). You will see the lens is covered in something that looks like camouflage. The reason for this is when I take wildlife photos, I have to make sure they don't spot me, and it seemed like the perfect surface to test my questions on. I took a video feed of my lens and then extracted every 10th frame to put into ColMap. I then utilized MeshLab to view the .ply generated from ColMap. Here are my results!

Original Image:

Point Clouds and Trajectories:



Dense Point Cloud



We can see that we got surprisingly good results and we were able to create a fairly convincing point cloud and dense map. I think we could improve this if I took more controlled photos in a better lighted environment and I hope to keep experimenting with this!