- What was the goal of your project?
    - The goal of the project is to construct a three-dimensional point cloud (similar to a laser scan) based on multiple views of the same object. This technique is often referred to as structure from motion, and it exploits various algorithms that are built-in to OpenCV. Its main application is to generate complex 3D images based on changes in the image view - there are a couple of mobile apps that implement this algorithm using the inertial sensor in the phone.
- Describe how your system works. Make sure to include the basic components and algorithms that comprise your project.
    - We were able to construct a 3D depth cloud by using just two different image views of the object. Given two different images from the Neato taken at different angles, these are passed into the following algorithm:
        - Calculate optical flow (Farneback's method) between the two images and save the keypoints for each image in a variable
        - Use the keypoints from image #1 and the corresponding keypoints from image #2 to calculate the essential matrix
        - Perform SVD on the essential matrix to calculate camera #2's projection matrix. This matrix contains the translational and rotational information that brings camera #1's position and orientation to camera #2's position and orientation.
        - Triangulate the matched keypoints from the two images, using this projection matrix, in order to obtain the 3D point cloud.
    - Many of the methods that were called in our code were from OpenCV, and the plotting code for the scatter plot was done in matplotlib.
- Describe a design decision you had to make when working on your project and what you ultimately did (and why)? These design decisions could be particular choices for how you implemented some part of an algorithm or perhaps a decision regarding which of two external packages to use in your project.
    - A major decision we had to make was which methods of keypoint detection and matching to use. We initially looked into implementing SIFT, which is a method of finding keypoint descriptors. To find matching keypoints from SIFT, we used a brute-force algorithm, called knnMatch. As the name suggests this was not a particularly efficient process, nor was it accurate. Because of this, we decided to research optical flow, which tracks keypoints and returns "flow vectors", or the amount of translation of each keypoint from one image to another. Within optical flow, we looked into implementing a few different methods of finding keypoints. We primarily tested out the Lucas Kanade method and the Gunnar Farneback method. Essentially, the two algorithms both use a process of localization to determine

keypoint flow; the main difference between the two is that the Lucas Kanade method is a sparse solver, while the Farneback method is a dense solver. We ended up using the Farneback method because this characteristic made it more accurate (and with only two images, time complexity was not a huge concern).

- How did you structure your code?
    - Our code was split into two files: one file (construct_twoimages.py) collects image data from the Neato and passes that to the other file (sfm.py), which performs the image processing. The code for getting data from the Neato was structured as a class, containing all of the subscriber information. The other file was written as a group of functions that split up the processes into reasonable chunks.
    - There is a third file (farneback.py), which we used to test the optical flow code to see that it worked as intended.
- What if any challenges did you face along the way?
    - For this project, it was difficult to split up the work into manageable chunks, especially since it was hard to see how the system worked in pieces rather than as a whole. As a result, we decided to work on the code together by doing pair programming. However, that was also sort of difficult because everybody had different understandings about how each algorithm worked.
- What would you do to improve your project if you had more time?
    - If we had more time, we would be able to delve deeper into optimizing the math behind developing an accurate projection matrix for an object. Under certain circumstances, the reconstruction would not look like the object at all, particularly when the object has a complicated geometry. We also would have liked to look into using more than 2 images to get more views and perspectives. However, due to time constraints and complexity, there would not have been enough time to both understand and effectively implement calculations for combining multiple images' keypoints. It also would have been interesting if we incorporated the LIDAR data into our program, as well, in order to make some form of visual SLAM.
- Did you learn any interesting lessons for future robotic programming projects? These could relate to working on robotics projects in teams, working on more open-ended (and longer term) problems, or any other relevant topic.
    - We realized later on that we should have unit tested our code more extensively than we did, and for future projects, this should definitely be more of a critical step in the process. Because we did not have a thorough process for testing each individual piece of code, it took much longer to pinpoint bugs in our system. Additionally, it was difficult to find the proper documentation for calling functions from OpenCV because there are so many

different versions and because not all of the C++ functions are implemented for the Python version. Now that we understand this, we feel more confident that we can find the correct online sources to learn how to call functions properly.