Structure From Motion with Planar Homography

David Gonzalez
Department of Computer Science
Stanford University
davidgzz@stanford.edu

Josue Solano Romero
Department of Computer Science
Stanford University
jsolanor@stanford.edu

Kenneth Cabrera
Department of Computer Science
Stanford University
cabrerak@stanford.edu

Abstract

Structure from motion is a form of 3D reconstruction. One issue with traditional algorithms for structure from motion is the extended runtime involved with constructing these structures due to certain steps in the process, such as creating a mesh of points. In this project, we recreate a proposed solution for this problem from a study performed by The Society for Modeling and Simulation International that uses planar homography in its structure from motion process. We do so to improve the reconstruction's runtime to real-time levels.

1. Introduction

Structure from motion is a process that is used for taking objects from a 2D image to a 3D structure and has many uses such as obtaining the location of a point or simulating the way an object would look in the real world. However, traditionally, it is a process that takes a long time to perform as the process behind constructing 3D structures is resource-intensive. In particular, the creation of dense point clouds and turning that cloud into a mesh reconstruction are the most time-consuming.

This is fine for small objects since their size makes it so that they take less time, but when covering many objects at once, such as a city block with dozens of buildings on that block, the time and resources needed for a 3D reconstruction become too great. This pushes us to look for another method. Instead of computing the entire 3D reconstruction of the figure in the images provided using traditional structure from motion, we instead combine structure from motion, and planar homography transforms. This results in a much simpler 3D design that is smaller in size and takes much less time to compute. This is done by creating generalized planes through triangulation, on which we can then display a reconstructed image on top of the planes. This makes it so that instead of connecting a cloud of points to turn into a mesh figure, we only need to find the planes that are represented by the images.

The runtime of such an algorithm would be much faster to run in real-time and provide us with a rough estimate of the 3D reconstruction for the figure. With this, we would be able to provide quick 3D reconstruction for large cities and other great masses of land with many figures on it that have simple shapes such as that of cubes or rectangular prisms.

The process behind the proposed algorithm would involve first mapping the location of each feature in one image to the same feature in the other image. We would then want to use an approximation technique such as RANSAC to calculate and find the essential braces. The next step would then be to rectify both images so that we can triangulate the 3D points and create a point cloud. After this, we would begin to stray from the traditional method to our recreation of the proposed algorithm. Due to the use of the RANSAC algorithm, we would need to recover absolute camera position by finding our camera matrix. Finally, we would replace the refinement and texturizing of point cloud meshes with plane detection and homography transformations.

1.1. Background and Related Work

This project is meant to recreate the paper *Structure from motion with planar homography estimation: a real-time low-bandwidth, high-resolution variant for aerial reconnaissance* written by Christian Arnold, Scott Nykl, Scott Graham, and Robert Leishman [2]. In this research paper, a new algorithm variant for Structure from Motion is proposed, whose primary purpose is to enable real-time image processing of scenes of images taken by drones. With this autonomous vision-based-navigation goal, a faster 3D reconstruction using planar homography was implemented, although with less complex reconstructions [2]. Our goal is to replicate the paper and get similar results to create 3D models efficiently.

1.2. Problem Statement

The main problem we are trying to solve is developing a lower runtime than that of traditional structure from motion by producing more simple reconstructions. The proposed algorithm intends to use a combination of traditional structure from motion and planar homography to accomplish those goals. To go a little deeper, we will be doing the first couple of steps of traditional structure from motion. Then, once we create a point cloud, we will be using this point cloud to create generalized planes through triangulation, on which we can then display a reconstructed image on top of the planes. This makes it so that instead of connecting a cloud of points to turn into a mesh figure, we would only need to find the planes that are represented by the images, and this would then reduce the time needed to recreate a 3D model.

2. Technical Approach

Approaching this problem, we first need to run a traditional structure from motion pipeline to generate a sparse point cloud. We would then need to introduce planar detection and planar homography to create an estimated reconstruction of the object in the images.

Given the complexity of a full-scale SFM pipeline, we implemented our structure from motion using OpenCV, an optimized library within Python consisting of computer vision functions. Furthermore, we took inspiration from the open-source GitHub repository "Structure from Motion - Implementation in Python" by Ashok M [3].

2.1. Feature Detection

Although there are several methods for feature extraction, some of them take too much time processing and cannot be used in real-time applications. As a result, we opted to use the corner feature detection method, Feature from Accelerated Segment Test (FAST). FAST is an algorithm initially published and developed by Edward Rosten and Tom Drummond. The algorithm starts by selecting a pixel **P** and determines if the pixel is an interest point (corner). Then, the algorithm evaluates the intensity (brightness) of the pixels in the border of a pixel circle with a radius of 3. In total, there are 16 pixels surrounding **P**. To speed up computations, the algorithm evaluates the pixels at the four cartesian locations in the circle. If three out of the four pixels meet a certain intensity threshold, then **P** is set as a corner point. For a detailed explanation of the algorithm, refer to reference [8]. Figure 2 demonstrates the feature points detected by using FAST.

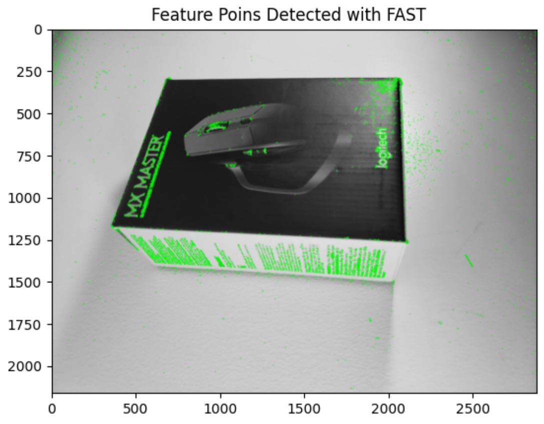


Figure 2: Feature points detected with OpenCV FAST detector

2.2. Feature Matching

For feature matching, we used the Fast Library for Approximate Nearest Neighbors (FLANN), a library that contains optimized algorithms for nearest neighbor search. Reference [7] provides a detailed description of the library. After using FLANN, we produced feature pairs between pairs of images. For our implementation in our code, we used OpenCV FLANN based matcher, which matches keypoints from the first image to the best two keypoints on the second image. Then, using Lowe's ratio test [5], we keep track of all the keypoint matches that have sufficient difference between the best and second-best matches. Figure 3 demonstrates our results.

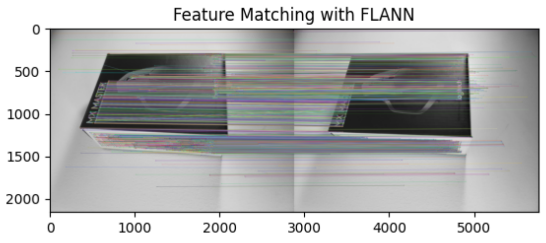


Figure 3: Key point correspondences between two images using OpenCV FLANN based matcher

2.3. Generating Point Cloud

With our key point correspondences between images, we can complete the steps in the structure from motion pipeline to generate a sparse point cloud reconstruction. In order to do this we estimate the essential matrix, move on to calculating the correct projection matrices, and then triangulating between pairs of images.

Using OpenCV and RANSAC, we estimated the essential matrix on the best feature correspondences. To calculate the camera projection matrices,

$$P_i = K[R|T]$$

we recovered the rotation and translation matrices between images using OpenCV's recoverPose() function within Python. With our calibrated camera, we can obtain our intrinsic parameters K. Then, we can triangulate each keypoint using each camera's projection matrix P_i. The following is the resulting point cloud produced from two images.

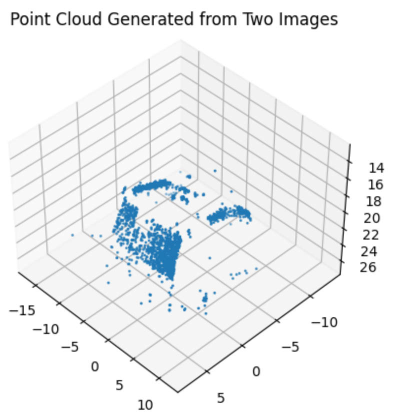


Figure 4: Generated sparse point cloud from two images

2.4. Plane Detection

Our next step was to detect the different planes within the point cloud. To do this would require us to segment the point cloud into the different planes that make it up. Generally, for line fitting or plane-fitting problems like this one, a couple of methods could be used: least-squares and RANSAC (Random Sample Consensus).

Although both methods can be viable in different situations, for our purposes, we opted for the RANSAC method because of its robust approach to outliers and missing data. Furthermore, since structure from motion point clouds may have noisy data and outliers, it is more beneficial to use RANSAC.

RANSAC works by randomly selecting a subset of the data and fitting a model to that subset. Then, using this model, the number of outliers is determined. This process is repeated a set number of times to find the equation that best fits the data.

The RANSAC algorithm we implement iteratively chooses three points and counts the number of points along that plane using some RANSAC threshold. If that plane is not a duplicate and is the "best" plane for those points, it will be added to a list of detected planes. This process is repeated until most points have a respective plane.

Using open-source code from the GitHub repository "Multiple_Planes_Detection" by Yueci Deng [9], we were able to use RANSAC to find each plane in the point cloud. Using these methods, we could access each plane's equation and the points that belong to each plane.

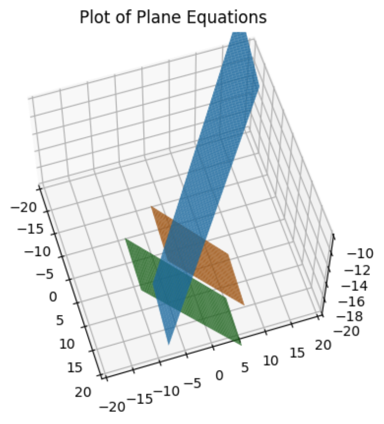


Figure 5: Calculated planes from point cloud

2.5. Planar Homography

The final step in the process is to map the planes from the original images onto their respective planes in the reconstruction. To do this, we will need to find the planar homography between each plane and project the points from the images to the plane using the calculated homography matrix.

To begin with, we need to find the best corresponding image that can be projected that has the closest representation of the plane. We can get a good estimation of the plane's location in each image by triangulating the plane's corner points back to each image. Now that we know where the plane is in each image, we select the best image to project onto our plane. One idea for finding the best image is choosing the image plane with the largest area. A different approach would be to choose the image taken by the camera closest to each plane's norm. Given that we know the equation of each plane

$$ax + by + cz = d$$

We can calculate the norm of each plane as (a, b, c). Then we take images taken by the camera with the smallest

Euclidean distance to the norm. In our case, since we only use a pair of images, we manually choose the images that we would use for the planar homography of each plane.

Once we identify the four corners in the image that represent the plane and four corners of the plane, we can find the homography matrix between these two planes.

The homography matrix relates points from one view to points in another view. It is a full rank three matrix. Since the homography is defined up to a scale, c, we have eight degrees of freedom, meaning there are eight unknowns to solve for. Therefore having the 4 points from each plane will be sufficient to solve for the homography matrix. Let p be the points in the image and p' be the points for the plane reconstruction. We need to solve the equation

$$Hp = cp'$$

where H is the homography matrix and c is the scaling factor. If we eliminate c, we end up with and equation in the form

$$Ah = 0$$

which allows us to use Singular Value Decomposition (SVD) to find a unique solution [1].

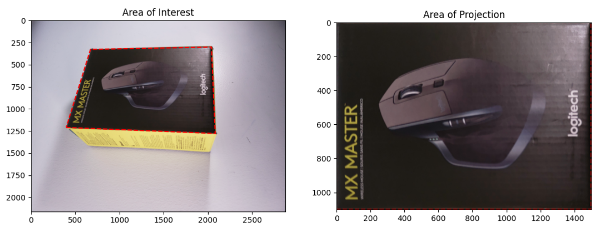


Figure 6: Planar homorgraphy on the top plane box produced using Manansala's article in reference [6]

Now that the planar homography has been completed, we can project those images onto the plane and use those to model the scene.

3. Experiment Results

Although we aimed to implement a full structure from motion pipeline, we could not produce reliable point clouds. In our SFM pipeline, we did not use bundle adjustment as it would increase the effective runtime of the sparse point cloud reconstruction. Furthermore, estimations in our camera parameters, essential matrix, and undetected outliers introduced noise in our triangulation. As a result, the error in our reprojections compounded through each image pair. Therefore, we compromised and decided to use the point cloud reconstruction from the first image pair.

Towards the end of project, we learned about Blender and its robust camera virtual environment. Future work would include transferring our project over to Blender and OpenCV functionality.

3.1. Detecting Key Points

Before settling on FAST, we experimented with different keypoint detectors such as SIFT (Scale Invariant Feature Transform) or ORB (Oriented FAST and Rotated BRIEF). As tested by the Faculty of Engineering and Applied Science [4], SIFT is a robust feature detector that performs the best in different scenarios, while ORB is a faster algorithm. Since we aim for 'real-time' performance, we experimented with the ORB feature detector.

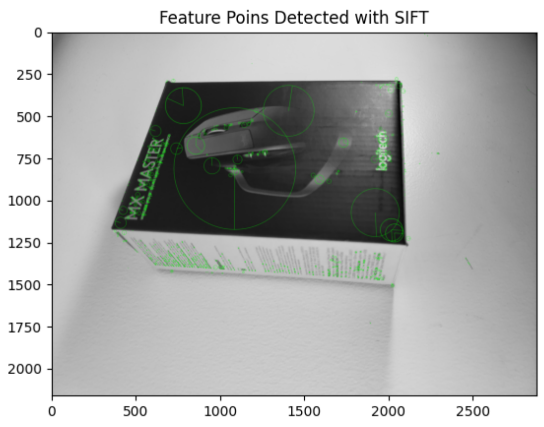


Figure 6: Feature detection with SIFT

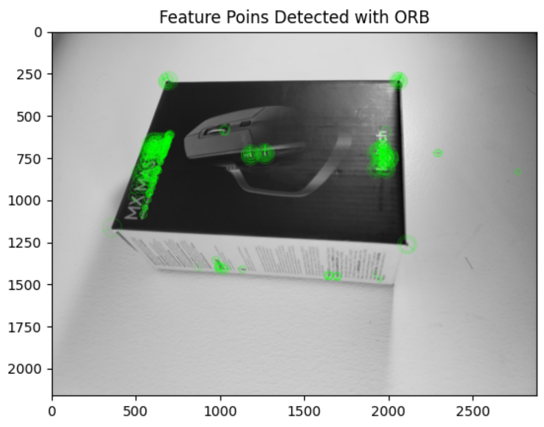


Figure 7: Feature detection with ORB

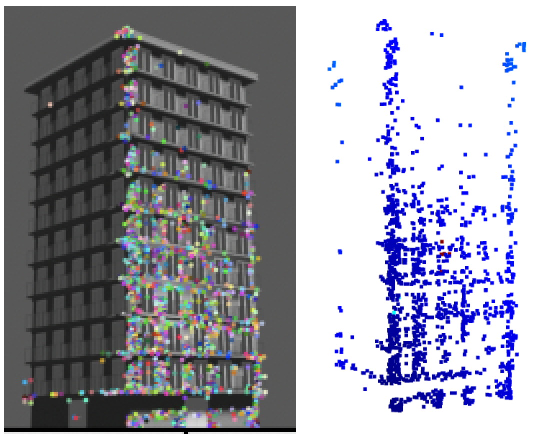
From Figure 7, we can see that the key points are concentrated on specific areas. However, when it comes to plane detection, our feature points should be spread evenly across planes. Therefore, we decided to use FAST as it provides similar speeds to ORB while assigning evenly spread out points.

3.2. Feature Matching

Before using FLANN, we experimented with OpenCV Brute-Force matcher. As opposed to FLANN, Brute-Force matcher compares the descriptors of one keypoint and compares it with every other keypoint. Then, the closest feature points are returned. In ‘real-time’ applications, a Brute-Force matcher will not be a viable solution for feature detection, especially on large scale image sets.

3.3. Point Cloud

Before compromising by using two images, we experimented with various image sets and camera calibration setups. For example, we noticed that the feature points detected were affected by the lighting present where the image was taken. So we experimented with non-Lambertian objects. Similarly, objects with corners performed better than objects with arbitrary textures. So we experimented with paper models of buildings. After failing to produce consistent point clouds, we decided to explore different approaches to obtaining point clouds in a controlled environment. This was when we came across Blender. Initial testing on two images produced promising results. The following is a point cloud generation from triangulation on our code.



Promising point cloud generate using virtual camera in Blender

As we learned about Blender late into the project, we could not transfer our project over to Blender. Given more time, we would attempt to obtain a full point cloud reconstruction on Blender and apply our proposed plane detection method.

3.4. Plane Detection

Since we only used two images to produce our point cloud, we only detected the 3 planes, as seen in Figure 5. This meant that we had to calculate the corner points of each plane manually. Ideally, to calculate our corner points, we need three plane intersections per point. This would require a full sparse point cloud reconstruction.

Furthermore, to obtain the dimensions of each plane (width and height), we need at least four bounding planes. From the cloud point we produced, we could estimate the height of the left side of the box, but we could not estimate the plane's width. Therefore, we manually established the dimensions of each plane.

3.5. Planar Homography and Projection

With the image width and height, we can calculate the aspect ratio of each image plane. As explained previously, we had to manually input the image's aspect ratio produced after the homography transformation.

4. Evaluation

Our algorithm only works on point clouds that contain all six planes of a box. First, we will evaluate our runtime, assuming we have a full point cloud. Then we will compare our estimation to Regard3D, an open-source SFM software, on an image set of 10 images.

Since our implementation is split into two working environments, we estimate the entire runtime of our project. We produced a sparse point cloud and calculated equations for three planes in 0.966 seconds from two images. Assuming we use four more image pairs to detect points on the remaining image planes, we expect a runtime of .966   5 = 4.83 seconds. Since we manually calculated the dimension of each plane instead of reprojecting all eight corners of the box, we will add 5 seconds. Then, since we render our planes and add image textures in Blender, we expect an additional 30-60 seconds, depending on the size of each image. In total, we estimate a total runtime of 29.83-39.83 seconds for a reconstruction of the entire box.

As a comparison, we ran Regard3D on a set of 10 images of our box that captured two planes. In total, Regard3D took a runtime of 1.16 minutes. Then, since we estimated 39.83-69.83 seconds for a reconstruction of 6 planes, we expect a runtime of 19.83-29.83 for a reconstruction of two planes. This is a significant improvement compared to Regard3D reconstruction even when the manual actions are taken into account.

The following images compare Reagard3D's final reconstruction to our estimated reconstruction.

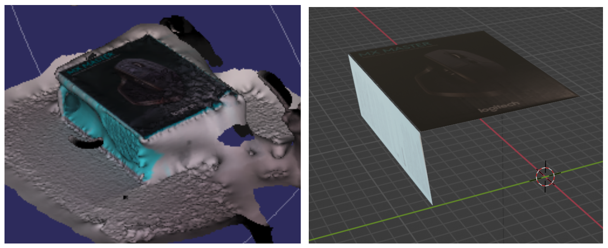


Figure 8: Left image is a 3D reconstruction produced by Regard3D using traditional SFM and MeshLab. The image on the right is our generated reconstruction using Blender

5. Future Work

Although the final reconstruction that we end up with appears to be accurate and the process is efficient, there are lots of avenues to improve and end up with a more complete solution to the problem.

To begin with, a more consistent SFM algorithm that generates more accurate point clouds would be extremely beneficial and make the processes that come after it much easier. For example, plane detection and figuring out the size of planes would be much easier with a complete point cloud.

In addition, various tasks were done manually yet can be automated, allowing the method to be performed in real-time. Instead of choosing which images to use for projection manually, algorithms can be created to find the best image for the projection of each plane. Ideas include selecting the image with the largest area of interest or selecting the image taken from the camera closest to the normal of the plane. Selecting the corners and edges can also become automated instead of finding the pixel location of each corner manually. Perhaps, Hough line detection could be used to find corresponding lines and, subsequently, corner points. It would also be helpful; to have everything implemented in Blender or a similar environment to create and texture planes automatically.

Lastly, to achieve a closer similarity to the original research paper, it could be helpful to implement the algorithm dynamically. This way, planes are detected 'in-real-time' as a drone would capture images as it conducts its aerial reconnaissance.

6. Conclusion

This paper is able to successfully create a 3D reconstruction using structure from motion and planar homography. Although not exactly like the original paper we reference, this algorithm is similar in its approach to the problem of improving the runtime and decreasing the bandwidth of traditional structure from motion.

References

[1] Agarwal, Siddharth. "Homography - And How to Calculate It?" *Medium*, 25 Jan. 2020, https://medium.com/all-things-about-robotics-and-computer-vision/homography-and-how-to-calculate-it-8abf3a13ddc5.

[2] Arnold, Christian, et al. "Structure from Motion with Planar Homography Estimation: A Real-Time Low-Bandwidth, High-Resolution Variant for Aerial Reconnaissance." *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, 2021, p. 154851292110628., https://doi.org/10.1177/15485129211062880.

[3] Ashok93. "Structure from Motion - Implementation in Python." *GitHub*, 2019, https://github.com/Ashok93/Structure-From-Motion-SFM-#readme. Accessed 2022.

[4] Karami, Ebrahim, et al. "Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison for Distorted Images." https://doi.org/https://arxiv.org/pdf/1710.02726.pdf. Accessed 16 Mar. 2022.

[5] Lowe, David G. "Distinctive Image Features from Scale-Invariant Keypoints." International Journal of Computer Vision, vol. 60, no. 2, 2004, pp. 91–110., https://doi.org/10.1023/b:visi.0000029664.99615.94.

[6] Manansala, Jephraim. "Image Processing with Python: Image Warping Using Homography Matrix." *Medium*, 30 Jan. 2021, https://medium.com/swlh/image-processing-with-python-image-warping-using-homography-matrix-22096734f09a. Accessed 14 Mar. 2022.

[7] Muja, Marius, and David Lowe. "FLANN - Fast Library for Approximate Nearest Neighbors User Manual." Cs.ubc.ca, 27 Feb. 2009.

[8] Viswanathan, Deepa. "Features from Accelerated Segment Test (FAST)." 2011.

[9] Yuecideng. "Multiple Planes Detection." *GitHub*, 2022, https://github.com/yuecideng/Multiple_Planes_Detection. Accessed 2022.