3D Reconstruction by Linear Triangulation

Michael Cruz
University of Central Florida
Orlando, USA
mi484725@ucf.edu

1 Introduction

3D reconstruction of images in two views is a sophisticated computer vision problem involving back projecting rays from matching image points. Once classical technique to accomplish this is Linear Triangulation. This experiment implements estimating the fundamental matrix and the linear triangulation algorithms from scratch. Each scene to be reconstructed in an individual folder containing two image views and a calibration script calib.m that uploads the intrinsic camera parameters, both images, and other image related information. To change the image directory being reconstructed, change the image_dir variable. The implementation will be described in the following portions.

2 Implementation

Triangulating image points to positions in 3D space requires multiples steps. First the fundamental matrix between the two views must be calculated from a set of matching image points from each view. Next, camera projection matrices must be computed for each image view using the camera's intrinsic parameters, rotation matrix, and translation vector. For this experiment, the intrinsic camera parameters are provided in the calib.m files in each image directory. Finally, using the set of matching points and camera projection matrices from each view as well as the fundamental matrix between the two views, 3D points for each matching point can be calculated and stored in a point cloud model. I will discuss each of these steps in detail in the following sections.

2.1 Matching Points

Finding pairs of matching points between the two image views is the first step in linear triangulation. SURF features were extracted from each image, and the 1000 strongest points were selected for each image. Subsequently, the function getMatchingPoints() uses feature mapping in order to find matching features between the two images and extracting those points. These will be the set of matching points used through the remained of the experiment.

2.2 Estimating the Fundamental Matrix

The fundamental matrix was estimated using a modified version of RANSAC, where the total set of matching points were iteratively decreased by projectively transforming the current set of matching points and finding inliers in the set of projected points. This process continued until there were only 8 points left or the number of inliers remained the same between two consecutive iterations. The set of inliers from the final iteration are used as the set of matching points to calculate the fundamental matrix. For each set of matching points m = (x, y) and m' = (x', y'), a 1x9 correspondence vector was constructed as follows:

A = [x'x x'y x' y'x y'y y' x y 1]

The correspondence vector for each set of matching points is stacked in a correspondence matrix. The fundamental matrix is estimated by solving the equation Af = 0, where A is the previously constructed Nx9 correspondence matrix in which N is the number of matching points and f is a 9x1 vector of unknowns representing the fundamental matrix F. To solve the systems of linear equations, the singular value decomposition of A is computed, returning matrices U, D, and V. The last column of the V matrix is the estimation of the fundamental matrix. This column is extracted and reshaped into a 3x3 homography F. To enforce the singularity constraint of the fundamental matrix, the singular value decomposition of F is computed. After, the last column of the D matrix is changed to 0 creating matrix D'. This matrix is used to compute the closest approximation to the fundamental matrix.

2.2.1 Camera Projection Matrix. To find the camera projection matrices P1 and P2 from each image view, the focal length, principal point, and image size are extracted from the intrinsic camera parameters given in the calib.m file. Using these metrics, a camera intrinsic object is instantiated. Using the intrinsic parameters K, the fundamental matrix F, and the set of matching points from each view, the orientation and location of camera 2 are calculated in reference to camera 1. These are used to find the rotation matrix R and the translation vector t. Finally, the camera projection matrix for each camera is computed via the following equations:

(1) $P_1 = K[I|0]$
(2) $P_2 = K[R|t]$

2.3 Linear Triangulation

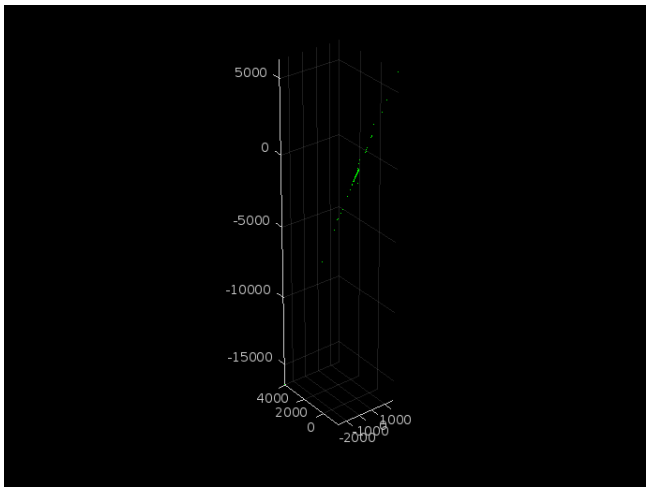

Figure 1: Linear Triangulation result from Newkuba when keeping all six equations in matrix construction

The final step to 3D reconstruction is linear triangulation. In the function get3DPoints(), the 3D points for each pair of matching points are calculated. For each matching point m = (x, y) and m' = (x', y') along with the camera 3x4 camera projection matrix from each view P1 and P2, the 3D coordinates for each point are calculated by first constructing the following matrix:

A = $xp^{3T} - p^{1T}$
$yp^{3T} - p^{2T}$
$x'p'^{3T} - p'^{1T}$
$y'p'^{3T} - p'^{2T}$

In the matrix, p represent a row from the P matrix, so p3 would be the third row from the P matrix. The singular value decomposition of this matrix is computed, resulting in matrices U, D, and V. The last column of V is the 3D coordinate of the 2D point. To put the 3D coordinate in it's homogeneous representation, the coordinate is divided by the last value making the final 3D point in the form M = (X, Y, Z, 1). This process is repeated for each matching point and the 3D coordinate from each point is stacked into a single point cloud matrix. Once finalized, the point cloud model is displayed.

While constructing the matrix for each matched point, I only kept 2 equations from each point x and x' and discarded the last equation. As demonstrated in Figure 1, the reconstructed point cloud model of the Newkuba image directory led to results that were much different than the model produced from the Matlab Triangulation function which are be displayed in the results section.

3 Results

The result from the linear triangulation algorithm implemented yielded results that were nearly identical to the results computed from the MatLab Triangulation function. The SNR values between renderings was consistently above 50dB.

4 Resulting figures

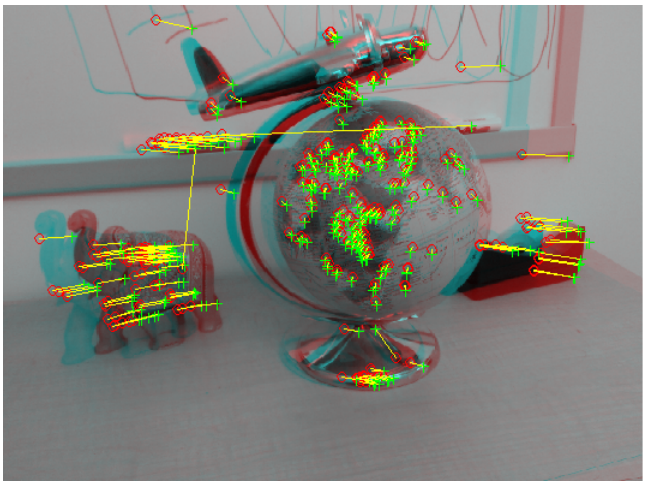

Figure 2: Globe set of matching points

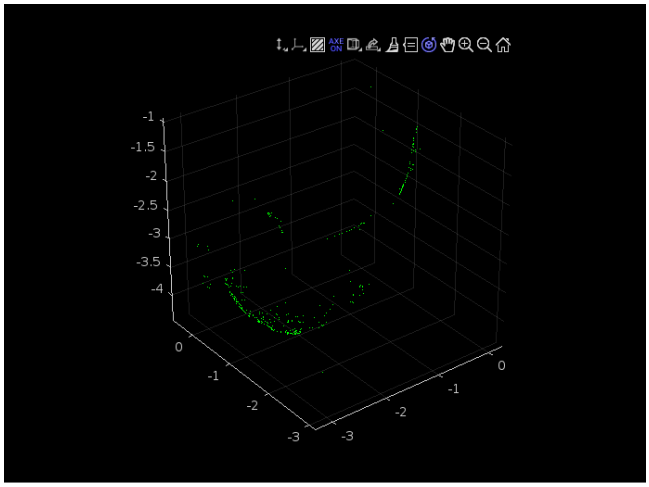

Figure 3: Globe cloud model rendering

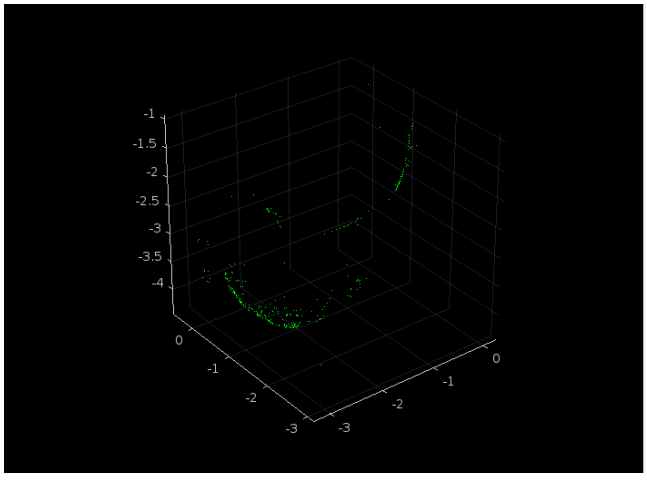

Figure 4: Globe rending from MatLab built-in function

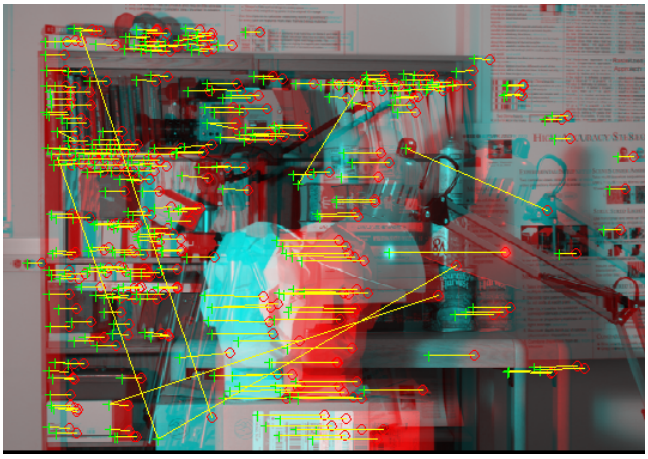

Figure 5: Newkuba set of matching points

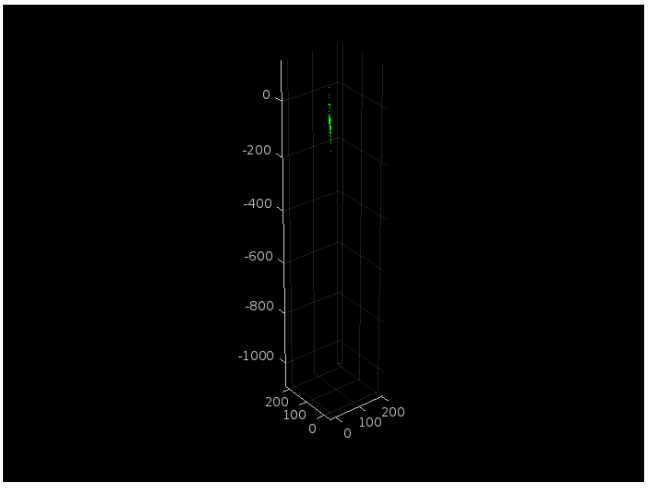

Figure 6: Newkuba cloud model rendering

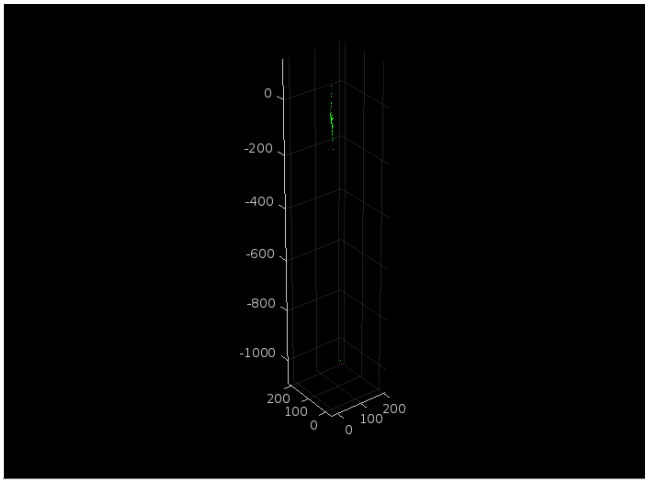

Figure 7: Newkuba rendering from MatLab built-in function

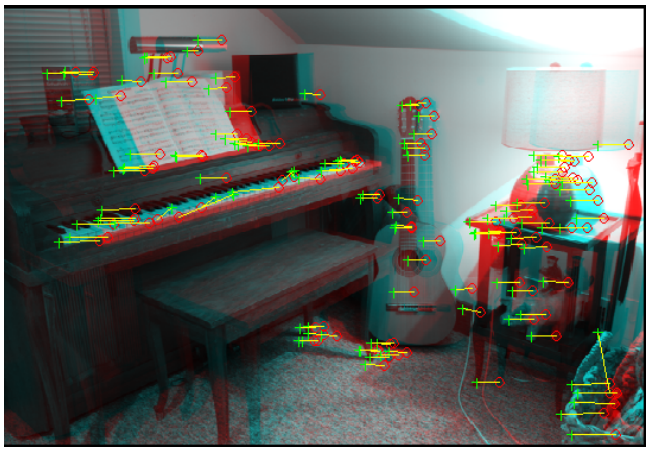

Figure 8: Piano set of matching points

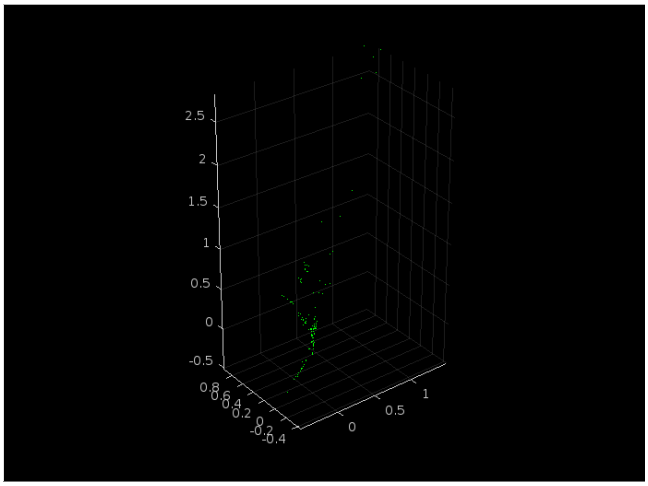

Figure 9: Piano cloud model rendering

Figure 11: Playroom set of matching points

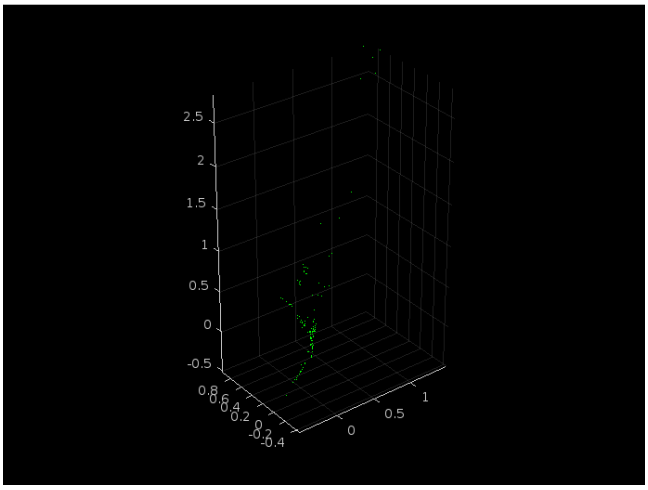

Figure 10: Piano rendering from MatLab built-in function

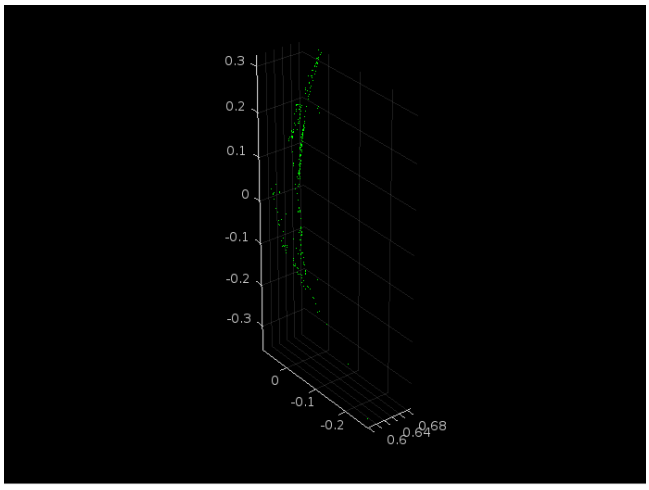

Figure 12: Playroom cloud model rendering

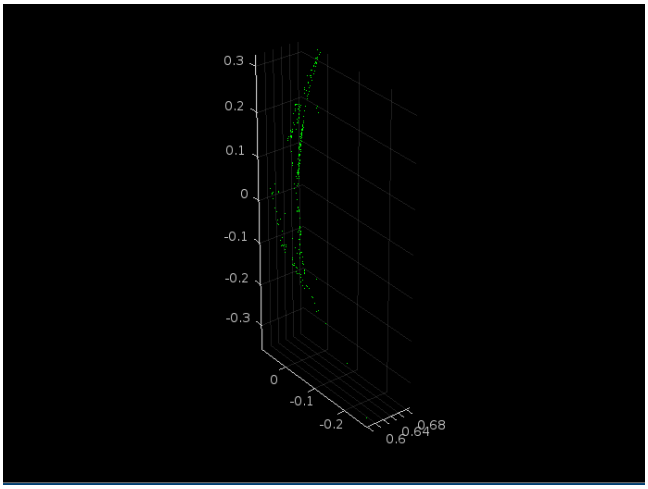

Figure 13: Playroom rendering from MatLab built-in function