Sparse 3D Reconstruction on an iOS device

Aishanou Osha Rait
Robotics Institute, CMU
Pittrbusgh, PA
arait@andrew.cmu.edu

Astha Prasad
Robotics Institute, CMU
Pittsburgh, PA
asthap@andrew.cmu.edu

Abstract

The goal of this project was to implement the entire Bundle Adjustment pipeline for obtaining the N-view 3D reconstruction of an object with sufficient features. The user is asked to take a video of the object of interest using an iOS device, which is then processed to generate the 3D reconstructed points.

1. Introduction

The pipeline for feature based 3D reconstruction involves the following major steps:

- ➢ Detect features in images (SIFT, SURF, BRIEF, ORB etc.)
- ➢ Find feature correspondences in the images
- ➢ Compute Essential matrix using the 5-point algorithm
- ➢ Decompose the Essential matrix into Rotation(R) and Translation(t) (up to scale)
- ➢ Compute the Camera projection matrices given Intrinsics (K)
- ➢ Triangulate points to obtain 3D locations
- ➢ Perform Bundle Adjustment to refine the camera poses and 3D points locations.

Since 3D reconstruction requires computationally heavy non-linear optimization, the feasibility of realistically performing this pipeline on a mobile device is rather untested. Current mobile solutions lean on mobile devices for the sole purpose of capturing object images with ease. The collected images of the object are then uploaded to the cloud where the 3D reconstruction is performed, thus taking the mobile's processing capabilities out of the equation. The following report outlines our attempts to implement the sparse 3D reconstruction pipeline on a mobile device.

2. Background

Some of the popular open source Structure for Motion packages available are:

- Bundler (Windows/Linux)
- VisualSFM (Windows/Linux/Mac)
- SFMedu (MATLAB)

While all the above offer solutions for Bundle Adjustment, none of the above are developed for iOS devices and the speed ups used by them cannot be easily ported onto a mobile device. Thus we decided to implement our own basic pipeline using openCV's functions on an iOS device and try to optimize at an algorithmic level.

3. Approach

3.1. Algorithm

3D reconstruction on the iOS device is performed using images taken from the video of an object captured by the user. Using OpenCV's keypont detection method, SURF keypoints are detected for each camera frame. This is followed by keypoint matching between pairs of images to obtain point correspondences. It is assumed that all points are visible from all camera frames. To satisfy this assumption, we consider the best matches obtained using the first pair of images and use the corresponding keypoints to find matches in the subsequent images.

Using these keypoints and matches, we are able to obtain the Essential matrix 'E' between two images using the 5-point algorithm.

The Essential Matrix E is decomposed to obtain the relative pose between the two images. The Singular Value Decomposition is performed on the essential matrix, as

$$E = U\, diag(1,1,0)V^T \tag{1}$$

For a given first camera matrix $P_1 = [I|0]$, there are four possible choices for the second camera matrix P_2

$$\begin{aligned}\mathbf{P}_2 &= \left[\mathbf{U}\mathbf{W}\mathbf{V}^T \middle| +\mathbf{u}_3\right]\\ \mathbf{P}_2 &= \left[\mathbf{U}\mathbf{W}\mathbf{V}^T \middle| -\mathbf{u}_3\right]\\ \mathbf{P}_2 &= \left[\mathbf{U}\mathbf{W}^T\mathbf{V}^T \middle| +\mathbf{u}_3\right]\\ \mathbf{P}_2 &= \left[\mathbf{U}\mathbf{W}^T\mathbf{V}^T \middle| -\mathbf{u}_3\right]\end{aligned} \qquad \mathbf{W} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (2)$$

These correspond to the following four possible solutions:

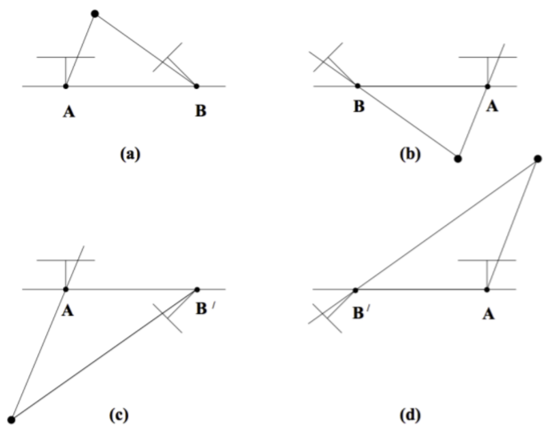

Figure 1: The four possible solutions for calibrated reconstruction from E. Between the left and right sides there is a baseline reversal. Between the top and bottom rows, the camera B rotates 180 degrees about the baseline. Note, only in (a) is the reconstructed point in front of both cameras [1]

The correct solution is selected by determining which solution results in maximum number of 3D points in front of the camera. Given the pose and intrinsics, the keypoints can be triangulated using linear triangulation to obtain the 3D locations as shown by Figure 2, where R_n and t_n are the rotation and translation of each camera pose.

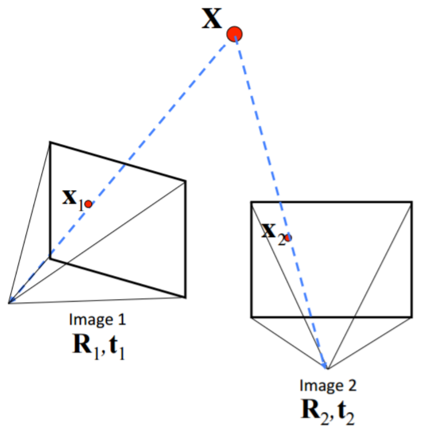

Figure 2 : Triangulation from two views

These locations are considered to be in the frame of the first camera for each pair. For bundle Adjustment, all the poses and 3D points need to transformed into a common reference frame, say the 1st frame of the N views. This was obtained using pose transformations using homogeneous transformation matrices. These poses and the 3D locations obtained from linear triangulation act as the initial estimates for optimization. The optimization works towards minimizing the reprojection error, given by

$$\min \sum_i \sum_j \left(\tilde{\mathbf{x}}_i^j - \mathbf{K}\left[\mathbf{R}_i \middle| \mathbf{t}_i\right]\mathbf{X}^j \right)^2 \qquad (3)$$

For this purpose, we used the Ceres-Solver[2] and OpenCV wrapper for Sparse Bundle Adjustment library[3]. However, neither of them were successfully ported to the iOS device due to limited admin rights to a Mac device.

The same pipeline, when tested on a Linux device with an Intel i5 processor using Ceres-Solver for optimization takes 1 minute to run.

3.2. iOS App Implementation

The above algorithm was encapsulated in an iOS mobile application that allows a user to either capture a video from within the app or choose one from the camera gallery and perform the sparse 3D reconstruction on frames grabbed from the video. The application is implemented using 3 view controllers that are managed by the navigation controller. This is exhibited by the storyboard for the app, shown in Figure 3.

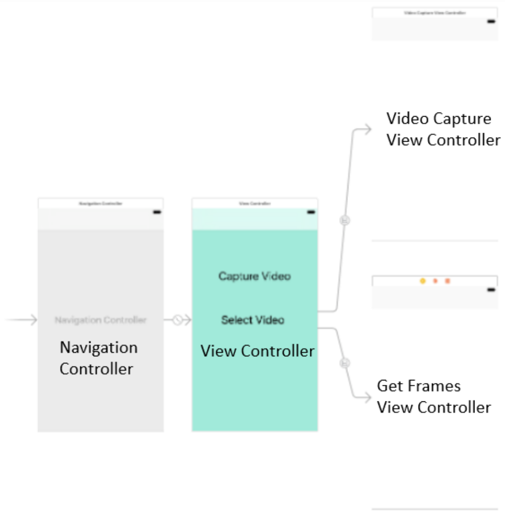

Figure 3 : Storybord of iOS application

The Video Capture View Controller is invoked by pressing the 'Capture Video' button on the main View Controller. It uses the UIImagePickerController to launch the camera from within the app. Once the video has been captured, it employs the ImagePickerController callback function that is programmed to save the video in the gallery.

The 'Select Video' button on the main View Controller opens up the phone gallery and allows the user to choose the video they wish to use for 3D reconstruction. Once the video has been selected, the main View Controller invokes the Get Frames View Controller programmatically and passes the video to it as an AVURLAsset object.

Once in the Get Frames View Controller, the app uses the AVFoundation framework functions to grab 4 frames from the video at equal time steps. These images are displayed on the screen as shown in Figure 4. The 3D reconstruction pipeline is then implemented on the images and the 3D locations obtained using linear triangulation are written into a text file.

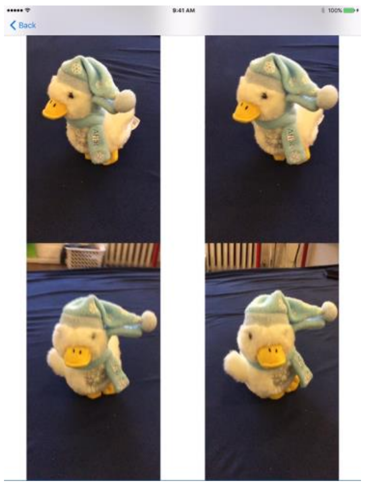

Figure 4 : Get Frames View Controller

The file sharing settings are enabled, thus allowing access to the text file on itunes.

4. Results

The 3D points available as a text file under the app on iTunes are used to visualize the sparse 3D reconstruction. The same images are fed to SfMToy library that performs optimization using the Ceres Solver and the results are shown in Figure 6. The dense 3D reconstruction using SfMedu's MATLAB implementation for the same images is also visualized. The results for the images shown in Figure 4 are presented in Figure 5. Similarly, the comparisons amongst different implementations for the head statue image are shown in Figure 6.

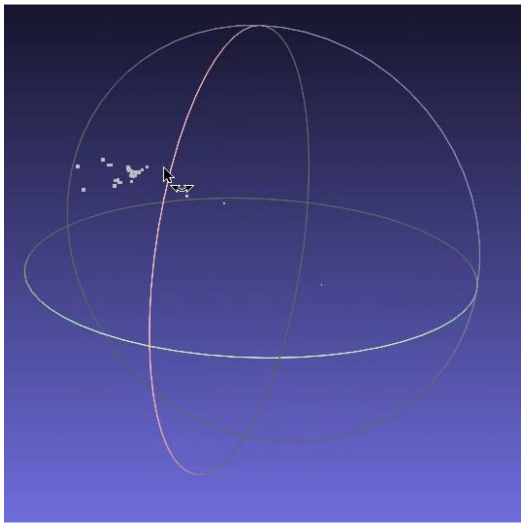

Figure 5a: Sparse Toy reconstruction using iOS app

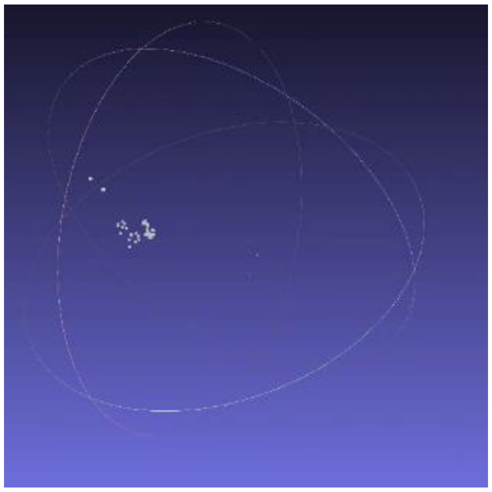

Figure 6b: Sparse Toy reconstruction using MATLAB

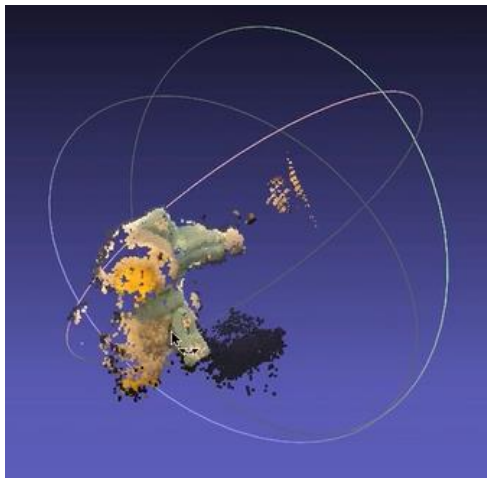

Figure 7c: Dense Toy reconstruction using MATLAB

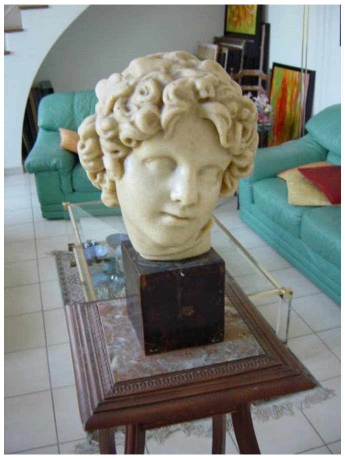
Figure 6a: Sample input image

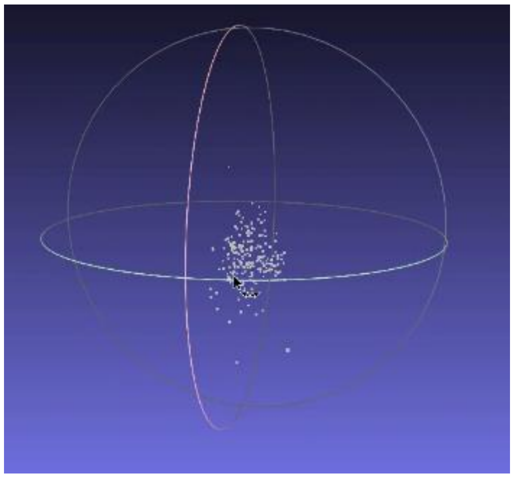
Figure 6b: Sparse Head reconstruction using iOS app

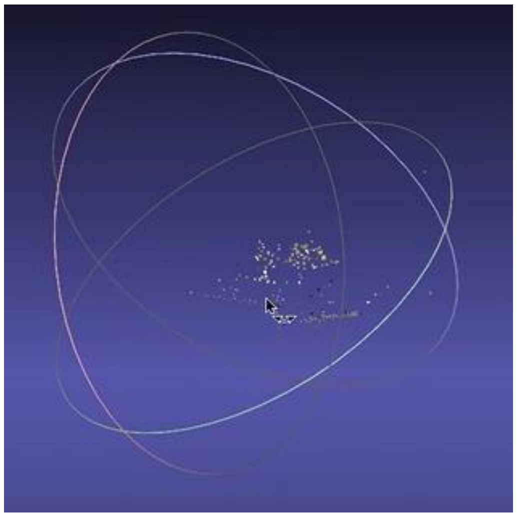
Figure 6c: Sparse Head reconstruction on Linux laptop

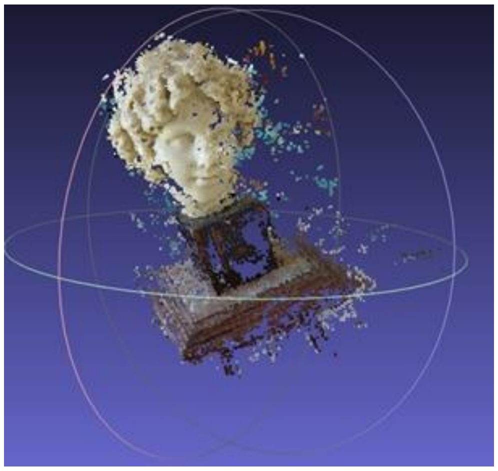
Figure 6d: Dense Head reconstruction using MATLAB

5. Path Forward

On comparing the presented implementation with the other implementations such as the SFMedu MATLAB implementation, it was observed that the following additions to the basic implementation would help in achieving better results:

- Calculating reprojection error after linear triangulation and removing the outliers
- Not using the key points which were outliers during essential matrix calculation
- Instead of selecting the 1st frame as the base frame, selecting the frame with maximum number of matches with other frames as the base frame

Nevertheless, the quality of results for both SFMedu[4] and SfmToy library [5] were not significantly better than what we obtained. The bundle adjustment on the available implementations failed to converge in many cases. Thus, in order to get good results many other points need to be considered, some of them being:

- Robust estimation of camera intrinsics K for an iOS device. It is necessary to maintain a fixed distance between the camera and the object because auto-focus causes the intrinsics to vary between the images. Alternatively, the full intrinsics could also be optimized for i.e. including the principal point.

- Using some other method of triangulation such as minimization of geometric error, Sampson approximation or the “optimal triangulation method” [6]

- Forcing the key points to be selected from different parts of the object instead of concentrating in only one part, using techniques similar to the ones used in ORB SLAM.

Further it is difficult to gauge the quality of results by looking at the sparse point cloud. By creating a semi-dense point cloud the visualization and comprehension could be improved.

6. List of Work

Equal work was performed by both members

7. GitHub Page

The code for this project can be found on:
https://github.com/aorait/mobile_3D_Reconstruction

Since the Checkpoint, we have implemented the remainder of the pipeline and created the xcode project detailed above.

8. References

[1] Page 260, Hartley, R., & Zisserman, A. (2003). Multiple view geometry in computer vision. Cambridge, UK: Cambridge University Press.

[2] Ceres Solver. (n.d.). Retrieved December 09, 2016, from http://ceres-solver.org/

[3] Cvsba: An OpenCV wrapper for sba library. (n.d.). Retrieved December 09, 2016, from https://www.uco.es/investiga/grupos/ava/node/39

[4] Princeton Vision Group - Prof. Jianxiong Xiao. (n.d.). Retrieved December 10, 2016, from http://vision.princeton.edu/courses/SFMedu/

[5] R. (2016). Royshil/SfM-Toy-Library. Retrieved December 09, 2016, from https://github.com/royshil/SfM-Toy-Library

[6] Page 318, Hartley, R., & Zisserman, A. (2003). Multiple view geometry in computer vision. Cambridge, UK: Cambridge University Press.