Plane Extraction on Surfaces

Connie Wu
Stanford University
wuconnie@stanford.edu

Abstract

In this paper we were able to achieve tracking of a plane section on any well-textured surface. Our approach achieves stability in tracking and we have optimized the algorithm's speed enough so that we could run it on an Android device. After tracking a planar section, we were able to place 3D objects on the planar selection. This is demonstrated with a cube projection. By calibrating the camera, we are able to determine the scale of the plane so that we can properly scale the 3D object projections.

1. Introduction

Augmented reality is where a 3D object is overlayed in a captured scene. Oftentimes this is most easily achieved by placing the object on a surface. The most challenging aspect of any augmented reality application is accurate and robust dynamic image registration. Dynamic registration is the accurate alignment of virtual and real images that minimizes jitter and lag [1]. Furthermore, we must also consider the challenge of making the algorithms computationally efficient enough so that they can run on smart phones or other mobile embedded devices.

There are many approaches to this problem both marker-based and marker-less. There are direct methods, which use photometric error to track features as well as indirect methods, which usually comprise of feature detection specifically corners. Furthermore, depending on the number of cameras there are also stereo-based methods that provide more information for model mapping as well as depth information.

For our paper, we wanted to focus on a monocular indirect marker-less tracking system.

2. Related Work

Here we will discuss prior work in marker-less tracking, which is a relatively new and still developing field. One of the first real-time marker-less method was RAPiD (Real-time Attitude and Position Determination) created by Harris [4]. In RAPiD the model must be known ahead of time and the 6-DOF pose is estimated relative to this model by estimating distance from model edges between frames. It requires very little computation because it only tracks 20-30 edge points per frame.This however does not work well with new scenes.

Marchand et al used a 2D affine motion model to estimate motion of the sequences [5]. They optimized the pose by minimizing the new frame's gradient intensities over the model edges. It performs better than RAPid but is computationally expensive and would not work on a mobile device.

Then in the work of Vachetti [6] and Rosten & Drummond [7] they showed that in contrast to edges, points tend to be more accurate because they contain more information and can have descriptors associated with them. This provides the motivation for our method development.

3. Methods

To approach this problem, we tried three different initial approaches

3.1. Overview

3.1.1 ORB Feature Extraction/Descriptor Matching

Initially, we wanted to use Oriented FAST and Rotated BRIEF (ORB) features because they detect corners and also provide a good way to do feature matching via the descriptors[2]. First we computed the keypoints and then matched them using RANSAC.

3.1.2 Optical Flow with Shi-Tomasi Corner Detection

We then wanted to try a method that would allow for tracking stability frame-to-frame and decide to use Lucas-Kanade optical flow in combination with Shi-Tomasi corners, which are an optimized version of Harris Corners as described in their paper[8]. For every five frames we would detect the corners in the image and use optical flow to find the trajectory of those points. This proved to be more stable than the method above because optical flow was less likely to match points incorrectly and as a result allowed for more stability in the homography calculation.

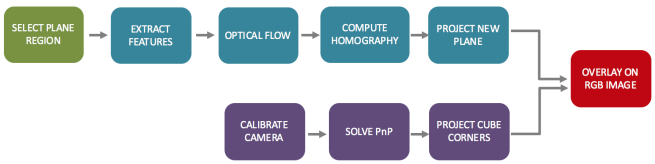

Figure 1: Method Pipeline

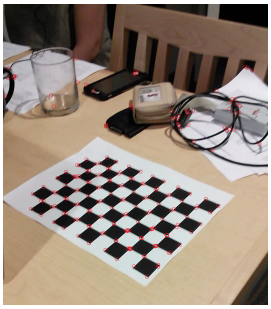

Figure 2: Corner Detection with Shi-Tomasi

3.1.3 Optical Flow with FAST

This method is similar to the one above, but instead of Shi-Tomasi corners, we used FAST corner detection. This was done because FAST gave a slight speed improvement which was crucial in the Android implementation.

Because of experiments we chose the latter two methods which have a similar pipeline represented in Figure 1. The details of which are summarized below:

3.2. Plane Tracking

1. Rectangular Selection: The user selects a rectangular region on top of a surface. From this selection we save the four points that form the corners of the shape and keep track of those corners to reconstruction the quandrangular shape in subsequent frames.

2. Extract Features: We extract the corner features using either FAST or the Shi-Tomasi corner detection in Figure 5. There was the option to only track features inside the quandrangular region, but we found that for scenes with few features in the region it was better to track the surrounding features as well, provided that the majority of the features are in the same plane for the reason described in step 4. This step is performed every 5 frames to allow for additional new features to be added during a recording sequence.

3. Optical Flow: In order to reduce computation, we used the Lucas-Kanade optical flow algorithm to determine the corner point locations between frames. This algorithm estimates the velocity vector of the given points.

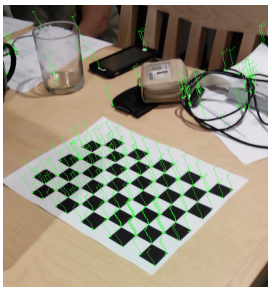

Figure 3: Optical Flow Trajectory

Figure 4: Plane Projection

We used a 15×15 patch which 2 pyramid levels. This means that for motions greater outside the patch, optical flow will fail to find the corresponding point. This will be evident during the mobile implementation.

4. Compute Homography: A homography is computed between each frame to compute the pixel locations of the quandrangle coordinates in the new image so that we can draw the plane.

3.3. 3D Cube Projection

1. Camera Calibration: Using the checkerboard camera calibration method we were able to calculate the camera matrix for 320×240 and 1920×1080 resolution [3]. At the resolution of 320x240 there was a focal length of 334.72 with a camera center at (169.32,136.29) with RMS error of 1.54px. . For a resolution of 1920×1080 the focal length was 1338.88 and the camera center was (717.292,584.38) with RMS error of 0.74px.

2. Solve Perspective-n-Point: Using the coordinates of the first rectangular region selection, we can compute the rotational and translation vectors of the perspective transformation using PnP to get the 2D projection of 3D points. Then given an input of the unit 3D corners of the cubic object, we can use these vectors to com-

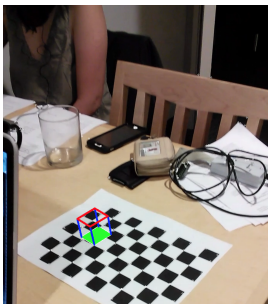

Figure 5: Cube Projection

pute the 2D perspective location of the cube corners above the bottom face.

3. Project Cube Corners: We then projected the cube vertices onto the RGB images to yield the projection of the 3D cube.

3.4. Mobile Implementation

On Android we had to take a different approach regarding the parameters of the method. We had to bound the number of features of the Shi-Tomasi corner detector to 50 features. We also had to decrease the resolution of the frames down to 320x240 at 30 FPS. We found that by doing this accuracy is mostly retained, but there is loss of good corners. Also since the frame rate is lower, optical flow does not perform as well as on PC, and we saw an increase in pixel error.

4. Results

To properly assess the success of the tracking we wanted to quantitatively judge the tracking accuracy. This was determined by a combination of translational pixel error and the retention rate of features from frame-to-frame.

4.1. Evaluation

To calculate the translational pixel error, this will vary depending on the method chosen. In the first method with ORB features and ORB descriptors we are using RANSAC to do the feature matching. From RANSAC we can get the p2 norm of the matched pixels. This will be our pixel error.

For the optical flow methods, we can compute pixel error from the error output of the OpenCV function calcOpticalFlowLK.

To calculate the retention rate, the following formula was used:

$$\frac{\text{Num features matched with less than 1px error}}{\text{num features in previous frame}}$$

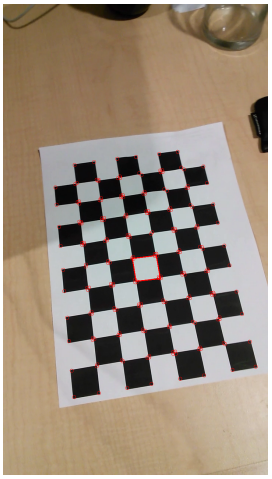

Figure 6: Tracked Tile in the 100th Frame Example (Opt-Flow with Shi-Tomasi)

We evaluated the performance of these methods on a simple checkerboard recording which should have relatively easy tracking. From this experiments we yielded the follow results:

Because of the results in Table 1, it was clear that we should be using the Optical Flow methods over the ORB. There retention rate of approx. 99% is much higher than that of ORB at approx. 27%.

4.2. Tracking Error Validation

Given that the basic sequence has a checkerboard in it, we were able to manually create a ground truth measurement of the coordinates of the center tile of the 6th row of the checkerboard. We measured the actual corners of this tile for the 50th, 100th, 150th, and 200th frames. Then we calculated the RMS p2 distance error of the tracker's coordinates from the ground truth and yielded the following.

This experiment verifies that using optical flow with either FAST or Shi-Tomasi has negligible difference between the two as can be seen in Table 2. However, we can see that ORB loses tracking very quickly and has very large pixel error. In fact on the 200th frame ORB cannot compute a

Metrics			
Metrics	ORB	OF S-T	OF FAST
Pixel Error	27.679	0.071	0.548
Retention Rate	0.275	0.995	0.991
Num Features	40	1462	205

Table 1: Optical Flow Methods Seem Better at Tracking

quandrangle because so many features have been dropped.

5. Conclusion

In well-textured areas, we were able to achieve stable tracking of a plane and as a result we were also able to place 3D object on top of the plane region. Running on PC allowed for optical flow to perform very well in estimating feature locations so there was very little jitter and feature dropping. This was due to a higher frame rate of 90FPS. On Android, however, sometimes frames(as high as 30 frames) would get dropped and as a result there was more pixel error in the feature matching because of higher perceived motion.

For future work, we can possibly overcome this problem by using the on-board GPU to speed up computation. Another way to increase speed is to use direct methods, which look at pixel intensities instead of indirect methods (SIFT, ORB, etc.), which result in faster computation of features. Similarly, since we are using a mobile device we can also utilize the IMU on board to estimate the camera's translation and rotation using the accelerometer and gyroscope, respectively.

With respect to improving pose estimation, adding loop closure allows the algorithm to identify when a familiar image is seen and re-calibrate the camera's pose, reducing drift error over time [9]. In general SLAM algorithms will yield more accurate results because they construct either a dense or sparse model of the environment. Because our approach is frame-to-frame, any error in the sequence will propagate to subsequent frames, which is not desirable. But in SLAM algorithms they will compare keypoints to the model and reconstruct pose based on that information.

Overall, with the computation constraints on Android CPU, this paper's method does well for plane tracking in textured scenes.

References

[1] Klein, Georg. Visual tracking for augmented reality. Diss. University of Cambridge, 2006.

[2] Rublee, Ethan, et al. "ORB: An efficient alternative to SIFT or SURF." 2011 International conference on computer vision. IEEE, 2011.

[3] Harris, Chris, and Carl Stennett. "RAPID-a video rate object tracker." BMVC. 1990. APA

[4] Ren, Shaoqing, et al. "Faster R-CNN: Towards real-time object detection with region proposal networks." Advances in Neural Information Processing Systems. 2015.

[5] Marchand, Eric, et al. "Robust real-time visual tracking using a 2D-3D model-based approach." IEEE Int. Conf. on Computer Vision, ICCV'99. Vol. 1. 1999. APA

[6] Vacchetti, Luca, Vincent Lepetit, and Pascal Fua. "Combining edge and texture information for real-time accurate 3d camera tracking." Mixed and Augmented Reality, 2004. ISMAR 2004. Third IEEE and ACM International Symposium on. IEEE, 2004. APA

[7] Rosten, Edward, and Tom Drummond. "Fusing points and lines for high performance tracking." Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1. Vol. 2. IEEE, 2005. APA

[8] Shi, Jianbo, and Carlo Tomasi. "Good features to track." Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on. IEEE, 1994. APA

[9] Ho, Kin Leong, and Paul Newman. "Detecting loop closure with scene sequences." International Journal of Computer Vision 74.3 (2007): 261-286. APA

Tracking Error			
Frame Index	ORB	OF S-T	OF FAST
50	46.8402	5	5
100	487.4002	4	4.1231
150	209.7284	3.4641	3.4641
200	n/a	5.8310	5.5678
Total	743.9688	18.2951	18.2951

Table 2: RMS Error in Pixels