Model Based Tracking for Augmented Reality on Mobile Devices

Michael Lowney
Department of Electrical Engineering
Stanford University
Stanford, CA 94305, USA
Email: mlowney@stanford.edu

Abhilash Sunder Raj
Department of Electrical Engineering
Stanford University
Stanford, CA 94305, USA
Email: abhisr@stanford.edu

***Abstract*—In this paper we describe our methods for implementing real-time edge based tracking on an Android tablet. We implement a basic tracking algorithm, chosen for both its simplicity and speed. . The performance of our system was evaluated and its advantages and flaws have been pointed out. Suggestions for further improvement are made.**

I. Introduction

In recent years, there has been a lot of interest in the field of Augmented Reality. With companies coming out with wearable AR headsets and AR mobile applications, there is a pressing need for the development of real-time, high quality AR interfaces. Such interfaces would ideally involve achieving a seamless blend between the real world and the 2D or 3D virtual objects projected onto it. This is a very challenging technical problem which requires the rendering of the virtual objects to be both geometrically (correct placement of the objects, accurate scaling, detection of occlusions, etc.) and photometrically (shadowing, mutual reflections, adapting to scene illumination, etc.) consistent.

The issue of correctly positioning virtual objects in the real world scene can be solved by tracking the 3D environment with respect to the camera. Therefore, real-time camera pose estimation and tracking is a key part of augmented reality applications. Broadly, all optical tracking methods can be classified into two categories: marker-based tracking and marker-less tracking. Marker-based tracking relies on artificial patterns placed in the scene to estimate the camera pose. This category of methods has been well researched. While these methods are fast and pretty robust, placing specialized markers in a natural scene is not ideal, especially for creating a seamless blend between the real and virtual worlds. As such, most of the recent research in this area has been focused on developing efficient marker-less tracking methods. This class of methods makes use of natural features in the scene to estimate the camera pose. On the downside, marker-less AR is much more complicated and computationally expensive.

In this paper, we adopt a subset of marker-less tracking, namely a model-based tracking approach for solving the problem. Model-based techniques make use of a known 3D object in the scene for estimating the pose of the camera.

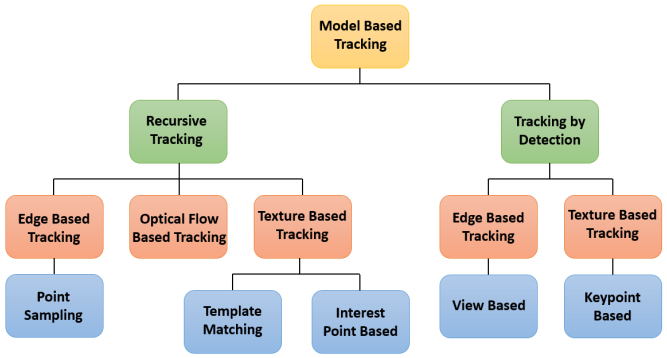

Fig. 1. Taxonomy of model-based tracking methods for Augmented Reality

Unlike marker-based tracking, this class of tracking methods does not rely on artificial patterns but rather on natural objects found in the scene. By tracking a simple cube placed in the scene, we show that model based tracking can be implemented in real-time on an Android based mobile device.

II. Related Work

Model-based tracking can be classified into two categories: recursive tracking and tracking by detection (Fig.1). In recursive tracking, the previous camera pose is used as an estimate to calculate the current camera pose [2][4][5][7]. Due to their recursive nature, they are not very computationally expensive and require relatively low processing power. Tracking by detection, on the other hand, can calculate the pose without any prior state knowledge [3][6], but these methods are usually computationally expensive and require high processing power.

Recursive tracking techniques can be further classified into three categories based on the type of feature used for tracking: edge-based methods, which tries to match a wire-frame 3D model of the object with the edges of the real-world object [2]; optical flow based tracking, which uses the temporal information extracted from the relative movement of the object's projection in the image plane [4]; and texture-based methods, which uses the texture of the object for tracking

[5][7].

Tracking by detection is further classified into two categories: edge-based methods, where the pose is calculated by matching the current frame with 2D views of the object, previously obtained from various positions and orientations [3]; and texture-based methods, which uses keypoint (like SIFT and SURF features) detection to calculate the camera pose in every frame [6].

In [1], the authors implement, evaluate and compare multiple model-based tracking methods. The algorithm testing in all the above papers has been done on computers. Since our goal was to implement a real-time tracking system on an Android mobile device (which has limited computational capability), we decided to use an algorithm requiring relatively low processing power. The algorithm implemented in this paper is the Point Sampling algorithm, a recursive edge-based tracking method.

III. Methods

The full pipeline of the recursive edge-based tracking algorithm we implemented is shown in Fig. 2. Each stage of the algorithm is explained below.

A. Camera Calibration

The first step of the implementation is camera calibration. Accurate camera calibration is essential since the algorithm later uses the camera matrix in order to project the 3D model of the object onto the 2D image plane. We calibrated our camera using the Camera Calibration Toolbox for MATLAB. The toolbox estimates the following camera parameters:

- The focal lengths of the camera in the x and y directions (denoted by f_x and f_y respectively).
- The position of the camera center in the image plane (given by c_x and c_y)
- The lens distortion (characterized by four variables $k_,k_2, k_3$ and k_4.

Using these parameters, the 3×3 camera matrix, K can be constructed as follows:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (1)$$

It is important to note that images taken with the Android tablet's camera app are different from the images that are displayed using the OpenCV Camera View. We believe this is due to the fact that the camera app already knows the camera matrix and applies some pre-processing to the raw image collected. It is for this reason that we collected the calibration images by taking screenshots of the OpenCV Camera View in order to estimate the camera matrix.

B. Initial Pose Estimation

An initial pose of the object is required to start the tracking algorithm. The 3D wire-frame model of the object is rotated and translated into a pose that the user can easily view the object from. The 3D model is then projected onto the image plane using the following equation

$$p_i = K[R|T]P_i \quad (2)$$

Where P_i is a 3D point (represented in homogeneous coordinates), K is the 3×3 camera matrix, R is the 3×3 rotation matrix of the camera, T is the 3×1 translation vector of the camera, and p_i is the projection of the 3D point onto the image plane (in homogeneous coordinates). The outline of the object is then drawn on the screen using the `line()` function in OpenCV. The user can align the outline with the actual object and tap the screen to begin the tracking algorithm.

C. Visible Edge Detection

Once the pose is determined, the system must determine which edges of the object are visible, and which edges are occluded. First, the model is rotated and translated according to the last known pose. Next, the normal vector for each surface is found. A dot product is taken between each normal vector and the vector from the camera to the surface in question. The sign of this dot product will be negative if the face is visible, and positive if the face is occluded. The visible edges are chosen from the known visible faces. This method works well for our simple cube model, but more elaborate methods are needed for complex shapes.

D. Control Points Sampling

With the visible edges of our model known, we must next sample control points along these edges. Control points are points of interest on the model that will be compared to the incoming image from the camera. The sampling rate along each visible edge in the 3D model is proportional to the Euclidean distance of the projection of that edge in the image plane. This ensures that the easily observable edges of our model have more importance in our final pose estimation. The control points are sampled on the 3D model and labeled according to which edge they were sampled from.

E. Edge Map

The gradient information of the image is needed to estimate correspondences. In order to extract an edge map from the latest image frame, we first blur the image with a Gaussian filter to remove noise and to reduce the detection of false edges in the scene. A $7x7$ Gaussian kernel was applied with a standard deviation of 20. Once the image is blurred, Canny edge detection is used to extract the edges from the image. Canny edge detection was chosen for its accuracy for detecting our object's edges and its relatively fast computation time.

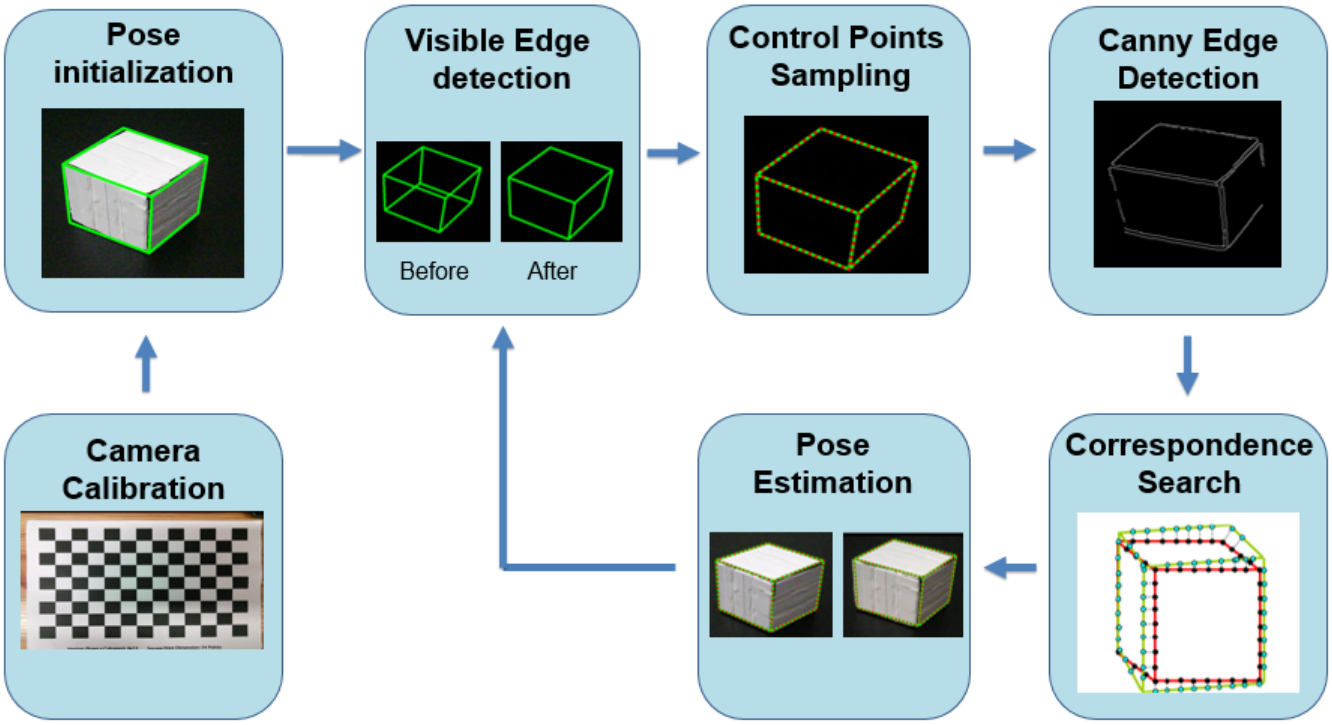

Fig. 2. Algorithm Pipeline

F. Correspondence Matching

In order to estimate the change in pose between successive frames, we must find the correspondences between the 2D points in the most recent frame to the 3D control points. First, the 3D control points are projected onto the image plane using equation 2. Next we will determine what direction we must search in to find a correspondence for each of the projected control points. Corresponding pairs are found by searching in the direction orthogonal to the edge that each control point lies on. The search directions are calculated based on the projection of the model onto the image plane. To determine if a point in the image is a correspondence, we will examine its gradient information in the image extracted from Section III-E. A corresponding point is the closest pixel in the orthogonal direction with a value of 1 in the edge map. If there is no corresponding point within a fixed search distance then the control point is removed and no longer used. In our application we used a search distance of 25 pixels. Increasing the search distance allows for greater movement between frames, but has the potential to find false edges when the true edge is not detected in the edge map.

G. Pose Estimation

Now that we have a set of correspondences between 3D points on the model to 2D points in the current image frame, we can update our estimate of the rotation and translation of the 3D model. This is done by finding the rotation and translation that minimizes distance between the projected 3D

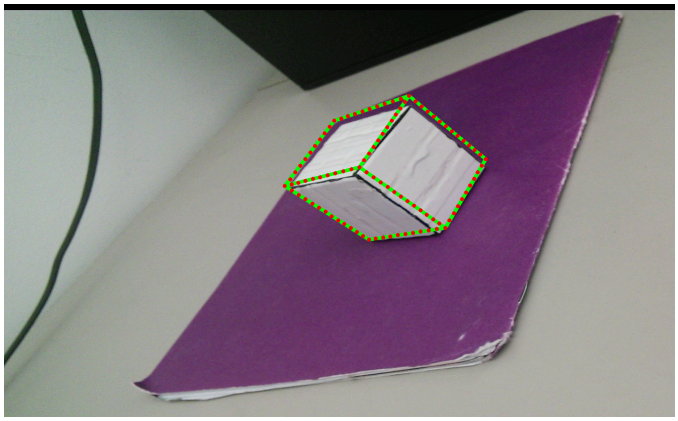

Fig. 3. Accurate tracking of cube

control points and their 2D correspondences as defined by the following equation

$$[R|T] = \arg\min_{[R|T]} \sum_{i=0}^{n} d(p_i, K[R|T]P_i)^2 \tag{3}$$

where P_i is a point on the 3D model, p_i is the estimated 2D correspondence, and d is the Euclidean distance. The objective function is minimized using OpenCV's `solvePnP()` function, which leverages the Levenberg-Marquardt algorithm to optimize this non-linear equation.

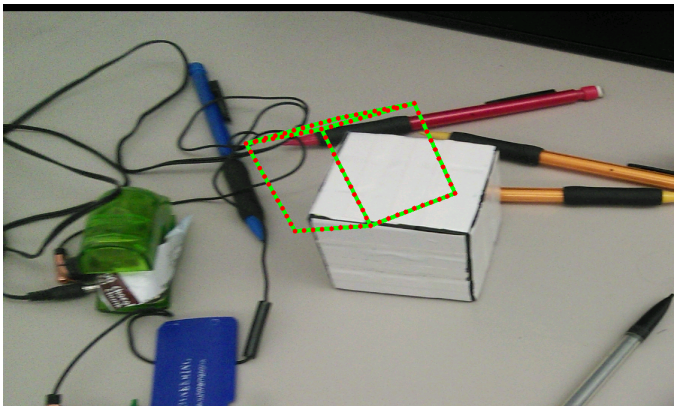

Fig. 4. Poor tracking occurs in a cluttered scene

IV. Results & Discussion

A. Qualitative

We tested our algorithm on a 2012 Nexus 7 Android tablet. An example of good performance is shown in Figure 3. The algorithm is capable of tracking a cube provided that the user moves slowly around the object. We believe that requirement for slow movement is mostly due to hardware limitations. We are confident that the algorithm will run much faster and much smoother on a newer device. This slow frame rate is the cause of most failures because as stated in Section III-F, the maximal search distance is limited to 25 pixels; so if the user moves past this threshold there will be no valid correspondences.

The tracking is very sensitive to the amount of clutter in the scene. If there is a relatively empty scene, with good contrast between the object and the background, then good tracking is observed. However, as more objects are added to the scene the performance starts to degrade as shown in 4. The correspondence search will start to think that the gradients of other objects are good matches for the control points, which causes a very poor pose estimation.

Many of the problems with cluttered scenes can be resolved if we implement a more elegant correspondence search. The correspondence search that we implemented was a single hypothesis model, in which we only have a single estimate of the correspondence. A multiple hypothesis model looks for multiple correspondences and finds the optimal combination of these potential matches in order to be more resilient to noise and clutter. Multiple hypothesis models have been shown to provide more stable tracking at the cost of longer run time [1][2].

The fact that we limit ourselves to orthogonal search directions is the cause of two main sources of error. The first is due to the fact that only certain edges will move in an orthogonal direction. For example, if the camera is translated horizontally, only the vertical edges will be tracked by our single hypothesis model. The second source of error is when points from one edge are matched to a neighboring edge. This can occur if the edge map is missing parts of an edge, and the camera angle is tiled so that there is a small pixel distance between an edge on the front of the cube, and an edge on the rear of the cube. Along with a multiple hypothesis model we can introduce an improved search method to avoid these kind of errors.

TABLE I
Camera Jitter Error

Coordinate	MSE
X	0.6006 (mm)
Y	0.2787 (mm)
Z	10.449 (mm)
ω_1	3.97x10-4 (radians)
ω_2	4.34x10-5 (radians)
ω_3	3.13x10-5 (radians)

B. Quantitative

Our algorithm was able to run at around 3-5 fps on the tablet. While this is not the real-time speed we were hoping to achieve, we know that it is capable of running much faster on a more modern device. For reference, our tablet ran the basic OpenCV camera viewer code at 16 fps.

To determine the accuracy of tracking algorithms a ground truth of the camera movement must be known. In many cases a synthetic scene is generated so that the exact rotation and translation is predefined. Since our algorithm was run on a mobile device we were not able to implement a synthetic ground truth. To get a rough quantitative measure of the error, we measured how much the estimated translation and rotation varied when both the camera and object were stationary. We measured the estimated translation in x, y, and z, as well as the three parameters of the Rodriguez rotation vector ω_1, ω_2, and ω_3, for 100 frames. We then subtracted the average value of each variable and calculated the mean squared error to quantify how much the tracking estimate changes. The results of our camera jitter measurements are shown in Table I.

V. Conclusion

In this paper we present our methods for implementing real-time edge based markerless tracking of a simple cube on an Android device. The algorithm is capable of tracking the object under slow camera movement and with the absence of clutter in the scene. We show that there is reasonable tracking, and list the flaws of our system. We suggest that it can be greatly improved by allowing for multiple hypotheses in the correspondence search.

Acknowledgment

The authors would like to thank Prof. Wetzstein, Jean-Baptiste Boin, and Hershed Tilak for all their help and guidance throughout the quarter.

REFERENCES

[1] Lima, J. Paulo, et al. "Model based markerless 3D tracking applied to augmented reality." Journal on 3D Interactive Systems 1 (2010).

[2] Wuest, Harald, Florent Vial, and D. Strieker. "Adaptive line tracking with multiple hypotheses for augmented reality." Fourth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'05). IEEE, 2005.

[3] Wiedemann, Christian, Markus Ulrich, and Carsten Steger. "Recognition and tracking of 3d objects." Joint Pattern Recognition Symposium. Springer Berlin Heidelberg, 2008.

[4] Basu, Sumit, Irfan Essa, and Alex Pentland. "Motion regularization for model-based head tracking." Pattern Recognition, 1996., Proceedings of the 13th International Conference on. Vol. 3. IEEE, 1996.

[5] Vacchetti, Luca, Vincent Lepetit, and Pascal Fua. "Stable real-time 3d tracking using online and offline information." IEEE transactions on pattern analysis and machine intelligence 26.10 (2004): 1385-1391.

[6] Skrypnyk, Iryna, and David G. Lowe. "Scene modelling, recognition and tracking with invariant image features." Mixed and Augmented Reality, 2004. ISMAR 2004. Third IEEE and ACM International Symposium on. IEEE, 2004.

[7] Jurie, Frdric, and Michel Dhome. "A simple and efficient template matching algorithm." Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on. Vol. 2. IEEE, 2001.

APPENDIX

The work for this project was distributed equally. Both authors were involved in all aspects of developing Android App. A working demo of the Android App can be found at: https://youtu.be/-Ffzi-qL5Co.