Real-Time Camera Pose Estimation for Virtual Reality Navigation

Arthur Louis Alaniz II, Christina Marianne Mantaring
Department of Electrical Engineering
Stanford University
Stanford, California, USA
Email: {aalaniz, cmgmant}@stanford.edu

***Abstract*—In this paper, we present a method for using a mobile device (in our case, the Motorola Droid) as both a navigational interface and a display device for navigating virtual worlds. Using the camera as our input, we extract meaningful corners and calculate corner correspondences between successive frames. From these correspondences, we calculate a homography transform, and then decompose this into rotation and translation vectors. Finally, we use these vectors to properly re-project a virtual world on the screen. Various methods for speeding up the process are investigated in order to make the entire process fast enough for real-time use.**

Keywords - Shi-Tomasi corners; homography; RANSAC;

I. INTRODUCTION

Virtual reality navigation can be done using a wide variety of methods, ranging from joystick controls, to the use of sensors and accelerometers, to more sophisticated and interactive methods. The simpler methods have the advantage of being fast, which makes them popular for gaming and other such applications. On the other hand, the use of sensor suits and other such equipment, while very process-intensive, makes the experience a lot more natural and seamless.

Any sort of virtual reality application on a mobile device is thus more likely to make use of the simpler methods. At most, they might use the compasses or accelerometers that come built-in into the newer smartphones. Still, not all mobile devices have these sensors. What most of them do have, however, is a video camera. Thus, we wanted to use the image capture capabilities of mobile phones to achieve a more natural, but still feasible, experience.

II. PRIOR AND RELATED WORK

Virtual reality is very closely tied to augmented reality (AR), which is an area that has received a lot of focus in the recent years. Augmented reality involves mixing real and virtual objects, and to be able to do this seamlessly, the virtual objects must be transformed according to the point-of-view of the person. Thus, camera pose estimation is a very critical part of augmented reality.

The methods for estimating camera pose fall into two categories. The first category is marker-based estimation, wherein visual markers on the scene are used to localize the camera. These markers are of known dimensions, and contain known keypoints, and by comparing the projected markers with these known dimensions, camera parameters can be extracted.

The second category of methods involve markerless algorithms. Instead of using visual markers, these methods make use of natural features on the scene. These make it easier for the end user, but harder for the application developers. To overcome the fact that the 3D scene is completely unknown, a lot of algorithms employ an initialization step to gain information about the scene.

Real-time tracking has been achieved by quite a number of people, but pretty much all of these methods use a PC. Real-time tracking on a mobile phone usually uses marker-based methods, since these are faster and simpler to implement. However, the need for visual markers makes it less user-friendly. Thus, we wanted to focus on using markerless methods to achieve real-time tracking on a mobile phone. Some of the methods that we studied are presented below.

A. Feature Detection and Matching

To be able to calculate camera pose between successive video frames, one of the crucial steps is to calculate and match keypoints. One of the most popular algorithms for doing this is Lowe's SIFT [1], and the reason why it is so popular is because it is very robust to changes in scale, illumination, and rotation. One of the downsides of SIFT, however, is that it is very computationally expensive, making it impossible to achieve real-time performance. Also, since the movement between frames would be very small, we didn't really need a matching algorithm that was this robust.

The original SIFT algorithm can be heavily modified to achieve real-time performance [2]. One of the modifications is using a FAST corner detector [3] instead of the Difference-of-Gaussians (DoG) method to look for keypoints. The FAST corner detector evaluates the 16-pixel region surrounding a possible keypoint to determine whether or not it is a valid corner. As its name suggests, it is indeed capable of running at very high speeds. However, to be able to robustly evaluate keypoints and discard noise, a training step is needed. We found that using FAST corners with the Motorola Droid without this training step made it very sensitive to noise, and

since we did not have time to train our detector properly, we decided not to use this method.

After surveying through the other work on AR, we found that a number of methods used Harris corners [4] to detect features. Harris corner detection has the advantage of being simple and fast. It is not as robust as scale-invariant features, but given that our keypoints weren't going to be changing much between frames, we found that this was sufficient for our purposes. One disadvantage of Harris corners is that it uses some threshold to calculate cornerness, and we found that this threshold can vary depending on the lighting conditions.

B. Mapping Environments

One problem with pose estimation is that often, no prior knowledge is known about the immediate environment. This can make it hard to properly localize the camera. To deal with this uncertainty, maps are often used to "learn" something about one's surroundings.

Simultaneous Localization and Mapping (SLAM) refers to a family of techniques and algorithms that are often employed by robots to navigate through unknown environments. As they go through the environment, they use the information around them to build a map of where they've gone through. At the same time, this information is also used to let them determine where they are.

While SLAM is a pretty powerful technique, one of its downfalls is that it can be pretty resource-intensive. Maps are needed to properly localize one's position, yet in the very beginning, this map does not exist. Thus, a lot of statistical analysis and machine learning goes into implementing SLAM, especially if there is no prior knowledge of the world being navigated.

An alternative to SLAM is George Klein's Parallel Tracking and Mapping (PTAM) [11], where the two main functions of SLAM are split into parallel threads. PTAM can track the 3D motion of a camera in real time, and it is used primarily in augmented reality applications. An initialization step allows a map of the world to be created, and it is this map that is first used to localize the camera. As the camera moves, the map is updated with new points. Finally, bundle adjustment helps prevent camera drift, which is the accumulation of all the minute errors in the frame-to-frame pose calculations.

PTAM is worth mentioning because it is one of the few camera pose estimation algorithms that has been successfully ported to a mobile device (in this case, an iPhone), although very heavy modifications were made to the original algorithm to allow it to reach real-time performance. While it would have been ideal to implement this on the Motorola Droid, we deemed any form of mapping too complex, given the amount of time that we had to accomplish our project.

C. Multiview Geometry

The matrix transformation relating 2D views of a general 3D scene is known as the fundamental matrix. This matrix can in turn be decomposed into a camera's intrinsic and extrinsic parameters. Intrinsic camera parameters relate the camera's internal frame with the image's frame. They take into account distortion caused by the focal length of the camera, the skew of the camera's frame, and the pixel ratio of the camera. On the other hand, the extrinsic parameters relate the image's frame of reference to the world frame of reference. Extrinsic parameters can be further decomposed into a rotation and translation matrix.

Since our scene would be in a general 3D configuration, estimating the fundamental matrix from matching keypoints in successive frames would have been the most ideal method. A variety of methods use five to eight matching keypoints to produce anywhere from one to ten possible solutions (more keypoints means less possible solutions, but more intensive computations). One drawback of fundamental matrix estimation is that you need your keypoints to be in general configuration (ie, not coplanar), and this is a constraint that is not always easy to achieve, especially if the depth of your scene is small compared to the camera distance.

A special case of the fundamental matrix would be a homography. This kind of transform exactly describes the relationship between keypoints if the scene is planar, or if the motion is purely rotational. Homographies are easier to extract than fundamental matrices, and they also only require a four keypoint matches to produce an exact solution. Camera pose estimation has been done using homographies in both [4] and [10].

III. ALGORITHM DESCRIPTION

Based on the prior work done, the following algorithm was formulated to achieve real-time camera pose estimation.

A. Robust Corner Detection

To detect important image points, we used Shi-Tomasi (ST) corners. The Shi-Tomasi corner metric is very similar to the Harris corner metric. For each point in the image, the Hessian matrix is calculated as follows:

$$M = \begin{bmatrix} \langle F_X^2 \rangle & \langle F_X F_Y \rangle \\ \langle F_X F_Y \rangle & \langle F_Y^2 \rangle \end{bmatrix}$$

where the angular brackes (<>) denote a summation over a window (in our case, this window was 5x5). However, unlike Harris corners, ST corners use the minimum eigenvalue of the Hessian to determine cornerness.

After calculating the cornerness metric for the image, the local maxima are extracted from within circular regions of radius 8. These local maxima now serve as our corners.

B. Feature Matching

To match features with one another, a cost function using both correlation and distance was devised. First, the correlation between the 3x3 regions surrounding each corner was computed. Then, the distance between the corners was computed. The final cost function was

$$C = \frac{\text{correlation}}{\text{distance}}$$

Identify applicable sponsor/s here. *(sponsors)*

and all matches above a certain threshold were passed. This had the effect of sometimes matching a single keypoint to two other keypoints, which happened when a very similar keypoint was far away (high correlation), and not-so-similar keypoint was spatially close. Since both matches appeared equally likely, we decided to pass both matches, and just eliminate the faulty one in the model-fitting step.

C. *Model-Fitting and Outlier Detection*

Given a bunch of keypoint pairs, a robust model-fitting algorithm with the capability of handling outliers was needed to extract the correct model, and reject the outliers. In our case, we used RANSAC to do this.

First, the keypoint pairs generated in the previous step were normalized to have a mean of 0 and a standard deviation of sqrt(2). This normalization is commonly done to minimize the errors produced in model-fitting.

After normalization, four pairs of points are randomly selected, and these points are used to solve for our homography matrix H, such that

$$\begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

As mentioned earlier, a homography will only exactly relate the relationship between the keypoints if our scene is planar, or if the motion is purely rotational. However, since the movement between camera frames is so small, we found that we can almost always approximate some part of the scene as planar. For instance, if the ground is present, then it forms a plane. Also, if the depth of the scene is very shallow compared to the distance from the camera, then our scene appears planar.

To solve for H, we build the following matrix A:

$$A = \begin{bmatrix} 0 & 0 & 0 & -x_1 & -y_1 & -1 & y_1'x_1 & y_1'y_1 & y_1' \\ x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1'x_1 & -x_1'y_1 & -x_1' \\ -y_1'x_1 & -y_1'y_1 & -y_1' & x_1'x_1 & x_1'y_1 & x_1' & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

and then solve for Ax = 0. Once we get H, we de-normalize it using the same matrices that were used to normalize the points.

After obtaining H, the number of inliers is determined. Then, four other random points are chosen, and the process is repeated. We continue iterating until we have extracted the model that gives us the most number of inliers.

D. *Homography Decomposition*

A homography H can be expressed by the following expression

$$H = \lambda\left(R + \frac{1}{d}TNT^T\right) \qquad \in \Re^{3\times 3}$$

where R is the rotation matrix, T is the translation matrix, N is the normal vector of the plane with respect to the camera view, and lambda is the unknown scaling factor. It can be proven that lambda actually corresponds to the second largest singular value of H, and once we have determined lambda, the rest of the parameters can be extracted using linear methods.

Using the constraint that the normal vector of the plane must be in front of the camera (positive depth), we end up with two possible solutions. From these, the correct solution is extracted based on the small motion assumption; that is, we assume that the frame-to-frame movement is very small, and this should be reflected in our transformation matrices.

E. *Speed-up Techniques*

To make it more feasible for the entire algorithm to be implemented in real-time, a variety of speed-up techniques was used.

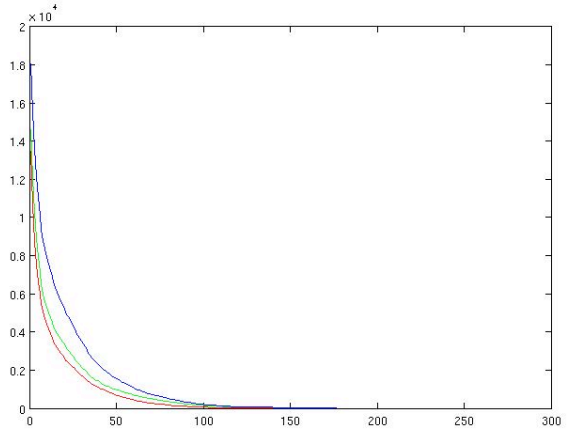
First, to lessen the number of calculations needed to perform corner detection, thresholds were used to determine whether or not to proceed with certain steps. For instance, if the individual differences in the x- and y- directions were too small, then the Hessian matrix would not be computed. Next, if the sum of the squared differences was less than a threshold, then the minimum Eigenvalue of the Hessian would no longer be calculated. Finally, if the cornerness value was too small, then it would not be included in the local maxima calculations.

Another speed-up technique was the early exiting of RANSAC. If a certain model had more than 80% of the points as inliers, then we stopped the iterations and passed the model as the best model.

Finally, a third speed-up technique involved rendering. If the calculated transformation values were very small, then we did not re-render the 3D world. This eliminated some of the camera shake produced by holding the camera with an unsteady hand, but had the disadvantage of making the motion more choppy.

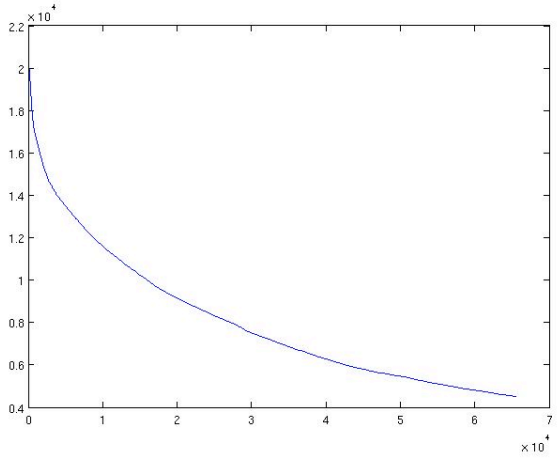
IV. EXPERIMENTAL RESULTS

No. of Calculations vs. First Threshold



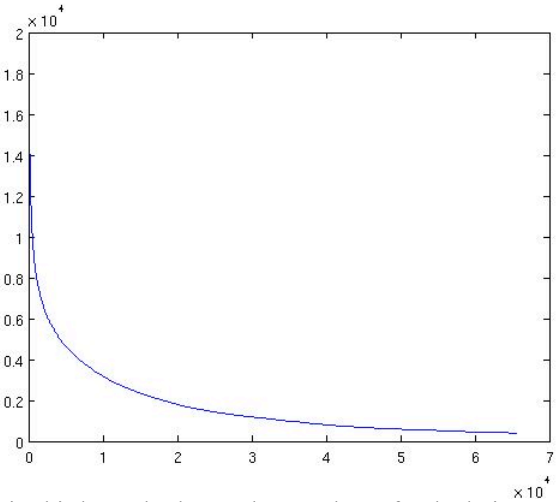
The above graph shows the decrease in calculations given the first speed-up threshold. The threshold that we used in our algorithm was 24, which used about a fourth of the total calculations

No. of Calculations vs. Second Threshold



The above graph shows the number of calculations given the second speed-up threshold. The final threshold used in the algorithm was 8192.

No. of Calculations vs. Third Threshold



This third graph shows the number of calculations versus the third speed-up threshold. The final threshold used was 2048.

The next two images show the comparison between the cornerness generated by the original Shi-Tomasi algorithm, and the speeded-up algorithm. We found that there was almost no change in the relative cornerness metrics.

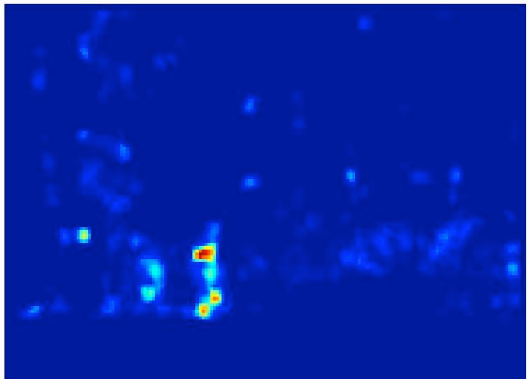


Figure 1. Cornerness metric produced by original Shi-Tomasi algorithm

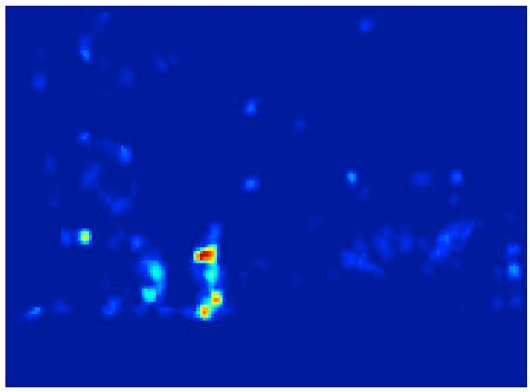


Figure 2. Cornerness metric produced by speeded-up algorithm.

Finally, the image below shows the regions that were included in the final corner detection step, where the local maxima are extracted as corners. The ratio of the included regions as compared to the whole image is very small, showing that we were greatly able to decrease the number of calculations.

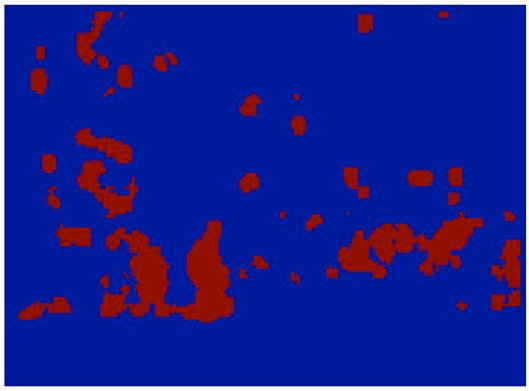
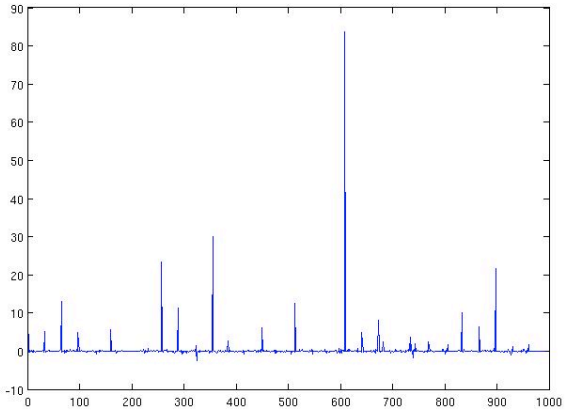


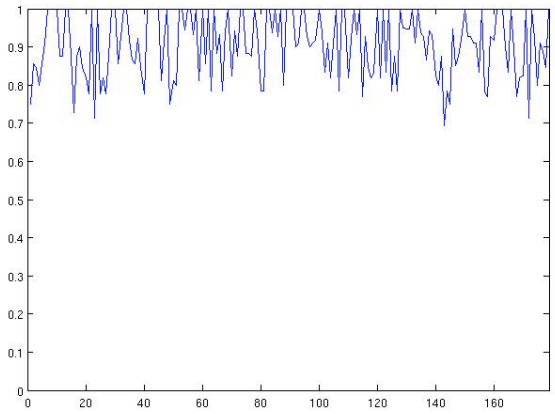
Figure 3. Regions included in local maxima detection

Next, we look at the results produced by our cost function in matching keypoints.



The above graph shows the cost values of each possible keypoint pair for one of our test images. The peaks correspond to values which are likely to be matches. After looking at this graph over a succession of frames, we determined our match threshold to be 5; that is, any pair whose cost was greater than or equal to 5 would be counted as a matched pair.

Finally, we examined the results produced by RANSAC for determining inliers.



The previous graph shows the percentage of inliers produced by RANSAC for a succession of frames. We see that most frames have an inlier percentage of over 80%. Thus, we set the early-exit score of RANSAC to be at 0.80. If any model fits more than 80% of the keypoints, then we can select it without going through the rest of RANSAC.

Finally, after extracting the homography model and decomposing them into rotation and translation, we were successfully able to render a 3D scene based on these parameters. Unfortunately, our Android program kept running into what we think are memory leaks, and would exit before any meaningful data could be extracted. Also, despite the speed-ups that we implemented, the performance of the algorithm was very slow.

We infer that several reasons may be behind the poor speed that we got. First, for each preview frame, Android has to allocate a new memory buffer. When old memory is freed up by the Garbage Collector, the system freezes momentarily. Aside from the preview frame buffers, we also allocate a number of buffers for our different algorithms, and these are allocated for every frame.

Another problem we had with the Droid phone is that, after the application hangs and is forcibly terminated by the Android OS, some of the camera parameters get permanently set to certain values. As a result, our camera image becomes overexposed or underexposed, leading to difficulty in detecting features. To undo this, the phone has to be restarted.

V. CONCLUSION

Real-time camera pose estimation is a difficult problem to solve. Although the methods for pose estimation exist, it is not easy to get them working in real time, especially on a mobile device. Using a variety of smart speed-ups greatly decreased the number of computations done, but in the end, the underlying limitations of using a certain platform must also be considered.

In the future, using a feature matching algorithm that was designed for real-time use may be helpful. Also, employing techniques like mapping and training for feature detection will help make the algorithm more robust.

REFERENCES

[1] David G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision,* 60, 2 (2004), pp. 91-110.

[2] Daniel Wagner , Gerhard Reitmayr , Alessandro Mulloni , Tom Drummond , Dieter Schmalstieg, "Pose tracking from natural features on mobile phones," Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality, p.125-134, September 15-18, 2008

[3] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *Proc. 9th European Conference on Computer Vision (ECCV'06)*, Graz, May 2006.

[4] Simon J. D. Prince , Ke Xu , Adrian David Cheok, "Augm ented Reality Camera Tracking with Homographies," IEEE Co mputer Graphics and Applications, v.22 n.6, p.39-- - 45, November 2002

[5] J. Shi and C. Tomasi. Good features to track. In *Proc. IEEE Intl. Conference on Computer Vision and Pattern Recognition (CVPR '94)*, pages 593-600. IEEE Computer Society, 1994.

[6] Martin A. Fischler and Robert C. Bolles (June 1981). "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". *Comm. Of the ACM* **24**: 381–395

[7] Richard Hartley and Andrew Zisserman (2003). *Multiple View Geometry in Computer Vision* (2nd ed.). Cambridge University Press

[8] Yi Ma, S. Soatto, J. Kosecka, and S. Sastry. *An Invitation to 3-D Vision*: From Images to Geometric Models. Springer-Verlag, November 2003

[9] C. Harris and M. Stephens (1988). "A combined corner and edge detector" (PDF). *Proceedings of the 4th Alvey Vision Conference*. pp. 147--151.

[10] Gilles Simon, Andrew W. Fitzgibbon and Andrew Zisserman. "Markerless Tracking Using Planar Structures in Scene". Robotics Research Group, Department of Engineering Science, University of Oxford.

[11] Klein, G., Murray, D.: Parallel tracking and mapping for small AR workspaces. In: Proc Intl. Symposium on Mixed and Augmented Reality (ISMAR 2007), Nara (November 2007)

APPENDIX

Divison of Labor:

- Arthur Alaniz
    - feature detection
    - feature matching
    - speed-ups
- Tina Mantaring
    - homography computation
    - homography decomposition
    - rendering