

Performance Analysis of Visual Tracking Algorithms For Motion-based User Interfaces on Mobile Devices

Stefan Winkler, Karthik Rangaswamy, Jefry Tedjokusumo, ZhiYing Zhou

Interactive Multimedia Lab, Department of Electrical and Computer Engineering
National University of Singapore
4 Engineering Drive 3, Singapore 117576

ABSTRACT

Determining the self-motion of a camera is useful for many applications. A number of visual motion-tracking algorithms have been developed till date, each with their own advantages and restrictions. Some of them have also made their foray into the mobile world, powering augmented reality-based applications on phones with in-built cameras. In this paper, we compare the performances of three feature or landmark-guided motion tracking algorithms, namely marker-based tracking with MXRToolkit, face tracking based on CamShift, and MonoSLAM. We analyze and compare the complexity, accuracy, sensitivity, robustness and restrictions of each of the above methods. Our performance tests are conducted over two stages: The first stage of testing uses video sequences created with simulated camera movements along the six degrees of freedom in order to compare accuracy in tracking, while the second stage analyzes the robustness of the algorithms by testing for manipulative factors like image scaling and frame-skipping.

1. INTRODUCTION

Tracking the camera's self-motion forms a major area of research in the field of computer vision. The majority of algorithms developed toward this goal employ visual cues in the form of features (shape, edges, color, corners etc.) to estimate the instantaneous pose of the camera in 3D space. Some of the algorithms require the object of reference to remain in the camera's field of view throughout run-time, while some others employ real-time active detection and tracking of features to compensate for lost landmarks in the scene. In either case, factors such as variation in lighting, shadows, occlusions, dynamic objects in the scene etc. introduce an element of uncertainty, which may affect the robustness of an algorithm in real-time scenarios.

In this work, we compare marker-based tracking^{1,2} with face tracking³ and feature-based MonoSLAM.⁴ The primary reason for choosing these three is that current-day algorithms not only perform well on the PC platform, but some have entered mobile platforms as well. There are quite a few augmented reality-based applications on the mobile phone that utilize marker and face tracking to ascertain the phone's pose in 3D space, for example. We have previously developed applications like games⁵ and map-navigation⁶ for mobile phones with built-in cameras that we propose to control with intuitive phone movements rather than buttons, for which efficient motion-tracking is required. Hence, our experiments are designed to evaluate the performance of the above-mentioned algorithms to the mobile platform with tests like reduction in resolution and frame skipping. Unfortunately, the processing power of current day mobile phones is insufficient to run all the above algorithms at equal frame-rates. Hence we had to test them on the PC platform. We hope that in the future, the gap between the processing power of PCs and the mobile devices narrows to allow more computationally complex algorithms like MonoSLAM to deliver better performance on mobile phones in terms of operational speed and accuracy.

The main difference between marker and face tracking when compared to MonoSLAM is the need for the reference object to remain in the scene for the former two, while the reference object is only required during initialization for the latter. Yet, for comparison purposes, the base-line motion was common for all the three algorithms with the reference object always in the field-of-view, and hence MonoSLAM's advantage is not fully exercised in this respect.

Corresponding author: Z.Y. Zhou (elezzy@nus.edu.sg).
S. Winkler is now with Symmetricom, San Jose, CA 95131.

We focus on simulated motion rather than real-world captures in the tests described here for easy access to precise ground truth. In a real-world scenario, a highly accurate tracking device would be required to record the actual trajectory of the camera, which was not available to us.

The paper is organized as follows: Section 2 explains the basic methodology behind each algorithm, Section 3 discusses the experimental setup, and Section 4 explains the tests conducted and the results obtained. The paper concludes with Section 5.

2. ALGORITHMS

The objects of reference for marker, face and MonoSLAM-based motion tracking are a square marker, a face and random salient image patches respectively. The following sub-sections explain each algorithm briefly.

2.1 Marker Tracking

A fiducial marker of known size is printed on a sheet of paper which is used to help track its 3D pose with a camera. The image of a marker observed by the camera is a projective transformation of the original physical marker, which includes rotation and translation. After thresholding the image observed by the camera, lines and corners are extracted using contour detection. The distorted sub-image within the square region can be matched with the original marker sub-image through normalization after it has been warped using the homography estimated between the camera and the marker. The pose of the marker with respect to the camera can be recovered using standard pose estimation techniques.²

2.2 Face Tracking

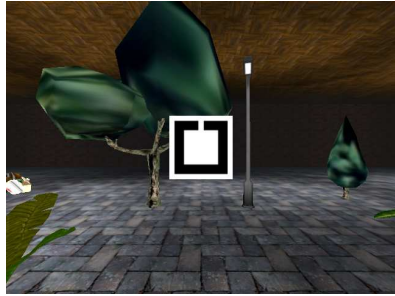
Human faces are a common feature in our daily life, and at least the face of the device user will always be available for tracking. We use CamShift,³ an algorithm that needs an initial face position before it can continually do the tracking. To automate this process, we use the OpenCV implementation of a face detection algorithm based on Haar object detection.^{7,8} CamShift uses probabilistic color histogram to do the tracking, hence it tracks any distinguished uniformly colored object, but it can be easily distracted by other objects with similar color (e.g., if there is more than one face in the image). Details on how the algorithm reacts to distraction, occlusion, lighting condition, etc. have been discussed by Bradski.³

2.3 MonoSLAM

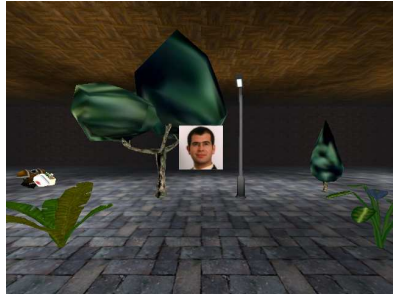
MonoSLAM developed by Davison⁴ addresses ego-motion estimation of a single active camera with respect to its surroundings in real-time. The process is initialized with a known target with which the camera is pre-calibrated. The states of detected features and the camera position are modeled as a multi-variate Gaussian distribution. A Kalman filter is used for predicting the instantaneous location of the camera, assuming a Gaussian distributed linear and angular acceleration. At every step, the positions of existing features in view as well as the camera's pose are measured and their uncertainties are updated. The features used are salient image patches⁹ found in the environment of the camera, which are stored indefinitely to serve as long term landmarks. These features are tracked by exhaustive correlation search. A new feature is added if the number of features in view drops below a pre-defined threshold. Figure 1(f) shows a screenshot of MonoSLAM tracking salient image patches in the scene.

3. EXPERIMENT SETUP

We generate a virtual world using OpenGL with several virtual objects (tree, plant, street lamp, etc.) inside it. We also embed the following reference objects: a marker for marker tracking, a photo for face tracking, and a black rectangular image for MonoSLAM initialization in the center of the camera view (see Figure 1). We generate 360 frames of size 640×480 for 6 different basic types of virtual camera movement with pre-defined limits: translation along x , y , and z axes and yaw, pitch and roll (see Table 1). We record the output of the algorithms for these images and compare them with the actual movement of the virtual camera.



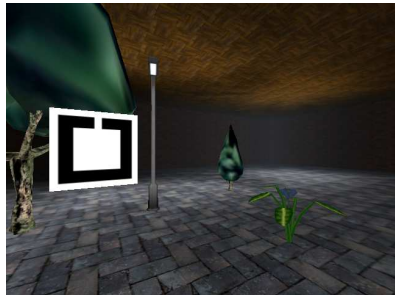
(a) Marker seen from the camera's initial position



(b) Face seen from the camera's maximum z translation



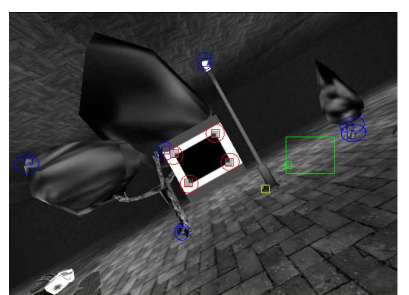
(c) MonoSLAM patch seen from camera's maximum x translation



(d) Marker seen from camera's maximum yaw



(e) Face seen from camera's maximum pitch



(f) MonoSLAM patch seen from camera's maximum roll with markings on the tracked features

Figure 1. Screenshots of the virtual world environment

| Frame Number | Camera Movement | Maximum value |
|--------------|-----------------|---------------|
| 1-60 | z translation | 3 units |
| 61-120 | x translation | 3 units |
| 121-180 | y translation | 3 units |
| 181-240 | Yaw rotation | 30 degrees |
| 241-300 | Pitch rotation | 30 degrees |
| 301-360 | Roll rotation | 30 degrees |

Table 1. Movement of the virtual camera in the test sequence

4. RESULTS AND DISCUSSION

We now compare the performance of the three algorithms. The error is defined as the difference between the algorithm's output and the actual virtual camera movement.

4.1 Accuracy

This test is conducted with a video resolution of 640×480 pixels to compare the accuracy in tracking for the three algorithms. Figure 2 shows the error graph of position and rotation along each axis for 360 frames of computer generated images.

Figure 2(a) shows the x translation error. Marker tracking and face tracking work very well, except when they incorrectly detect the camera's yaw as x translation movement. They also have the same problem with y translation error, see Figure 2(b). MonoSLAM does not have this problem, but for most cases its error is larger than that incurred by face and marker tracking, especially during z translation and roll rotation.

Figure 2(c) shows the z translation error. Marker and face tracking have significant errors with yaw and pitch movement. Figures 1(d) and 1(e) show the images where marker and face tracking have the largest error. The algorithms misinterpret the virtual camera's motion as moving closer to the reference object. MonoSLAM does not exhibit this kind of problem, because it also tracks other features in the view aside from the reference object.

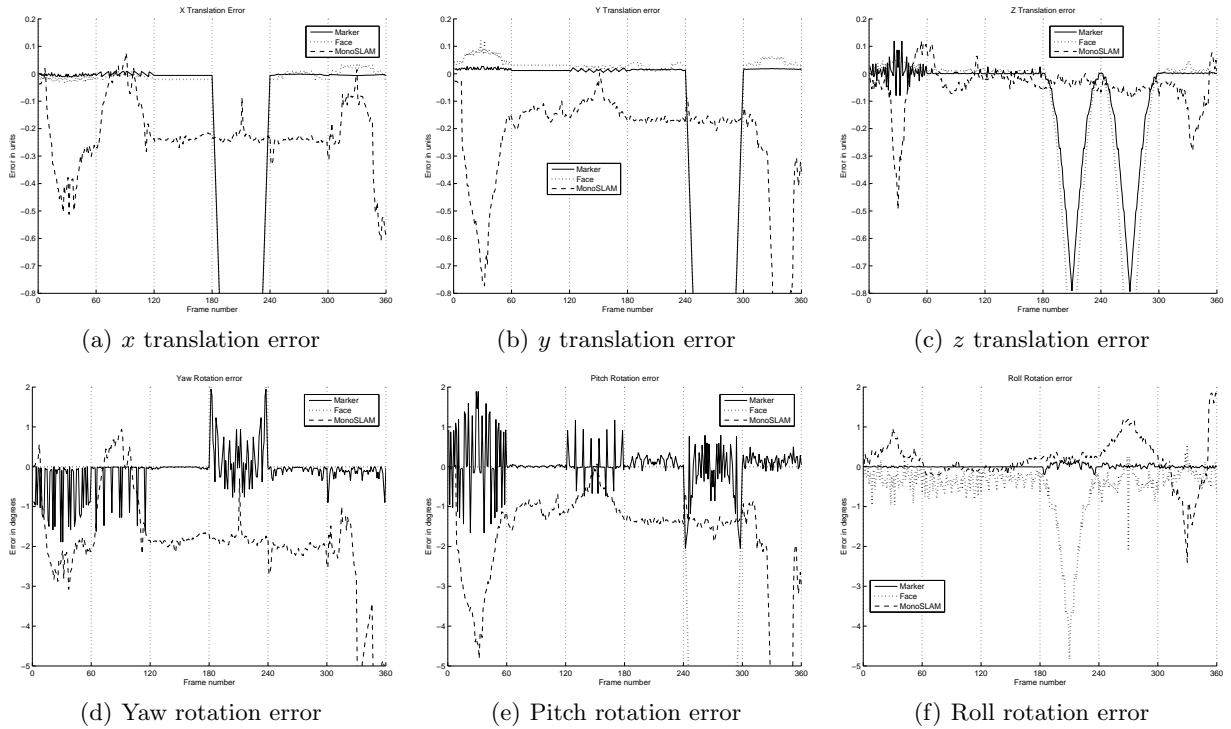


Figure 2. Comparison of errors in accuracy for marker tracking, face tracking and MonoSLAM

Figure 2(d) shows the yaw error. The absolute mean error for marker tracking is 0.44° , while for MonoSLAM it is 1.09° . Face tracking has large errors because the algorithm could not detect yaw rotation.

Figure 2(e) shows the pitch error. Again the marker has the smallest absolute mean error of 0.45° , while it is 1.98° for the MonoSLAM. Once more, face tracking cannot detect pitch rotation.

Figure 2(f) shows the roll error, marker tracking has the smallest error, followed by MonoSLAM, and face tracking has the largest error. Yet, their overall performances are within acceptable limits here.

To conclude, marker tracking has the highest accuracy for rotation. It also has good accuracy for translation except that it misinterprets yaw and pitch rotation as translation. Face tracking is as precise as marker tracking, except it is not capable of detecting yaw and pitch movements. MonoSLAM tracking is not as smooth and accurate as marker tracking, although the plus point for MonoSLAM is that it does not misinterpret yaw and pitch rotation as translation movements. Of course, MonoSLAM does not depend on a specific object of reference, as it can pick any natural features as reference once initialized.

4.2 Resolution Scaling

We choose three different resolutions to compare the performances of the algorithms for resolution scaling. Figures 3, 5, and 6 show the plots of absolute mean errors calculated at intervals of 60 frames, to highlight the differences in accuracy due to resolution scaling.

For marker tracking, we use the following resolutions: 640×480 , 480×360 , and 320×240 . The graphs in Figure 3 shows no significant difference in resolution scaling. The 640×480 gives the lowest absolute mean error although the error differences between the resolutions are not significant. Scaling down the resolution to 320×240 causes loss of marker tracking. Further decreasing the resolution causes progressive increase in the number of frames for which the marker is not detected. The results are shown in Table 4.2 and Figure 4.

For face tracking, we use the following resolutions: 640×480 , 480×360 , and 320×240 . Similar to the marker statistics above, the graphs in Figure 5 show that tracking accuracy increases with resolution, although the relative differences are small.

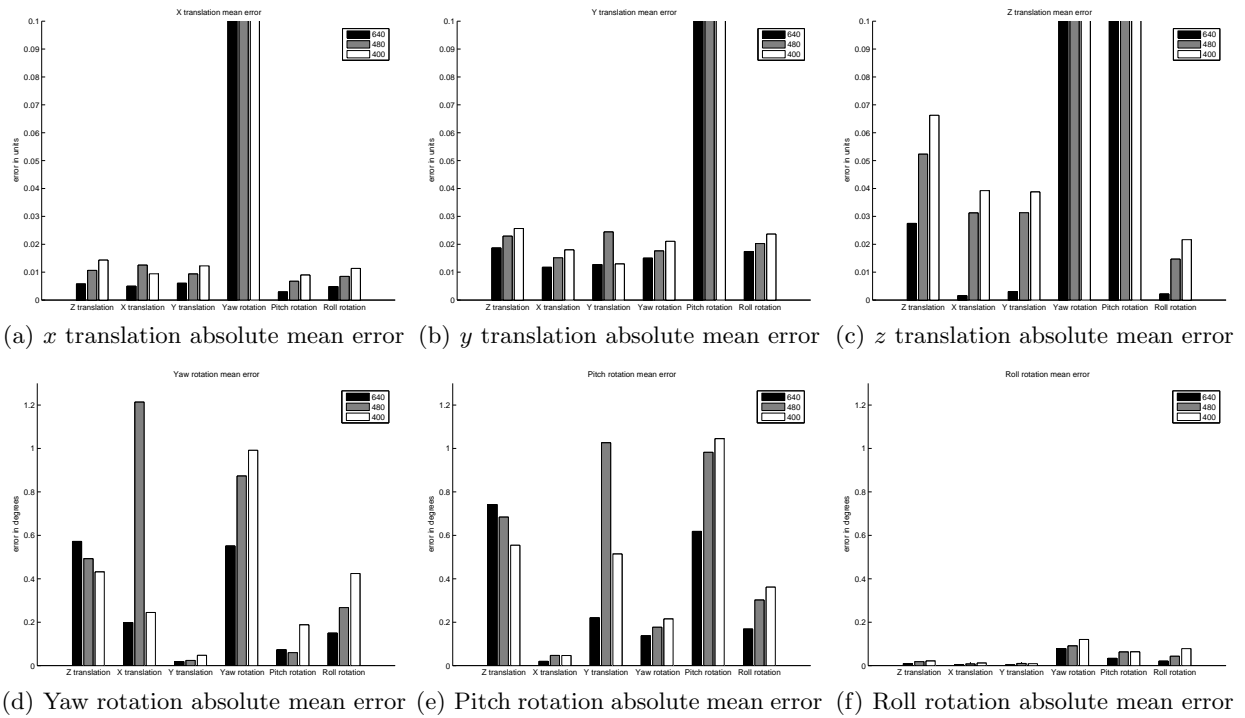


Figure 3. Comparison of errors at different resolutions for marker tracking

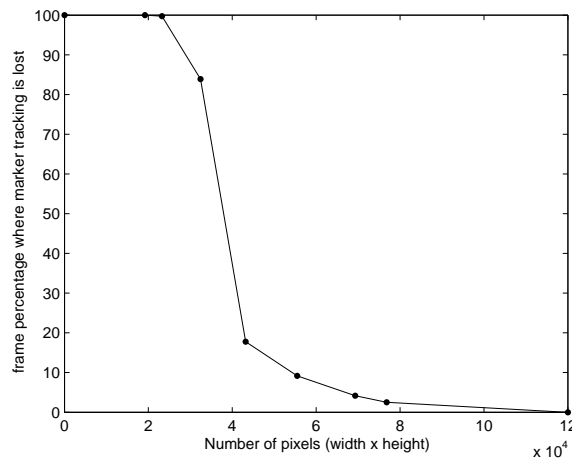


Figure 4. Robustness vs. frame resolution

The means and standard deviations calculated for the data shown in Figure 6 did not give any statistically conclusive decisions about MonoSLAM from a comparison point of view. Yet, the mean errors incurred seem well-bounded within acceptable levels of performance. The complete behavioral study of MonoSLAM could be a whole topic in itself, as there are a number of factors that it depends on, like number of detectable salient image patches in the scene, different sequences of movement (the order of camera movement directions could be rearranged to test again) etc.

Table 3 and Figure 7 show the processing time for marker tracking, face tracking and MonoSLAM for different resolutions. Marker and face tracking run on a 3 GHz Pentium D processor, whereas MonoSLAM runs on 2.8 GHz Pentium D processor. The processing time of all the three algorithms increases with resolution. The MXRTToolkit implementation of marker tracking is very efficient. MonoSLAM performs on par with marker tracking when

| Resolution | Number of frames where Marker is not detected |
|------------------|---|
| 160×120 | 360 |
| 176×132 | 359 |
| 208×156 | 302 |
| 240×180 | 64 |
| 272×204 | 33 |
| 304×228 | 15 |
| 320×240 | 9 |
| 400×300 | 0 |

Table 2. Robustness vs. frame resolution

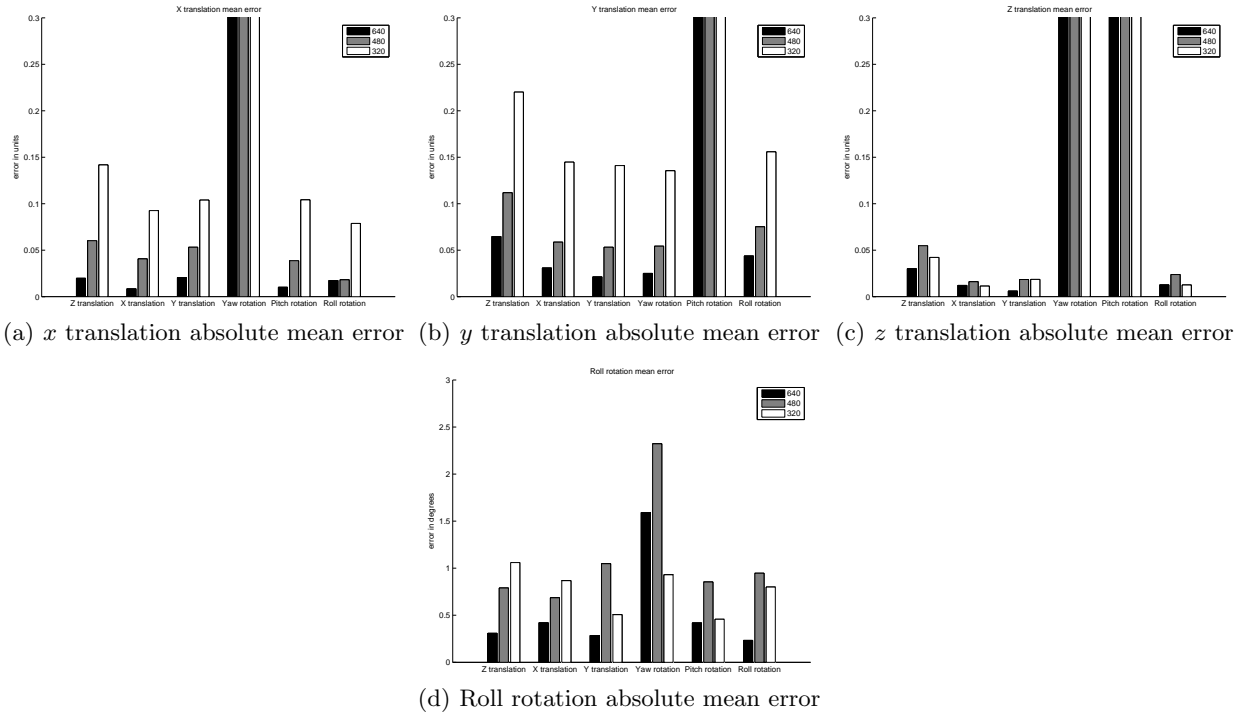


Figure 5. Error comparison of different resolutions for face tracking

looking at the varying trend in the processing time curves. The OpenCV-based CamShift algorithm for face tracking takes a much longer time to process, as it needs several stages to differentiate the skin color of the face from the environment.

| Resolution | Marker tracking (ms) | Face tracking (ms) | MonoSLAM (ms) |
|------------------|----------------------|--------------------|---------------|
| 400×300 | 3.64 | 10.88 | 5.28 |
| 480×360 | 4.50 | 15.64 | 7.47 |
| 560×420 | 5.31 | 21.26 | 8.00 |
| 640×480 | 6.31 | 27.74 | 10.30 |

Table 3. Processing time for different resolutions

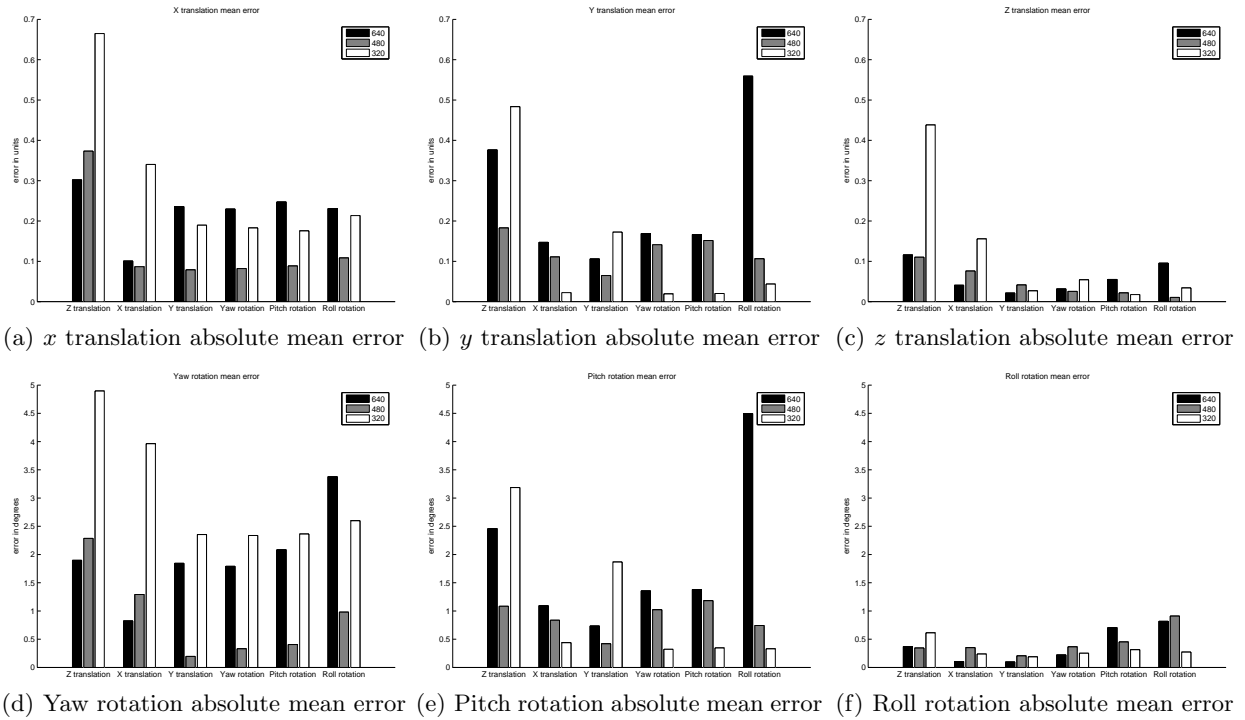


Figure 6. Error comparison of different resolutions for MonoSLAM

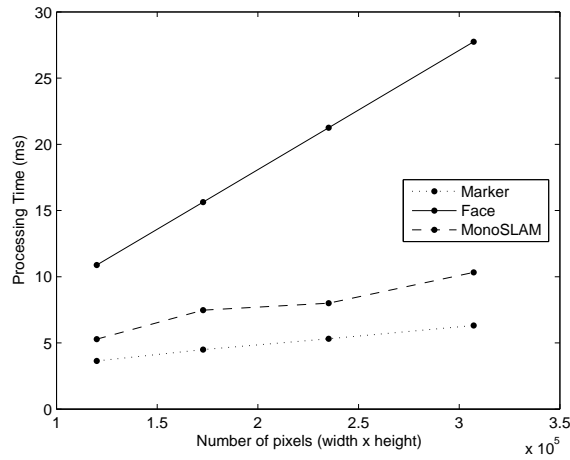


Figure 7. Processing time vs. resolution

4.3 Frame Dropping

Most tracking algorithms assume smooth movement from frame to frame. To test the robustness of the algorithms, we conduct several tests, in which we gradually lower the frame rate by dropping frames.

In four different test clips, we process every frame, every 2nd, 4th and 6th frame, respectively. The results show no significant differences for marker and face tracking. The CamShift implementation that we use actually depends on smooth continuous movement, but it works quite well with discrete movement in our experiment because we do not have any other skin-colored objects in the virtual world. MonoSLAM still works quite well for every alternate frame being dropped, but loses track when two or more frames are dropped, i.e., at lower frame rates.

5. CONCLUSION

We have compared three major motion tracking algorithms to estimate camera's ego-movement namely marker-tracking, face-tracking, and MonoSLAM using a simulated virtual environment. Marker-based tracking is the most accurate, followed by face tracking (although it only support 4 degree of freedom). MonoSLAM also performs well, even though there are a lot of factors that affect its robustness in performance that still need to be tested more thoroughly in order to establish its performance bounds. Resolution scaling has small impact on accuracy, but it significantly influences processing time. Marker and face tracking are robust toward fast discrete movement, while MonoSLAM needs a smoother continues movement to minimize uncertainties in 3D pose estimation.

REFERENCES

1. MXRToolkit. <http://mxrtoolkit.sourceforge.net/>.
2. H. Kato and M. Billinghurst, "Marker tracking and HMD calibration for a video-based augmented reality conferencing system," in *Proc. 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR)*, pp. 85–94, (San Francisco, CA), 1999.
3. G. R. Bradski, "Computer vision face tracking for use in a perceptual user interface," *Intel Technology Journal* **Q2**, 1998.
4. A. J. Davison, "Real-time simultaneous localisation and mapping with a single camera," in *Proc. 9th IEEE International Conference on Computer Vision (ICCV)*, p. 1403, (Nice, France), 2003.
5. S. Winkler, K. Rangaswamy, and Z. Zhou, "Intuitive user interface for mobile devices based on visual motion detection," in *Proc. SPIE Multimedia on Mobile Devices*, **6507**, (San Jose, CA), January 2007.
6. S. Winkler, K. Rangaswamy, and Z. Zhou, "Intuitive map navigation on mobile devices," in *Proc. 12th International Conference on Human-Computer Interaction, Lecture Notes in Computer Science* **4555**, pp. 605–614, (Beijing, China), July 2007.
7. R. Lienhart, A. Kuranov, and V. Pisarevsky, "Empirical analysis of detection cascades of boosted classifiers for rapid object detection," in *Proc. DAGM Symposium, Lecture Notes in Computer Science* **2781**, pp. 297–304, 2003.
8. P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 511–518, (Kauai, HI), 2001.
9. J. Shi and C. Tomasi, "Good features to track," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 593–600, 1994.