# Model-Based 3D Object Tracking Using an Extended-Extended Kalman Filter and Graphics Rendered Measurements

Hua Yang
*Computer Science Department*
*Univ. of North Carolina at Chapel Hill*
*Chapel Hill, NC 27599*
*yanghua@cs.unc.edu*

Greg Welch
*Computer Science Department*
*Univ. of North Carolina at Chapel Hill*
*Chapel Hill, NC 27599*
*welch@cs.unc.edu*

***Abstract***

*This paper presents a model-based 3D object tracking system that uses an improved Extended Kalman filter (EKF) with graphics rendering as the measurement function. During tracking, features are automatically selected from the input images. For each camera, an estimated observation and multiple perturbed observations are rendered for the object. Corresponding features are extracted from the sample images, and their estimated/perturbed measurements are acquired. These sample measurements and the real measurements of the features are then sent to an extended EKF (EEKF). Finally, the EEKF uses the sample measurements to compute high order approximations of the nonlinear measurement functions, and updates the state estimate of the object in an iterative form. The system is scalable to different types of renderable models and measureable features. We present results showing that the approach can be used to track a rigid object, from multiple views, in real-time.*

## 1. Introduction

Vision-based object tracking typically involves estimating the state of a moving object from a sequence of camera images. The state typically includes the object pose (position and orientation) and derivatives (velocity and acceleration) parameters. Model-based object tracking makes the assumption that some prior information of the tracked object is available in terms of a model, that can be used to generate a estimated observation of the object at some predicted pose. Thus the state of the object can be estimated by comparing the real observation to the predicted one. The comparison normally involves matching features extracted from different observations.

In this paper, we present a model-based object tracking system. The input of the system is a graphics model of an object and image sequences from single or multiple cameras observing the object. The output is the state estimation of the object. For a single-camera system, in each frame, features are selected from the input image at runtime. A synthetic image is rendered to estimate the observation (from the specific camera) of the object at its predicted state. The real and estimated measurements are acquired by finding positions of the corresponding features in the real and synthetic images. Similarly, more synthetic images of the object are rendered at perturbed poses around its state estimate, features are extracted and their measurements are derived. These perturbed feature measurements are later used to compute high order approximations of the nonlinear measurement functions. This provides an efficient way to compute the estimated measurement of a feature and the Jacobian matrices of the the feature's measurement function. Finally, the EEKF

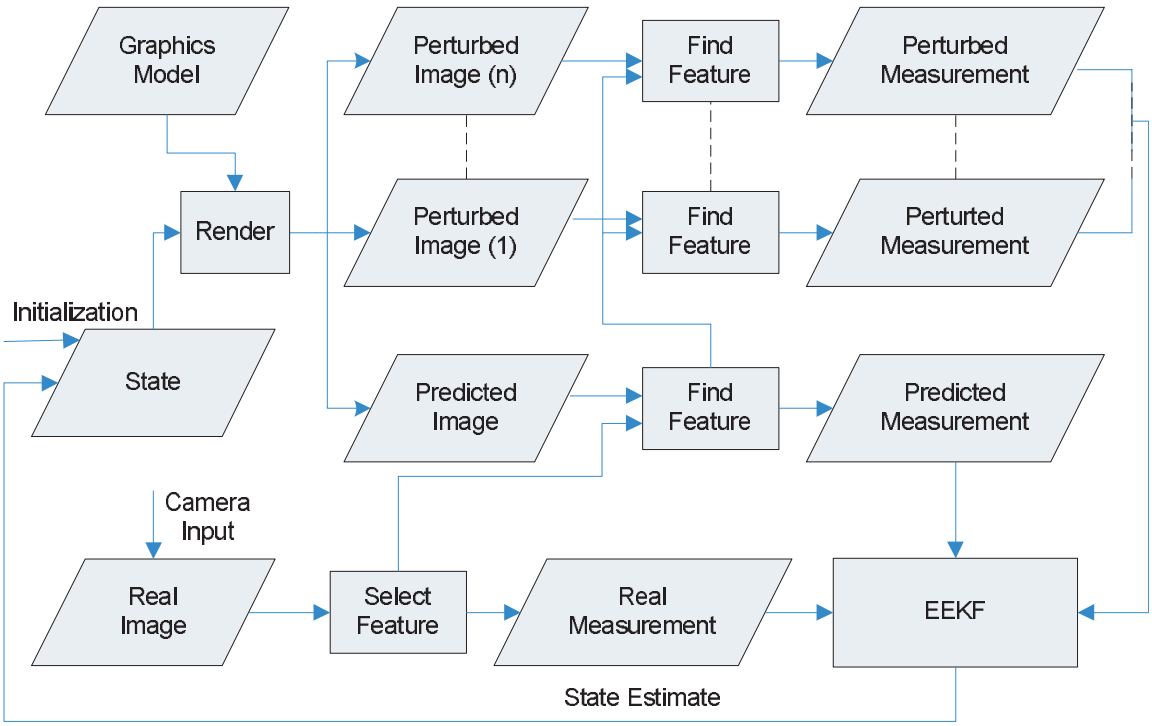


**Figure 1. Flowchart of the tracking system.**

uses the measurement function approximations and the real measurements to estimate and update the state of the object. An overview of the procedure is shown in Fig. 1. For a multi-camera system, such a procedure proceeds sequentially for all the camera views.

The proposed system prototype can be generalized into a framework of model-based object tracking using graphics rendering as the measurement function. As long as the model is renderable and the features are measurable, this framework can be applied. As shown in the experiment results, a textureless skull with a shape model is tracked using silhouette information, and a checkerboard with an appearance model is tracked using Kanade-Lucas-Tomasi (KLT) features [12]. Although it is not implemented in our system, this framework has the potential to perform model refinement. In that case, the state of the object should be extended to include the model.

Section 1.1 presents a brief survey of related literature on model-based object tracking. Section 2 reviews the the Extended Kalman filter approach to state estimation. An extended EKF (EEKF) is introduced. In Section 3 an overview of the proposed system is given with emphasis on selecting features and computing Jacobian matrices through graphics hardware accelerated rendering. Section 4 describes the detailed approach and the associated results. Section 5 offers concluding and future work thoughts.

### 1.1. Previous Work

Different model-based object tracking methods have been proposed. Classical approaches use geometric or appearance models with features defined in the model space. For example, Gennery [3] and Bray [1] exploited a wire frame model to track polyhedral objects. Dellaert *et al.* [2] proposed a Kalman filter based system that simultaneously estimates the pose and refines the texture of a planner patch. Nickels and Hutchinson [8] tracked articulated objects through matching feature

templates. Although these methods achieve good results in their own applications, they are prone to have the problem of establishing the image-model correspondence for projective-variant features. Moreover, some features can not be defined in the model space. For instance, the silhouette of an object.

One solution to the problem is to use a view-dependent representation of the object. In [15] a novel 3D object tracking system was developed, in which the model of the object is a pre-acquired light-field. An alternate approach is to select features directly from images. For instance, Rehg and Kanade [6] tracked articulated hand motion using a generalized cylinder model of the hand. Stenger *et al.* [11] also proposed a method to capture hand motion. Their model is built from truncated quadrics. Both systems extract the contours of the hand from images and use them as the visual cues for tracking.

Although defining features in the image space is a natural way to solve the problem of false matching for projective-variant features, it has not been adopted by any real-time Extended Kalman filter based 3D object tracking system. We believe this is due to the fact that in order to propagate distributions of the EKF one needs to compute the derivatives of the measurements of the features with respect to the motion of the object. Conventionally, computing the derivative of the measurement function of a feature requires its coordinate in a 3D space. Which is hard (if not impossible) to compute for features extracted from a 2D camera image.

## 2. Nonlinear Filtering

The problem of tracking an object from input measurements can be formulated as optimizing the state estimate of a nonlinear system. A number of Bayesian techniques have been proposed to perform the optimization. When Gaussian distribution is assumed, commonly used approaches include the EKF and the Unscented Kalman filter (UKF) [4]. To handle multi-modal distribution, the particle filter is used [9]. An EKF computes the Jacobian matrices and derives a linear approximation to the nonlinear function. While the UKF and the particle filter avoid computing the Jacobian matrices and propagate the distribution by sampling. In this paper, we assume an unimodal distribution, and use an extended EKF (EEKF) to achieve a balance between the computation and the performance.

### 2.1. Extended Kalman filtering

The extended Kalman filter addresses the problem of estimating the state of a nonlinear system. There are two basic Kalman filter equations for an uncontrolled system.

$$X_k = f(X_{k-1}, w_{k-1}) \tag{1}$$

$$Z_k = h(X_k, v_k) \tag{2}$$

Equation (1) is the state-transition or the process equation. Where $X$ is the state of the system, $k$ is the time stamp, $f$ describes the process model of the system, $w$ is the process noise. In the measurement equation (2), $Z$ is the measurements of the system, $h$ is a nonlinear function that relates the state of the system $X$ to the measurements $Z$, and $v$ is the measurement noise. Notice that in order to apply the EKF, the additive process and measurement noise of the system should be independent, white and with Gaussian probability distributions. We denote them as $p(w) \sim N(0, Q)$ and $p(v) \sim N(0, R)$. In practice one does not know the real values of state as well as the noise. However, we can approximate the state and the measurement as:

$$\tilde{X}_k = f(\hat{X}_{k-1}, 0) \tag{3}$$

$$\tilde{Z}_k = h(\tilde{X}_k, 0) \tag{4}$$

where $\hat{X}$ is the a posteriori estimate of the state from the previous time stamp $k-1$.

In a nonlinear system, process function $f$ and measurement function $h$ can be linearized about the current state estimate of the system by computing their partial derivatives. Thus Equation (1) and (2) can be transformed into:

$$X_k = \tilde{X}_k + A_k(X_{k-1} - \hat{X}_{k-1}) + W_k w_{k-1} \tag{5}$$

$$Z_k = \tilde{Z}_k + H_k(X_k - \tilde{X}_k) + V_k v_k \tag{6}$$

Where $X$ and $Z$ are the actual state and measurement vectors, $\tilde{X}$ and $\tilde{Z}$ are the approximate state and measurement computed from (3) and (4), $A$ and $H$ are the Jacobian matrices of partial derivatives of $f(\hat{X}_{k-1})$ and $h(\hat{X}_{k-1})$ with respective to $X$, and $W$ and $V$ are the Jacobian matrices of partial derivatives of $f$ and $h$ with respective to $w$ and $v$.

Given the linearization of the system, the EKF perform state estimation using a prediction-correction mechanism, as shown in Algorithm 1. At each frame, the filter predicts the current state of the system and correct this estimated state using measurements of the system. The updated estimate will then be used to predict the state for the next frame. The reader is referred to [13] for the derivation of the Kalman filter equations.

**Algorithm 1: Extended Kalman filter**

---

1) Predict the state:
$\hat{X}_k^- = f(\hat{X}_{k-1}, 0)$
2) Compute the error covariance:
$P_k^- = A_k P_{k-1} A_k^T + W_k Q_{k-1} W_k^T$
3) Compute the Kalman gain:
$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + V_k R_k V_k^T)$
4) Update the state estimate
$\hat{X}_k = \hat{X}_k^- + K_k(Z_k - h(\hat{X}_k^-, 0))$
5) Update the error covariance
$P_k = (I - K_k H_k) P_k^-$

---

### 2.2. Iterated Extended Kalman Filtering

The core of the extended Kalman filter is the linearization of the nonlinear system around its current estimated state. This linearizing process can cause significant error if the measurement function is highly nonlinear. One approach to address this problem is the *iterated extended Kalman filter* (IEKF). The key observation that leads to the IEKF is that, after each iteration, the EKF generates a better estimate of the state of the system. Which can be used to derive a better approximation of the measurement function. The IEKF iteratively estimates the state of the system. The estimation result of one iteration is used to re-linearize the measurement function. Thus in the next iteration, the state of the system will be computed using the updated Jacobian matrix and measurement estimate. A detailed description of the IEKF can be found in [14].

By re-linearizing the measurement function, the IEKF derives a better approximation of the measurement and generates a better state estimate. However, the re-linearization process introduces

additional computation. This computation can be trivial if an analytical representation of the measurement function $h$ is available. In this case, the $H$ matrice can be computed directly. Otherwise, a numerical differentiation has to be applied using Equation (7).

$$H(\hat{X}^-) = \frac{Z|_{X=\hat{X}^-+\varepsilon} - Z|_{X=\hat{X}^-}}{\varepsilon} \tag{7}$$

The numerical computation of $H$ can be expensive, since it has to be applied for every measurement and for every iteration. This can preclude the use of the IEKF in real-time applications. To solve this problem, we propose a new Kalman-based approach, which we call the Extended Extended Kalman filter (EEKF). Like the IEKF, the EEKF iteratively sents measurements to a standard EKF to estimate the state of the system. But it avoids the excessive numerical differentiation by pre-computing a nonlinear approximation of the measurement function. This is done through acquiring multiple samples of the measurement at states around current state estimate.

### 2.3. Extended Extended Kalman Filter

In practice, an analytical representation of the measurement function is not always available. However, one can always samples the measurement function and compute a Taylor expansion for it.

$$\begin{aligned} &h(\hat{X}) = h(\hat{X}^-) + \textstyle\sum \frac{1}{i!} h^i(\hat{X}^-)(\hat{X} - \hat{X}^-)^i (i = 1..n-1) \\ &where \\ &h(X_1) = Z_1|_{X=X_1} \\ &..... \\ &h(X_n) = Z_n|_{X=X_n} \end{aligned} \tag{8}$$

In general, given $n$ sample measurements $Z_1, ..., Z_n$ acquired at states $X_1, ..., X_n$ that are close to the current state estimate $\hat{X}^-$, one can numerically solve Equation (8) and compute the derivatives of $h|_{X=X_1}$ to the order of $n-1$: $h', h'', ..., h^{n-1}$ or $H, H', ..., H^{n-2}$. The high order approximation enables efficient re-linearization of the measurement function $h$ at a different state. Again using a Taylor expansion, one can approximate the Jacobian matrix $H$.

$$H(\hat{X}) = H(\hat{X}^-) + \sum \frac{1}{i!} H^i(\hat{X}^-)(\hat{X} - \hat{X}^-)^i (i = 1..n-2) \tag{9}$$

The Extended Extended Kalman filter (EEKF) is designed to estimate the state of a nonlinear system, when an analytical representation of its measurement function is not available. The intuition of the EEKF is sampling and approximating the nonlinear system. At each time stamp, multiple measurement samples are generated at states around current state estimate. A high order ($\geq 2$) approximation of the measurement function is computed using these samples. After that, the EEKF performs state estimation iteratively. In each iteration, the measurement function is relinearized at the state estimated from previous iteration, a standard EKF is then applied to update that state estimate. The algorithm of the EEKF is shown in Algorithm 2.

Both the IEKF and the EEKF iteratively relinearize the measurement function and apply an EKF to perform state estimation. In fact, they are quite similar. The difference between them lies in the relinearization. The IEKF relinearizes the original measurement function. In the absence of an analytical representation of the measurement function, relinearization involves numerical differentiation, an expensive process. On the other hand, the EEKF relinearizes the approximated measurement function. Numerical differentiation is not required anymore, since we already have an analytical measurement function, an approximated one though.

**Algorithm 2: Extended Extended Kalman filter**

---

1) Predict the state $\hat{X}^-$.
2) Compute the error covariance $P^-$.
3) Generate multiple samples of *Z* around $\hat{X}^-$.
4) Derive an high order approximation of *h* using Eq(8).
5) Compute the Kalman gain *K*.
6) Update the state estimate $\hat{X}$.
7) Update $\tilde{Z}$ and *H* using Eq(8) and (9).
8) Assign $\hat{X}$ to $\hat{X}^-$.
9) $|\hat{X} - \hat{X}^-| > thresh$ ? Goto (5) : Continue
10) Update the error covariance *P*.

---

## 3. Object Tracking

In this section, the EEKF is applied to accomplish the task of object tracking. Feature selection and matching and graphics hardware accelerated measurement sampling are described.

### 3.1. System Models

The goal of tracking an object is to estimate and update its state. Typically, the state of an object consists of its pose (position and orientation) and its velocities (translation and rotation). In our system, we assume constant-speed motion of the object between consecutive frames. Thus the state $X$ of a rigid object in the 3D space is defined as $[x\ y\ z\ \phi\ \psi\ \gamma\ x'\ y'\ z'\ \phi'\ \psi'\ \gamma']$. Applying this state representation to the Kalman filter equations, we acquire a linear state transition function *f* (Equation (1)). Its derivative matrix can be written as:

$$A = \left| \begin{array}{cc} I_{6\times6} & \Delta t I_{6\times6} \\ 0 & I_{6\times6} \end{array} \right|$$

Where $\Delta t$ is the time between two consecutive frames.

The input of the system are camera images from one or multiple cameras. Features are extracted from these images. For any feature, its coordinate in the image space $(ix, iy)$ is a measurement $Z$ of the system. Given the state of the object *X*, the camera parameters $C$ and the coordinate of the feature in the object model space $P$, the image coordinate of a feature can be computed using a 3D-2D projection, a highly nonlinear process. The measurement function *h* in Equation (2) can be written as:

$$Z = h(X, C, P, v)$$

Notice that $v$ is white noise and $C$ is constant for a fixed camera. Thus *h* is a function of $X$ and $P$. If $P$ is also given, the linearization of $h$ can be computed directly. Otherwise the derivative of $h$ has to be computed numerically using Equation (8). Multiple images of the object at neighboring states are generated as described in Section 3.3.

### 3.2. Feature Selection and Matching

Conventional methods define features in the model space. So when an estimate of the object's pose is available, the estimated measurement of the feature can be determined by projecting the

model onto the image plane, using internal camera parameters. This approach works well for tracking geometric features defined by a wire-frame model, such as vertices and edges. However, it has problem with features whose appearances are subject to the viewpoints. Most types of the texture features fall into this category. For these features, it is hard to match the one extracted from the real image to its template defined in the model space. Besides, some important types of features, such as the silhouette of the object, can not be defined in the model space.

In the proposed system, the features are automatically selected from the input images at run time. This method has several advantages. Self-occluding is easily solved. Silhouette of the object can be detected. Moreover, extracting features from the image space increases the accuracy of feature matching. Because a feature extracted from a camera image is presumably one of the most distinguishable features from the camera's viewpoint, and it is used to search for corresponding ones in images that are acquired or generated from nearby perspectives.

When a feature is defined in the image space, a numerical method has to be applied to compute the derivative of its measurement function, for its coordinate in the model space is generally not available. Back-projection is not feasible for lack of the depth information. In a multi-camera system, one may recover the 3D position of a feature by triangulation. However, the triangulation requires accurate matching of features over two views that are considerably apart from each other. For projective variant features, such a task is hard to achieve. Besides, the triangulation can not be applied to features like the silhouette.

### 3.3. Generating Measurement Samples Using Graphics Rendering

As described earlier, for a feature extracted from the image, multiple sample measurements of the feature need be acquired to derive an approximation of its measurement function. The pose of a 3D object is a vector of 6 DOF. To generate a second order approximation, at least three (one center/estimated and two perturbed) samples have to be acquired along each of the 6 dimensions. The centered sample can be shared by all 6 dimensions. Thus a total of 13 samples have to be acquired. And they are only the samples for one feature. If a system consists of $m$ cameras and an average of $n$ features are selected for each of the camera view, a total of $13mn$ feature samples need to be acquired. That is a challenging task for a realtime system.

When a graphics model of the object is available, one can generate feature samples using graphics hardware accelerated rendering. For each camera, one center/estimated image is rendered using the estimated pose parameters. Besides, 12 other images are rendered at perturbed poses. These perturbed poses are derived by adding a perturbation along one of the 6 dimensions to the center pose. After that, corresponding features are extracted from all 13 images and and their measurements are found. Compared with the CPU based method, the GPU based method is more efficient. The modern graphical processing unit is fast. A moderate graphics model that consists of thousands of polygons can be rendered in milliseconds. As shown in the experiment result, 13 sample images are rendered and read back to the memory in 100 ms using a NVIDIA GeForce 6800. Moreover, graphics rendering is a parallel method, samples of features from the same view are generated simultaneously. Thus one can expect a better feature-per-second performance when the number of features increases.

Recall that samples of features are used to generate functions that approximate the measurement functions locally. Thus the perturbation of the object's pose should be small enough to derive relatively accurate approximations. However, the perturbation should not be too small, for the difference between two measurements of the sampled feature should be measureable. The magnitude of an appropriate perturbation is determined by the size of the object and the distance between the

object and the camera. In our current implementation, given the prior knowledge on the system's working volume and the object's size, a predefined fixed perturbation is chosen to make the difference between measurements to be of the magnitude of several pixels. A more practical way would be to let the system choose the magnitude of the perturbation adaptively and automatically.

### 3.4. Initialization

To track an object, an initial state estimate needs to be provided. While automatic state initialization is feasible, for our prototype implementation, our current prototype assumes the object to be static at the first frame and computes its pose using a manual method. A set of $N$ ($N \geq 3$) features are chosen from the object model. The coordinates of these features in the model space are acquired. Markers are added to the real object at the corresponding positions. At the first frame, these markers are manually identified in the input images. Their positions are computed by triangulation. At this point, the coordinates of the markers in both the world space and the model space are known. One can then use a linear solver to compute a minimum sum of square solution for the transformation matrix between the two 3D spaces. The pose of the object can be derived from the linear 3D transformation.

## 4. Experimental Results

The imaging device of our tracking prototype consists of four calibrated and synchronized Pointgrey Dragonfly digital cameras. The system is implemented in C++ using GLUT and Opencv libraries. The algorithm runs on an AMD Athlon 64 FX-55 processor (2.61 GHz) with PCI express and a NVIDIA GeForce 6800 videocard. The time performance of the prototype is shown in Fig. 2:

| | | |
|---|---|---|
| Render and read-back | /image | 6.67 ms |
| Silhouette feature extraction and matching time | /image | 0.23 ms |
| Number of synthetic images samples | /frame | 52 |
| Number of real images | /frame | 4 |
| Render and read-back time | /frame | 347 ms |
| Silhouette feature extraction and matching time | /frame | 13 ms |
| Kalman filtering time | /frame | 3 ms |
| Total processing time | /frame | 380 ms |

**Figure 2. Time performance of model-based tracking. About 500 silhouette features per image, and 13 images per camera, and 4 cameras per frame are used.**

We test the performance of the system on three experiments using both synthetic and real data. In the first experiment, the prototype uses four sequences of graphics generated images to track the motion of a synthetic solid cube, as shown in Fig. 3. To demonstrate the capability of the EEKF, we perform model-based tracking using both the standard EKF and the EEKF. The estimation error using both methods are compared in Fig. 4. One can see that the EEKF surpasses the EKF with more accurate pose estimates.

In the second experiment, a checkerboard is tracked using $640 \times 480$ image sequences from four cameras. Texture features are extracted from the images using KLT method. The tracking result is

illustrated in Fig. 5. A synthetic image of the checkerboard model is rendered from one of the four viewpoints of the cameras, using the estimated pose parameters. Which is then blended with the real image from the same camera. The third experiment demonstrates the tracking of a textureless skull model in front of a dark background. The measurement is the outline of the skull in the image space. To be more specific, the measurement is the samples of the skull's outline on a number of horizontal and vertical scanlines. The resulting images are shown in Fig. 6. In each of the images (b)–(f), a synthetic image of the skull model is rendered using state estimate, and then blended with the real image.

## 5. Conclusion and Future Work

In this paper, we present a prototype of object tracking system. The system prototype uses graphics hardware accelerated rendering to acquire sample measurements of the tracked object. These measurements are sent to an Extended Extended Kalman filter. Which generates nonlinear approximations of the measurement functions then estimates object motion iteratively. The system can be generalized into a model-based object tracking framework that supports different types of renderable models and measureable features. Experiment results show that, using inputs from multiple cameras, the prototype can be applied to track a rigid object robustly and accurately in real-time.

Some future works can be attempted to make the system more practical. For instance, the automatic initialization and the adaptive perturbation-decision may be implemented. Besides, our current system requires an accurate model of the object. In the future, we would build a *smarter* system that starts tracking with a coarse model of the object and performs online model-refinement. For a multi-camera system, the coarse model can be acquired using the stereo reconstruction or the convex-hull technique.

## Acknowledgements

At UNC-Chapel Hill we want to acknowledge Jim Mahaney and John Thomas for their technical support. We also want to thank Andy Christensen of Medical Modeling Corporation for his help with the MRI-based physical skull models shown in Fig. 6.

This effort is primarily supported by National Library of Medicine contract N01-LM-3-3514: "3D Telepresence for Medical Consultation: Extending Medical Expertise Throughout, Between and Beyond Hospitals," and in part by NSF grant EIA-0303590: "Ubiquitous Pixels: Transforming Collaboration & Teaching with Pervasive Wall Displays."

## References

[1] A. J. Bray, "Tracking objects using image disparities," *Image and Vision Computing*, Vol. 8(1), pp. 4–9, February 1990.

[2] F. Dellaert and S. Thrun and C. Thorpe, "Jacobian images of superresolved texture maps for model-based motion estimation and tracking,"*In Proceedings of the Fourth Workshop on Applications of Computer Vision*, October 1998

[3] D.B. Gennery, "Visual tracking of known three-dimensional objects," *International Journal of Computer Vision*, Vol. 7(3), pp. 243–270, April 1992

[4] S. J. Julier and J. K. Uhlmann, "A new extension of the Kalman filter to nonlinear systems," *In Proceeding of the International Symposium on Aerospace/Defense Sensing, Simulation and Controls, Orlando, FL*, 1997.

[5] I. A. Kakadiaris and D. Metaxas, "Model-Based Estimation of 3D Human Motion with Occlusion Based on Active Multi-Viewpoint Selection," *In Proceeding of the International Conference on Computer Vision and Pattern Recognition*, pp.81–87, 1996.

[6] J. M. Rehg and T. Kanade, "Model-Based Tracking of Self-Occluding Articulated Objects,"*In Proceedings of the Fifth International Conference on Computer Vision*, June 1995

[7] D. G. Lowe, "Object recognition from local scale-invariant features," *Proceedings of the 7th International Conference on Computer Vision*, pp. 1150–1157, 1999.

[8] K.Nickels and S. Hutchinson, "Model-based tracking of complex articulated objects," *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION*, Vol. 17(1), pp. 28–36, 2001.

[9] D.J. Salmond N.J. Gordon and A.F.M. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," *Radar and Signal Processing, IEE Proceedings F*, Vol. 140(2), pp. 107-113, April 1993.

[10] Y. Bar-Shalom and T.E. Fortmann, "Tracking and Data Association," *Academic Press*, Boston, 1988.

[11] B. Stenger, P. R. S. Mendonca and R. Cipolla, "Model-based 3D tracking of an articulated hand," *In Proceeding of the International Conference on Computer Vision and Pattern Recognition*, December 2001.

[12] C. Tomasi and T. Kanade, "Detection and Tracking of Point Features," *Carnegie Mellon University Technical Report, CMU-CS-91-132*, April 1991.

[13] G. Welch and G. Bishop, "An Introduction to the Kalman Filter," *Technical Report, TR 95-041, Department of Computer Science, University of North Carolina at Chapel Hill*, December 1995.

[14] Z. Zhang, "Parameter Estimation Techniques: A Tutorial with Application to Conic Fitting," *International Journal of Image and Vision Computing*, Vol. 25, pp. 59-76, 1997.

[15] M. Zobel, M. Fritz and I. Scholz, "Object Tracking and Pose Estimation Using Light-Field Object Models," *VISION, MODELING, AND VISUALIZATION*, pp. 371–378, November 2002.

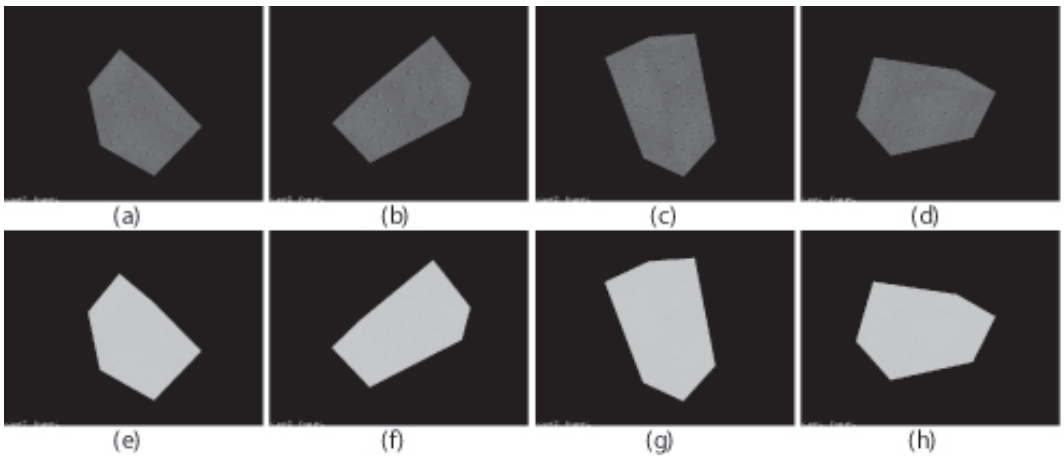


**Figure 3. Images (a-d) show the input images at certain frame, generated by four virtual cameras. Images (e-h) are the resulted images rendered from the viewpoints of the same cameras using the state estimate of this frame.**

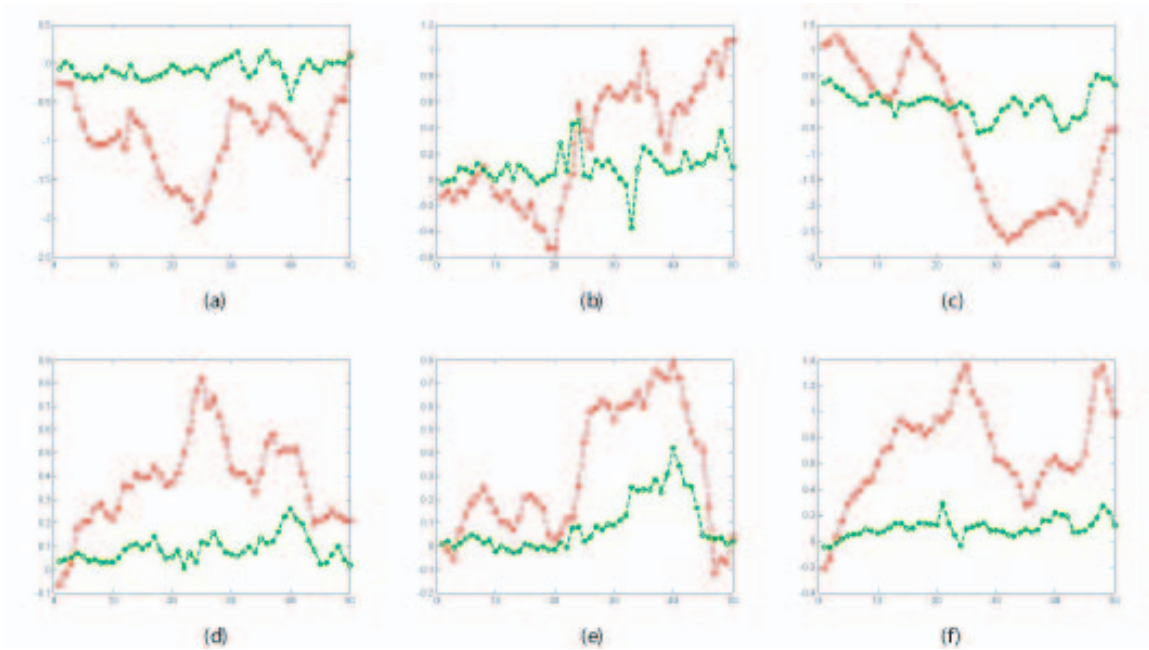


**Figure 4. Estimation error of the pose of the synthetic cube over 50 frames. The cube is tracked using synthetic image sequences from 4 virtual cameras. Image (a-c) show the error of the position (in mm). Image (d-f) show the error of the orientation (in degree). The red curves represent estimation error using the standard EKF. The green curves represent estimation error using the EEKF.**

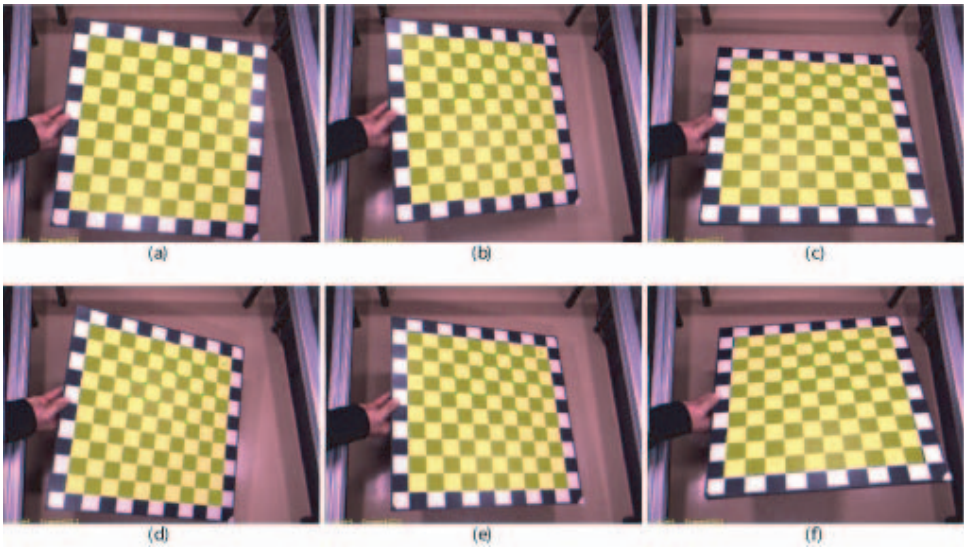


**Figure 5. Illustration of tracking a checkerboard. The center part of the checkerboard ($10 \times 10$ grid) is rendered from the viewpoint of one calibrated camera, using the estimated pose parameters. The resulting synthetic image is then blended with the real image from the same camera. Four cameras are used, here we only demonstrate the viewing results from one of them.**

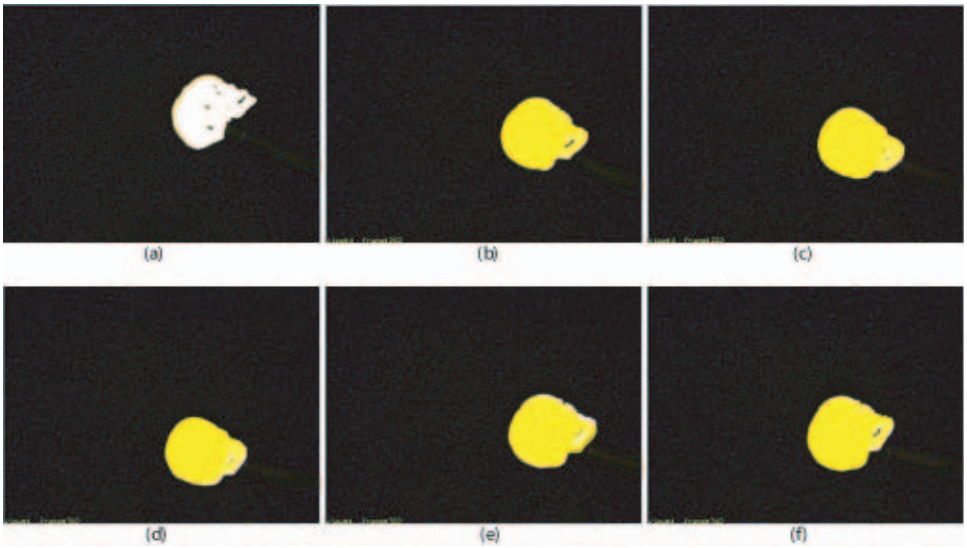


**Figure 6. Illustration of tracking a skull using samples of its outline on a number of vertical and horizontal scanlines. Four cameras are used; here we illustrate the viewing results from the perspective of one of them. Image (a) is the real image of the skull. The dark spots on the skull are the markers used for initialization. In images (d-f), synthetic images of the skull model are rendered using state estimate and are blended with the real images. (MRI-based skull models from Andy Christensen of Medical Modeling Corporation.)**