Real-time Camera Pose and Focal Length Estimation

Sumit Jain and Ulrich Neumann
Computer Science Department, University of Southern California
{sumit,uneumann}@graphics.usc.edu

Abstract

This paper presents a novel approach to estimate the changing internal and external parameters of the camera in real time using a few 3D-2D point correspondences. We approach the problem by constructing two filters which can individually track motion and zoom respectively in a robust manner. Then, the Viterbi algorithm is used to select a filter depending on whether the camera is undergoing motion or zoom. We assume that the camera cannot move and zoom simultaneously, though our method can tolerate some hand held movement during zoom. The application of our approach is demonstrated on synthetic and real data.

1. Introduction

The subject of vision based registration methods has received considerable attention in the past few years and many robust techniques have been developed for the same. However, these approaches assume that the camera internal parameters are fixed and hence only address the issue of pose estimation of the camera. Simultaneous estimation of internal and external camera parameters is a hard problem because zoom variations can be confused with camera motion along the optical axis which can lead to poor registration. In our approach, we lift this assumption of fixed internal parameters and estimate them along with the camera pose.

The complexity of the problem is reduced by assuming that camera motion and zoom do not occur at the same time, although some hand held camera motion during zoom is tolerated by incorporating it as noise in our model. This assumption is quite reasonable as it is human tendency to keep the camera stable while zooming. We extend the approach of pose tracking from the work of Davison [4] in which Extended Kalman filter (EKF) is used to localize camera pose and new features are added to the scene map simultaneously. Another EKF is built, which can track only the changing focal length but not the pose. We name these two EKFs as *MotionModel* and *ZoomModel* respectively. The $MotionModel$ works robustly when the camera is moving (said to be in state *motion*) while $ZoomModel$ works robustly when the camera is zooming (said to in state *zoom*). The system is tracked by both the models at all times, but only one model will work correctly at any time as per our assumption. Hence, the system will either be in $motion$ state or $zoom$ state at any time step. Our goal is to obtain the correct sequence of states the system goes through. The *Viterbi* algorithm [6] is employed to estimate this. The output of this algorithm is an optimal sequence of states through which the system should evolve. However, one may argue that the optimal sequence can be detected only after the completion of the tracking sequence, but we show later that it can be done on-line in our case.

The algorithm is initialized with a known configuration, i.e. both the external and internal camera parameters are known (These can be obtained by simple calibration techniques). The camera is then allowed to move and zoom freely, given the 3D-2D correspondences of a few feature points (typically 6-10). Our algorithm also depends on few other parameters, like process noise covariances and the weights in the Viterbi algorithm, which can be tuned easily. We emphasize that our method is independent of the 2D feature tracker and can work with natural feature tracking as well as marker based tracking. This allows us to incorporate our framework in any existing pose tracking method simply by replacing the pose tracking framework by ours thereby relaxing the constraint of fixed internal parameters. This is very useful in Structure from Motion (SfM) and Simultaneous Localization and Mapping (SLAM) problems.

The rest of the paper is organized as follows: Section 2 describes previous work done related to ours. Section 3 highlights the conventional pose tracking model and its extensions that we developed for pose and zoom tracking. Section 4 then describes how the two models are combined to output an optimal state vector. In Section 5, our results on simulation and real data are presented. Finally, in Section 6, we conclude our paper and discuss potential for future work.

2. Previous Work

Most of the recent work in active vision has been in the field of vision-based and hybrid registration methods for augmented reality [4, 8]. All these methods focus on 3D pose estimation of the camera and assume the internal parameters to be fixed and known. To the best of our knowledge, simultaneous on-line estimation of internal and external parameters has not been explored. Although many advances has been made in this direction like self-calibration [11, 12, 14, 9, 7, 5], all these approaches are typically offline.

Our motivation for pose tracking and extension to zoom tracking comes from the work of Davison [4], in which they presented real time robust pose estimation. They assumed a "constant linear velocity, constant angular velocity" (i.e. zero mean Gaussian accelerations) EKF formulation, which works pretty well for general camera motion.

Some work has been done in the past in modeling a zoom lens camera. Sturm [13] made an attempt to model the internal parameters like aspect ratio and principal point as a function of focal length by 'pre-calibration'. Simon and Berger [12] used an affine model to model the zoom lens camera internal parameters. We use simplifying assumptions, like fixed principal point and aspect ratio, which hold good for all practical purposes, as discussed in Section 3.1.

Multiple Kalman filter framework has been used mostly in signal processing community and airborne surveillance tracking problems. In methods like Interacting Multiple Model estimator (IMM) [2, 10] (a suboptimal hybrid filter), the state of a dynamic system is estimated using several behavior modes which can *switch* among each other. Another technique called Mixture of Experts [3] is used, in which a bank of filters is maintained and each filter is modeled with a different realization of the unknown system parameters such as process and measurement noise. Weights are then given to individual filter estimates based on performance. We discuss in Section 4.1, why these schemes fail to work in our case.

3. Models for pose and focal length tracking

We briefly describe the EKF model used in [4]. A "constant velocity, constant angular velocity" model is used which means that the statistical model of motion expects that undetermined accelerations occur with a zero mean Gaussian distribution. This implies that very large accelerations are highly unlikely.

The overall state of the system is represented as a vector, $\mathbf{x}$, which can be partitioned into the state $\mathbf{x}_v$ of the camera and the states $\mathbf{y}_i$, the 3D positions of the feature points in the map. This state vector is accompanied by a covariance matrix, P, which can also be partitioned as follows:

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_v \\ \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \end{pmatrix}, P = \begin{pmatrix} P_{x_v x_v} & P_{x_v y_1} & P_{x_v y_2} & \cdots \\ P_{y_1 x_v} & P_{y_1 y_1} & P_{y_1 y_2} & \cdots \\ P_{y_2 x_v} & P_{y_2 y_1} & P_{y_2 y_2} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

where the state vector $\mathbf{x}_v$ of the camera is represented as

$$\mathbf{x}_v = \left(\mathbf{r}^W, \mathbf{q}^{WC}, \mathbf{v}^W, \boldsymbol{\omega}^W\right)^T \tag{1}$$

where $\mathbf{r}^W$ represents the position of the camera in the world coordinate system, $\mathbf{q}^{WC}$ represents the rotation quaternion from world reference frame to camera reference frame, $\mathbf{v}^W$ and $\boldsymbol{\omega}^W$ represent the velocity and angular velocity of the camera respectively in world reference frame. We extend this model thereby incorporating the varying focal length.

3.1. Modeling the zoom lens camera

We argue that only one parameter is sufficient to model the changing camera internal parameters during zoom. Sturm [13] modeled the interdependence of internal parameters (called pre-calibration) by empirically expressing parameters like aspect ratio, α, and principal point, $(u0, v0)$, as a function of the focal length, f. In general, we assume skew, s, to be zero.

We tried to follow this approach to pre-calibrate a AXIS 213 PTZ network camera and a JVC GR-D30U digital video camera. For both the cameras, we performed calibration (using Matlab Calibration Toolbox [1]) at various zoom levels. The aspect ratio remained almost constant for all zoom levels, but functional dependence of principal point on focal length could not be established. Sturm's model [13] doesn't appear to hold for all cameras, possibly due to variations in optics and mechanics or zoom systems. Thus, our approach is to calibrate focal length by fixing the principal point at the center of the image, and our tests indicate that this approximation is adequate i.e. re-projection error is still sub-pixel. So, we are left with estimating only one parameter, f, to fully determine our internal camera matrix.

Simon and Berger [12] used an affine model for internal parameters to model the zoom. This is not suitable in our case as it doesn't fit our model and is not applicable for online purposes due to its high computation time.

3.2. EKF framework for modeling zoom

To estimate zoom along with motion, the first instinct is to simply insert the focal length parameter in the state vector for tracking. We tried this by including the focal length, f, and focal velocity, v_f, in the camera state vector, $\mathbf{x}_v$, following the "constant velocity" model. But this has some limitations as discussed below.

Simon and Berger [12] experimentally showed that in a true zoom sequence, re-projection error for the first few frames is not necessarily lower in the case of modeling zoom (keeping the pose fixed), than in the case of modeling motion (keeping focal length fixed). Similar behavior is observed for a true motion case. This leads to lot of error in pose and focal length estimation even though the re-projection error is still low. Thus, we need more complex and a robust way of estimating zoom. We form two separate EKFs for modeling motion and zoom called *MotionModel* and *ZoomModel* respectively.

For $MotionModel$, we use the same camera state vector as in equation 1 with the addition of focal length, f, but not focal velocity, because we assume a "constant focal length" model, which means that focal velocity follows a zero mean Gaussian distribution. This helps the focal length to converge at correct value if it is estimated incorrectly. Therefore the new state estimate is given by

$$\mathbf{f}_v(\mathbf{x}_v, \mathbf{n}) = \begin{pmatrix} \mathbf{r}^W + (\mathbf{v}^W + \mathbf{V}^W)\Delta t \\ \mathbf{q}^{WC} \times \mathbf{q}((\boldsymbol{\omega}^W + \boldsymbol{\Omega}^W)\Delta t)^1 \\ \mathbf{v}^W + \mathbf{V}^W \\ \boldsymbol{\omega}^W + \boldsymbol{\Omega}^W \\ f + F \end{pmatrix} \tag{2}$$

where $\mathbf{n} = (\mathbf{V}^W, \boldsymbol{\Omega}^W, F)^T$ is the noise vector given by $(\mathbf{a}^W \Delta t, \boldsymbol{\alpha}^W \Delta t, v_f \Delta t)^T$, Δt being the time step and $\mathbf{a}^W$, $\boldsymbol{\alpha}^W$ and v_f being the linear acceleration, angular acceleration and focal velocity respectively (processes of zero mean Gaussian distribution). All noise components are assumed to be uncorrelated.

For $ZoomModel$, we delete linear and angular velocities from the state vector and insert focal length, f, and focal velocity, v_f, because we assume a "constant position, constant orientation, constant focal velocity" model, which means that linear and angular velocities and focal acceleration follow zero mean Gaussian distributions. The new state estimate is given by

$$\mathbf{f}_v(\mathbf{x}_v, \mathbf{n}) = \begin{pmatrix} \mathbf{r}^W + \mathbf{R}^W \\ \mathbf{q}^{WC} \times \mathbf{q}(\mathbf{Q}^W) \\ f + (v_f + V_f)\Delta t \\ v_f + V_f \end{pmatrix} \tag{3}$$

where $\mathbf{n} = (\mathbf{R}^W, \mathbf{Q}^W, V_f)^T$ is the noise vector given by $(\mathbf{v}^W \Delta t, \boldsymbol{\omega}^W \Delta t, a_f \Delta t)^T$, Δt being the time step and $\mathbf{v}^W$, $\boldsymbol{\omega}^W$ and a_f being the linear velocity, angular velocity and focal acceleration respectively. The addition of positional noise helps to tolerate noisy movement as in a hand held camera. The EKF equations can be now set up in a usual fashion by computing the appropriate Jacobians as in [4].

Therefore, by using $MotionModel$, we can track the camera motion and by using $ZoomModel$, we can track the camera zoom robustly. In the next section, we describe how to track the camera parameters in a video sequence in which both camera motion and zoom occur.

[1] $\mathbf{q}(\mathbf{x})$ is a quaternion with angle of rotation $|\mathbf{x}|$ about an axis $\frac{\mathbf{x}}{|\mathbf{x}|}$

4. Combining outputs of the two EKFs

We are now left with the problem of combining the outputs of the two EKFs. Ideally we want the output of $MotionModel$ when camera is in motion and the output of $ZoomModel$ when camera is zooming. We discussed in the previous section that this decision cannot be solely based on re-projection error of a single frame.

4.1. IMM and Mixture of Experts

Methods like IMM [2, 10] and Mixture of Experts [3] are typically used for such scenarios in which switching between various filters based on their performance is desired. The bottom-line of these algorithms is that the weighted combination of output of each filter is fed as input to each filter for the next step. These weights are adaptive and are based on filters' performance. These methods work rather poorly in our case, due to the following reasons. Firstly, since only one model is correct at any time, mixing the outputs of the two models introduces error in the state vector and the associated covariances, which makes the system unstable. Secondly, the weights are updated depending on the current re-projection error, hence, incorrect importance is given to the filters which leads to poor estimation of parameters even though the re-projection error is minimized (as discussed in Section 3.2).

4.2. Employing the Viterbi algorithm

The Viterbi algorithm [6] is very popular in the signal processing community and has been widely used in applications like speech recognition etc. because it is an optimal yet computationally efficient algorithm for signal detection. It is a dynamic programming algorithm for finding the most likely sequence of hidden states that result in a sequence of observed events.

In our case, $MotionModel$ works robustly when the camera is moving (said to be in state *motion, M*) while $ZoomModel$ works robustly when the camera is zooming (said to in state *zoom, Z*). The system is tracked by both the models at all times. But only one model will work correctly at any time as per our assumption. Hence, the system will either be in the *motion* state or the *zoom* state at any time step. So our problem statement becomes: Given the image measurements of feature points, find the sequence of states (M or Z), such that the re-projection error is minimized. We now explain how the algorithm works in our setup (see figure 1). A transition $X \rightarrow Y$ ($X, Y \in \{M, Z\}$) is defined

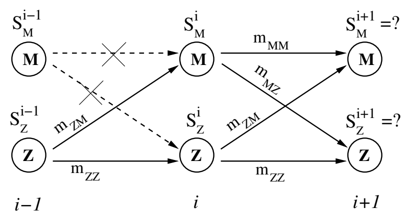


Figure 1. State transitions in Viterbi algorithm

as state vector of X being tracked by EKF of Y. Since at each time step, we are tracking state vectors by two filters, we have four possible transitions. Let us denote the state $X \in \{M, Z\}$ at i^{th} step by X^i. Now we have to select two transitions such that they are the most 'profitable' ways to reach states M^{i+1} and Z^{i+1} respectively from either M^i or Z^i. Now to evaluate a transition $X \rightarrow Y$, we define a metric m_{XY}. It is defined as $w_{XY} \times l_{XY}$, where w_{XY} is the fixed weight and l_{XY} is the 'likelihood' of the estimate. We assign higher weight to transitions of type $X \rightarrow X$ because they are more likely. The likelihood, l_{XY}, is based on the innovation, **z**(measurement)-**h**(prediction), and its associated covariance, S, obtained from the EKFs. We define the likelihood function as $N(\mathbf{z}\text{-}\mathbf{h}; \mathbf{0}, S)$, where $N(\mathbf{x}; \bar{\mathbf{x}}, \Sigma)$ is a Gaussian density function of **x** with mean $\bar{\mathbf{x}}$ and covariance Σ. At time step i, for each state X, we maintain S_X^i, sum of metrics of the previous states from which we have reached X^i. So, we select transitions $X_1 \rightarrow M$ and $X_2 \rightarrow Z$, where $X_1 = argmax_{a \in \{M,Z\}}\{S_a^i + m_{aM}\}$ and $X_2 = argmax_{a \in \{M,Z\}}\{S_a^i + m_{aZ}\}$. Then $S_M^{i+1} = S_{X_1}^i + m_{X_1 M}$ and $S_Z^{i+1} = S_{X_2}^i + m_{X_2 Z}$. We label these transitions at time step i as an ordered pair (X_1, X_2).

4.3. Obtaining the optimal sequence

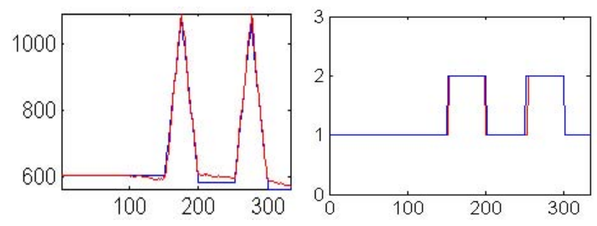
After completion of the sequence, we select the state, X, with higher S_X and trace back to get the optimal sequence of states. Thus one may argue that a post processing step is required. However, we show how we can do better. We keep high process noise variances so that the system rapidly adapts to a state transition. By testing many examples, we found that mostly transitions of type (X, X) occur because exactly one model holds at any time instant. Also, transitions (X, Y), $X \neq Y$, occur mostly whenever a true transition occurs and holds for 2-3 states in succession. Note that transitions (X, X) clearly imply that state X will be in the optimal sequence, thus we can output it on-line. When transitions of type (X, Y) occur, we wait until transitions of type (X, X) occur and output the pending sequence by tracing back. Thus, we can always output the state that will be present in the optimal sequence with a constant delay of say three frames without waiting for the complete sequence.

Without further effort, the system outputs the optimal state within a fixed short delay. However, we can also trade-off delay and an optimal solution. If a delay is not tolerable, we can output a sequence which will be sub-optimal but very close to the original solution. If we output the states without delay, error can only occur when the transitions (X, Y) occur. In such a case, we output the last known optimal state until we observe transitions (X, X). Since transitions (X, Y) are infrequent, the on-line output sequence is very close to the optimal solution. Also note that any errors are transient and do not propagate in the output.

5. Results

First, we apply our approach on simulation data and demonstrate its robustness by comparing the estimated pose and focal length with the (known) ground truth.

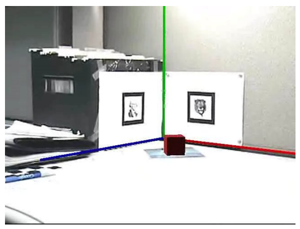
We used two sets of four coplanar feature points (each on a vertex of a square of side 0.9 units, 1 unit=10 cms). Random noise, uniformly distributed between $[-5, 5]$ pixels, was added to the 2D measurements of the features. Noise was also added to the initial focal length. The camera was set to rotate and translate along each axis. At selected times, camera was stopped and allowed to zoom in and out. Experiments with varied camera paths were run and the errors in estimated parameters were noted as depicted in figure 2. We observed an absolute value of camera position error per frame with mean 0.12 units and standard deviation 0.06 units. Figure 2(a) depicts that estimated focal length is close to the true value, with an error of less than 5%. Figure 2(b), shows the accuracy of our system in selecting the appropriate model (1 indicates *motion* and 2 indicates *zoom*).

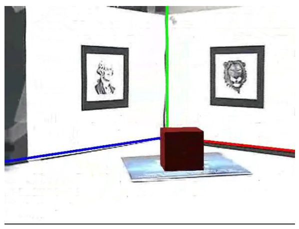


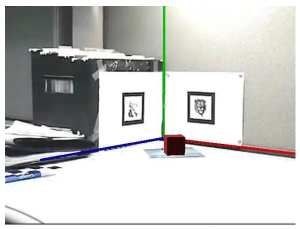
(a) Focal length (b) Sequence of states

Figure 2. Simulation results: blue and red indicate the ground truth and the estimate resp.

Now, we present results on real imagery. We used marker based tracking with the corners of the markers as our feature points. A JVC GR-D30U digital video camera was used to track two markers. We achieved a frame rate of 15 fps on a video sequence of resolution 640×480. To initialize the system, we calibrated the internal camera parameters with the Matlab Calibration Toolbox [1] and measured the approximate position of markers and the camera with a simple ruler. For visualizing the run-time estimated parameters, we augmented a virtual cube (red) in the video sequence on an image placed on the table. The image is only there to clearly show the stability of the cube position and plays no part in estimation. Figures 3(a) and 3(b) show snapshots before and after camera motion towards the targets. Figures 3(c) and 3(d) show snapshots before and after zooming in. The cube position on the image is consistent for both camera motion and camera zoom, which confirms that the estimated parameters are correct. Any jitter in estimation is primarily caused by high noise variances in the tracking model and not by incorrect selection of model.


(a) away from markers (frame 931)


(b) close to markers (frame 1333)


(c) no zoom (frame 931)

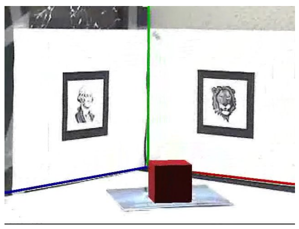
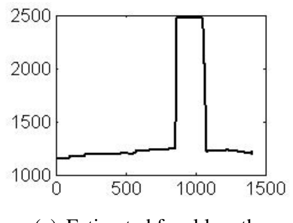

(d) zoomed in (frame 961)

Figure 3. Moving and zooming camera



(a) Estimated focal length

frames	camera
1-610	general motion
611-939	move away
940-961	zoom in
962-1140	small motion
1140-1170	zoom out
1171-1380	move close

(b) Camera parameters

Figure 4. Results on real data

Figure 4(a) shows the estimated focal length and figure 4(b) shows how the camera parameters varied over the video sequence. Thus the curve in figure 4(a) shows that zoom changes are estimated at the right times.

6. Conclusion and Future work

We presented a novel algorithm that estimates pose and focal length of a camera in real time using the Viterbi algorithm. The Viterbi algorithm provides the advantage of not biasing the state selection decision by the reprojection error of the current frame, but instead takes previous frames into account. Also, the filters remain pure and are not adulterated by mixing filter outputs.

The most important limitation of the current work is the assumption that motion and zoom cannot occur simultaneously. This is an area to explore further.

References

[1] *Camera Calibration Toolbox for Matlab*. http://www.vision.caltech.edu/bouguetj/calib_doc/.

[2] H. A. P. Blom and Y. Bar-Shalom. The interacting multiple model algorithm for systems with Markovian switching coefficients. *IEEE Transactions on Automatic Control*, 33:780–783, Aug 1988.

[3] W. S. Chaer, R. H. Bishop, and J. Ghosh. A Mixture-of-Experts Framework for Adaptive Kalman Filtering. *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, 27(3):452–464, Jun 1997.

[4] A. J. Davison. Real-Time Simultaneous Localisation and Mapping with a Single Camera. In *ICCV*, 2003.

[5] L. de Agapito, E. Hayman, and R. I. Hartley. Linear self-calibration of a rotating and zooming camera. In *CVPR*, 1999.

[6] G. D. Forney. The Viterbi algorithm. *Proceedings IEEE*, 61:268–278, Mar 1973.

[7] E. Hemayed. A survey of camera self-calibration. In *IEEE Conference on Advanced Video and Signal Based Surveillance*, pages 351– 357, 2003.

[8] B. Jiang, S. You, and U. Neumann. A Robust Tracking System for Outdoor Augmented Reality. In *IEEE Virtual Reality (VR)*, March 2004.

[9] E. Malis and R. Cipolla. Camera self-calibration from unknown planar structures enforcing the multiview constraints between collineations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(3):1268– 1272, Sep 2002.

[10] E. Mazor, A. Averbuch, Y. Bar-Shalom, and J. Dayan. Interacting Multiple Model Methods in Target Tracking: A Survey. *IEEE Transactions on Aerospace and Electronic Systems*, 34(1):103–123, Jan 1998.

[11] J. Mendelsohn and K. Daniilidis. Constrained Self-Calibration. In *CVPR*, pages 2581–2587, 1999.

[12] G. Simon and M.-O. Berger. Registration with a Moving Zoom Lens Camera for Augmented Reality Applications. In *ECCV*, pages 578–594, 2000.

[13] P. Sturm. Self-Calibration of a Moving Zoom-Lens Camera by Pre-Calibration. *Image and Vision Computing*, 15(8):583–589, Aug 1997.

[14] P. Sturm. Critical Motion Sequences for the Self-Calibration of Cameras and Stereo Systems with Variable Focal Length. *Image and Vision Computing*, 20(5-6):415–426, Mar 2002.