Robust 3D Hand Tracking for Human Computer Interaction

Victor Adrian Prisacariu
Department of Engineering Science
University of Oxford
victor@robots.ox.ac.uk

Ian Reid
Department of Engineering Science
University of Oxford
ian@robots.ox.ac.uk

***Abstract*— We propose a system for human computer interaction via 3D hand movements, based on a combination of visual tracking and a cheap, off-the-shelf, accelerometer. We use a 3D model and region based tracker, resulting in robustness to variations in illumination, motion blur and occlusions. At the same time the accelerometer allows us to deal with the multimodality in the silhouette to pose function. We synchronise the accelerometer and tracker online, by casting the calibration problem as a maximum covariance problem, which we then solve probabilistically. We show the effectiveness of our solution with multiple real-world tests and demonstration scenarios.**

I. Introduction

With the advent of processing power and the internet, the computer has become a gaming and media hub and part of the centre of our social life. Still, interaction with it is very limited to mouse and keyboard. This paper describes a natural and intuitive way of interacting with a virtual environment, by employing a vision and inertial sensor-based system, to recover the *3D non-articulated* pose of a human hand in real time, while still creating a cost effective system.

Fast and reliable hand tracking has usually been achieved using glove-based approaches. For example, the ShapeHand data glove [8] uses accelerometers, gyroscopes and flex sensors, while in [15] the authors use a specially coloured glove (but no sensors) to obtain the full 3D articulated pose of a hand. Both these systems show very good results. However the ShapeHand data glove is both intrusive and expensive, and the authors in [15] limit themselves to clean, clutter-free environments.

Our method combines a vision based 3D tracker, similar to the one presented in [11], with a single off-the-shelf accelerometer. This gives it the following significant advantages over previous hand tracking work: (i) it works in real time and in real-world environments (cluttered and with large amounts of motion blur and occlusions) and (ii) it is much less intrusive when compared to glove-based approaches.

Vision based 3D hand tracking can be split into *model-based* and *appearance-based* tracking [10]. A model-based technique has a number of important elements: a 3D model, a set of image features, an error function and a non-linear optimisation algorithm. At every frame the set of features is extracted from the image and an error function is minimised. This energy function relates the pose of the 3D model to the features extracted from the image. The features may be simple (edges, points) or complex (3D depth information). Most often edges and colour are combined to form a silhouette. The 3D model may be anything between a coarse geometric model and a detailed 3D reconstruction (including shading and texture information) of the user's hand. For example [12] uses a model built only from quadratics while in [4] both texture and lighting are used. To form the error function the 3D model may be projected down or the image features may be projected up. The minimisation may lead to one or to several hypotheses. Finally sometimes the hand motion dynamics are used, therefore predicting the pose of the next frame from the poses at the current frame and previous frames.

Appearance-based algorithms propose to learn a direct mapping between image features and pose, so only two steps are required when a new frame is presented: extract the image features and obtain the pose from the feature–pose mapping. These methods need no initialisation and are (theoretically) faster, because most of the processing effort is put into offline training, rather than in the online phase. The mapping between feature and pose can either be learnt inside a database in [1], by using a tree-based filter in [12] or via a Relevance Vector Machine in [3]. These methods, while maybe producing good results in a controlled environment where the segmentation is very good, do not perform well in real-world scenarios.

Our visual tracker is model and region based: we use simple 3D meshes as models (so no texture or lighting information) and a region based energy function. This energy function uses the implicit representation of the contour of the projection of the known 3D model, by embedding it inside a level set function. We maximise the posterior per-pixel probability of foreground and background membership as a function of pose, directly, bypassing any separate segmentation phase. We represent the region statistics by variable bin size, colour histograms and adapt these online. Since we are using only the silhouette of the projection, the energy function will be multimodal. To help deal with the ambiguities, we place an accelerometer on the hand.

The remainder of this article is structured as follows: in Section II we present the 3D tracker, in Section IV we detail the way in which we extract the orientation from the accelerometer and correlate it with the tracker and in Section V we show various results. We conclude in Section VI.

II. 3D Tracker

This section presents the details of our 3D tracker, similar to the one published in [11]. Here we review that work, then in subsections II-D, II-E and II-F we describe several improvements. We first set up the notation. We then show our

3D model and detail the tracking algorithm and the region statistics.

A. Notation

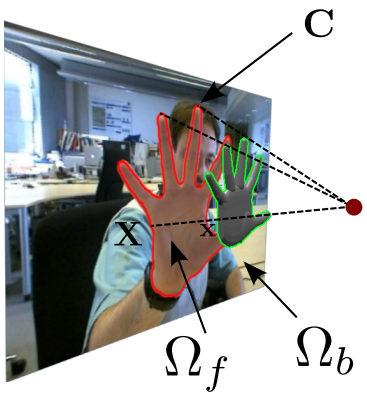

Fig. 1. Notation – the contour around the visible part of the 3D model (green) and its corresponding projection **C** (red), the foreground region Ω_f and the background region Ω_b, a 2D point **x** on the contour and its corresponding 3D point in the object coordinate frame, **X**

Let the image be denoted by **I**, and the image domain by $\Omega \subset \mathbf{R}^2$ with the area element $d\Omega$. An image pixel $\mathbf{x} = [x, y]$ has a corresponding image value $\mathbf{I}(\mathbf{x}) = \mathbf{y}$ (in our experiments a RGB value), a corresponding 3D point $\mathbf{X} = [X, Y, Z]^T = \mathbf{R}\mathbf{X_0} + \mathbf{T} \in \mathbf{R}^3$ in the camera coordinate frame and a point $\mathbf{X_0} = [X_0, Y_0, Z_0]^T \in \mathbf{R}^3$ in the object coordinate frame. **R** and **T** are the rotation matrix and translation vector representing the unknown pose and are parametrised by 7 parameters (4 for rotation and 3 for translation), denoted by λ_i.

We assume the intrinsic parameters of the camera to be known. Let (f_u, f_v) be the focal length and (u_o, v_o) the principal point of the camera.

The contour around the visible part of the object in 3D (marked with green in Figure 1) projects to the contour **C** in the image (marked with red). We embed **C** in the zero level-set function $\Phi(\mathbf{x})$. The contour **C** also segments the image into two disjoint regions: foreground denoted by Ω_f and background denoted by Ω_b. Each region has its own statistical appearance model $P(\mathbf{y}|M), M \in \{M_f, M_b\}$.

Finally, by $H_e(x)$ we denote the smoothed Heaviside step function and by $\delta_e(x)$ the smoothed Dirac delta function.

B. 3D Model

As stated in the introduction, we use 3D triangle meshes as models for the tracker. Figure 2 shows the 3D model we employed throughout this paper. This model was obtained by combining several pictures of the hand using iModeller 3D [6], into a coherent 3D model. Note we only use a rough approximation of the hand. Our algorithm does not need the model to be very accurate in order to produce good results. However, the error in the recovered pose is proportional to the difference between the model and the tracked object.

C. Tracking

The algorithm used for tracking in this work is similar to the PWP3D algorithm of [11], whose biggest advantage

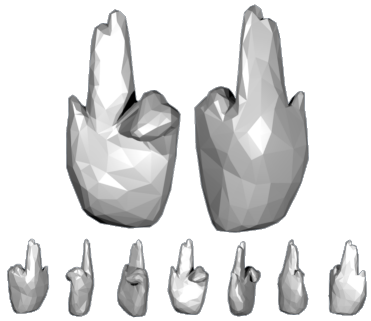

Fig. 2. 3D Model

is its robustness to motion blur, occlusions and cluttered background. The main differences are the different formulations for the region statistics (using temporal consistency and variable bin size histograms) and their online adaptation.

PWP3D maximises the log (posterior) probability of the shape of the contour (of the projection of the 3D model), encoded by an embedding function Φ, given the image data:

$$P(\Phi|\Omega) = \prod_{\mathbf{x}\in\Omega} \Big(H_e(\Phi) P_f + \big(1 - H_e(\Phi)\big) P_b \Big) \Rightarrow \quad (1)$$

$$E(\Phi) = -\sum_{\mathbf{x}\in\Omega} \log \Big(H_e(\Phi) P_f + \big(1 - H_e(\Phi) P_b\big) \Big) \quad (2)$$

where P_f and P_b are the posterior probabilities respectively:

$$P_f = \frac{P(\mathbf{y}|M_f)}{\eta_f P(\mathbf{y}|M_f) + \eta_b P(\mathbf{y}|M_b)} \quad (3)$$

$$P_b = \frac{P(\mathbf{y}|M_b)}{\eta_f P(\mathbf{y}|M_f) + \eta_b P(\mathbf{y}|M_b)} \quad (4)$$

with η_f and η_b being the areas of the foreground and background regions respectively:

$$\eta_f = \sum_{\mathbf{x}\in\Omega} H_e\big(\Phi(\mathbf{x})\big) \quad \eta_b = \sum_{\mathbf{x}\in\Omega} 1 - H_e\big(\Phi(\mathbf{x})\big) \quad (5)$$

We differentiate this energy function with respect to λ_i:

$$\frac{\partial E}{\partial \lambda_i} = P(\Phi|\Omega) \sum_{\mathbf{x}\in\Omega} \frac{P_f - P_b}{H_e(\Phi) P_f + \big(1 - H_e(\Phi)\big) P_b} \frac{\partial H_e(\Phi)}{\partial \lambda_i} \quad (6)$$

$$\frac{\partial H_e\big(\Phi(x,y)\big)}{\partial \lambda_i} = \frac{\partial H_e}{\partial \Phi} \left(\frac{\partial \Phi}{\partial x} \frac{\partial x}{\partial \lambda_i} + \frac{\partial \Phi}{\partial y} \frac{\partial y}{\partial \lambda_i} \right) \quad (7)$$

Every 2D point on the contour of the projection of the 3D model has at least one corresponding 3D point **X**, for which:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -f_u \dfrac{X}{Z} - u_0 \\ -f_v \dfrac{Y}{Z} - v_0 \end{bmatrix} \quad (8)$$

Therefore:

$$\frac{\partial x}{\partial \lambda_i} = -f_u \frac{\partial}{\partial \lambda_i} \frac{X}{Z} = -f_u \frac{1}{Z^2} \left(Z \frac{\partial X}{\partial \lambda_i} - X \frac{\partial Z}{\partial \lambda_i} \right) \quad (9)$$

The differential $\frac{\partial y}{\partial \lambda_i}$ is analogue to $\frac{\partial x}{\partial \lambda_i}$ while $\frac{\partial X}{\partial \lambda_i}$, $\frac{\partial Y}{\partial \lambda_i}$ and $\frac{\partial Z}{\partial \lambda_i}$ follow trivially by differentiating $\mathbf{X} = \mathbf{R}\mathbf{X_0} + \mathbf{T}$ with respect to λ_i. For more details the reader is refereed to [11].

D. Temporal Consistency

The probability of a pixel being foreground or background should not change instantly i.e. if a pixel was foreground at time $t-1$, the probability of it being foreground at time t should be higher than the probability of it being background. This can be written formally, using a recursive Bayes filter:

$$P(M_j^t|\mathbf{Y}^t) = \frac{P(\mathbf{y}^t|M_j^t, \mathbf{Y}^{t-1})P(M_j^t|\mathbf{Y}^{t-1})}{P(\mathbf{y}^t|\mathbf{Y}^{t-1})} \tag{10}$$

where $M_j,\ j \in \{f,b\}$ is the foreground/background model, $\mathbf{y}^t$ the value of pixel $\mathbf{y}$ at time t and $\mathbf{Y}_t = [\mathbf{y}^t, \mathbf{y}^{t-1}, \ldots]$ the values of pixel $\mathbf{y}$ up to time t.

Assuming conditional independence we can write:

$$\begin{aligned} P(M_j^t|\mathbf{Y}^t) &= \frac{P(\mathbf{y}^t|M_j^t)P(M_j^t|\mathbf{Y}^{t-1})}{P(\mathbf{y}^t|\mathbf{Y}^{t-1})} = \\ &= \frac{P(\mathbf{y}^t|M_j^t)\sum_{i\in\{f,b\}}\left(p(M_i^t|M_i^{t-1})p(M_i^{t-1}|\mathbf{Y}^{t-1})\right)}{P(\mathbf{y}^t|\mathbf{Y}^{t-1})} \end{aligned} \tag{11}$$

where:

$$P(\mathbf{y}^t|\mathbf{Y}^{t-1}) = \sum_{i\in\{f,b\}} P(\mathbf{y}^t|M_i^t)P(M_i^t|\mathbf{Y}^{t-1}) \tag{12}$$

The recursion is initialised by $P(M_0) = P(M_j)$.

To estimate the per pixel values for $P(M_f^t|M_f^{t-1})$ and $P(M_b^t|M_b^{t-1})$ we define two classes of pixels γ_f and γ_b representing the unchanged (between the previous and current frame) foreground and background pixels respectively. The following posteriors can be obtained:

$$\begin{aligned} P(M_f^t|M_f^{t-1}) =& P(\gamma_f|\mathbf{y}^t, \mathbf{y}^{t-1}) = P(\mathbf{y}^t, \mathbf{y}^{t-1}|\gamma_f)P(\gamma_f) = \\ =& P(\mathbf{y}^t|M_f)P(\mathbf{y}^{t-1}|M_f)P(\gamma_f) \\ P(M_b^t|M_b^{t-1}) =& P(\gamma_b|\mathbf{y}^t, \mathbf{y}^{t-1}) = P(\mathbf{y}^t, \mathbf{y}^{t-1}|\gamma_b)P(\gamma_b) = \\ =& P(\mathbf{y}^t|M_b)P(\mathbf{y}^{t-1}|M_b)P(\gamma_b) \end{aligned} \tag{13}$$

with

$$P(\gamma_f) = \frac{\zeta_f}{\zeta} \quad P(\zeta_b) = \frac{\zeta_b}{\zeta} \tag{14}$$

where ζ is the total number of pixels in the image, ζ_i, $i \in \{f,b\}$ is the number of pixels that didn't change type between the previous and current frames i.e. had probability of foreground/background bigger than that of background/foreground and the pixel colour didn't change by more than a fixed threshold.

Here we only consider $\mathbf{Y}^t = \mathbf{y}^t$, which means that the current value the foreground/background posterior depends only on the value at the previous time step.

E. Variable Bin Size Histograms

In [11] and [2] the authors use a standard appearance model consisting of two 32 bin histograms (one for foreground and one for background).

We have empirically observed that the dynamic response of the sort of consumer grade webcams is such that bright colours will have a higher variance to changing lighting conditions than dark colours. Standard histograms use the same number of bins for both bright and dark colours, which leads to undersampling for dark colours and oversampling for bright colours. Put differently, if we build the histogram of an object of bright constant colour, under uneven lighting conditions, rather than having a single peak, the histogram will tend to flatten. If the object is dark we will end up having a single peak inside the histogram, when it should actually be flatter. One solution to this problem would be to separate brightness from colour, by using a different colour space (HSV, HSL, etc.). This would imply an extra processing step and these colour spaces have singularities at dark colours.

Our solution is to keep using the RGB colour space, but vary the number of bins in the histogram, according to the brightness of the pixel: bright colours will use fewer bins while dark colours use more bins. In our implementation we use 4 histograms (of 8, 16, 32, 64 bins per channel).

F. Online Adaptation

In [11] the authors updated the histogram only when the minimisation of the energy function had converged. This does not guarantee that the histograms will not be corrupted, because the minimisation of the energy function might have converged to an incorrect pose. Here we use the distance transform as a measure of contour uncertainty: points that are far from the contour, and inside the projection will most likely be foreground, while pixels far from the contour and outside the projection will most likely be background. We evaluate the distance transform in a band around the contour so we update the foreground histogram only with the pixels outside the band but inside the contour and the background one with pixels outside the band and outside the contour. This might mean that we do not update with some pixels that actually are foreground / background but it allows us to use much higher adaptation rates while still ensuring stable tracking. In [11] the adaptation rate was usually less than 1%. Here we can easily go as high as 5% learning rate without the histograms getting corrupted. This allows more rapid changes in appearance which in turn produces more reliable tracking.

III. Implementation and Timings

To be useful in real-world applications, a 3D hand tracker has to work in real time. To this end we used a highly-parallel GPU implementation similar to that of [11]. The energy function from Equation 2 is minimised using gradient descent, for expedience and ease of implementation. Each gradient descent iteration proceeds as follows:

- The 3D model is rendered using the current estimate of the pose.

- The contour of the rendering and its exact signed distance transform are computed.
- The partial derivatives of the energy function with respect to the pose parameters are calculated.
- A step change is made, in the direction of the gradient. The step size is fixed.

In [11] the authors parallelised everything except the 3D rendering step. This ended up being the bottleneck in their implementation, because the rendering and transfer to the GPU operations took as much as the rest of the processing (distance transform and energy function evaluation). An OpenGL based implementation would have actually been slower (at least for models with few triangles) [11].

Here we developed our own NVIDIA CUDA-based 3D rendering engine. We chose to use a separate thread for each triangle in the model. For the ZBuffer implementation we use global memory write atomic operations to avoid any possible read-write-modify hazard. As a result we are able to achieve speeds of **2-3ms per iteration** (compared to 4-6 in [11]). Reliable tracking can be done at **15-20 frames per second** when running a fixed number of iterations. Convergence is usually faster than the fixed number of iterations so the speed increases to **20-25 frames per second** when checking for convergence. In this paper we used a NVIDIA GTX 285 video card and an Intel Xeon E5420 CPU.

IV. Accelerometer

Fig. 3. Silhouette - Pose ambiguities

The function mapping silhouette to pose is multimodal: very different poses will end up generating the very similar silhouettes, as shown in Figure 3. One solution to overcome this problem would be to consider multiple hypotheses at every frame (i.e. use a particle filter). This would make real time performance difficult to achieve. Our solution is to augment the 3D tracker with data from a simple off-the-shelf accelerometer, mounted on the hand. In this section we present our method of extracting data from the accelerometer and of calibrating the accelerometer and tracker.

A. Extracting Hand Orientation from the Accelerometer

We use the Freescale ZSTAR3 kit, consisting of a sensor board and an USB dongle. The sensor board is mounted on the top of the hand (Figure 4) or on the palm of the hand, and it transmits acceleration data to the USB dongle.

Most works (for example [5]) extract the orientation from the accelerometer by computing three Euler angles, pitch (denoted with ρ), roll (denoted with ϕ) and the angle of the Z axis relative to the gravity) (denoted with θ). Note that yaw cannot be measured, because a change in yaw does not alter

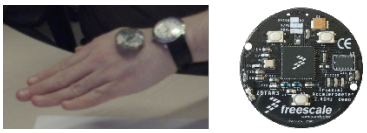

Fig. 4. Hand with accelerometer sensor board

the gravity measurements of the accelerometer. Therefore:

$$\rho = \arctan \frac{A_x}{\sqrt{A_y^2+A_z^2}} \quad \phi = \arctan \frac{A_y}{\sqrt{A_x^2+A_z^2}} \quad \theta = \arctan \frac{\sqrt{A_x^2+A_y^2}}{A_z} \tag{15}$$

The problem with this approach is that it suffers from gimbal lock. Our solution was to use the axis-angle representation for rotation, by interpreting the acceleration data from the accelerometer as a (normalised) force vector. Assuming a preset reference r, the accelerometer force vector a, θ the axis and Θ the angle, we can write:

$$\Omega = a \times r \quad \theta = \arccos(a \cdot r) \tag{16}$$

We always define the reference as the acceleration vector at the previous frame, which means we use the accelerometer to measure rotation between consecutive frames.

B. Accelerometer - Tracker Calibration

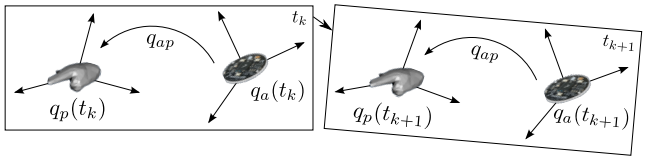

Fig. 5. Rotation quaternions and coordinate systems for the tracker and the accelerometer

Both the 3D tracker and accelerometer produce rotation quaternions, in the reference coordinate system (which in our case is the camera coordinate system). The accelerometer can be positioned in many ways on the hand which means that the two quaternions will generally be different. The difference will be constant because the accelerometer does not move with respect to the hand. The calibration process attempts to recover this difference, i.e. the quaternion that rotates the accelerometer coordinate system into that of the tracker. Figure 5 shows the definitions of the coordinate systems: q_a is the rotation quaternion of the accelerometer, in the world coordinate system, q_p is the rotation quaternion of the tracker, in the world coordinate system and q_{ap} is the quaternion which rotates one into the other:

$$q_a q_{ap} = q_p \tag{17}$$

and t_k and t_{k+1} are the frames at the current and next time step.

We assume that there is no translation difference between the tracker and accelerometer coordinate systems. While this might not be true all the time, the differences will be small and the tracker is iterated after it is combined with the accelerometer data, so any errors will be compensated there.

Several visual-inertial calibration methods exist. Generally this problem is solved as one of linear or nonlinear least squares [7], [9]. For example in [9] the authors start from the fact that q_{ap} should be constant and minimise:

$$q_{pa} = \arg\max_{q}\{q^T\Big(\sum_{i=1}^{t}\overline{Q}^T_{\Delta q_a}(t)Q_{\Delta q_p}(t)\Big)q\} \qquad (18)$$

where $q_{pa} = q_{ap}^{-1}$, Q and $\overline{Q}$ are the matrix representations of Δq_a and Δq_p, and:

$$\Delta q_p = q_p^{-1}(t_1)q_p(t_2) \quad \Delta q_a = q_a^{-1}(t_1)q_a(t_2) \qquad (19)$$

Our solution assumes the user moves the hand in a predefined pattern (up, down, left, right), to pass through the entire range of valid accelerometer values. We compute q_{ap} at every frame and then the eigenvectors of the covariance matrix built by stacking together all the q_{ap} values. Since eigenvectors norm to 1, all eigenvectors will be valid quaternions. The calibrated quaternion, relating q_p to q_a, is then the eigenvector with the maximum eigenvalue i.e. the one that maximises the variance among all the values of q_{ap}. This process could also be seen as running Principal Component Analysis (PCA) on the quaternion dataset.

Least squares based methods and PCA assume all data points to be i.i.d. and corrupted by i.i.d. noise. However the noise in our case is not i.i.d. A quaternion describes a single rotation, making the quaternion parameters interlinked. The noise in the quaternion parameters can therefore be approximated more accurately by spherical noise. Therefore in this work, rather than running PCA on the quaternion dataset we run Probabilistic PCA [14], which considers noise as being spherical Gaussian.

C. Accelerometer - Tracker Integration

Ideally we would operate with visual data alone, but for the reasons we have already discussed, a limited amount of additional information such as that provided by a cheap accelerometer can resolve visual ambiguities. Here we have taken an expedient route that uses the accelerometer data to aid the starting point for the visual tracker iterations; this increases the speed and reliability of convergence of the visual tracker as well as overcoming many of the visually ambiguous poses, but places little faith on the actual output of the accelerometer (in line with our expectation that overall it is not especially reliable). More specifically, we begin with the previous pose estimate from the visual tracker, update it with the differential motion obtained from the accelerometer, and use this as the starting point for the visual tracker's iterations. We have found that the differential output of the accelerometer is aided by filtering the raw acceleration data prior to computing the change in motion. We do this with a Kalman Filter that assumes the acceleration is constant.

V. Results

In this section we present the results of applying our algorithm to a multitude of video sequences. We also show the benefits brought by of each part of the algorithm (using the accelerometer, using the variable bin size histograms, imposing temporal consistency for the posteriors). We then compare our calibration method to the standard least-squares approach of [9]. Finally we detail a proof-of-concept user interface based on our 3D tracker.

With Figure 6, we begin by showing an example where the accelerometer helps resolve the silhouette - pose ambiguities, making tracking more reliable and accurate. In this example the hand was moving up - down and then left - right, very quickly. The figure also shows our system working in a cluttered environment. Each motion took up to 5 frames at 30 fps video (so 1/6 seconds), followed by a short (around 60 frames) period of stability. We allowed a fixed number of iterations at each frame. For the first frame, where there are no ambiguities, the results are similar, with and without the accelerometer. At the second frame it can already be seen that the visual tracker alone is showing some error in the recovered pose. This is happening because the visual tracker does not have time to converge. The accelerometer helps it by recovering most of the rotation, allowing it to converge to a much more accurate pose. At the next frame the pose recovered by the visual tracker is very far from the correct hand pose. The straight on view of the hand is an ambiguous pose i.e. a change in hand rotation will not alter the silhouette. The visual tracker was already in the wrong position at the previous frame and when the hand came out of the ambiguous pose, the difference in rotation between the pose known by the tracker and the correct pose was at least 20 degrees. The accelerometer helped deal with the ambiguities. The same behaviour can be noticed in the left - right rotation (next 4 frames). The visual tracker alone can't deal with the ambiguities in the silhouette - pose function and with the high speed of the motion.

Next we show the usefulness of variable bin size histograms and temporal consistency for the posteriors. In Figure 7 we plot $P_f - P_b$ where $P_f - P_b > 0$ with P_f and P_b defined according to Equations 3 and 4. This is a measure of how effective the regions statistics are at separating foreground from background. Figure 7 shows an extreme case where the variable bin size histograms help exclude pixels from the foreground and achieve better pose tracking results. When using fixed bin size histograms a lot of the shadow of the hand on the table ends up being considered foreground. Adding temporal consistency to the posteriors improves the results further. Note that we do not use any kind of background subtraction in any of our tests.

In Figure 8 we compare our calibration method against the one of [9]. Our method is probabilistic (we use probabilistic PCA to compute the calibration quaternion) while the one presented in [9] uses least squares. We start with two rotations over 500 frames which produce two sets of 500 quaternions $q_1(t)$ and $q_2(t)$. The offset between the two rotations is constant and known. We apply uniformly distributed random noise, at each frame and on each axis, within the $[-5,5]$ degree range. We use both methods to recover the offset between the two rotations q_c, where $q_2(t) = q_1(t)q_c$. At this noise level both method yield similar results. We then add a secondary source of noise, a uniformly

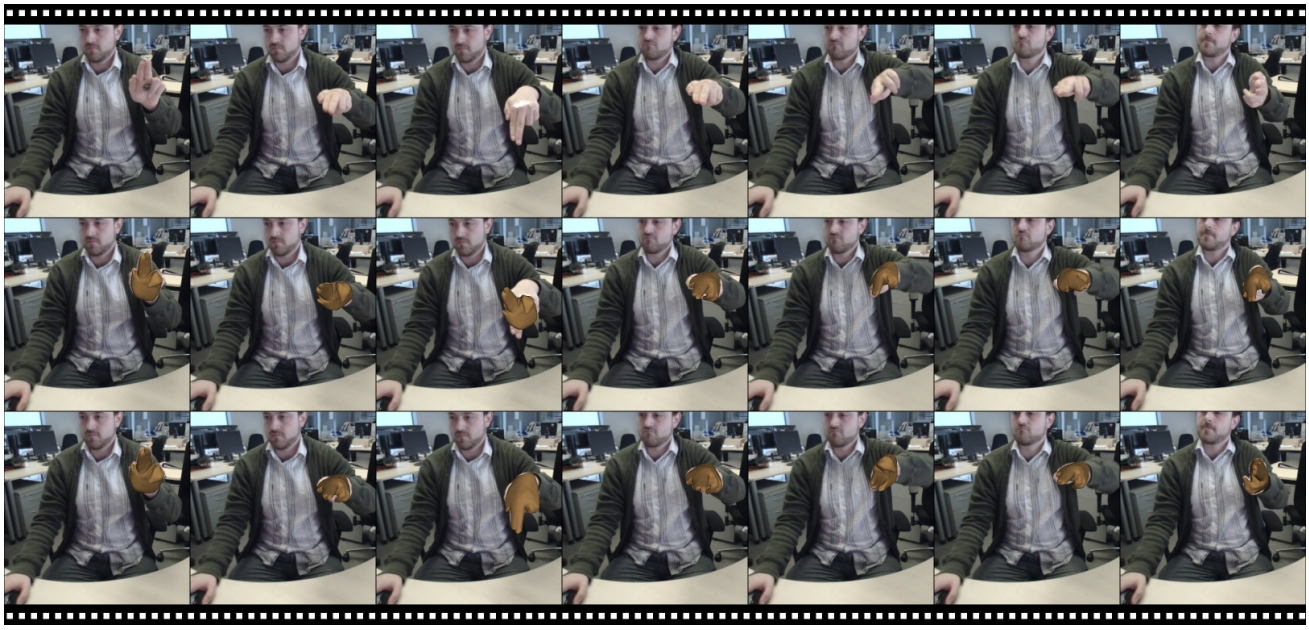

Fig. 6. Filmstrip showing tracking results with and without the accelerometer. The visual tracker alone cannot deal with ambiguities in the silhouette (where a change in pose does not alter the silhouette). The accelerometer provides enough information to discriminate between the ambiguous poses.

distributed random noise, at every 60th frame and on each axis, within the $[-30, 30]$ degree range. At this noise level our method can successfully obtain q_c while the method of [9] fails.

In Figures 10 and 11 we show filmstrips from two proof-of-concept examples of how our hand tracker could be used as part of a human-computer interface. The user first has to mark a plane. Here marking the plane is done by moving the hand along the edges of a rectangular board (Figure 9). The board (or its colour) does not influence the hand tracking in any way. We chose this shape as a visual cue to the user and because it provides an easy and structured way of displaying information. The motion made by the user could have been different (i.e. a circle). The plane does not need to lie on a real surface i.e. it could have been marked in mid air. In order to extract the plane position we used RANSAC on the 3D positions of the tip of the index finger (which we computed using the pose of the entire hand and the 3D model). Once the user has marked the plane, the user is able to "click" the virtual surface. A click is signalled when this distance between the tip of the index finger and the plane is 0 or negative. We display a menu interface on the virtual surface, which the user navigates by clicking. The UI is displayed on the computer screen but could also be projected on the board or shown through a pair of AR glasses.

In the first example (Figure 10) the user is first prompted to place a DVD on the marked surface. We used two video game DVDs as examples. The system identifies the DVD and displays relevant information on the virtual surface. Recognition is achieved by using the TinEye API [13] and the information is extracted from sources such as Amazon and Wikipedia. Depending on the DVD placed on the virtual surface the user is also shown a 3D object. In the case of the first DVD the user is shown a model of the main starship from the game. In the case of the second DVD the user is shown the model of planet Mars. The user can interact with the 3D object. In the case of the first DVD the user can place the hand under the starship which then "climbs" on top of the hand. The user can move the ship by moving the hand. In the case of the second DVD the user can rotate the model of planet Mars by moving the hand.

In the second example (Figure 11) the user first draws a 2D closed shape on the virtual surface. The 2D shape can be turned into a 3D object by clicking it and raising the hand above the virtual surface. Just like in the previous example, the user can also move the 3D object by placing the hand under it and waiting for it to "climb" on top of the hand.

VI. Conclusions

In this article we proposed a system for real-time 3D human computer interaction, via the use of 3D hand tracker. Our system does not need specially coloured gloves or markers and can work in cluttered environments. It combines a region based visual tracker with a a single off-the-shelf accelerometer, making it robust to motion blur and occlusions while allowing it to differentiate between ambiguous hand poses. Variable bin size histograms and temporal consistency for the foreground/background membership probabilities help improve the foreground/background separation when subjected it changing lighting conditions.

A straightforward extensional to our work is the ability to track multiple hands, from multiple cameras. A more compelling extension would be the ability to track the 3D pose of individual fingers, along with that of the full hand. In our ongoing research we are looking at combining our tracker with a prelearned appearance-based mapping between

Fig. 7. Filmstrip showing the per pixel difference between the foreground and background probabilities $P_f - P_b$, where $P_f - P_b > 0$, and the resulting recovered pose, when using fixed bin size histograms (rows 2 and 3), variable bin size histograms (rows 4 and 5) and variable bin size histogram combined with temporal consistency (rows 6 and 7).

silhouette and pose, which would make 3D articulated tracking tractable.

REFERENCES

[1] V. Athitsos and S. Sclaroff. Estimating 3d hand pose from a cluttered image. In *CVPR 2003*, pages II–432–9 vol.2, 2003.

[2] C. Bibby and I. Reid. Robust real-time visual tracking using pixel-wise posteriors. In *ECCV 2008*, pages 831–844, 2008.

[3] T. E. de Campos and D. W. Murray. Regression-based hand pose estimation from multiple cameras. *CVPR 2006*, 1:782–789, 2006.

[4] M. de La Gorce, N. Paragios, and D. J. Fleet. Model-based hand tracking with texture, shading and self-occlusions. *CVPR 2008*, 0:1–8, 2008.

[5] Freescale. Tilt sensing using linear accelerometers, an3461.

[6] U. GmbH. imodeller 3d professional. 2009.

[7] J. D. Hol, T. B. Schön, and F. Gustafsson. Modeling and calibration of inertial and vision sensors. *IJRR*, 29(2-3):231–244, 2010.

[8] M. Inc. Shapehand data glove, 2009.

[9] P. Lang and A. Pinz. Calibration of hybrid vision / inertial tracking systems. In *InverVis 2005*, April 2005.

[10] V. I. Pavlovic, R. Sharma, and T. S. Huang. Visual interpretation of hand gestures for human-computer interaction: A review. *IEEE T-PAMI*, 19(7):677–695, 1997.

[11] V. Prisacariu and I. Reid. Pwp3d: Real-time segmentation and tracking of 3d objects. In *BMVC 2009*, September 2009.

[12] B. Stenger, A. Thayananthan, P. H. S. Torr, and R. Cipolla. Model-based hand tracking using a hierarchical bayesian filter. *IEEE T-PAMI*, 28:1372–1384, 2006.

[13] TinEye. Tineye commercial api http://www.tineye.com, 2010.

[14] M. E. Tipping and C. M. Bishop. Probabilistic principal component analysis. *JRSS*, 61:611–622, 1999.

[15] R. Y. Wang and J. Popović. Real-time Hand-tracking with a Color Glove. *ACM TOG*, 28(3), 2009.

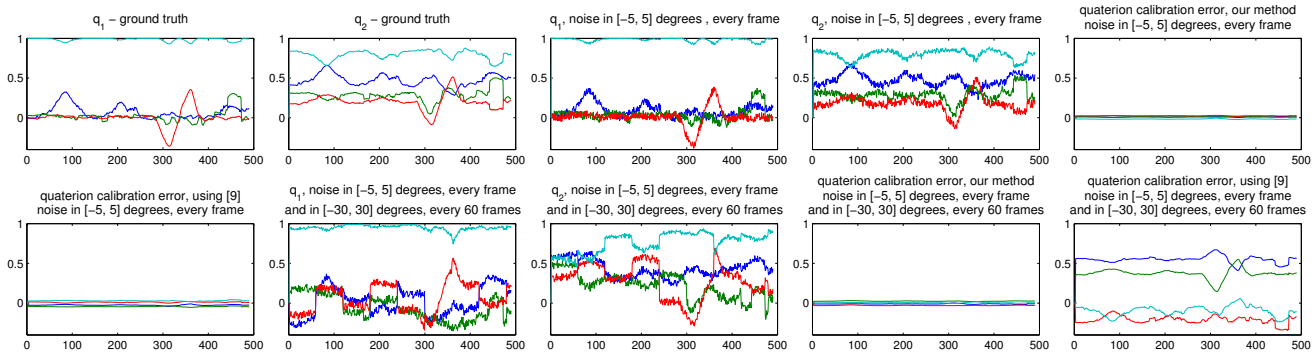

Fig. 8. Charts comparing our calibration method to the one from [9]. We use two motions (denoted with q_1 and q_2) over 500 frames as our ground truth. The calibration quaternion between these two is known and constant. We first add a small amount of noise noise at each frame (a randomly size angle, between -5 and 5 degrees, at every frame) and compute the calibration quaternion using the two methods. Both methods are able to recover the calibration quaternion. We then keep this source of noise and add another, bigger, one every 60 frames (a randomly sized angle, between -30 and 30 degrees). Our method is still able to compute the correct calibration quaternion, while the one from [9] fails.

Fig. 9. Example human-computer interface based on our tracker – the user first marks a plane by moving the hand along the edges of the board.

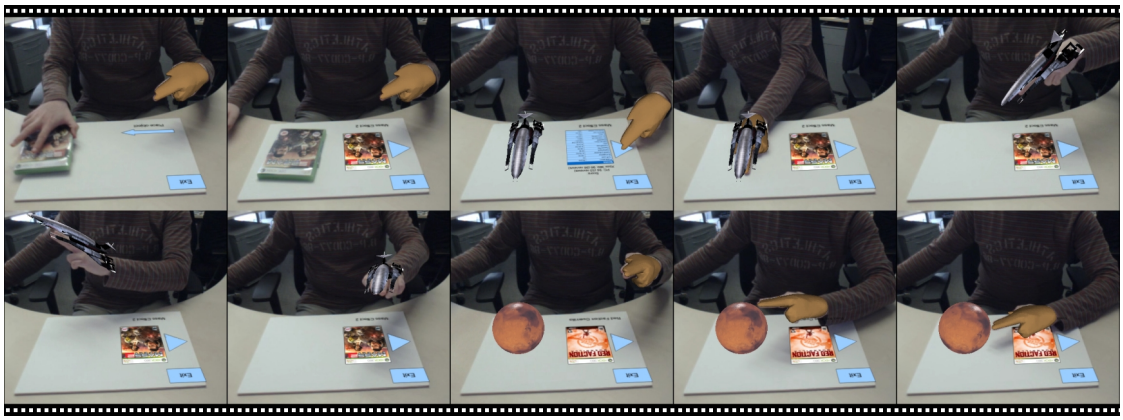

Fig. 10. Example human-computer interface based on our tracker – the user places a DVD on the surface, prompting the system to show facts about the DVD and a 3D object related to it, which he/she can manipulate with the hand. In the case of the first DVD the user sees and can be pick up and move a 3D model of a starship. In the case of the second DVD, the user sees and can rotate a model of planet Mars.

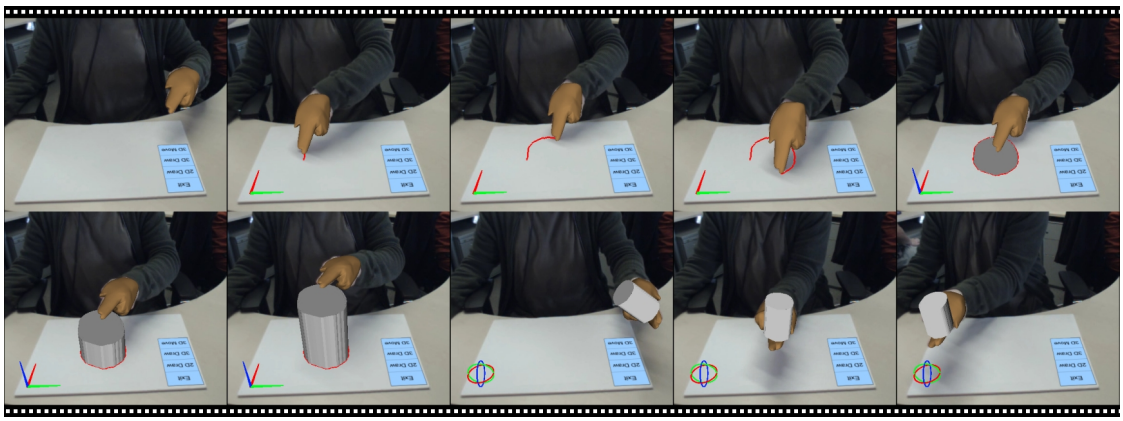

Fig. 11. Example human-computer interface based on our tracker – the user can click and draw on the 2D surface. He/she can transform the 2D contour into a 3D object by clicking inside the contour and lifting the hand. The user can also pick up and move the object.