Efficient Tracking of 3D Objects Using Multiple Orthogonal Cameras

Enrique Muñoz
Dep. Sistemas Informáticos y Computación,
U. Complutense de Madrid, Spain
http://www.dia.fi.upm.es/~pcr

José M. Buenaposada
Dep. Ciencias de la Computación,
U. Rey Juan Carlos, Spain
http://www.dia.fi.upm.es/~pcr

Luis Baumela
Dep. de Inteligencia Artificial,
U. Politécnica de Madrid, Spain
http://www.dia.fi.upm.es/~pcr

Abstract

We introduce a multi-view direct procedure for efficiently tracking 3D objects. It is an extension of Hager and Belhumeur's factorisation approach to the case of three-dimensional objects and multi-camera setup. By tracking a 3-D object we mean to estimate the pose and location of the object through a video sequence. A novel parameterisation of the object texture allow us to compute the Jacobian that emerges in the minimisation in a way it has a large constant part. The pixels viewed by each camera determine the rows of the Jacobian used for tracking. We perform qualitative and quantitative experiments confirming the validity of the approach.

1 Introduction

Efficiently tracking 3D objects has been a topic of interest in Computer Vision for years, with applications in augmented reality, advanced human-machine interfaces and robotics. Tracking is achieved by estimating the parameters of a function representing the relative position between camera and object. This can be achieved by matching a sparse collection of features (feature-based approaches) or by directly minimising the difference in image intensity values (direct approaches). The main advantage of feature-based approaches is the possibility of working with very large inter-frame motion [9]. This make them best suited for target detection or for recovery after a complete loss. Direct approaches assume that inter-frame motion is small, as is the case in video sequences. Tracking is usually posed as a Gauss-Newton-like optimisation process, minimising a similarity measure between a reference model and the target region [8]. Their main advantage is accuracy, since usually all pixels in the region contribute to the estimation. This is a key feature, for example, for applications in virtual reality and robotics in which tracking jitter must be minimised.

Many applications of tracking (e.g. robot navigation [8], augmented reality, face tracking [7]) also require real-time video processing capabilities. So far, two main re-

BMVC 2008 doi:10.5244/C.22.100

search paths have have been explored to increase the efficiency of direct image alignment methods:

a) *Reduce the computational cost.* The computational cost of each Gauss-Newton iteration can be reduced by precomputing part of the image Jacobian, as done by Hager and Belhumeur [7], or all of it, as in Baker and Matthews' Inverse Compositional Image Alignment (ICIA) algorithm [1]. Computational requirements may also be lowered by discarding pixels that do not contribute significantly to the minimisation. These pixels are normally located in low-textured image regions [5].

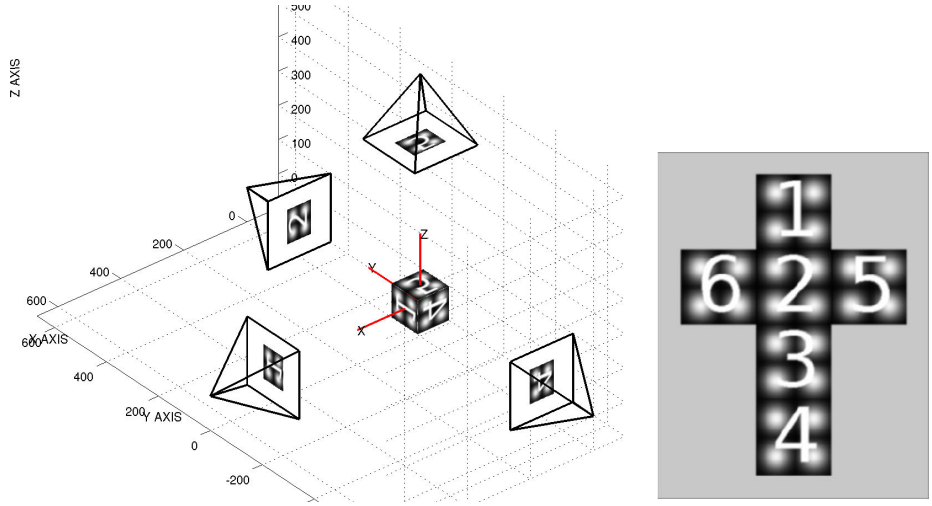
b) *Improve the convergence properties.* Efficiency has also been improved by increasing the convergence rate of the minimisation algorithm. Benhimane and Malis [4] propose the Efficient Second order Minimisation procedure (ESM) which converges faster and with a larger convergence region than Gauss-Newton, without the need of computing the Hessian matrix. Faster convergence rate and larger convergence region may also be achieved by selecting pixels which verify the assumption of linearity w.r.t. the motion parameters in the minimisation [3].

In this paper we introduce a multi-view direct procedure for efficiently tracking 3D objects. It is an extension of Hager's [7] factorisation approach to the case of three-dimensional objects and multi-camera setup. Our factorisation is closely related to the solution introduced by W. Sepp in [12]. His tracker, nevertheless, only works in the vicinity of the reference image. Our tracker is based on a 3D model of the target. It is composed of a textured 3D point cloud, which is valid for any relative orientation between camera and object.

Most previous approximations to 3D tracking are monocular, but a number of recent approaches are based on multiple views. Devernay et al. [6] use a Lucas-Kanade-like procedure to track both 3D points and texture patches (surfels). In [13], pose is computed from both point matching and similarity measures from off-line key-frames (images) of the target. Baker et al. determines object's motion simultaneously from several cameras using an Active Appearance Model (AAM) on each camera constrained globally by a single 3D model [10].

In our multi-view procedure, tracking is based on a direct approach that minimises the discrepancy between the sequence of image values and the pixel intensities (texture) of the target. This texture and its derivatives w.r.t. object's motion will be defined for each 3D point (vertex) of the object, even for those not visible in the first frame of the sequence (as opposed to [12] and [6]). These derivatives, crucial for 3D motion estimation, will also be independent of the camera position, which enables us to use any number of cameras with a single Jacobian. In our approach, each target point has associated one texture derivative. Each camera determines the subset of object points that are visible to the tracker and will be used for tracking.

The paper is organised as follows: Section 2 introduces the object model and notation used through the paper. The efficient estimation procedure for 3D motion is presented in Section 3, and expanded with annotations in appendix A. Section 4 deals with the multi-view extension. Finally, in Sections 5 and 6 we describe the experiments conducted and draw conclusions.



(a) (b)

Figure 1: (a) Example of virtual cameras around the object. Each camera optical axis is oriented along the vertex normal attached to it. (b) Texture map for numbered cube. Notice that the texture covers all possible views of the object.

2 Model Description

Let $\mathscr{M}$ be our object model, $\mathscr{M} = \{\mathscr{V}, \mathscr{T}\}$, composed of 3D points (vertices) and intensity values (texture). The set of model vertices is defined by $\mathscr{V} = \{\mathbf{x}_i \in \mathbb{R}^3 | i = \{1, \ldots, V\}\}$, where each point is expressed in terms of a scene coordinate system with origin at $\mathbf{O}$. Each object vertex has a texture value, defined as $\mathscr{T} = \{T[\mathbf{x}_i] \in \mathbb{R} | i = \{1, \ldots, N\}\}$, by means of a *texture map* $T : \mathbb{R}^3 \mapsto \mathbb{R}$. Figure 1 shows both the texture object and its texture map represented as an image.

The object pose and location are parametrically defined by a *motion model* (or *warp*) $\mathbf{f} \in \mathbb{SE}(3)$. Motion in 3D is represented as a rigid body transformation with a rotation $\mathtt{R} \in \mathbb{SO}(3)$ and a 3D offset $\mathbf{t} \in \mathbb{R}^3$: $\mathbf{x'}_i = \mathtt{R}\mathbf{x}_i + \mathbf{t}, \forall \mathbf{x}_i$. Of course, both rotation and translation are common to the whole set of object's vertices. Rotation matrices are parameterised with an exponential map $\omega = (\omega_x, \omega_y, \omega_z)^\top$. These values are stacked together with the translation values in a parameter vector $\mu \in \mathbb{R}^6$: $\mu = (\omega^\top, t_x, t_y, t_z)^\top$. Let $I_t[\mathbf{u}]$ be the intensity value at the pixel location $\mathbf{u}$ of the image acquired at time t. Under Lambertian assumptions, the following *brightness constancy* equation holds

$$\mathbf{T}[\mathbf{x}] = \mathbf{I}_t[\mathbf{p}(\mathbf{f}(\mathbf{x}, \mu))], \tag{1}$$

where vector $\mathbf{I}_t$ is the result of stacking the intensity values of the projections of each vertex $\mathbf{x}_i$ in image I_t. The same applies to $\mathbf{T}$. Vertices are projected onto the image plane using an orthogonal projection function $\mathbf{p}$, that depends on the known camera intrinsics.

2.1 Texture equivalence

Now, we will derive the constancy equation using an alternate representation for the texture values of the object. We will use a set of virtual cameras (one per point) such the image intensities resulting from the projection of each point equals the texture values for that vertex. This is similar to Fua's key-frames representation, [13], but having one (virtual) key-frame per object vertex.

Let us suppose now that we have N orthogonal cameras around our object represented by the location of their optical centres, $\mathbf{C}_i, i = 1 \ldots N$. Each camera has its optical axis aligned with vector $\mathbf{n}_i$, the normal to the point $\mathbf{x}_i$ (see Figure 1). Point $\mathbf{x}_i$ is expressed in the reference coordinate system of camera $\mathbf{C}_j$ as $\mathbf{x}_i^j$ using

$$\begin{aligned} \mathbf{x}_i^j &= \phi_j(\mathbf{x}_i) = \mathtt{R}_j\mathbf{x}_i - \mathtt{R}_j\mathbf{t}_j, \\ \mathbf{x}_i &= {\phi_j}^{-1}(\mathbf{x}_i^j) = \mathtt{R}_j^\top \mathbf{x}_i^j + \mathbf{t}_j, \end{aligned} \tag{2}$$

where $\phi_j \in \mathbb{SE}(3)$ is a rigid body transformation between both coordinate systems, which is given by a rotation $\mathtt{R}_j \in \mathbb{SO}(3)$ and a translation $\mathbf{t}_j \in \mathbb{R}^3$. Note that point $\mathbf{x}_i^j$ always has the form $\mathbf{x}_i^j = (0,0,z_j)^\top$ (expressed in camera coordinates). Let I_j be the image captured by $\mathbf{C}_j$. Points are orthogonally projected onto the image plane by means of function $\mathbf{p}_j$, which depends on the camera intrinsics. Each camera may have different intrinsics. Point $\mathbf{x}_i^j$ is projected onto the principal point of I_j, so its intensity values equals the texture value of the vertex.

$$T[\mathbf{x}_i] = I_j[\mathbf{p}_j(\mathbf{x}_i^j)] \quad \forall \mathbf{x}_i. \tag{3}$$

Combining equations (1), (2) and (3) results in a new brightness constancy equation expressed in terms of each virtual camera $\mathbf{C}_j$,

$$I_j[\mathbf{p}_j(\mathbf{x}_i^j)] = I_t[\mathbf{p}(\mathbf{f}(\phi_j^{-1}(\mathbf{x}_i^j), \mu_t))] \quad \forall \mathbf{x}_i^j, \tag{4}$$

where μ_t is the vector of parameters that optimally correspond to the object pose for time t.

Using the above assumption, we can pose our tracking problem in terms of a minimisation of the motion parameters μ,

$$\min_{\mu_{t+1}} \mathscr{J}(\mu) = ||\mathbf{I}_j[\mathbf{p}_j(\mathbf{x}^j)] - \mathbf{I}_{t+1}[\mathbf{p}(\mathbf{f}(\phi_j^{-1}(\mathbf{x}^j), \mu_{t+1}))]||^2. \tag{5}$$

Assuming incremental changes in our motion parameters between two consecutive time instants, we can rewrite equation (5) as

$$\min_{\delta\mu_t} \mathscr{J}(\mu) = ||\mathbf{I}_j[\mathbf{p}_j(\mathbf{x}^j)] - \mathbf{I}[\mathbf{p}(\mathbf{f}(\phi_j^{-1}(\mathbf{x}^j), \mu_t + \delta\mu_t))]||^2. \tag{6}$$

Making a Taylor series expansion at (μ_t, t), we can rewrite the right term of (6) as

$$\min_{\delta\mu_t} \mathscr{J}(\mu) = ||\mathbf{e}(t) - \underbrace{\left.\frac{\partial \mathbf{I}_t[\mathbf{p}(\mathbf{f}(\phi_j^{-1}(\mathbf{x}^j), \mu)])}{\partial \mu}\right|_{\mu=\mu_t}}_{\mathtt{J}(t)} \delta\mu_t||^2, \tag{7}$$

where and $\mathbf{e}(t)$ is the vector of image differences, $\mathbf{e}(t) = \mathbf{I}_j[\mathbf{p}_j(\mathbf{x}^j)] - \mathbf{I}_j[\mathbf{p}(\mathbf{f}(\phi_j^{-1}(\mathbf{x}^j), \mu_t + \delta\mu_t))]$, and $\mathtt{J}(t)$ is the Jacobian matrix relating the instantaneous change of image values with the motion parameters, both at time instant t. With least-squares we can compute the minimum of $\mathscr{J}$ as $\delta\mu_t = (\mathtt{J}(t)^\top \mathtt{J}(t))^{-1}\mathtt{J}(t)^\top \mathbf{e}(t)$. Usually, this estimation is iteratively refined (Gauss-Newton minimisation) until a stop criterion is reached.

3 Efficient Tracking

The major limitation of the tracking procedure described above is the computational cost of recomputing the image derivatives for each image in the sequence, since the Jacobian matrix $\mathtt{J}(t)$ depends on I_t. We will alleviate this computational burden extending the factorisation scheme proposed in [7] to the case of a 3D textured object. The key idea here is to express intensity changes due to object's motion in terms of the texture map of the object instead of the image values at instant t. Taking derivatives in (4) w.r.t. $\mathbf{x}^j$ we have[1],

$$\left.\frac{\partial \mathbf{I}_j[\mathbf{p}_j(\hat{\mathbf{X}})]}{\partial \hat{\mathbf{X}}}\right|_{\hat{\mathbf{X}}=\mathbf{x}^j} \approx \left.\frac{\partial \mathbf{I}_t[\mathbf{p}(\mathbf{f}(\phi_j^{-1}(\hat{\mathbf{X}}), \mu_t))]}{\partial \hat{\mathbf{X}}}\right|_{\hat{\mathbf{X}}=\mathbf{x}^j}. \tag{8}$$

And applying the chain rule to the right side of (8) leads us to

$$\begin{aligned}\left.\frac{\partial \mathbf{I}_t[\mathbf{p}(\mathbf{f}(\phi_j^{-1}(\hat{\mathbf{X}}), \mu_t))]}{\partial \hat{\mathbf{X}}}\right|_{\hat{\mathbf{X}}=\mathbf{x}^j} = &\left[\left.\frac{\partial \mathbf{I}_t[\mathbf{p}(\hat{F})]}{\partial \hat{F}}\right|_{\hat{F}=\mathbf{f}(\phi_j^{-1}(\mathbf{x}^j),\mu_t)}\right] \cdot \left[\left.\frac{\partial \mathbf{f}(\hat{\mathbf{Y}}, \mu_t)}{\partial \hat{\mathbf{Y}}}\right|_{\hat{\mathbf{Y}}=\phi_j^{-1}(\mathbf{x}^j)}\right] \\ &\cdot \left[\left.\frac{\partial \phi_j^{-1}(\hat{\mathbf{X}})}{\partial \hat{\mathbf{X}}}\right|_{\hat{\mathbf{X}}=\mathbf{x}^j}\right]\end{aligned} \tag{9}$$

We can move the two rightmost regular matrices of (9) to the other side of equation, resulting in

$$\begin{aligned}\left[\left.\frac{\partial \mathbf{I}_t[\mathbf{p}(\hat{F})]}{\partial \hat{F}}\right|_{\hat{F}=\mathbf{f}(\phi_j^{-1}(\mathbf{x}^j),\mu_t)}\right] \approx &\left[\left.\frac{\partial \mathbf{I}_j[\mathbf{p}_j(\hat{\mathbf{X}})]}{\partial \hat{\mathbf{X}}}\right|_{\hat{\mathbf{X}}=\mathbf{x}^j}\right] \cdot \left[\left.\frac{\partial \phi_j^{-1}(\hat{\mathbf{X}})}{\partial \hat{\mathbf{X}}}\right|_{\hat{\mathbf{X}}=\mathbf{x}^j}\right]^{-1} \\ &\cdot \left[\left.\frac{\partial \mathbf{f}(\hat{\mathbf{Y}}, \mu_t)}{\partial \hat{\mathbf{Y}}}\right|_{\hat{\mathbf{Y}}=\phi_j^{-1}(\mathbf{x}^j)}\right]^{-1}\end{aligned} \tag{10}$$

On the other hand, we can expand $\mathtt{J}(t)$ using the chain rule,

$$\left.\frac{\partial \mathbf{I}_t[\mathbf{p}(\mathbf{f}(\phi_j^{-1}(\mathbf{x}^j), \mu))]}{\partial \mu}\right|_{\mu=\mu_t} = \left[\left.\frac{\partial \mathbf{I}_t[\mathbf{p}(\hat{F})]}{\partial \hat{F}}\right|_{\hat{F}=\mathbf{f}(\phi_j^{-1}(\mathbf{x}^j),\mu_t)}\right] \cdot \left[\left.\frac{\partial \mathbf{f}(\phi_j^{-1}(\mathbf{x}^j), \mu)}{\partial \mu}\right|_{\mu=\mu_t}\right]. \tag{11}$$

[1] Note here that we assume that there is an extension of the texture value out of the object surface, so the derivative exists. Since our projection is orthogonal, $\left.\frac{\partial \mathbf{I}_k[\mathbf{p}_k(\hat{\mathbf{X}})]}{\partial z}\right|_{\hat{\mathbf{X}}=\mathbf{x}^k} = 0$ for any point on the object surface.

Plugging equation (10) into (11) results in a expression for $\mathtt{J}$ that does not depend on $\mathbf{I}_t$,

$$\left.\frac{\partial \mathbf{I}_t[\mathbf{p}(\mathbf{f}(\phi_j^{-1}(\mathbf{x}^j),\mu))]}{\partial \mu}\right|_{\mu=\mu_t} \approx \left[\left.\frac{\partial \mathbf{I}_j[\mathbf{p}_j(\hat{\mathbf{X}})]}{\partial \hat{\mathbf{X}}}\right|_{\hat{\mathbf{X}}=\mathbf{x}^j}\right] \cdot \left[\left.\frac{\partial \phi_j^{-1}(\hat{\mathbf{X}})}{\partial \hat{\mathbf{X}}}\right|_{\hat{\mathbf{X}}=\mathbf{x}^j}\right]^{-1} \cdot \left[\left.\frac{\partial \mathbf{f}(\hat{\mathbf{Y}},\mu_t)}{\partial \hat{\mathbf{Y}}}\right|_{\hat{\mathbf{Y}}=\phi_j^{-1}(\mathbf{x}^j)}\right]^{-1} \cdot \left[\left.\frac{\partial \mathbf{f}(\phi_j^{-1}(\mathbf{x}^j),\mu)}{\partial \mu}\right|_{\mu=\mu_t}\right]. \tag{12}$$

This equation can be further refined so our Jacobian matrix can be represented as $\mathtt{J}(t) = \mathtt{M}_0\Sigma(t)$. Matrix $\mathtt{M}_0$ is such that it depends on the vertices and the texture map, whereas $\Sigma(t)$ is a matrix that depends on the motion parameters and therefore it must be recomputed for each t. Details on derivation can be found in appendix A. Optimal parameters at time t are efficiently computed as

$$\delta\mu_t = (\Sigma(t)^\top(\mathtt{M}_0^\top\mathtt{M}_0)\Sigma(t))^{-1}\Sigma(t)^\top\mathtt{M}_0^\top\mathbf{e}(t). \tag{13}$$

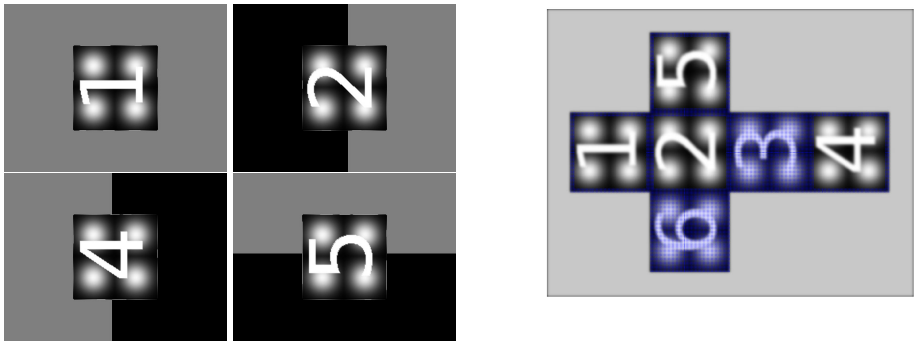
Notice that the large $N \times 30$ matrix $\mathtt{M}_0$ is constant whereas time-changing $\Sigma(t)$ is just 30×6 size, so finding our optimum has been considerably speeded up as much of these values can be precomputed.

4 Multiple Camera Tracking

From equation (15),we know that the Jacobian matrix is defined for the whole set of vertices of the object, but at time instant t only a portion of them are visible. This implies that only some rows of $\mathtt{J}(t)$ will be used: those corresponding to the visible vertices projected onto I_t. Let us suppose we have two or more cameras. Detailed inspection of equation (15) shows that matrix $\mathtt{J}(t)$ does not depend on the camera position at time t, but on the pose of the virtual cameras and the texture map values. Then, at time instant t we could use those rows of $\mathtt{J}(t)$ that are deemed as visible points at each camera. Figure 2 shows the visibility map for the camera setup of Figure 1.

Again, from (15), the matrix row $\mathtt{J}(t)_i$ at time t depends only on the texture map $\mathbf{T}_{\mathbf{x}_i}$ iff equation (8), the derivative of the brightness constancy, holds. This is only true when the image I_t corresponds to the virtual camera attached to $\mathbf{x}_i$. Thus, for each given camera, we could only use those rows of $\mathtt{J}(t)$ corresponding to points whose normal have the same orientation of the optical ray of the camera. However, we can relax the condition on the brightness constancy so that a larger number of rows per camera are selected. The larger the angle difference between the optical axis and the point normal, the lesser the accuracy of the brightness constancy assumption,and hence, the worse will be the approximation to our true Jacobian matrix. On the other hand, notice that the more cameras we have, the more rows of $\mathtt{J}(t)$ will be available, and hence, the better will be the tracking.

Notice as well that some terms from (13) must be recomputed for each time instant because of changes of the visibility of the vertices. However, recomputing consist of deleting or adding rows of $\mathtt{M}_0$ and then operating between the matrices but the values of $\mathtt{M}_0$ remain constant.



(a) (b)

Figure 2: (a) Views from the cameras located as described in Figure 1. (b) Visibility map for computing matrix $\mathtt{J}(t)$ at the given setup. The vertices that are not visible in any camera are overlaid in blue.

5 Experimental Validation

The goal of these experiments is to empirically validate our algorithm. This is achieved by using a sequence of synthetically created images where the object's motion is known with absolute accuracy. The sequence is 600 frames long and comprises a textured cube simultaneously rotating and translating in the three axis of coordinates.

The cube is 100 units side and has a Gaussian pattern with a different number attached to each face (see Figure 1) and it is placed at the origin of the scene reference system. We simulate four cameras located at 4000 units from the object at different orientations (again, see Figure 1). Initially, each camera looks at a different face of the object and they all share the same intrinsics. We simulate a orthographic projection camera by using a focal length of 20*mm* together with the considerable object-to-camera distance. The cube spins 360 degrees around each one of it axis of rotation while simultaneously translates through the scene. Snapshots of several frames are shown in Figure 3.

For each frame of the sequence we compute the motion parameters using the proposed algorithm. The iterative procedure minimises the texture values corresponding to each visible vertex with the images values captured from different cameras. Qualitative results are presented in Figure 3. We overlay onto each image of the sequence a wire-frame model of the object. The model is placed using both the ground-truth and the estimated values of the motion parameters, which allow us to compare them visually. We also present quantitative results in Figure 4, where we plot ground-truth parameter values against the estimations computed from the algorithm. Estimation for rotation parameters is quite accurate whereas the 3D offset is precise enough in most of the sequence. Notice that the estimated values diverge from the ground-truth for some frames, i.e., the "valley" of t_y. This is caused by a special configuration of the cube's faces in which the normal of all six faces depart considerably from the four cameras optical axes. In this case $\mathtt{J}(t)$ is not accurately estimated.

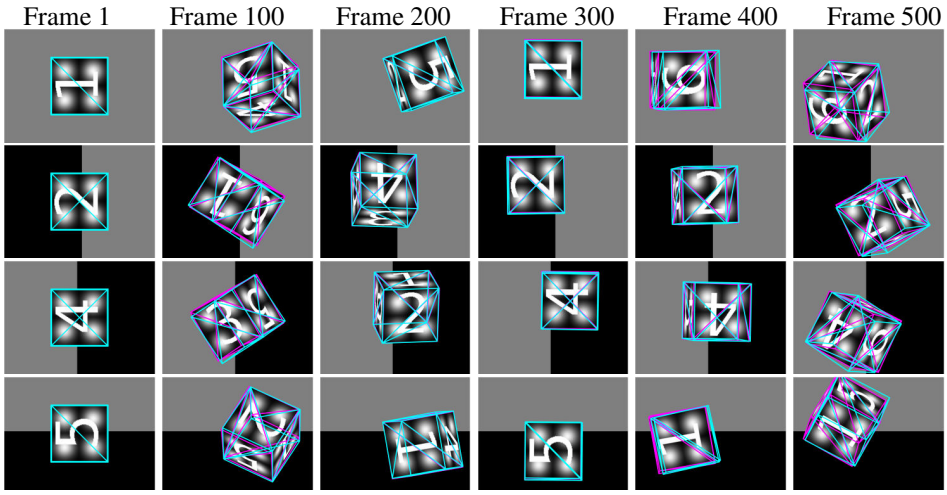


Figure 3: Selected frames from the synthetic sequence. Each row corresponds to a different camera which, initially, looks at a different face of the cube. We overlay onto each image a wire-frame model of the object using the ground-truth parameters (solid magenta) and the estimated ones (solid blue).

6 Conclusions

We introduced an algorithm for efficiently estimating the 3D motion of a known target using multiple orthogonal cameras. The algorithm is efficient since a major portion of the Jacobian involved in the minimisation is precomputed an remains constant over time. The algorithm relies on an object model based on a textured set of object points, which is independent of the camera pose. This allows us to precompute off-line the relationship between object's motion and the change in image intensities (the image Jacobian matrix), even for points of the object that are not initially visible. Moreover, we can extend this approach to multiple cameras due to the independence of the Jacobian matrix of the camera pose. For this to be true, some constraints must be satisfied: a) the cameras must be orthographic; b) only those pixels whose normal orientation coincides (or is close to) that of the camera optical axis are eligible for tracking. Neglecting this constrains leads to a loss of accuracy in the tracking.

Acknowledgements

Authors were funded by the Spanish *Ministerio de Educación y Ciencia*, under contract `TRA2005-08529-C02-02`.

References

[1] Simon Baker and Iain Matthews. Lucas-kanade 20 years on: A unifiying framework. *International Journal of Computer Vision*, 56(3):221–255, 2004.

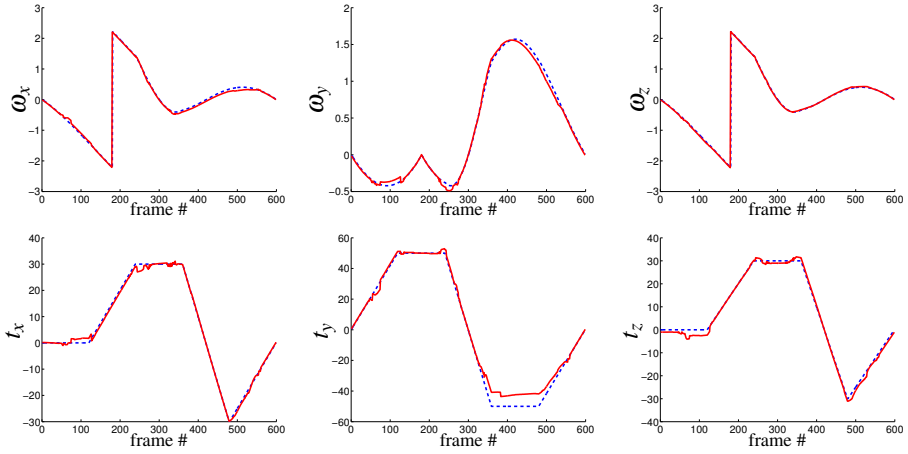


Figure 4: Estimated (red) vs. ground-truth (dotted blue) parameters. Top: Plotted values correspond to the three values of the exponential map that represents object's rotation. Bottom: Plotted values correspond to the object's 3D translation in the scene.

[2] Simon Baker, Raju Patil, Kong Man Cheung, and Iain Matthews. Lucas-kanade 20 years on: Part 5. Technical Report CMU-RI-TR-04-64, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, November 2004.

[3] S. Benhimane, A. Ladikos, V. Lepetit, and N. Navab. Linear and quadratic subsets for template-based tracking. In *Proc. of CVPR*, 2007.

[4] S. Benhimane and E. Malis. Homography-based 2d visual tracking and servoing. *International Jounal of Robotics Research*, 26(7):661–676, July 2007.

[5] F. Dellaert and R. Collins. Fast image-based tracking by selective pixel integration. In *ICCV99 Workshop on frame-rate applications*. IEEE, 1999.

[6] F. Devernay, D. Mateus, and M. Guilbert. Multi-camera scene flow by tracking 3-d points and surfels. In *Proc. of CVPR*, volume II, pages 2203– 2212, 2006.

[7] Gregory Hager and Peter Belhumeur. Efficient region tracking with parametric models of geometry and illumination. *Trans. on PAMI*, 20(10):1025–1039, 1998.

[8] C. Mei, S. Benhimane, E. Malis, and P. Rives. Constrained multiple planar template tracking for central catadioptric cameras. In *Proc. BMVC*, volume II, pages 619–628, 2006.

[9] Julien Pilet, Vincent Lepetit, and Pascal Fua. Real-time non-rigid surface detection. In *Proc. of CVPR*. IEEE, 2005.

[10] Krishnan Ramnath, Seth C Koterba, J. Xiao, C. Hu, Iain Matthews, Simon Baker, Jeffrey Cohn, and Takeo Kanade. Multi-view aam fitting and construction. *International Journal of Computer Vision*, 76(2):183–204, Feb 2008.

[11] S. Romdhani and T. Vetter. Efficient, robust and accurate fitting of a 3d morphable model. In *Proc. of ICCV*, volume 1, pages 59–66, 2003.

[12] Wolfgang Sepp and Gerd Hirzinger. Real-time texture-based 3-d tracking. In *Proc. of Deutsche Arbeitsgemeinschaft für Mustererkennung e.V.*, volume 2781 of *LNCS*, pages 330–337. Springer, 2003.

[13] L. Vacchetti, V. Lepetit, and P. Fua. Stable 3–d tracking in real-time using integrated context information. In *Conference on Computer Vision and Pattern Recognition, Madison, WI*, June 2003.

A Derivation of the Factorisation Scheme

Equation (12) can be further simplified using the definitions for both constancy equations and functions $\mathbf{f}$ and ϕ_j. First, we assume that derivatives onto the image plane are equal to derivatives onto the texture map, i.e., $\left[\left.\frac{\partial \mathbf{I}_j[\mathbf{p}_j(\hat{\mathbf{X}})]}{\partial \hat{\mathbf{X}}}\right|_{\hat{\mathbf{X}}=\mathbf{x}_i^j}\right] = \nabla \mathbf{T}_{\mathbf{x}_i}$ for object vertex i. From our warp definition we have that $\left[\left.\frac{\partial \mathbf{f}(\hat{\mathbf{Y}},\mu_t)}{\partial \hat{\mathbf{Y}}}\right|_{\hat{\mathbf{Y}}=\phi_j^{-1}(\mathbf{x}^j)}\right]^{-1} = \mathtt{R}(t)^\top$, and from equation 2, $\left[\left.\frac{\partial \phi_j^{-1}(\hat{\mathbf{X}})}{\partial \hat{\mathbf{X}}}\right|_{\hat{\mathbf{X}}=\mathbf{x}^j}\right]^{-1} = \mathtt{R}_j$. Taking partial derivatives of the warp function w.r.t. the motion parameters we can rewrite the $i-th$ row of equation (12), $\mathtt{J}_i$, as

$$\mathtt{J}_i(t) = \nabla \mathbf{T}_{\mathbf{x}_i} \mathtt{R}_j \mathtt{R}(t)^\top \left[\left.\frac{\partial \mathtt{R}(\hat{\omega}_x)}{\partial \hat{\omega}_x}\right|_{\hat{\omega}_x=\omega_x(t)} \mathbf{x}_i \;\middle|\; \left.\frac{\partial \mathtt{R}(\hat{\omega}_y)}{\partial \hat{\omega}_y}\right|_{\hat{\omega}_y=\omega_y(t)} \mathbf{x}_i \;\middle|\; \left.\frac{\partial \mathtt{R}(\hat{\omega}_z)}{\partial \hat{\omega}_z}\right|_{\hat{\omega}_z=\omega_z(t)} \mathbf{x}_i \;\middle|\; \mathtt{I}_3 \right], \tag{14}$$

with $\mathtt{I}_3$ being the 3×3 identity matrix. We can reorder this equation as a matrix multiplication in the form:

$$\begin{aligned} \mathtt{J}_i = \nabla \mathbf{T}_{\mathbf{x}_i} \mathtt{R}_j & \begin{bmatrix} \mathbf{x}_i^\top & 0 & 0 & \mathbf{x}_i^\top & 0 & 0 & \mathbf{x}_i^\top & 0 & 0 & 1 & 0 & 0 \\ 0 & \mathbf{x}_i^\top & 0 & 0 & \mathbf{x}_i^\top & 0 & 0 & \mathbf{x}_i^\top & 0 & 0 & 1 & 0 \\ 0 & 0 & \mathbf{x}_i^\top & 0 & 0 & \mathbf{x}_i^\top & 0 & 0 & \mathbf{x}_i^\top & 0 & 0 & 1 \end{bmatrix} \\ & \cdot \begin{bmatrix} \mathtt{vec}(\mathtt{R}(t)^\top \dot{\mathtt{R}}_{\omega_x(t)}) & 0 & 0 & 0 \\ 0 & \mathtt{vec}(\mathtt{R}(t)^\top \dot{\mathtt{R}}_{\omega_y(t)}) & 0 & 0 \\ 0 & 0 & \mathtt{vec}(\mathtt{R}(t)^\top \dot{\mathtt{R}}_{\omega_z(t)}) & 0 \\ 0 & 0 & 0 & \mathtt{R}(t)^\top \end{bmatrix}, \end{aligned} \tag{15}$$

where $\mathtt{0}$ is a padding matrix of zeros of the appropriate size and $\mathtt{vec}(\mathtt{A})$ is the vectorised form of matrix $\mathtt{A}$. Derivatives of the rotation matrix are expressed in dot form, i.e., $\dot{\mathtt{R}}_{\omega_x(t)} = \left.\frac{\partial \mathtt{R}(\hat{\omega}_x)}{\partial \hat{\omega}_x}\right|_{\hat{\omega}_x=\omega_x(t)}$. Notice that the rightmost matrix depends on the motion parameters at time t, but is common to every single vertex in the object. This matrix will be known as $\Sigma(t)$. The leftmost matrix depends only on the $i-th$ vertex of the object and its texture. We can stack all these matrices into a constant matrix $\mathtt{M}_0$, such that $\mathtt{J} = \mathtt{M}_0\Sigma(t)$.