Fully Automatic Real-Time 3D Object Tracking using Active Contour and Appearance Models

Giorgio Panin, Alois Knoll
Chair for Robotics and Embedded Systems, Technical University of Munich
Boltzmannstrasse 3, 85748 Garching bei Muenchen, Germany
Email: {panin,knoll}@in.tum.de

***Abstract*— This paper presents an efficient, robust and fully automatic real-time system for 3D object pose tracking in image sequences. The developed application integrates two main components: an extended and optimized implementation of a state-of-the-art local curve fitting algorithm, and a robust global re-initialization module based on point feature matching. By combining the two main components, together with a trajectory coherence and image cross-correlation check modules, our system achieves full automatic and robust object tracking in real world situations, requiring very small manual intervention from the user. The developed application relies upon a few standard libraries available on most platforms, and runs at video frame rate on a PC with standard hardware equipment.**

I. Introduction

This paper presents a novel real-time 3D object tracking system. The system efficiently combines local 3D contour estimation, in order to get fast and accurate frame-to-frame tracking, with advanced feature matching and object pose reconstruction, in order to obtain a robust global re-initialization.

Object contour tracking in image sequences is, on its own, a very interesting and important topic in computer vision, that can alone provide crucial information for many image understanding problems and, at the same time, allows the development of efficient and useful working applications in many fields of interest. We refer the reader to [2] for a survey of these applications and the related references.

Among the currently available methodologies, a very interesting one is the *Contracting Curve Density* (CCD) algorithm: this method has been developed and presented in [3] as a state-of-the-art improvement over other advanced techniques such as the Condensation algorithm [4], and it has been shown to overperform them in many different estimation tasks. Nevertheless, its higher complexity has been initially considered as an obstacle to an effective real-time implementation, even in the simplified form named Real-Time CCD from the same authors [5], and a working online version still had to be investigated.

Moreover, in order to realize an autonomous and robust tracking system, some important additional issues have to be taken into account; one of them is the possibility of automatic initialization and reinitialization of the system, both at the beginning of the tracking, and in case of tracking loss. Nevertheless, one also expects that a "well-behaving" tracking system does not need a global estimation procedure during most of the task, or, in the ideal case, only at the beginning. The global initialization module is computationally more demanding than the online tracker, and a frequent reinitialization would significantly slow down the performance of the system.

Therefore, a considerable effort in realizing such a system have been focused on better performances of the local tracking module, in terms of speed, accuracy and *convergence area* for the parameter search.

All of the above mentioned issues, and other relevant aspects, constitute the main motivation of the present work, which constitutes an extension and improvement of a recently published work by the same Authors [1], and contributes to the state-of-the-art in 3D tracking systems with the following achievements:

- The local real-time CCD algorithm [5] has been reimplemented with a significant number of critical *speedups* with respect to the originally proposed version.
- A global initialization and reinitialization module has been developed and introduced into the system
- A coherence check module, and a trajectory predicion module based on simple kinematic models, provide the robust binding between the two main estimation modules, and give the possibility of reliably checking for a tracking loss condition at any time.

By combining the global and local estimation modules together with the check and prediction modules in a fully automatic way, we thus achieve both robust and real-time tracking performances; as a result, we were able to build a working application, that runs at video frame rate on a PC equipped with standard video hardware.

The present paper is organized as follows: in Section 2, a general overview of the tracking system will be given; Section 3 states the common geometric framework, used by all of the system modules; Section 4 will then focus on the local CCD tracking algorithm, and the most important implementation speedups over the existing version will be described; Section 5 describes the global initialization/search module; Section 6 deals with the image cross-correlation check and the trajectory prediction modules; experimental results are given in Section 7.

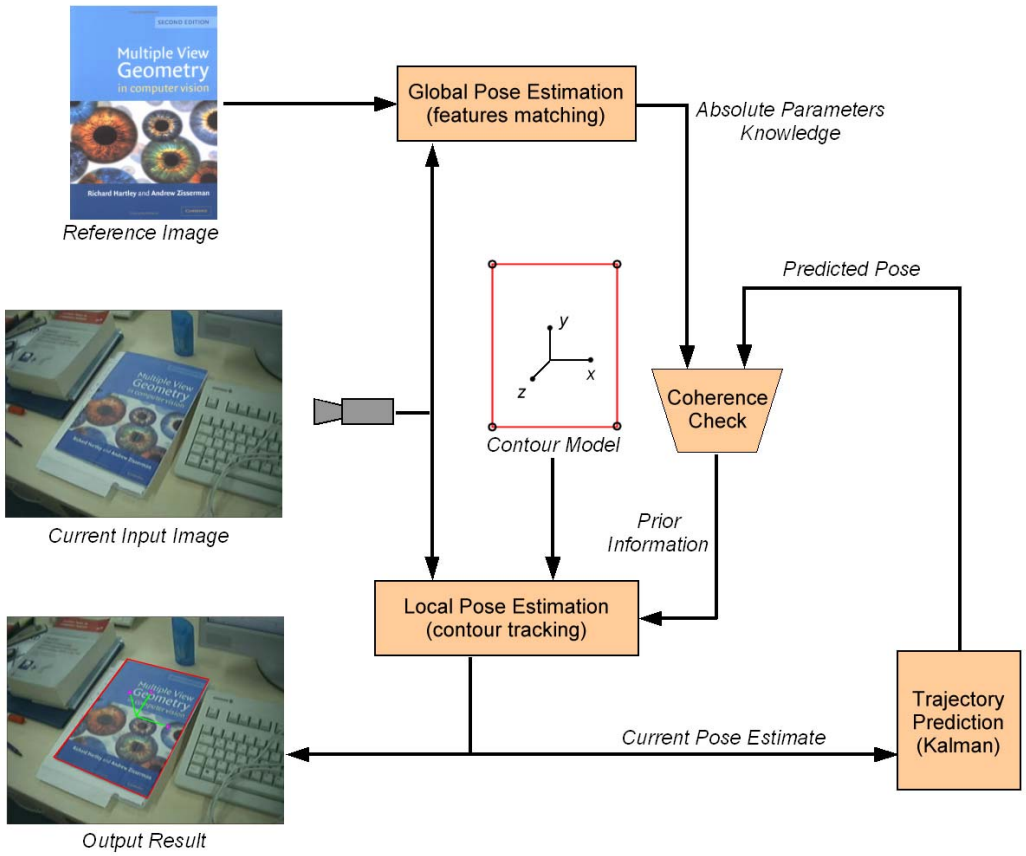

Figure 1. Overall system architecture

II. System Overview

A general overview of the system is here given, particularly focusing on the main purpose and interactions of the different modules involved, and the information flow between them. Referring to Fig. 1, we can see the interacting modules inside the system, exchanging online and offline data.

A description of the system behavior is then given by the following abstract procedure:

System Setup: At startup, the user needs to specify a basic *model description* for the object contour; this operation needs to be done only once for a given object to be tracked. The base contour shape of the object is taken by simply placing a few *control points* of a spline curve on a *reference image*, in order to satisfactory match the visible object contour. The result of this process is the base *contour shape model* and the reference image itself, that are both stored and used in the subsequent tracking steps.

Global Estimation: When activating the tracking engine, the system first searches for the object over the whole incoming image, by means of robust *feature matching* against the reference image. The obtained pose parameters and their estimation uncertainty constitute the *absolute* knowledge of the object pose, in absence of reliable predictions from the contour tracker.

Local Estimation: The local estimation module holds the given contour model, and receives as well the online color image stream coming from the video camera. It refines the pose estimation coming from the prior knowledge (either the absolute or the predicted one) through fast and accurate contour matching. The output parameters are both given to the end user, and used by the system in order to make a pose *prediction* and *check* for the next timestep.

Trajectory Prediction: The trajectory prediction is accomplished by a simple but suitable *kinematic model*, via a steady-state *Kalman Filter*. The filter uses the current estimated pose from the CCD algorithm as state measurement, and provides a prediction of the next state.

Coherence Check: After each pose estimation (either global or local), a check for the coherence of the result is performed. The coherence check module employs the reference model appearance, by rendering the textured model surface at the estimated pose onto the image, and comparing them via a normalized cross-correlation index. If the coherence index at the given timestep do not satisfy a given treshold, the global estimation module will be called in order to reinitialize the system. Otherwise, the predicted pose will be accepted as prior knowledge for the next frame local pose estimation.

III. The Common Geometric Transformation Framework

The overall geometric mapping between the object space and the image plane consists of a rigid roto-translation, followed by perspective projection on the image screen; for this purpose, it is first necessary to specify in advance the internal camera image acquisition model, given by a set of fixed *intrinsic* parameters $\mathbf{\Psi}$ that are obtained off-line via a common calibration procedure. The *extrinsic* transformation parameters will instead be denoted with a vector $\mathbf{\Phi}$, so that the overall transformation from a space point $\mathbf{x}$ to a screen point $\mathbf{q}$ is given abstractly by

$$\mathbf{q} = T(\mathbf{x}, \mathbf{\Phi}, \mathbf{\Psi}) \tag{1}$$

The general form of this 3D/2D rigid transformation is in turn given, in homogeneous coordinates, by

$$T(\mathbf{x}, \mathbf{\Phi}, \mathbf{\Psi}) = P(\mathbf{\Psi}) [R(\mathbf{\Phi}) \mathbf{x} + \mathbf{t}(\mathbf{\Phi})] \tag{2}$$

being P the (3×3) intrinsic *projection matrix* of the camera, and $(R, \mathbf{t})$ the extrinsic rotation matrix and translation vector.

The projection matrix P contains generally linear and nonlinear calibration parameters, accounting for several distortion effects, the video resolution of the system, and so on; for sake of simplicity, the nonlinear distortion effects have been here neglected. We assume a simple camera calibration to be performed off-line with one of the commonly available tools, e.g. the OpenCv camera calibration functions in [6], so that the matrix P is given off-line and considered as a constant parameter for the system.

Regarding the extrinsic parameters in (2), we decided in particular to represent rigid rotations by means of a common *Euler angles* parametrization

$$R(\alpha, \beta, \gamma) = R_z(\gamma) R_y(\beta) R_x(\alpha) \tag{3}$$

being R_x, R_y, R_z the elementary rotation matrices around x, y, z axes, with angles α, β, γ.

In order to avoid singularities, the current frame rotation parameters have been in our implementation referred to the *previous* estimated frame attitude, instead of the fixed camera frame, so that actually

$$R^t(\alpha, \beta, \gamma) = R^{t-1}(dR_z(\gamma) dR_y(\beta) dR_x(\alpha)) \tag{4}$$

at timestep t.

After each successful estimation, the current rotation matrix is updated

$$R^{t-1} \rightarrow R^t \tag{5}$$

and the orthogonality constraint over the matrix is enforced, in order to avoid numerical drifts.

Finally, our *6-dof* transformation vector will be given by

$$\mathbf{\Phi} \equiv [\alpha, \beta, \gamma, t_x, t_y, t_z]; \quad \mathbf{\Phi} \in \Re^6 \tag{6}$$

The above defined transformation model is subsequently used by all of the system modules; therefore the object contour, the individual feature points on the surface, and the whole surface image itself, will be all transformed and mapped to the image according to (1), given the current parameters vector $\mathbf{\Phi}$ in (6) and the current rotation matrix R^t.

IV. Local Tracking: The Improved Real-Time CCD Algorithm

This Section considers in some detail the main speed-up improvements over the original CCD algorithm [3] [5], starting from a hypothetical initial estimate with a given uncertainty.

In order to describe the algorithm improvements, we recall as well its main underlying principles.

A. Contour Model Parametrization

As already mentioned in Section 2, the object contour model is defined as a parametric model, that describes the shape of the object in terms of a base shape and a limited number of degrees of freedom (*dof*), chosen in order to specify the allowed displacements and deformations of the base curve.

In this implementation, we consider planar rigid objects with a single contour, moving in 3D space, and we model the contour as a continuous, differentiable and open *quadratic B-Spline* in $\Re^3$. The contour representation is given by a set of N_C *control points* $C = \{\mathbf{c}_0, \mathbf{c}_1, ..., \mathbf{c}_{N_C-1}\}$ and a piecewise quadratic *Basis Function* $B(s)$, so that the curve equation is given by

$$\mathbf{q}(s) = \sum_{n=0}^{N_C-1} \mathbf{c}_n B(s-n) \tag{7}$$

A set of K *sample points* $\{\mathbf{q}_{10}, ..., \mathbf{q}_{K0}\}$, and normal vectors $\{\mathbf{n}_1, ..., \mathbf{n}_K\}$ which are uniformly distributed w.r.t. the contour parameter *s*, is computed in advance by using (7), being the parameter K off-line fixed.

In order to search for the best matching pose, the contour points $\mathbf{q}$ are then transformed in 3D space according to the *6-dof* parametrized mapping (1), where the parameters vector $\mathbf{\Phi}$ can be regarded as the *state-space* of the contour.

B. Step 1 of CCD: Learn Local Statistics

Once the model parameter space has been specified, the optimization algorithm is initialized by setting the global estimated parameter mean and covariance matrix $(\mathbf{m}_{\mathbf{\Phi}}, \mathbf{\Sigma}_{\mathbf{\Phi}})$ to the prior knowledge $(\mathbf{m}^*_{\mathbf{\Phi}}, \mathbf{\Sigma}^*_{\mathbf{\Phi}})$, where the statistics are modeled by multivariate Gaussian distributions.

The next step consists of taking a set of points along the normal directions on both sides of the curve, for each sample position k. We decided to set a *fixed distance*

h along the line segments on which the set of points are collected and evaluated, according to the hypothetical uncertainty of the current evaluation step, by following the heuristic rule

$$h^2 = \left[\det\left(\mathbf{\Sigma}_{\mathbf{\Phi}}\right)\right]^{1/N} \tag{8}$$

with $N = 6$ the parameter space dimension. This idea, which differs from the original CCD formulation, allows a significant computational saving by giving a constant shape for the local statistics *weighting function* at each sample point, thus avoiding many expensive nonlinear function evaluations. For a contour that encloses a limited area, moreover, some attention has been paid to limit the search distance on the internal side, in order to avoid "crossing" the opposite boundary, thus sampling pixels from the wrong area.

The normal segments are then uniformly sampled along their length, by dividing each of them into an overall number of L equally spaced sample points ($L/2$ for each contour side)

$$\mathbf{q}_{kl} = \mathbf{q}_{k0} + d_l \mathbf{n}_k \tag{9}$$

where $d_l = d\left(\mathbf{q}_{kl}, \mathbf{q}_{k0}\right)$ is the distance to the curve (with sign) along the normal.

After the search distance h has been set, we proceed by assigning two suitable weighting functions w_1, w_2 to the pixels $\mathbf{q}_{kl}$ along the normal for the two sides of the curve. By following the suggested rules in [3], we decided to define these functions as

$$w_{1/2}\left(d_l\right) := Z\left(\frac{a_{1/2}\left(d_l\right) - \gamma_1}{1 - \gamma_1}\right)^6 \left[e^{-d_l^2/2\hat{\sigma}^2} - e^{-\gamma_2}\right]^+ \tag{10}$$

where Z is a *normalization* factor ensuring that $\sum_l w_{1/2}\left(d_l\right) = 1$, and

$$a_1\left(d\right) := \frac{1}{2}\left[erf\left(\frac{d}{\sqrt{2}\sigma}\right) + 1\right]; \qquad a_2 := 1 - a_1 \tag{11}$$

the smooth *assignment* functions to the two sides of the curve, with $\gamma_1 = 0.5$ (disjoint weight assignment) and $\gamma_2 = 4$ for the truncated Gaussian in (10). Moreover, the standard deviation $\hat{\sigma}$ has been chosen such as to cover the specified distance h.

That means

$$\hat{\sigma} = \max\left[\frac{h}{\sqrt{2\gamma_2}}, \gamma_4\right]; \qquad \sigma = \frac{1}{\gamma_3}\hat{\sigma} \tag{12}$$

with the two additional constants $\gamma_3 = 6$ (linear dependence between σ and $\hat{\sigma}$) and $\gamma_4 = 4$ (minimum weighting window width). The L distances d_l, the assignment and weighting function values are computed and stored as well.

Afterwards, the sample points set $\mathbf{q}_{kl}$ will be transformed according to the current hypothesis $\mathbf{\Phi} = \mathbf{m}_{\mathbf{\Phi}}$, by applying the geometric transformation (1).

The result is a set of local image positions $\mathbf{v}_{kl}$

$$\mathbf{v}_{kl} = T\left(\mathbf{q}_{kl}, \mathbf{\Phi}\right) \tag{13}$$

The $2K$ local unnormalized RGB statistics up to the second order, are then collected as specified in the original CCD algorithm (see [3] and [5])

$$\begin{aligned} \nu_{ks}^{(0)} &= \sum_{l=1}^{L} w_{kls} \\ \nu_{ks}^{(1)} &= \sum_{l=1}^{L} w_{kls}\mathbf{I}_{kl} \\ \nu_{ks}^{(2)} &= \sum_{l=1}^{L} w_{kls}\mathbf{I}_{kl}\mathbf{I}_{kl}^T \end{aligned} \tag{14}$$

with $s = 1, 2$ (for each curve side), $k = 1, ..., K$, and $\mathbf{I}_{kl}$ the observed pixel colors at the sample point locations.

In order to perform the *space-blurring* step along the contour, as recommended in [3] with exponential filtering, we managed to simplify the operation by using a *constant* set of filtering coefficients for the whole contour, independent from the current shape $\mathbf{\Phi}$: every contour point $k = 1, ..., K$ will receive a contribution from each other point k_1 given by a coefficient

$$b\left(k - k_1\right) = \left(\lambda/2\right)\exp\left(-\lambda\left|k - k_1\right|\right) \tag{15}$$

with $\lambda = 0.4$ in our implementation, so that

$$\nu_{ks}^{(o)} \leftarrow \sum_{k_1=1}^{K} b\left(k - k_1\right)\nu_{k_1 s}^{(o)}; \qquad o = 0, 1, 2 \tag{16}$$

The space-filtered statistics are then also smoothed in *time*, by means of a simple first-order exponential decay rule

$$\widetilde{\nu}_{ks}^{(o)}\left(t\right) = \tau\nu_{ks}^{(o)} + \left(1 - \tau\right)\widetilde{\nu}_{ks}^{(o)}\left(t - 1\right) \tag{17}$$

and finally, the resulting quantities are *normalized*, thus giving a set of expected color values and covariance matrices for each sample position k and each side of the contour

$$\bar{\mathbf{I}}_k^{(s)} = \frac{\widetilde{\nu}_{ks}^{(1)}\left(t\right)}{\widetilde{\nu}_{ks}^{(0)}\left(t\right)}; \qquad \bar{\mathbf{\Sigma}}_k^{(s)} = \frac{\widetilde{\nu}_{ks}^{(2)}\left(t\right)}{\widetilde{\nu}_{ks}^{(0)}\left(t\right)} \tag{18}$$

The complete normalized local statistics set will be indicated with $S := \left(\bar{\mathbf{I}}_k^{(s)}, \bar{\mathbf{\Sigma}}_k^{(s)}\right)$ with $2K$ color mean vectors $\bar{\mathbf{I}}_k^{(1)}, \bar{\mathbf{I}}_k^{(2)}$ and (3×3) positive definite covariance matrices $\bar{\mathbf{\Sigma}}_k^{(1)}, \bar{\mathbf{\Sigma}}_k^{(2)}$, for each side of the curve and each sample position k.

C. Step 2 of CCD: Compute the Cost Function and its Derivatives

Here the collected local statistics are used in order to obtain the matching function, its gradient and the Hessian matrix, which will be used to update the pose parameters.

By following [3], we write the general cost function formulation, with a slightly different notation:

$$E = E_1\left(\mathbf{\Phi}, \mathbf{m}^*_{\mathbf{\Phi}}, \mathbf{\Sigma}^*_{\mathbf{\Phi}}\right) + E_2\left(\mathbf{\Phi}, S\right) \tag{19}$$

where

$$E_1 := \frac{1}{2}\left(\mathbf{\Phi} - \mathbf{m}^*_{\mathbf{\Phi}}\right)^T \left(\mathbf{\Sigma}^*_{\mathbf{\Phi}}\right)^{-1} \left(\mathbf{\Phi} - \mathbf{m}^*_{\mathbf{\Phi}}\right) + Z\left(\mathbf{\Sigma}^*_{\mathbf{\Phi}}\right) \tag{20}$$

is the prior log-likelihood according to a Gaussian distribution (with normalization factor Z) and

$$\begin{aligned} E_2 &:= -\log \prod_{k,l} p\left(\mathbf{I}_{kl} | \mathbf{\Phi}, S\right) = \\ &= \sum_{k,l} \left[\tfrac{1}{2} \left(\mathbf{I}_{kl} - \hat{\mathbf{I}}_{kl}\right)^T \hat{\mathbf{\Sigma}}^{-1}_{kl} \left(\mathbf{I}_{kl} - \hat{\mathbf{I}}_{kl}\right) + Z\left(\hat{\mathbf{\Sigma}}_{kl}\right) \right] \end{aligned} \tag{21}$$

with normalization $Z\left(\hat{\mathbf{\Sigma}}_{kl}\right)$, is the log-probability of the observed colors $\mathbf{I}_{kl}$ w.r.t. the pixel statistics $\left(\hat{\mathbf{I}}_{kl}, \hat{\mathbf{\Sigma}}_{kl}\right)$, with $\hat{\mathbf{I}}_{kl}$ given by

$$\hat{\mathbf{I}}_{kl} = a_1\left(d_l\right) \bar{\mathbf{I}}^{(1)}_k + \left(1 - a_1\left(d_l\right)\right) \bar{\mathbf{I}}^{(2)}_k \tag{22}$$

and the local covariance matrices $\hat{\mathbf{\Sigma}}_{kl}$

$$\hat{\mathbf{\Sigma}}_{kl} = a_1\left(d_l\right) \bar{\mathbf{\Sigma}}^{(1)}_k + \left(1 - a_1\left(d_l\right)\right) \bar{\mathbf{\Sigma}}^{(2)}_k \tag{23}$$

as proposed in [3].

Regarding the derivatives, we decided instead to neglect the dependence of (23) on $\mathbf{\Phi}$, because this leads to a high computational cost for the overall derivatives of E_2[1].

The last implementation choice allows to formulate the cost function E_2 in a standard *weighted nonlinear least squares* form

$$E_2 = \frac{1}{2} \sum_{k,l} \left(\mathbf{I}_{kl} - \hat{\mathbf{I}}_{kl}\left(\mathbf{\Phi}\right)\right)^T \hat{\mathbf{\Sigma}}^{-1}_{kl} \left(\mathbf{I}_{kl} - \hat{\mathbf{I}}_{kl}\left(\mathbf{\Phi}\right)\right) \tag{24}$$

where the only dependence on $\mathbf{\Phi}$ has been put into evidence, and the normalization term of the sum (21) has therefore also been neglected for the optimization.

With this formulation, the Gauss-Newton approximation to the Hessian matrix can be adopted. First, the gradient is computed as

$$\nabla_{\mathbf{\Phi}} E_2 = -\sum_{k,l} J^T_{a_1} \hat{\mathbf{\Sigma}}^{-1}_{kl} \left(\mathbf{I}_{kl} - \hat{\mathbf{I}}_{kl}\right) \tag{25}$$

with

$$J_{a_1}\left(\mathbf{\Phi}, \mathbf{v}_{kl}\right) := \left(\bar{\mathbf{I}}^{(1)}_k - \bar{\mathbf{I}}^{(2)}_k\right) \left(\nabla_{\mathbf{\Phi}} a_1\left(d_l\right)\right)^T \tag{26}$$

Afterwards, the Gauss-Newton approximation to the Hessian matrix is given by

$$H_{\mathbf{\Phi}} E_2 = \sum_{k,l} J^T_{a_1} \hat{\mathbf{\Sigma}}^{-1}_{kl} J_{a_1} \tag{27}$$

The overall gradient and Hessian matrices for the optimization are then obtained by adding the prior cost function derivatives

$$\begin{aligned} \nabla_{\mathbf{\Phi}} E_1 &= \left(\mathbf{\Sigma}^*_{\mathbf{\Phi}}\right)^{-1} \left(\mathbf{\Phi} - \mathbf{m}^*_{\mathbf{\Phi}}\right) \\ H_{\mathbf{\Phi}} E_1 &= \left(\mathbf{\Sigma}^*_{\mathbf{\Phi}}\right)^{-1} \end{aligned} \tag{28}$$

and the Newton optimization step can finally be performed as

$$\mathbf{m}^{new}_{\mathbf{\Phi}} = \mathbf{m}_{\mathbf{\Phi}} - \left(H_{\mathbf{\Phi}} E\right)^{-1} \nabla_{\mathbf{\Phi}} E \tag{29}$$

The covariance matrix is updated as well by an exponential decay rule (see [3])

$$\mathbf{\Sigma}^{new}_{\mathbf{\Phi}} = c\mathbf{\Sigma}_{\mathbf{\Phi}} + \left(1 - c\right)\left(H_{\mathbf{\Phi}} E\right)^{-1} \tag{30}$$

with $c = 0.25$ in our implementation.

A *confirmation* check before every parameter update, as recommended in [3], is here employed as follows.

We compute a *confirmation value* given by

$$K^{new} := g\left(\mathbf{m}^{new}_{\mathbf{\Phi}} - \mathbf{m}_{\mathbf{\Phi}}, \mathbf{\Sigma}^{new}_{\mathbf{\Phi}} + \mathbf{\Sigma}_{\mathbf{\Phi}}\right) \tag{31}$$

being

$$g\left(\mathbf{m}, \mathbf{\Sigma}\right) := \frac{1}{\left(2\pi\right)^{d/2} \left|\mathbf{\Sigma}\right|^{1/2}} exp\left(-\frac{1}{2}\left(\mathbf{m}^T \mathbf{\Sigma}^{-1} \mathbf{m}\right)\right) \tag{32}$$

the multivariate Gaussian function; if this parameter is increasing over the previous evaluation $K^{new} > K^{old}$, the estimate $\mathbf{m}^{new}$ will be recorded as the current best estimate

$$\mathbf{m}^{ok}_{\mathbf{\Phi}} = \mathbf{m}^{new}_{\mathbf{\Phi}}; \qquad \mathbf{\Sigma}^{ok}_{\mathbf{\Phi}} = \mathbf{\Sigma}^{new}_{\mathbf{\Phi}} \tag{33}$$

and kept for the output.

The optimization then restarts from step 1 with the updated parameters

$$\begin{aligned} \mathbf{m}^{new}_{\mathbf{\Phi}} &\rightarrow \mathbf{m}_{\mathbf{\Phi}} \\ \mathbf{\Sigma}^{new}_{\mathbf{\Phi}} &\rightarrow \mathbf{\Sigma}_{\mathbf{\Phi}} \\ K^{new} &\rightarrow K^{old} \end{aligned} \tag{34}$$

and the whole procedure is repeated until convergence; we extensively verified that less than 10 iterations are always sufficient to give satisfactory results.

At the end, the best recorded estimate $\left(\mathbf{m}^{ok}_{\mathbf{\Phi}}, \mathbf{\Sigma}^{ok}_{\mathbf{\Phi}}\right)$ will be given as output of the algorithm.

For sake of clarity, a sketch of the whole local tracking algorithm is below given.

[1] In a previous implementation, as described in [1], a further approximation was introduced by substituting (23) with a *hard* $0/1$ assignment rule, therefore completely removing the dependence of $\hat{\mathbf{\Sigma}}_{kl}$ on $\mathbf{\Phi}$. The present implementation, more faithful to the original formulation, considers this dependence up to the $0-th$ order, that is, neglecting the dependence only when computing the derivatives.

Algorithm 1 Local countour tracking algorithm (CCD)

Input: Contour points $\{\mathbf{q}_{10}, ..., \mathbf{q}_{K0}\}$, normal vectors $\{\mathbf{n}_1, ..., \mathbf{n}_K\}$ and *prior information* $(\mathbf{m}^*_{\mathbf{\Phi}}, \mathbf{\Sigma}^*_{\mathbf{\Phi}})$

Initialize: set current estimate $(\mathbf{m}_{\mathbf{\Phi}}, \mathbf{\Sigma}_{\mathbf{\Phi}})$ = $(\mathbf{m}^*_{\mathbf{\Phi}}, \mathbf{\Sigma}^*_{\mathbf{\Phi}})$

Main Loop: iterate until convergence

1: Compute uncertainty area h according to the current estimate (Eq. 8)
2: Compute the KL sample points in space $\mathbf{q}_{kl}$ (Eq. 9), and the assignment weights (Eqs. 10, 11), according to h
3: Project the sample points on the screen $\mathbf{v}_{kl}$ (Eq. 13)
4: Compute 2^{nd} order unnormalized local statistics $\nu^{(o)}_{ks}$ (Eq. 14)
5: Compute space-time filtered statistics at time t, $\widetilde{\nu}^{(o)}_{ks}(t)$ (Eq. 17)
6: Get the normalized statistics set $S = \left(\bar{\mathbf{I}}^{(s)}_k, \bar{\mathbf{\Sigma}}^{(s)}_k\right)$ (Eq. 18)
7: Compute $\hat{\mathbf{I}}_{kl}$ and $\hat{\mathbf{\Sigma}}_{kl}$ (Eqs. 22 and 23)
8: Compute the gradient $\nabla_{\mathbf{\Phi}} E_2$ and the Hessian $H_{\mathbf{\Phi}} E_2$ (Eqs. 25 and 27) and add the prior terms (Eq. 28).
9: Perform Newton optimization step to compute $\mathbf{m}^{new}_{\mathbf{\Phi}}$ and $\mathbf{\Sigma}^{new}_{\mathbf{\Phi}}$ (Eqs. 29 and 30)
10: Compute confirmation parameter K^{new} (Eq. 31), and evtl. accept $(\mathbf{m}^{new}_{\mathbf{\Phi}}, \mathbf{\Sigma}^{new}_{\mathbf{\Phi}})$ as best estimate
11: Update estimation parameters and confirmation number (Eq. 34)

Output: Best estimated contour pose $\left(\mathbf{m}^{ok}_{\mathbf{\Phi}}, \mathbf{\Sigma}^{ok}_{\mathbf{\Phi}}\right)$

V. Global pose initialization

For the (re-)initialization phase, a feature matching approach is used to estimate the position of the object (Fig. 2). In order to detect the object in the current image, the reference image is required.

We employed an implementation of the well-known SIFT algorithm [7] for keypoint detection and matching, followed by least-squares pose estimation:

- Significant feature point *descriptors* are first off-line extracted from the reference image, and stored in a database.
- Subsequently, the keypoints are matched to features found in the current image.
- Afterwards, the detected matches are used in order to reconstruct (in a least-squares sense) the 3D pose of the object, through the POSIT algorithm [8], following the implementation of the OpenCv library [6]. In order to perform this task, the reference keypoint positions in the database are measured in real-world units by knowing the conversion factor of the reference image [pixel/mm].

The obtained transformation finally gives the absolute estimate of the contour position, needed for the CCD optimization algorithm.

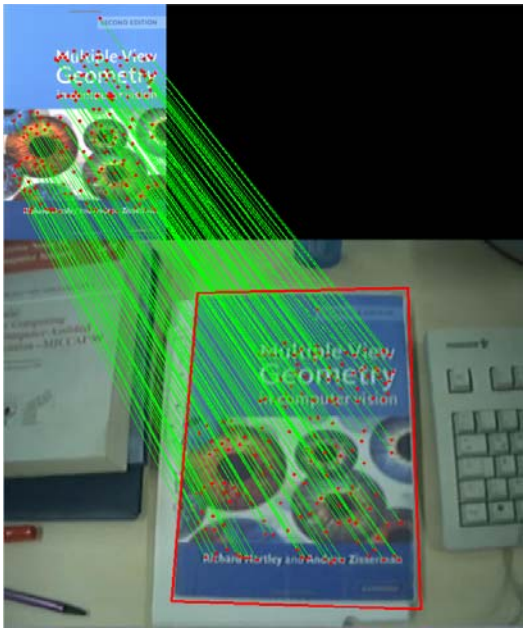

Figure 2. Initial pose estimation with features matching

VI. Prediction and Check modules

A. Image cross-correlation check

The coherence check module employs the model appearance information, by *rendering* the model surface at the estimated pose onto the image, and comparing the results via a normalized cross-correlation (*NCC*) index. *NCC* has been employed because of its independence from lighting, and relative smooth behavior with respect to occlusions and noise in the real object image.

NCC between two grayscale images R and I, is defined as

$$NCC = \frac{\sum_{x,y}\left(R(x,y)-\overline{R}\right)\left(I(x,y)-\overline{I}\right)}{\sqrt{\sum_{x,y}\left(R(x,y)-\overline{R}\right)^2 \sum_{x,y}\left(I(x,y)-\overline{I}\right)^2}} \tag{35}$$

where $\overline{R}$ and $\overline{I}$ are the mean values of R and I respectively, and the sums are performed over the overlapping pixel coordinates of the two images.

Our procedure would consist first in *warping* the reference image R onto the current image I according to the mapping (1) and the parameters $\mathbf{\Phi}$, and then computing (35), where the sums are performed over the superimposing pixels. Instead, in order to keep a constant number of pixels in the sums, independent of the pose parameters, we equivalently perform an *inverse* warp of the image I onto R, so that the sums extend over the constant pixel set of R. By doing so, the *NCC* measure (35) will be stable, and independent from the object pose $\mathbf{\Phi}$.

Therefore, we compute the following overall index

$$NCC = \frac{\sum_{x,y}\left(R(x,y)-\overline{R}\right)\left(I\left(T^{-1}(x,y,\mathbf{\Phi})\right)-\overline{I}\right)}{\sqrt{\sum_{x,y}R(x,y)^2 \sum_{x,y}\left(I\left(T^{-1}(x,y,\mathbf{\Phi})\right)-\overline{I}\right)^2}} \tag{36}$$

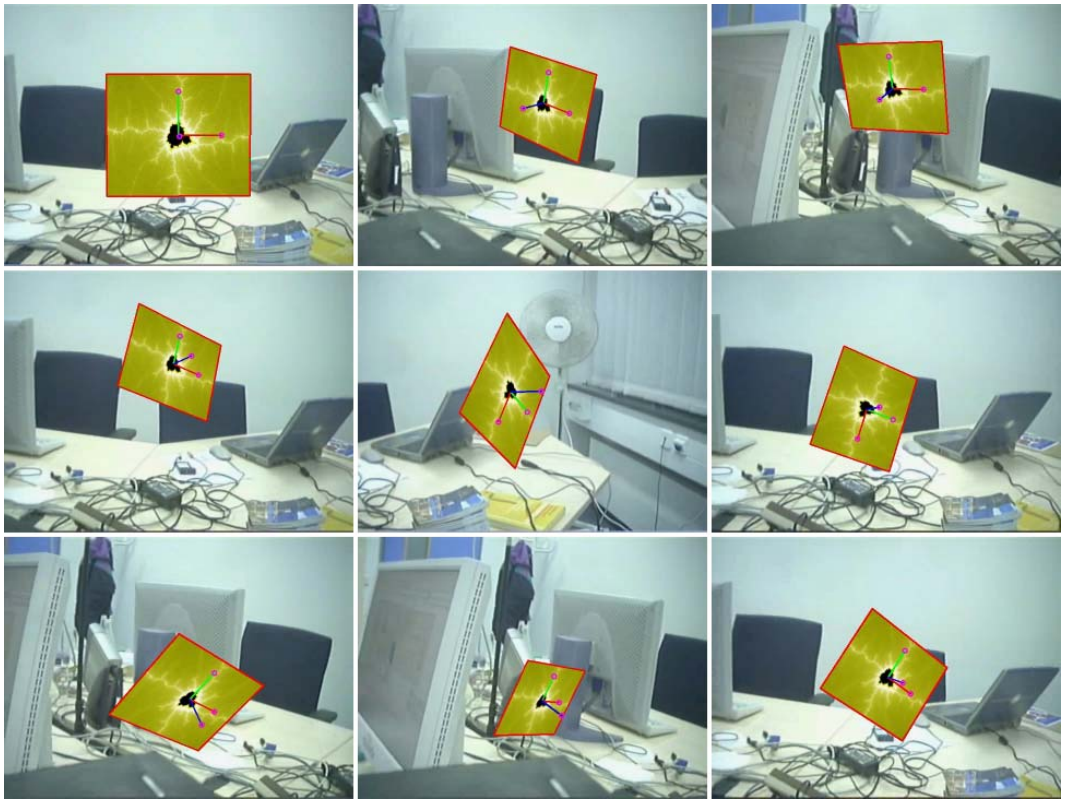

Figure 3. Results of the tracking on a simulated scenario

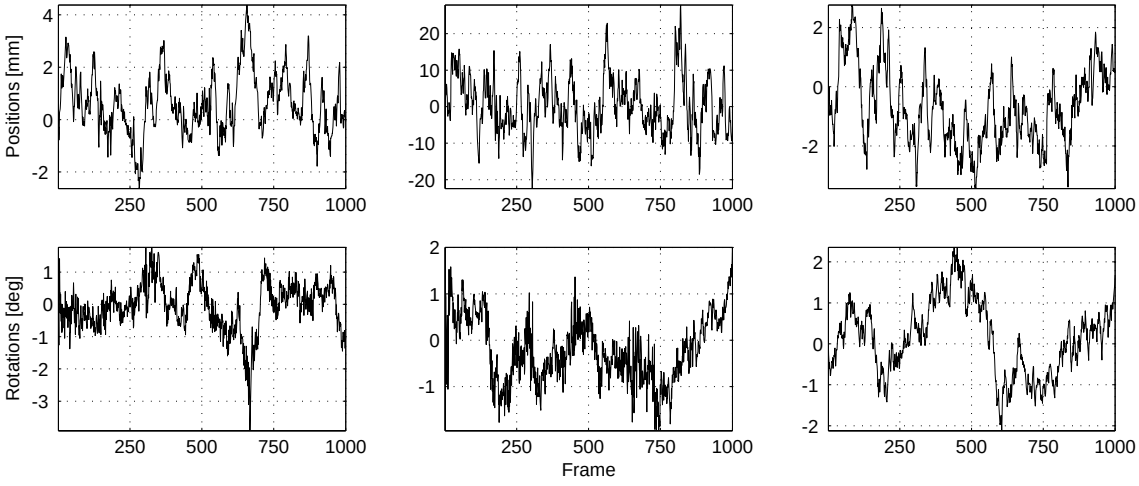

Figure 4. Position and orientation errors

The coherence check is easily performed by setting a suitable minimum treshold on (36), and calling for a re-initialization in case of a too low image coherence.

B. Trajectory prediction

In order both to obtain a robust trajectory estimate, and to be able to give a prediction over the next object position, we employ a standard Kalman Filter, that incorporates a "reasonable" noisy kinematic model of the trajectory, known as Discrete White Noise Acceleration Model [9]. For this system, the steady-state estimation filter (or *alpha-beta* filter) is employed, with suitable covariance parameters for each state variable.

This module, moreover, allows to furtherly check the tracking behavior by putting an upper threshold on the measurement residuals between the predicted variables and the estimated values (according to the CCD algorithm); if the residual error exceeds the threshold, the measured trajectory is considered as unreliable and a re-initialization call is issued. The reader is referred to [9] for more details on the topic.

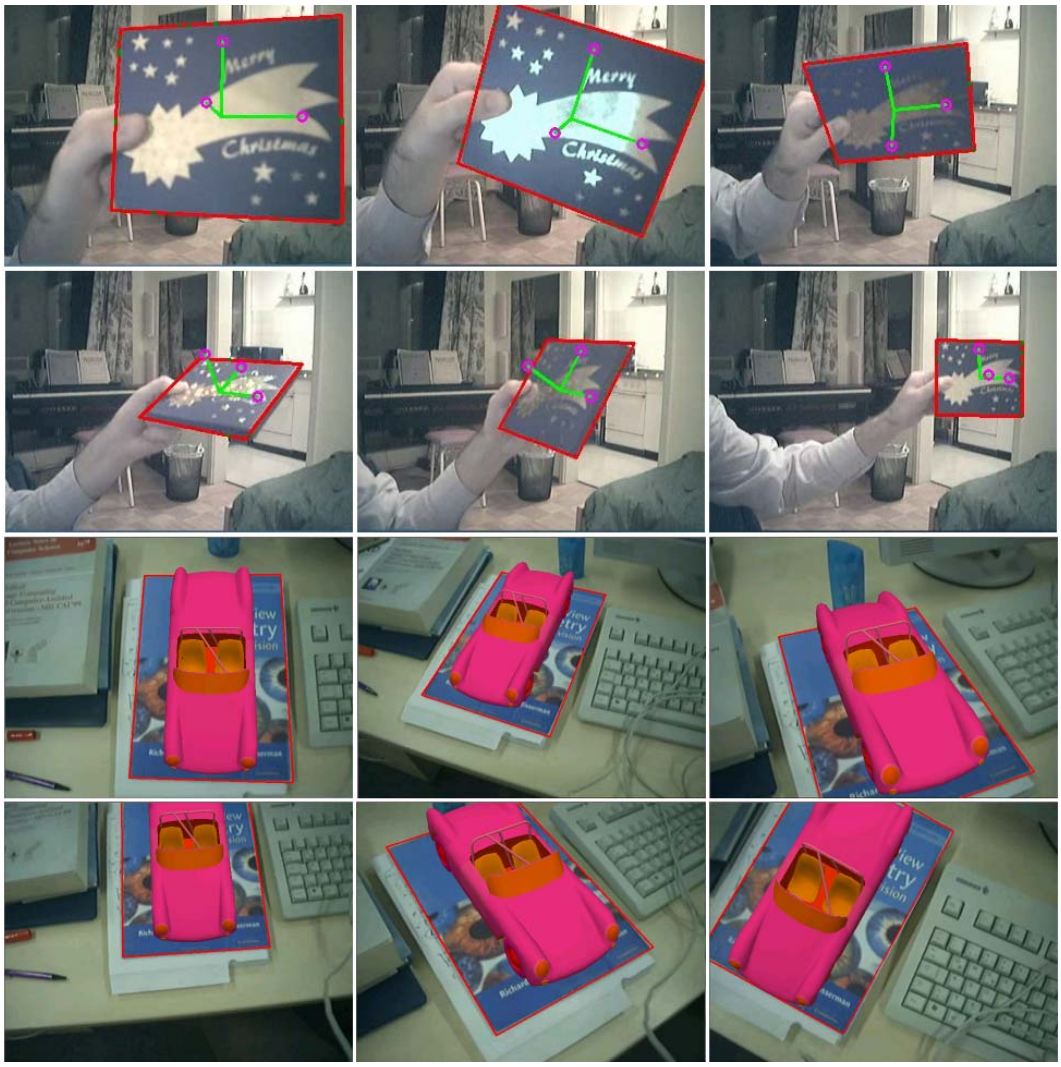

Figure 5. Results of two real-time tracking experiments

VII. Experimental Results

In order to evaluate the performance of the tracking system, a simulated sequence was generated by superimposing a fictitious planar object onto a moving background sequence taken from an office scenario. The object moves w.r.t. the camera by following a 2^{nd}-order *random walk* motion model, obtained by adding Gaussian acceleration noise to a linear AR state system, with independent dynamics for each dof. In Fig. 3 some frames taken from the simulation are shown, and both the estimated contour and 3D object frame positions are displayed.

The results of the tracking are shown in Fig. 4, where the 6 trajectory estimation errors w.r.t. the real values are shown; the translation errors are measured in [*mm*], and the orientation errors are expressed by using the equivalent axis-angle representation, where the three components of the rotation error vector are given in [*deg*]. The *rms* estimation error over the whole sequence is lower than 2*mm* for the 2 planar translations, 7*mm* for the depth component, and 1*deg* for the rotations.

Subsequently, we tested the tracking system on real-life scenarios, and in Fig. 5 a few pictures from two real-time experiments are shown. The frames show the online tracking performance after the initialization under different conditions, such as partial occlusion, cluttered background, changing lighting, and widely different object positions; in particular, in the second sequence a superimposed 3D car model shows a nice example of Augmented Reality application, made possible by the system.

For all the experiments, a standard webcam with a resolution of 640x480 has been employed, without any preprocessing of the input images. The tracking system runs in real-time on a 3 GHz PC, and the processing speed of the tracking loop provides an overall frame rate of more than 30 fps.

VIII. Conclusions

We presented a robust real-time system for 3D object pose tracking in image sequences, which integrates an extended and optimized implementation of a powerful local curve fitting algorithm, and a robust global re-initialization module based on point feature matching.

The two main components were effectively combined together with a trajectory coherence and image correlation check modules, in order to achieve full automatic and real-time object tracking in real world situations. Simulation and experimental results have been presented, through a stand-alone implementation running at video frame rate on standard hardware.

Several available fields of interest have been considered for this kind of application; we mention here a few of them, like as: Augmented Reality (AR) applications, visual-guided robot manipulation, games, robot navigation (self-localization) and target tracking (e.g. ball/robot tracking in the RoboCup competition).

Future developments of the system include the integration into a more complex tracking framework, that will allow more complex applications like as body parts (faces, hands) or full-body tracking, automatic assembly procedures for mechanical parts, and many others as well.

REFERENCES

[1] G. Panin, A. Ladikos, and A. Knoll, "An efficient and robust real-time contour tracking system." in *2006 IEEE International Conference on Computer Vision Systems (ICVS), New York, NY, USA*, 2006, p. 44.

[2] A. Blake and M. Isard, *Active Contours: The Application of Techniques from Graphics,Vision,Control Theory and Statistics to Visual Tracking of Shapes in Motion*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1998.

[3] R. Hanek and M. Beetz, "The contracting curve density algorithm: Fitting parametric curve models to images using local self-adapting separation criteria," *International Journal of Computer Vision (IJCV)*, vol. 59, no. 3, pp. 233–258, 2004.

[4] M. Isard and A. Blake, "Condensation – conditional density propagation for visual tracking," *International Journal of Computer Vision (IJCV)*, vol. 29, no. 1, pp. 5–28, 1998.

[5] R. Hanek, T. Schmitt, S. Buck, and M. Beetz, "Fast image-based object localization in natural scenes," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2002*, ser. Lausanne, 2002, pp. 116–122.

[6] OpenCV Library. http://www.intel.com/technology/computing/opencv/.

[7] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[8] D. DeMenthon and L. S. Davis, "Model-based object pose in 25 lines of code," in *ECCV '92: Proceedings of the Second European Conference on Computer Vision*. London, UK: Springer-Verlag, 1992, pp. 335–343.

[9] T. K. Y. Bar-Shalom, X. Rong Li, *Estimation with applications to Tracking and Navigation*. John Wiley and Sons, Inc., 2001.

Giorgio Panin received the M.Sc. degree in Electronics Engineering in 1998 and the Ph.D. degree in Robotics in 2002 from the University of Genoa, Italy. He joined the European IST Project Artesimit in 2002 as postdoctoral fellow at the Informatics Faculty of the Technical University of Munich, Germany, funded under 5th FWP (Fifth Framework Programme), IST-2000-29689. Since January 2005 he is Assistant Professor at the Chair for Robotics and Embedded Systems of TU Munich. His current research interests include anthropomorphic robot hands control, design and implementation of advanced real-time computer vision systems and human-computer interfaces (HCI) design.

Alois Knoll received the diploma (M.Sc.) degree in Electrical Engineering from the University of Stuttgart, Germany, in 1985 and his PhD (summa cum laude) in computer science from the Technical University of Berlin, Germany, in 1988. He served on the faculty of the computer science department of TU Berlin until 1993, when he qualified for teaching computer science at a university (habilitation). He then joined the Technical Faculty of the University of Bielefeld, where he was a full professor, and the director of the research group Technical Informatics until 2001. In March 2001 he was appointed to the board of directors of the Fraunhofer-Institute for Autonomous Intelligent Systems. Since autumn 2001 he has been a professor of computer science at the Computer Science Department of the Technical University of Munich. His research interests include sensor-based robotics, multi-agent systems, data fusion, adaptive systems and multimedia information retrieval. In these fields he has published over eighty technical papers and guest-edited international journals. He has been part of (and has coordinated) several large scale national collaborative research projects (funded by the EU, the DFG, the DAAD, the state of North-Rhein-Westphalia). He initiated and was the program chairman of the First IEEE/RAS Conference on Humanoid Robots (IEEE-RAS/RSJ Humanoids2000), which took place at the Mass. Inst. of Technology (MIT) in September 2000. Prof. Knoll is a member of the German Society for Computer Science (Gesellschaft fr Informatik (GI)) and the IEEE.