A Unifying Software Architecture for Model-based Visual Tracking

Giorgio Panin, Claus Lenz, Martin Wojtczyk, Suraj Nair, Erwin Roth, Thomas Friedlhuber, Alois Knoll

Technical University of Munich, Informatics Faculty
Boltzmannstrasse 3, D-85748 Garching bei Muenchen, Germany

ABSTRACT

In this paper we propose a general, object-oriented software architecture for model-based visual tracking. The library is general purpose with respect to object model, estimated pose parameters, visual modalities employed, number of cameras and objects, and tracking methodology. The base class structure provides the necessary building blocks for implementing a wide variety of both known and novel tracking systems, integrating different visual modalities, like as color, motion, edge maps etc., in a multi-level fashion, ranging from pixel-level segmentation, up to local features matching and maximum-likelihood object pose estimation. The proposed structure allows integrating known data association algorithms for simultaneous, multiple object tracking tasks, as well as data fusion techniques for robust, multi-sensor tracking; within these contexts, parallelization of each tracking algorithm can as well be easily accomplished. Application of the proposed architecture is demonstrated through the definition and practical implementation of several tasks, all specified in terms of a self-contained description language.

Keywords: M odel-based Computer Vision, Object Tracking, Maximum-Likelihood Parameter Estimation, Data Association, Multi-modal Data Fusion, Object Modeling

1. INTRODUCTION

Visual tracking consists in integrating fast computer vision and image understanding techniques, together with sequential estimation methodologies (tracking), for the purpose of localizing one or more moving objects in a video sequence. More or less detailed models can be used for this task, and an efficient integration of all available information greatly enhances the quality of the estimation result.

Model-based visual tracking has application in many fields of interest, including robotics, man-machine interfaces, video surveillance, computer-assisted surgery, navigation systems, and so on. Recent surveys about the wide variety of techniques already proposed have been attempted in the literature.[1,2]

The aim of this paper is to present an object-oriented, unifying software architecture, with a twofold goal: on one side, to resume and provide some more insight about existing state-of-the-art visual tracking approaches as well as to develop new ones, within a common framework; and at the same time to aid designing, describing and specifying a tracking system in a self-contained fashion, by using a compact description vocabulary and a few specifications of related parameters (system configuration).

The present work is organized as follows: Section 2 presents a high-level description of the library, motivated by a general view of the problem; the basic visual processing tools and data structures are briefly discussed in Section 3, while the model-related layer of the architecture is presented in more details in Section 4; Section 5 focuses on the main tracking pipeline for a single target and single sensor task, organized in agents and data structures. Examples of individual systems instantiated within the proposed architecture are given in Section 6, along with related experimental results; conclusions and planned improvements are finally given in Section 7.

Image Processing: Machine Vision Applications, edited by Kurt S. Niel, David Fofi,
Proc. of SPIE-IS&T Electronic Imaging, SPIE Vol. 6813, 681303, © 2008 SPIE-IS&T · 0277-786X/08/$18

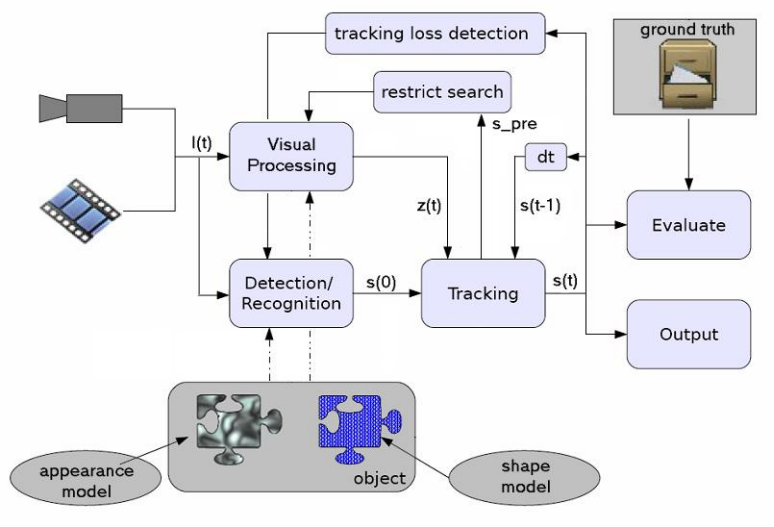

Figure 1. Single-target tracking flow diagram.

2. OVERVIEW OF THE TRACKING LIBRARY

In order to present the library, we start from a general discussion of model-based visual tracking. The general form considered here for a single-sensor, single-target system is resumed in Fig. 1.

The main information data, flowing within the system at time t, are:

- The sensor signal, I_t
- The measurement variable, z_t
- The object state, s_t

In a probabilistic framework, the tracking module follows the general *Bayesian tracking* scheme,[3,4] using Bayes' rule in order to update the state statistics s_t, by integrating the currently available measurement z_t together with the set Z^t of all measurement up to time t, resumed by the last estimation s_{t-1}.

This scheme, when extended to multiple objects, cameras and visual modalities, directly leads to develop a set of base classes that realize the desired functionalities in a *pipeline* fashion:

- A raw image (**sensor data**) is obtained from the cameras (**input sensors**)
- Images are processed in more or less complex ways (**visual processing** units), in order to provide more or less abstract **measurement** vectors z, of the most variable nature (**visual modalities**) and complexity: color blobs, motion fields, edge maps, LSE pose estimates, etc. There can be multiple processing units, running in parallel on the same input data, each one providing a single or multiple measurements $z_{i,t}$
- If required, simultaneous measurements from more cameras or modalities at each time can be synthesized in a "supermeasurement" Z_t before providing them to the tracker (**static measurement fusion**)
- In case of multiple objects and/or multiple measurements with uncertain origin, a **measurement-to-target** association mechanism is required, before updating the state.
- Fused/associated measurements are fed into **sequential estimators** (trackers) in order to update the **state** statistics of each independent object in a Bayesian framework (prediction-correction).
- Still in case of multiple cameras and/or visual modalities, the fusion process can take place after individual state updates, by means of a **dynamic fusion** mechanism (track-to-track).

- Predicted and updated states are provided as **output** for the user, for displaying the result and, if required, for providing an initial guess to the next visual processing step (**restrict search**).
- At the beginning ($t = 0$) and in case of any **track loss detection**, an optional **re-initialization** module is called, providing an external prior information for the Bayesian estimators.

In the above description, we put into evidence the on-line data and agents working within the pipeline; by taking into account also the *modeling* items which may be required for this complex task, we finally organize our structure in the following way, which will be detailed in the next Sections (Fig. 2)

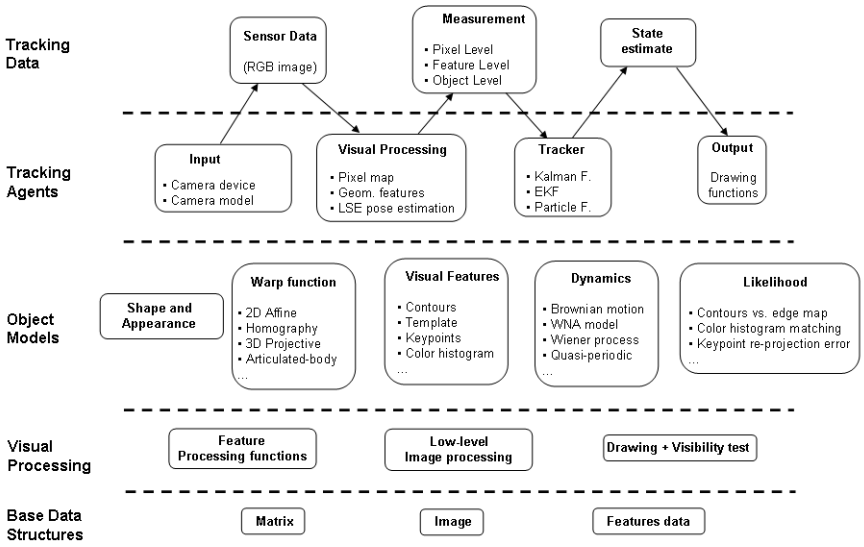

Figure 2. The library functional structure.

- Visual processing and data structures
  - Model-based vision algorithms
  - Image processing tools
  - Common data types (Matrix, Image, etc.) with internal manipulation and algebra functions
  - Specialized data structures, related to individual features (local keypoints, elliptical blobs, etc.)
- Object Models
  - Object shape and appearance
  - Warp function and pose parameters
  - Dynamic model
  - Visual features for tracking
  - Likelihood function (measurement model)
- Tracking Agents
  - Input device
  - Visual processing
  - Static/Dynamic measurement fusion
  - Measurement-to-target association

  - Sequential estimator (Tracker)
  - Output
- Tracking Data
  - Sensor data
  - Measurement
  - Object state

In what follows, the role and meaning of each layer and class will be detailed.

3. VISUAL PROCESSING AND DATA STRUCTURES

The first two layers include single image processing/understanding tools as well as general-purpose functions, together with the associated data structures.

Utility and general-purpose functions in these layers include matrix and image manipulation, algebra, visualization tools, visibility testing, least-squares optimization algorithms, and others as well.

Within the library, visual processing tools and data are organized in two main namespaces:

- Processing functions (model-free and model-based), operating at different levels: segmented pixel maps, geometric features detection and matching, and object pose estimation tools
- Processing data: storage structures for model and image features: invariant keypoints (location and descriptors), geometric primitives (lines, circles, elliptical blobs), etc.

Computer vision algorithms employ more or less generic model features, according to the required level of abstraction and specificity of the output: for example, color segmentation may need only a reference histogram in HSV space,[5] while invariant keypoints matching[6] require a database of reference features from one or more views of the object, and a contour-based LSE pose estimation[7] employs a full wireframe model of the shape.

For each algorithm, the output result is stored in a vector of specialized data structures: for example, a SIFT database[8] is stored as an array of structures, containing the 2D image coordinates and the respective invariant descriptors. These structures, together with more common types (Matrix, Image) constitute the base layer of Fig. 2.

4. MODEL LAYER

The object to be tracked, or the environment part (e.g. a ground-fixed marker), and the sensor device are modeled with respect of the most relevant aspects for tracking.

4.1 Ground object shape

For each object, a shape model can be of any kind, ranging from simple planar shapes like a rectangle, up to articulated structures with a skeleton, for modeling a human hand or a full body; the shape model can include rigid as well as deformable parts.

In a general form, the base shape can be fully represented or approximated by a base triangular mesh, moving an deforming in space according to the respective degrees of freedom. Articulated multi-body structures require additional parameters (e.g. Denavit-Hartenberg), specifying the intermediate frame locations and degrees of freedom between connected rigid bodies; this information can be stored as an additional skeleton model.

In this context, we call the base mesh and skeleton model also *ground shape* of the object, which is externally provided in a general-purpose CAD format (e.g. a VRML description).

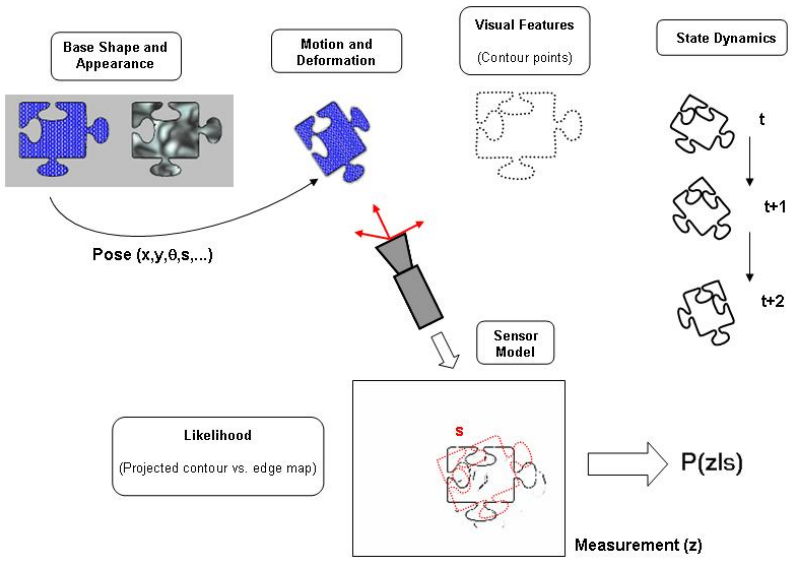

Figure 3. Model informations for tracking.

4.2 Warp function and pose parameters

Pose parameters specify the allowed (or expected) degrees of freedom of all body parts with respect to the ground shape, and they are included in the object model as well. Object motion and deformation is therefore compactly represented by a more or less large vector p.

The choice of pose parameters leads to the definition of the *warp* function, mapping points x from body space to sensor space y (image pixels), under a given hypothesis p:

$$y = W(x, p) \tag{1}$$

According to the choice of a calibrated or uncalibrated camera model (4) the parameter p may include, respectively, only the extrinsic part of the transformation (body-to-camera frame), or both extrinsic and intrinsic parameters.

Several examples of warps can be found in the literature, of variable complexity and dimensionality; many of them can be represented in homogeneous coordinates by the general form

$$\overline{y} = P(p)\,\overline{x} \tag{2}$$

with P a pose-dependent *projection matrix*.[9] Articulated models are described by multiple projection matrices $P(p, l)$ with l an integer index specifying the rigid body which x belongs to. And finally, more complex warps for deformable objects are piece-wise defined, with $P(p, x)$ depending also on the local point x.

In many situations, first derivatives of the Warp function w.r.t. the pose are also required

$$J(x, p) = \left.\frac{\partial W}{\partial p}\right|_{(x,p)} \tag{3}$$

where J is a $(2 \times N)$ Jacobian matrix, with N the number of degrees of freedom. Computation of J is included in this class as well, and usually available in exact (analytical) form.

4.3 Sensor model

In a general tracking system the sensing device is a physical transducer, obtaining from the environment a raw signal of variable nature (visual, acoustic, infrared, laser, thermal, and so on), which we call *sensor data*.

For calibrated CCD cameras, a *sensor model* is specified by the optical imaging parameters inside the intrinsic calibration matrix K_{int}.[9]

When available, this information is included into the Warp function, which in this case is split into an extrinsic and intrinsic mapping of the form

$$W(x, p) = K_{int} T_{ext}(p, x) x \tag{4}$$

with T_{ext} the object-to-camera transformation matrix.

4.4 Object appearance model

Together with the geometric model for shape and motion, many algorithms need also a model of the object appearance, specifying in more or less abstract terms how the object surface is expected to look like, from any given viewpoint.

An appearance model can as well be more or less complex and object-specific, ranging from very simple description of expected color statistics ("the surface at a given pose shows about a 70% green and 30% yellow color distibution"), to more detailed, point-wise information about the texture pattern, up to multiple appearance models, accounting also for light shading and reflectance effects (e.g. the AAM approach[10,11]).

Appearance models can generally be specified through one or more *reference images* of the object of interest, possibly together with texture coordinates of the base mesh.

If the appearance model includes only color or texture statistics information, the reference images are directly used in order to build the statistics (for example, an histogram in HSV color space), which is used as visual feature for tracking.[5]

When the appearance model includes a precise texture map, the reference images can be rendered by using standard techniques (e.g. OpenGL) and more specific model features can be obtained.

4.5 Visual features for tracking

The visual features class contains all of the model information required by a given modality. Examples range from reference histograms in color space to local invariant keypoints (e.g. SIFT[8]), model contour segments, a large spatial distribution of textured surface points (template), and so on.

A visual feature set for a given modality always consists of an array of geometric primitives, each one specified by a representative point in object space, together with a more or less specific *descriptor*, that may consist of a local grey-level pattern, a segment position and orientation, an elliptical blob size and principal axis, etc.

Visual features are warped onto the sensor space under a pose hypothesis p, and matched with corresponding features from the image I, in order to perform a matching task, to evaluate a probabilistic measure for tracking (*Likelihood* function), or to directly maximize the likelihood in pose-space.

A database of visible-from-camera features is automatically obtained from the object ground shape and appearance, and it can be pose-dependent: in fact, for some systems[6] reference features can be pre-computed and updated only after large rotational displacements of the camera view, while in other situations[7] a visibility test and features selection must be performed at each pose hypothesis.

In a tracking context, we also distinguish between *off-line* and *on-line* model features, where the latter are updated directly from the image stream under the estimated pose, while the former are obtained from the ground shape and appearance model only. Combining both informations generally leads to a better stability and robustness for tracking, as demonstrated for example in Ref. 12 for the case of local keypoints.

5. THE TRACKING PIPELINE (AGENTS AND DATA)

5.1 Visual processing and measurement

The input image is processed in more or less complex, eventually model-based ways, in order to obtain a more refined information about the target, called measurement variable z.

As it can be seen from the literature, there are many different levels of complexity at which the visual processor can operate.

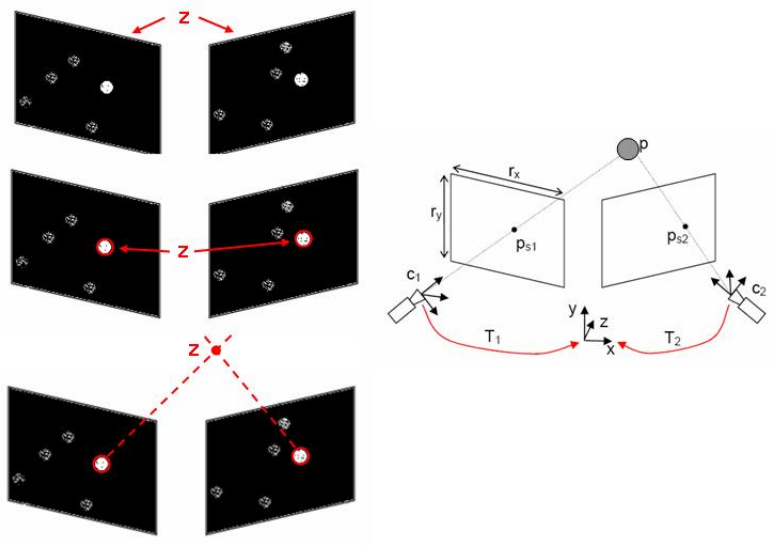

Figure 4. Three possible visual processing levels for the same tracking task.

In our framework, we decide to classify them in three most significant categories, according to the degree of abstraction of the resulting output z (see the example in Fig. 4).

1. *Pixel-level*: the raw image itself, or any kind of processing that produces information on a pixel basis, like as a binary color segmentation, a point-wise motion field, a pixel-wise edge map (e.g. the output of a Canny algorithm), etc.

2. *Feature-level*: the visual processor detects primitive shapes (connected segments, curves, elliptical blobs, up to complex invariant keypoints) in the image, matches them to the model features, and provides the result in a variable dimension array z

3. *Object-level*: z is a direct estimate of the object in pose-space p^*, obtained through an optimization algorithm (typically a nonlinear least-squares estimation), which also requires the model features in order to be performed. We observe here that the optimization of a cost functional C is in most cases equivalent to a Maximum-Likelihood approach applied to a Gibbs distribution:[13]

$$z_t = p^* = \min_p C(p, I_t) = \max_p \left[\exp\left(-\lambda C(p, I_t)\right)\right] \tag{5}$$

   In this case, we need to distinguish between this Likelihood function $P(I_t|\,p)$ and the tracker Likelihood $P(z|\,s)$ defined next.

Generally speaking, the last processing type is also the most complex one and the most computationally expensive, usually involving a nonlinear multivariate optimization problem, eventually requiring Jacobian matrices. For this purpose, more general and robust similarity functions, other than standard LSE, are given by M-estimators,[14] Normalized Cross Correlation (NCC)[15] or Mutual Information.[16] The result of an object-level measurement is also the most compact one, and directly commensurable with the object pose ($z \sim p$).

5.2 Sequential estimation module (Bayesian tracker)

Here the measurement variable z_t is employed in order to update the current state estimate s_t through a Bayesian prediction-correction scheme. This scheme can be here implemented with a Kalman Filter,[17] an Extended Kalman Filter or a Particle Filter,[18] following the general Bayesian tracking scheme[3]

$$P\left(s_t|\, Z^t\right) = kP\left(z_t|\, s_t\right) \int_{s_{t-1}} P\left(s_t|\, s_{t-1}\right) P\left(s_{t-1}|\, Z^{t-1}\right) \tag{6}$$

where $P\left(z_t|\, s_t\right)$ is the current measurement Likelihood, $P\left(s_t|\, s_{t-1}\right)$ is the object dynamical model, and k a normalization term.

In a single-target, single-measurement scenario, only one tracker and one visual processor operate during the whole sequence; in more general multitarget/multisensor environment multiple, parallel instances of each class are present, and additional data association/fusion modules are required in order to solve the target-to-measurement association issue, as well as to exploit redundant informations from the measurement set.[19]

For Bayesian tracking, two additional models are required.

5.2.1 Dynamical model

The object state vector s normally includes pose parameters p, possibly together with first or second time derivatives (velocity, acceleration), all referred to the camera frame. Object dynamics are usually specified by means of a time-dependent prediction function

$$s_t = f(s_{t-1}, \Delta t, w_t) \tag{7}$$

where w_t is a random variable with known distribution (process noise). This form is equivalent to the dual form $P\left(s_t|\, s_{t-1}\right)$.

The dynamic model is used in the Extended Kalman Filter in order to obtain the predicted state

$$s_t^- = f(s_{t-1}, \Delta t, 0) \tag{8}$$

together with its Jacobian matrices

$$F_t = \left.\frac{\partial f}{\partial s}\right|(s_{t-1}, \Delta t, 0); \quad W_t = \left.\frac{\partial f}{\partial w}\right|(s_{t-1}, \Delta t, 0) \tag{9}$$

for covariance prediction.

The same form (7) is also used in Particle Filters for simulating the *drift* process, by explicitly generating the random component w_t for Monte-Carlo sampling of the particle set.

A pre-defined set of standard linear models[20] is built inside the library; all of these models can be used with any pose representation (2D, 3D, articulated body, etc.), and the user can introduce a custom dynamical model, always provided in the abstract form (7).

5.2.2 Likelihood model

The Likelihood function of the tracker is defined as a probabilistic measure of fit between the measured variable z_t associated to a given target, and a state hypothesis s_t. It is expressed as well in one of two equivalent forms:

- As a proability distribution $P(\, z_t|\, s_t)$, used by Particle Filters in order to update the particle weights.
- As a measurement function $z_t = h(s_t, v_t)$ with random component v_t of known statistics (measurement noise); this form, with v_t a Gaussian noise, is appropriate for an (E)KF implementation.

Since the measurement can be of a very different type with respect to the state, the Likelihood model can also have a much variable complexity, which is again related to the processing level used for tracking, (object-feature- or pixel-level) and the visual modality employed (contour lines, points, color, etc.).

Generally speaking, as we can see a higher complexity of the visual processor is compensated by a lower complexity of the tracker, and vice-versa.

In the Computer Vision literature, many likelihood models have been proposed and, as one can see from the previous description, for the same problem different choices are possible, which are left to the experience of the designer.

6. APPLICATION EXAMPLES AND EXPERIMENTS

In this Section, we provide a set of examples of individual tracking systems that have been instantiated using our architecture, by means of a self-contained description language, directly referring to the classes depicted in Fig. 2.

All of the experiments have been performed on a Desktop PC with a dual-core 3GHz Intel processor and a programmable NVidia GPU, using standard FireWire cameras. The library is also platform-independent, and all of the experiments have been run with the same performances on both Linux and Windows operating systems.

As a first case, we describe a Condensation contour tracker[21] in 2D space. This corresponds to the following instances of base classes:

- **Task Description**: 2D contour-based object tracking using Particle Filters and the Canny edge map
- **Models**
  - *Shape and Appearance*: wireframe CAD model, without texture
  - *Visual Features*: a set of points (x, y) sampled over the external model edges (silhouette), with companion point indices $(i \rightarrow i+1)$[22] for computing the image normals on-line
  - *Warp Function*: 2D planar roto-translation and scale $y = aR(\theta)\, x + b$
  - *State Representation*: 4 pose parameters $p = [a, \theta, b_x, b_y]$ and first time derivatives
  - *Object Dynamics*: Constant velocity model (also called Continuous White Noise Acceleration[20])
- **Tracking Pipeline**
  - *Sensor*: Color camera with given pixel resolution, and uncalibrated acquisition model
  - *Visual Processing and Measurement*: Canny edge detection, providing a pixel-level measurement $z =$ the binary edge map
  - *Likelihood Model*: $P(z|s) =$ Project the visual features at pose hypothesis s, search corresponding points onto the edge map along the normals, and compute a multi-modal contour Likelihood[21]
  - *Bayesian Tracker*: SIR Particle Filter, with deterministic resampling; single-target, single-modality, without data association
  - *Output*: Compute the sample average of the particle set $\overline{s} = (\overline{p}, \overline{v})$, and display the projected CAD model at estimated pose $\overline{p}$
  - *Tracking Loss Detection*: Compute the sample covariance $\overline{\Sigma}$ and check against a threshold value $\det\left(\overline{\Sigma}\right) < d$; in case of loss, call the re-initialization procedure
  - *Re-initialization*: Provide an initializing particle set, by sampling from a prior distribution $P(s_0)$; for example, a large Gaussian around the image center, with suitable covariances for the 4 pose parameters and velocities.

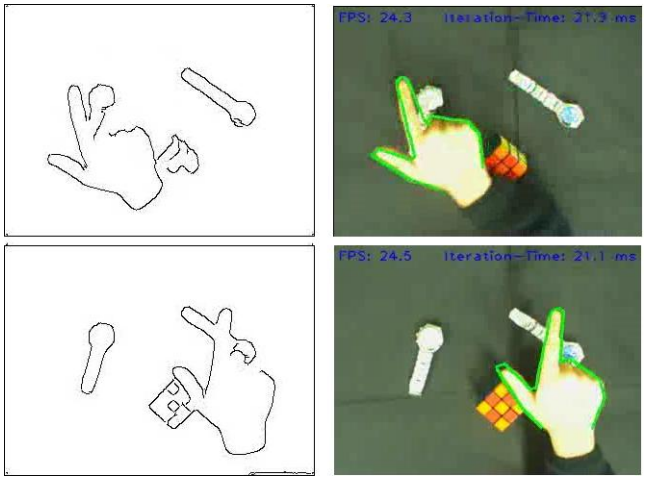

Figure 5. Contour-based Particle Filter for planar hand tracking.

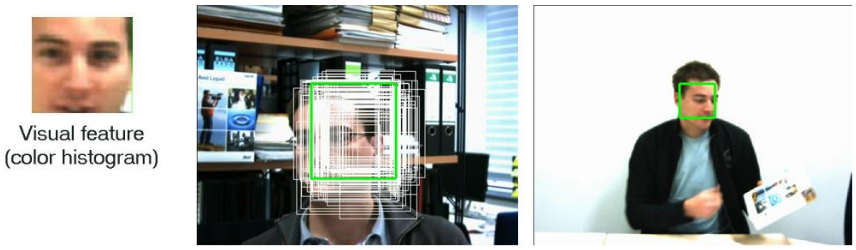

Figure 6. Color-based particle filter.

Fig. 5 shows some results obtained with the above described system, for planar hand tracking; in this application, a frame rate of 30 fps is obtained, where a set of 200 particles has been used, and each warp function and likelihood computation does not require any visibility test.

Other examples of real-time tracking applications realized through our library are synthetically presented in the following.

Fig. 6: a color-based Particle Filter[5] using a single reference image (appearance model), and a simple rectangular shape with planar warp (translation and scale). The visual feature is given by a two-dimensional reference histogram in HSV color space, where the first two channels (H,S) only are collected from the appearance image. Tracking is realized by means of single-target SIR Particle Pilter, by projecting the contour shape onto the color image (pixel-level measurement z) and computing the likelihood as the Bhattacharyya distance between expected and observed color histograms.[5] Dynamics are modeled as simple Brownian motion, with a pose-only state vector.

Fig. 7: a Kalman Filter tracker integrating motion and color; pose parameters are two-dimensional (x, y); two visual processing modules act in parallel, providing motion segmentation with the motion history image,[23] and color segmentation using the Fisher linear discriminant vector[24] in RGB color space. Both measurements z_1, z_2 are object-level, computed as the mass center of the respective segmented images. Dynamic data fusion[19,25] is performed here by stacking the two measurements in a supermeasurement

$$Z^t = \left[z_1^t, z_2^t\right]^T$$

which is fed into a standard Kalman Filter for updating the target state s_t.

Fig. 8: frame-by-frame 3D pose estimation of a planar object, by matching invariant keypoints against a single appearance model. The measurement for tracking is object-level, obtained after a nonlinear LSE estimation of Euclidean roto-translation parameters, represented as a 6-dimensional Euclidean *twist*,[26] with calibrated

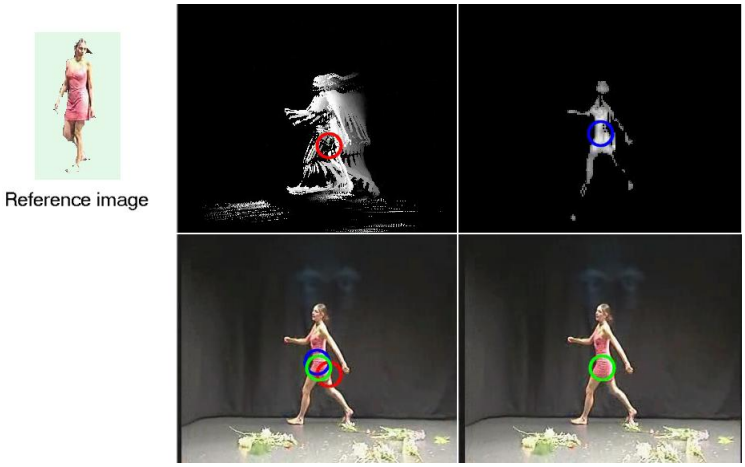

Figure 7. Integrating color and motion through dynamic data fusion.

Figure 8. 3D Pose estimation with invariant keypoints - GPU-based implementation.

camera model. In this context, the tracker is a standard Kalman Filter with constant velocity dynamics. The implementation of invariant keypoints detection and matching[8] is done for speed purposes on the GPU, following the paper in Ref. 27. This example also shows one multi-purpose hardware capability of the library.

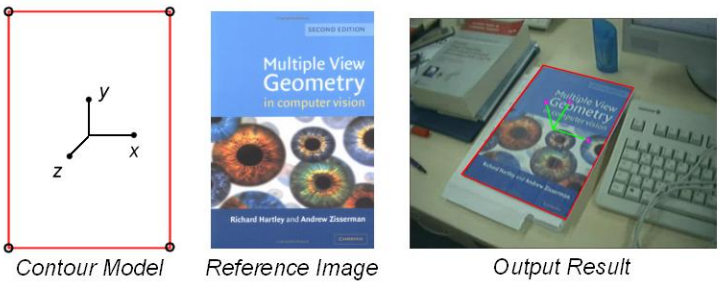

Figure 9. Contour-based tracking by optimizing local color statistics.

Fig. 9: Color-based contour tracking with keypoint re-initialization. The contour tracker employs the color-based CCD algorithm,[28] implemented in the real-time version,[29] providing a frame-rate of 25fps; as described in Ref. 29, re-initialization in provided through keypoint detection and pose estimation, and track loss is detected by computing an NCC index between off-line appearance and warped image pixels at the currently estimated

pose. Measurements are object-level pose parameters, feeded into a Kalman Filter with CWNA motion model for state estimation. The camera model is calibrated, and pose parameters are represented through the twist vector.

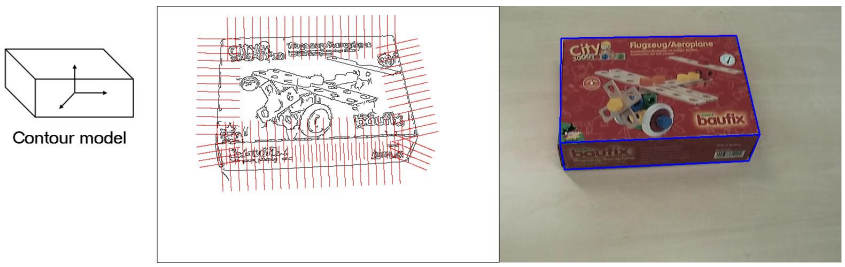

Figure 10. Object tracking with edge-based least-squares pose estimation.

Fig. 10: LSE edge-based tracking of 3D objects. This system corresponds to the well-known object tracker described in,[7,22] based on local least squares optimization of edge features in 3D space; as in the previous example, an object-level measurement $z_t = p_t^*$ is performed, by maximizing the standard (single-hypothesis) contour Likelihood function shown in the picture on the left side, starting from the predicted pose hypothesis p_t^-; for this purpose, only nearest edge points are detected along the normals in order to evaluate the overall error and its derivatives. 3D pose is represented again by a twist vector, and a Kalman Filter is employed for tracking.

Figure 11. Template-based tracking by Mutual Information maximization.

Fig. 11: Template tracking by maximizing Mutual Information as robust similarity index, using a single reference appearance under illumination changes; full details of the pose estimation algorithm are described in.[16] The estimated object pose is again fed into a twist-based Kalman filter for tracking.

Fig. 12: Face tracking in 3D integrating contour and template features.[30] This example shows a complimentary data fusion,[19] where the two visual modalities provide two disjoint parts of the full pose vector, namely 3D translation (contour) and rotation (template). The head contour is first estimated through the CCD algorithm above mentioned, using an elliptical model with 3 translational degrees of freedom; head rotations are subsequently estimated with fast Mutual Information template matching, working with the remaining degrees of freedom.

Both features for tracking are off-line obtained from the full textured 3D mesh provided by the user.

7. CONCLUSION AND PLANNED IMPROVEMENTS

We presented a general architecture for model-based visual tracking with the purpose of casting the problem in a common object-oriented framework, where a minimal number of classes and layers show to be sufficient for entailing a wide variety of existing approaches, as well as for developing new ones. At the same time, the structure allows multiple intances of base classes to work in parallel for distributed tracking in multi-camera/modality/object tracking problems.

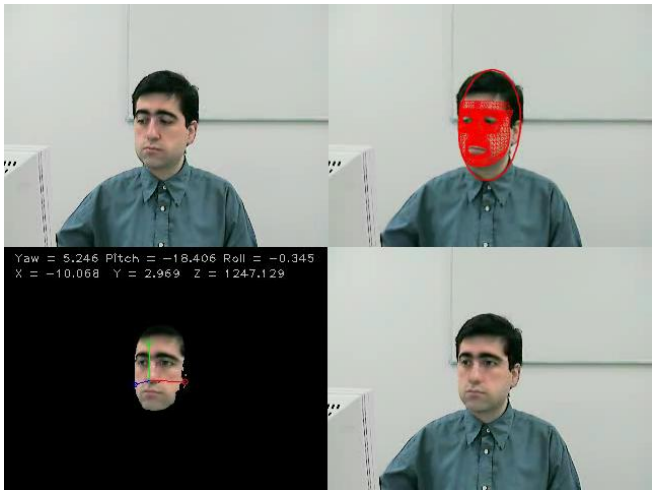

Figure 12. 3D Face tracking integrating contour and template visual modalities.

In this framework, off-line models of the object shape, appearance, deformation and temporal dynamics need to be externally provided, although in a common format throughout different tasks. Future developments of the architecture are planned in the direction of building in fully- or semi-automatic ways the models required: for example, 3D shape estimation from motion,[9] automatic contour modeling,[31] appearance model training,[11] and so on. The additional set of tools (generic recognition, classification, shape morphing, etc.) for object modeling can be added to the base structure of Fig 2 as part of the model layer.

Another issue concerns the possibility of on-line adaptation of model features (template, local keypoints, etc.), since for more robustness and adaptivity,[12] it can be desired to update the appearance model directly from the image stream I_t. This corresponds to an additional set of "model processing" tools, whose input is the current image I_t and estimated state s_t information, where the image is *back-warped* into the object space, and the on-line appearance model is updated accordingly; off-line and on-line features can then be easily combined either within the Likelihood model, or the cost function to be optimized.

REFERENCES

1. A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," *ACM Comput. Surv.* **38**(4), p. 13, 2006.
2. V. Lepetit and P. Fua, *Monocular Model-based 3d Tracking of Rigid Objects (Foundations and Trends in Computer Graphics and Vision(R))*, Now Publishers Inc, 2005.
3. L. D. Stone, T. L. Corwin, and C. A. Barlow, *Bayesian Multiple Target Tracking*, 1st. Artech House, Inc., 1999.
4. S. S. Blackman and R. Popoli, *Design and Analysis of Modern Tracking Systems*, Artech House Radar Library, 1999.
5. P. Pérez, C. Hue, J. Vermaak, and M. Gangnet, "Color-based probabilistic tracking," in *ECCV '02: Proceedings of the 7th European Conference on Computer Vision-Part I*, pp. 661–675, Springer-Verlag, (London, UK), 2002.
6. I. Skrypnyk and D. G. Lowe, "Scene modelling, recognition and tracking with invariant image features," in *ISMAR '04: Proceedings of the Third IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'04)*, pp. 110–119, IEEE Computer Society, (Washington, DC, USA), 2004.
7. T. Drummond and R. Cipolla, "Real-time visual tracking of complex structures," *IEEE Trans. Pattern Anal. Mach. Intell.* **24**(7), pp. 932–946, 2002.
8. D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision* **60**(2), pp. 91–110, 2004.
9. R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, Cambridge University Press, ISBN: 0521540518, second ed., 2004.

10. T. F. Cootes, G. J. Edwards, and C. J. Taylor, "Active appearance models," *Lecture Notes in Computer Science* **1407**, pp. 484–498, 1998.
11. I. Matthews and S. Baker, "Active appearance models revisited," Tech. Rep. CMU-RI-TR-03-02, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, April 2003.
12. L. Vacchetti and V. Lepetit, "Stable real-time 3d tracking using online and offline information," *IEEE Trans. Pattern Anal. Mach. Intell.* **26**(10), pp. 1385–1391, 2004.
13. S. Roth, L. Sigal, and M. J. Black, "Gibbs likelihoods for bayesian tracking," *cvpr* **01**, pp. 886–893, 2004.
14. P. Huber, *Robust Statistics*, Wiley, New York, 1981.
15. J. Lewis, "Fast normalized cross-correlation," in *Vision Interface*, pp. 120–123, Canadian Image Processing and Pattern Recognition Society, 1995.
16. G. Panin and A. Knoll, "Mutual information-based 3d object tracking," *submitted to the International Journal of Computer Vision* , 2007.
17. G. Welch and G. Bishop, "An introduction to the kalman filter," tech. rep., 2004.
18. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for on-line non-linear/non-gaussian bayesian tracking," *IEEE Transactions on Signal Processing* **50**, pp. 174–188, February 2002.
19. Y. Bar-Shalom and X.-R. Li, *Multitarget-Multisensor Tracking: Principles and Techniques*, YBS Publishing, 1995.
20. Y. Bar-Shalom, T. Kirubarajan, and X.-R. Li, *Estimation with Applications to Tracking and Navigation*, John Wiley and Sons, Inc., New York, NY, USA, 2002.
21. M. Isard and A. Blake, "Condensation – conditional density propagation for visual tracking," *International Journal of Computer Vision (IJCV)* **29**(1), pp. 5–28, 1998.
22. C. Harris, "Tracking with rigid models," pp. 59–73, 1993.
23. G. R. Bradski and J. W. Davis, "Motion segmentation and pose recognition with motion history gradients," *Mach. Vision Appl.* **13**(3), pp. 174–184, 2002.
24. C. M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, Oxford, 1995.
25. G. Welch and G. Bishop, "Scaat: incremental tracking with incomplete information," in *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pp. 333–344, ACM Press/Addison-Wesley Publishing Co., (New York, NY, USA), 1997.
26. R. M. Murray, Z. Li, and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation*, CRC, March 1994.
27. G. Ziegler, A. Tevs, C. Theobalt, and H.-P. Seidel, "Gpu point list generation through histogram pyramids," Research Report MPI-I-2006-4-002, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany, June 2006.
28. R. Hanek and M. Beetz, "The contracting curve density algorithm: Fitting parametric curve models to images using local self-adapting separation criteria," *Int. J. Comput. Vision* **59**(3), pp. 233–258, 2004.
29. G. Panin, A. Ladikos, and A. Knoll, "An efficient and robust real-time contour tracking system," in *ICVS*, p. 44, 2006.
30. G. Panin and A. Knoll, "Real-time 3d face tracking with mutual information and active contours," in *International Symposium on Visual Computing (ISVC)*, (Lake Tahoe, Nevada, USA), 2007.
31. 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*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1998.