ITrackU: An Integrated Framework for Image-based Tracking and Understanding

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

***Abstract*— Within the ITrackU project, a modular software architecture for model-based visual tracking and image understanding is being developed. The library is general-purpose with respect to object models, state-space parameters, visual modalities employed, number of cameras and targets, and tracking methodology. This provides the necessary building blocks for a seamless integration of a wide variety of both known and novel tracking systems, involving 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. Application of the proposed architecture is demonstrated through the definition and practical implementation of relevant tasks for the CoTeSys cluster, all specified in terms of a self-contained description language.**

I. 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].

The aim of this paper is to present an object-oriented, unifying software architecture, whose main goal is a seamless integration of existing state-of-the-art visual tracking approaches, as well as to develop new ones, within a common framework. In this way designing, describing and specifying any tracking system can be done 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.

Authors affiliation:
Technische Universität München, Fakultät für Informatik
Boltzmannstrasse, 3 - 85748 Garching bei München.
Email: {panin,eroth,roeder,nair,wojtczyk,firedlhu,knoll}@in.tum.de

II. 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.

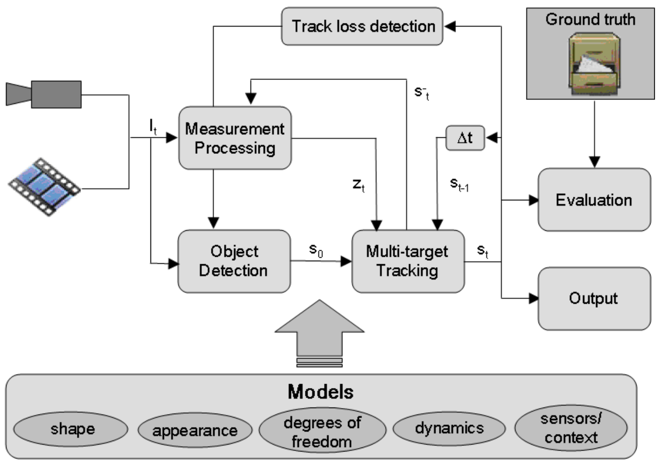

Fig. 1. Single-target tracking flow diagram.

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 [2], 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 a more or less abstract **measurement** z, of the most variable nature (**visual modalities**) and complexity: color blobs, motion fields, edge maps, LSE pose estimates, etc.
- In case of multiple objects and/or multiple measurements with uncertain origin, a **data association** mechanism is required, before updating the state.
- Simultaneous measurements from multiple cameras or modalities, associated to the same target, have to be integrated in some way before providing them to the tracker (**data fusion**)
- 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).
- 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.

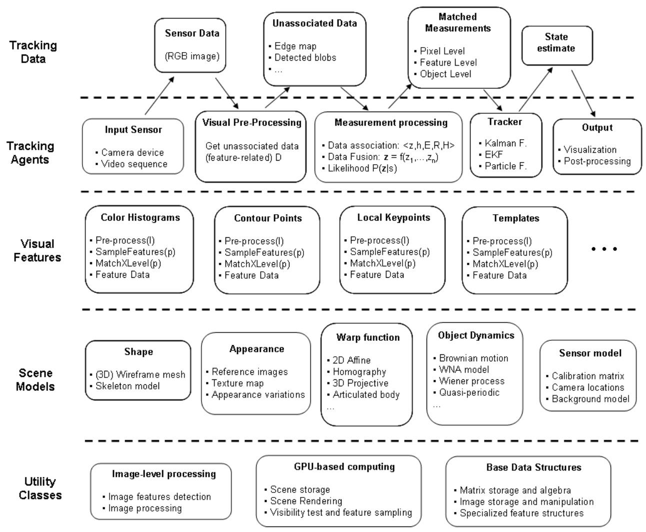

Fig. 2. The library functional structure.

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 as depicted in Fig. 2, and detailed in the next Sections.

III. 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 [3], while invariant keypoints like SIFT [4] require a database of reference features from one or more views of the object, and a contour-based LSE pose estimation [5] 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 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.

IV. 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.

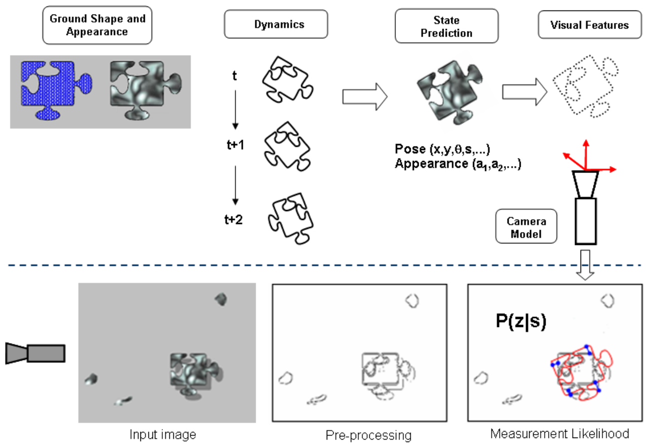

Fig. 3. Model informations for tracking.

A. 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).

B. 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:

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

In a generic 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, the *sensor model* is specified by the optical imaging parameters inside the intrinsic calibration matrix K_{int} [6]. Therefore, the warp function can be written in homogeneous coordinates as

$$\bar{y} = K_{int} T_{ext}(p) \bar{x} \tag{2}$$

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

According to the choice of a calibrated or uncalibrated camera model (2), as well as the dimensionality (2D or 3D) of the pose, both matrices T, K can be differently specified.

Articulated models are described by piece-wise defined transformations $T(p, l)$ with l an integer index specifying the rigid body which x belongs to. And finally, more complex warps for deformable objects are point-wise defined, with $T(p, x)$ 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.

C. 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 [7], [8]).

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 [3].

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.

D. 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 [4]), 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 the reference features can be pre-computed and updated only after large rotational displacements of the camera view, while in other situations [5] 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 [9] for the case of local keypoints.

V. The Tracking Pipeline (Agents and Data)

A. 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. In our framework, we classify them in three most significant categories, according to the degree of abstraction of the resulting output z:

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 [10]:

$$z_t = p^* = \min_p C(p, I_t) = \max_p [\exp(-\lambda C(p, I_t))] \quad (4)$$

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

B. 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 [11], an Extended Kalman Filter or a Particle Filter [12], following the general Bayesian tracking scheme

$$P\left(s_t|\, Z^t\right) = kP\left(z_t|\, s_t\right) \int_{s_{t-1}} P\left(s_t|\, s_{t-1}\right) P\left(s_{t-1}|\, Z^{t-1}\right) \quad (5)$$

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 [13].

For Bayesian tracking, two additional models are required.

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

$$s_t = f(s_{t-1}, \Delta t, w_t) \quad (6)$$

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

$$s_t^- = f(s_{t-1}, \Delta t, 0) \quad (7)$$

together with its Jacobian matrices

$$F_t = \left.\frac{\partial f}{\partial s}\right| (s_{t-1}, \Delta t, 0); \quad W_t = \left.\frac{\partial f}{\partial w}\right| (s_{t-1}, \Delta t, 0) \quad (8)$$

for covariance prediction.

The same form (6) 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 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 (6).

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 propability 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.

VI. 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 2.13GHz 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 [14] in 2D space. This corresponds to the following items in our library:

A basic shape model, without appearance, is provided in a CAD format. Visual features are obtained as a set of points (x, y) sampled over the external model edges (silhouette), with companion point indices $(i \rightarrow i+1)$ [15] for computing the image normals on-line.

Pose parameters are given by 2D planar roto-translation and scale $y = aR(\theta)x + b$, and first time derivatives are included in the state model. Object dynamics are modeled by a constant velocity with Gaussian random acceleration. Pre-processing is done by a Canny edge detection, providing a pixel-level measurement; by searching for corresponding edge points to the predicted contour along the normals, a multi-modal Likelihood is computed (see [14] for more details). With this model, a SIR Particle Filter with deterministic resampling is initialized and run. As output, the sample average of the particle set $\overline{s} = (\overline{p}, \overline{v})$ is computed, and displayed on screen Re-initialization of the filter is provided by a prior particle set $P(s_0)$ widely distributed around the image center, with suitable covariances for the 4 pose parameters and velocities.

Fig. 4 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.

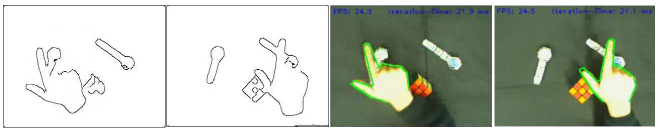

Fig. 4. Contour-based Particle Filter for planar hand tracking.

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

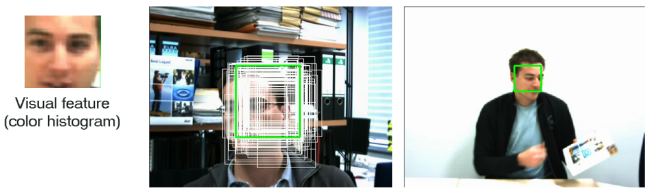

Fig. 5. Color-based particle filter.

Fig. 5: a color-based Particle Filter [3] 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 Filter, 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 [3]. Dynamics are modeled as simple Brownian motion, with a pose-only state vector.

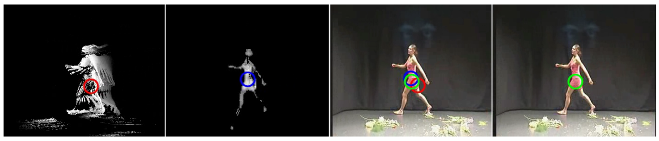

Fig. 6. Integrating color and motion through dynamic data fusion.

Fig. 6: 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 [16], and color segmentation using the Fisher linear discriminant vector [17] 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 [13] 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. 7. 3D Pose estimation with invariant keypoints - GPU-based implementation.

Fig. 7: 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* [18], with calibrated camera model. In this context, the tracker is a standard Kalman Filter with constant velocity dynamics. The implementation of invariant keypoints detection and matching [4] is done for speed purposes on the GPU, following the paper in [19]. This example also shows one multi-purpose hardware capability of the library.

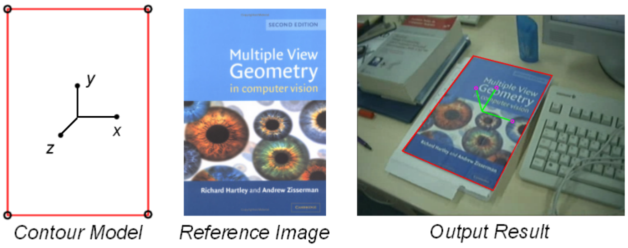

Fig. 8. Contour-based tracking by optimizing local color statistics.

Fig. 8: Color-based contour tracking with keypoint re-initialization. The contour tracker employs the color-based CCD algorithm [20], implemented in the real-time version [21], providing a frame-rate of 25fps; as described in [21], re-initialization is 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, fed 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.

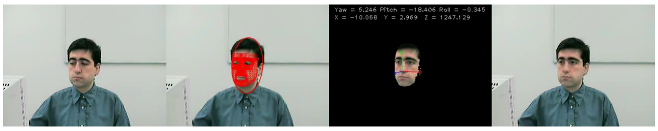

Fig. 9. 3D Face tracking integrating contour and template visual modalities.

Fig. 9: Face tracking in 3D integrating contour and template features [22]. This example shows a complimentary data fusion [13], 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.

VII. 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 instances of base classes to work in parallel for distributed tracking in multi-camera/modality/object tracking problems.

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 [6], automatic contour modeling [23], appearance model training [8], 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.

References

[1] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," *ACM Comput. Surv.*, vol. 38, no. 4, p. 13, 2006.

[2] S. S. Blackman and R. Popoli, *Design and Analysis of Modern Tracking Systems*. Artech House Radar Library, 1999.

[3] 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*. London, UK: Springer-Verlag, 2002, pp. 661–675.

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

[5] T. Drummond and R. Cipolla, "Real-time visual tracking of complex structures," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 7, pp. 932–946, 2002.

[6] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, ISBN: 0521540518, 2004.

[7] T. F. Cootes, G. J. Edwards, and C. J. Taylor, "Active appearance models," *Lecture Notes in Computer Science*, vol. 1407, pp. 484–498, 1998.

[8] I. Matthews and S. Baker, "Active appearance models revisited," Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-03-02, April 2003.

[9] L. Vacchetti and V. Lepetit, "Stable real-time 3d tracking using online and offline information," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 10, pp. 1385–1391, 2004.

[10] S. Roth, L. Sigal, and M. J. Black, "Gibbs likelihoods for bayesian tracking," *cvpr*, vol. 01, pp. 886–893, 2004.

[11] G. Welch and G. Bishop, "An introduction to the kalman filter," Tech. Rep., 2004.

[12] 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*, vol. 50, no. 2, pp. 174–188, February 2002.

[13] Y. Bar-Shalom and X.-R. Li, *Multitarget-Multisensor Tracking: Principles and Techniques*. YBS Publishing, 1995.

[14] 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.

[15] C. Harris, "Tracking with rigid models," pp. 59–73, 1993.

[16] G. R. Bradski and J. W. Davis, "Motion segmentation and pose recognition with motion history gradients," *Mach. Vision Appl.*, vol. 13, no. 3, pp. 174–184, 2002.

[17] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford: Oxford University Press, 1995.

[18] R. M. Murray, Z. Li, and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation*. CRC, March 1994.

[19] G. Ziegler, A. Tevs, C. Theobalt, and H.-P. Seidel, "Gpu point list generation through histogram pyramids," Max-Planck-Institut für Informatik, Saarbrücken (Germany), Research Report MPI-I-2006-4-002, June 2006.

[20] 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*, vol. 59, no. 3, pp. 233–258, 2004.

[21] G. Panin, A. Ladikos, and A. Knoll, "An efficient and robust real-time contour tracking system," in *ICVS*, 2006, p. 44.

[22] 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.

[23] 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.