A GPU-accelerated particle filter with pixel-level likelihood

Claus Lenz, Giorgio Panin, Alois Knoll

Robotics and Embedded Systems Lab
Computer Science Department
Boltzmannstrasse 3
85748 Garching bei München
Technische Universität München

Abstract

We present in this paper a GPU-accelerated particle filter based on pixel-level segmentation and matching, for real-time object tracking. The proposed method achieves real-time perfomance, while computing for each particle the corresponding filled model silhouette through the rendering engine of the graphics card, and comparing it with the underlying binary map of the segmentation preprocess. With the proposed approach, a better precision and generality is obtained with respect to related feature-level likelihoods such as color histograms, while keeping low computational requirements.

1 Introduction

Many examples of particle filters for single and multiple object tracking are well-known in the literature. Most of these systems use *feature-level* likelihoods, where model features (color statistics [16], contour points [12], etc.) are selected and matched with the current image under a given state hypothesis, with more or less robust likelihood models.

In particular, these filters achieve different degrees of precision and robustness, as well as tracking speed, according to the specifity of the feature being measured.

For real-time applications, simple statistics including color histograms are usually preferred [4, 14, 16], and provide good results for 2D problems where a precise shape localization is not required: usually a rough estimation of translation and scale parameters is obtained by using a rectangular or elliptical model, and planar rotations are not always estimated.

One reason for this limitation is that local statistics (such as histograms, GMMs, etc.) provide rather artificial information, obtained by accumulating individual contributions from all pixels inside the area.

Instead, more specific likelihoods can be defined when the information is compared at pixel-level: for example, after performing a binary image segmentation, the object shape can be projected and matched pixel-wise with the underlying map, by defining a distance function (SSD, or more robust indices) between images. Such an approach can provide a better localization, and more degrees of freedom (including planar rotations) can be estimated. Related work that has been done in the research field of motion and pose estimation using silhouettes on *feature-level* can be found in [18, 10, 3].

In the context of this work, image pre-processing can be performed with any modality (color, background, motion segmentation) with the output likelihood defined as above. Moreover, multiple modalities can be combined at pixel-level, by using a suitable decision methodology such as voting, fuzzy or Bayesian rules [11].

However, in a particle filter, such an approach involves computing and matching the filled *model silhouette* for each pose hypothesis, which can be extremely time-consuming for generic object shapes if performed on the CPU.

In this work, we have implemented the above computation on graphics hardware, where the execution time is much faster and almost object-independent. A well-known bottleneck of current GPUs is given by the back-transfer of the data on the main bus; therefore, we also compute on-board the image likelihood and transfer back the result to the CPU memory, in order to update the particle weights.

The proposed system achieves real-time performances regardless of the shape complexity (2D or

3D mesh), due to the rendering capabilities of current graphics hardware.

The present paper is organized as follows: Sec. 2 describes the particle filter (2.1), the pre-processing step (2.2), the object state parameters (2.3), the pixel-level likelihood (2.4), and the full software implementation (2.5); Sec. 3 provides experimental results and comparisons with an equivalent CPU implementation, and Sec. 4 concludes the work and proposes future system developments.;

2 The object tracking system

2.1 Particle filters for Bayesian tracking

The aim of our system is to follow skin-colored objects in time, by integrating past information with the current measurement in order to update the posterior state estimate. In the *Bayesian tracking* framework, knowledge about the object state is represented and propagated in a probabilistic way[2, 20], by means of two main steps:

1. *Prediction* (Kolmogorov-Chapman equation):

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

2. *Correction* (Bayes' rule):

$$P\left(s_t|\, Z^{t}\right) = kP\left(z_t|\, s_t\right) P\left(s_t|\, Z^{t-1}\right) \quad (2)$$

where the current state statistics s_t are updated by integrating the associated measurement z_t together with the set $Z^t = z_{1\ldots t}$ of all measurements up to time t, together with the last posterior distribution s_{t-1}.

In this scheme, the dynamical model $P\left(s_t|\, s_{t-1}\right)$ and the measurement likelihood $P\left(z_t|\, s_t\right)$ need to be specified, according to the chosen estimation filter. For nonlinear, pixel-level measurement models like the one that we present in this work, where Jacobian matrices are not available or too costly to compute, we consider in particular particle filters [1], where state statistics s_t are represented by a set of N weighted particles

$$\{s_t^n, \pi_t^n\};\ n = 1 \ldots N \quad (3)$$

where $\sum_n \pi_t^n = 1$.

Figure 1: An input video frame and the corresponding segmented image using GMM.

For our purposes, we chooose the standard sampling-importance-resampling (SIR) scheme [12]:

- Sample from previous posterior with the dynamical model: $s_t^n \sim P\left(s_t|\, s_{t-1}^n\right)$
- Weight the particle according to the likelihood $\pi_t^n \propto P\left(z_t|\, s_t^n\right)$ and normalize the weights $\sum_n \pi_t^n = 1$
- Resample particles: $s_t^{n'} \leftarrow s_t^n$, with n' randomly selected according to $\{\pi_t^n\}$; afterwards, reset the weights $\pi_t^n = 1/N$

2.2 Image pre-processing

As stated in the Introduction, the binary pixel map for likelihood computation can be obtained from any modality and segmentation procedure; without loss of generality, we consider in this paper a color segmentation.

For this purpose, the input image is first converted from RGB to HSV color space, which is well-suited for color object segmentation and tracking, because it seperates pixel luminance from the pure color channels (hue-saturation), providing more robustness against illumination changes.

After the conversion, each pixel is classified using a 2D Gaussian Mixture Model (GMM) on the two color channels (H,S). GMMs have been widely used for foreground segmentation [7, 8, 9, 21, 22], because of their efficient training and evaluation procedures.

In the example of Fig. 1, a GMM is used in order to model skin-colored pixels. A GMM is composed of K Gaussian probability density functions (pdfs), described by the following equation:

$$p(\mathbf{c}_j|C_{skin}) = \sum_{k=1}^{K} w_k p_k(\mathbf{c}_j|C_{skin}) \quad (4)$$

where p_k is the k^{th} mixture component, with

weights w_k normalized so that $\sum_{k=1}^{K} w_k = 1$, and each component p_k is described by a bi-variate Gaussian

$$p_k(\mathbf{c}_y|C_{skin}) = \frac{1}{2\pi\sqrt{|\Sigma_k|}} e^{-\frac{1}{2}(\mathbf{c}_y-\mu_k)^T \Sigma_k^{-1}(\mathbf{c}_y-\mu_k)} \tag{5}$$

with $\mathbf{c}_y$ the 2-dimensional (H, S) color of screen pixel y.

Mean and covariance matrix (μ_k, Σ_k) for each component, as well as the mixture weights w_k, are learned from a given training set, via the Expectation-Maximization algorithm [6].

In order to classify a color pixel y, its GMM likelihood is thresholded against a suitable value p_{min}

$$z(y) = \begin{cases} 1 & \text{if } p(\mathbf{c}_j|C_{skin}) > p_{min} \\ 0 & \text{if } p(\mathbf{c}_j|C_{skin}) \leq p_{min} \end{cases} \tag{6}$$

which results in a binary image (Figure 1), which constitutes our pixel-level measurement z_t for tracking.

Although the pre-processing step is performed only once per frame, it could result in pixel-wise expensive computations that leave less time for the subsequent tracking steps. Therefore, as it will be described in Sec. 2.5, both computations (4),(6) have been implemented on the GPU by using the OpenGL shader language [19], together with the pixel-level likelihood.

2.3 Object state parameters and dynamics

The sampling step for particle filters requires that a prior modle of object dynamics be specified, with its degrees of freedom modeled via a set of *pose parameters* which constiute the state vector s_t.

First, the geometric mapping between object space and screen coordinates is generically defined as

$$y = W(x, p) \tag{7}$$

where x is a point in object coordinates (2D or 3D), and y is the corresponding screen projection under pose parameters p.

Depending on the model complexity, as well as the kind of transformation in Eq. (7) (planar, 3D, articulated, etc.), this can be a very expensive computation on the CPU, but can be performed efficiently on general-purpose graphics hardware even for complex 3D meshes.

In our experiments, we use a planar warp, given by a similarity transform

$$y = \begin{pmatrix} s_x cos\theta & s_y sin\theta \\ -s_x sin\theta & s_y cos\theta \end{pmatrix} x + t \tag{8}$$

with θ a rotation angle, t a 2D translation vector, and the scaling factors s_x and s_y. This corresponds to 5 pose parameters $p \equiv (s_x, s_y, \theta, t_x, t_y)$.

For particle filtering, the explicit form of state dynamics

$$s_t = As_{t-1} + w_t \tag{9}$$

is required, in order to drift the particles and generate new pose hypotheses, by randomly sampling the process noise w from its known distribution.

In the present work, a simple unconstrained Brownian model is employed, where $A = I$ and w is zero-mean, normally distributed with covariance $\Lambda_w(\Delta t)$, increasing with the time interval Δt

$$P(s_t|s_{t-1}) = \mathcal{N}(s_{t-1}, \Lambda_w(\Delta t)) \tag{10}$$

so that predicted particle positions are obtained by

$$s_t^n = s_{t-1}^n + w(\Delta t) \tag{11}$$

More specific dynamics can be obtained by including in s_t the first or second time derivatives of the pose (velocity, acceleration) and providing suitable values for A and Λ_w, which can eventually be learned from ground truth sequences [17]. However, in our experiments the simple Brownian model (11) already gave satisfactory results.

2.4 Measurement likelihood

In the update step of Bayesian tracking, the weight π_n of each particle $n = 1 \ldots N$ is computed by comparing its state hypothesis s_t^n with the current measurement z_t, obtained by the pre-processing step (Section 2.2).

For pixel-level matching, the projected filled object silhouette h_t^n at each pose hypothesis has to be computed (Fig. 2, middle). For this purpose, the warp function (7) is used, providing a binary map which represents the expected measurement for an ideal, noise-free segmentation under the given pose s_t^n.

Afterwards, the residual with the current measurement z_t is computed by a simple SSD cost function

$$e_t^n = \sum_y \left[h_t^n(y) - z_t(y)\right]^2 \tag{12}$$

Figure 2: Left: Segmented image. Middle: generated filled silhouette hypothesis. Right: pixel-level residual.

For binary images h_t^n, z_t, this is equivalent to a pixel-wise XOR (right of Fig. 2) followed by a sum of the non-zero pixels.

The computation of h^n can be very expensive if performed on the CPU, and moreover no pixel parallelization can be exploited while comparing it with z_t. Therefore, we implement these operations on the GPU, using at the same time the power of the rendering engine and the parallel pixel-pipelines.

The residual value (12) is normalized in the range $[0, 1]$ by dividing it by the number of pixels, and the likelihood is evaluated with a Gaussian model:

$$\pi_n = P(z_t|s_t^n) = \exp(-\frac{e_t^n}{2r^2}) \qquad (13)$$

with a suitable measurement variance r, providing the new particle weights π_n, afterwards normalized so that $\sum_n \pi_n = 1$. Deterministic resampling of the particle set [12] is applied after each update, in order to keep a well-distributed particle set.

In order to estimate the output pose $\hat{s}_t$, the weighted average of the particle set is computed

$$\hat{s}_t = \sum_{n=1}^{N} \pi_h s_t^n \qquad (14)$$

and returned to the user.

2.5 Computation on the GPU

Modern graphics card are becoming very popular today, because of their computational power, the low costs, and the emerging of high-level languages such as CUDA [5], Cg [13], or the OpenGl Shader Language [19] that allow a customized use the graphics hardware. [15] presents a good survey of the posibilities and limitations of the graphics processing units (GPU). The main limitiation in porting algorithms to GPU is the fact that one needs to rethink the structure of the algorithm to fit into a parallel graphics pipeline. In this context it is not trivial, for example, to sum up pixels of a certain region, because each pixel is calculated separately.

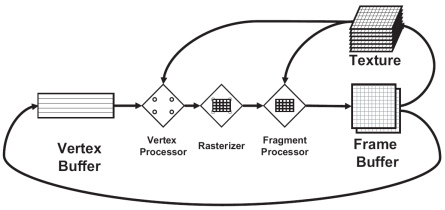

Figure 3: Graphics hardware pipeline. Image taken from [15]

When using graphics hardware for on-line image processing and related computations, another well-known bottleneck is given by the slow memory transfer from the GPU memory to the main memory. This is due to the fact that graphics hardware has originally been optimized for display purposes, which do not require any data transfer to the CPU memory.

In order to avoid this bottleneck, after uploading the color image to the GPU we develop all pixel-level computations on board, directly providing the image residual (12) to the CPU, in order to compute the likelihood. The algorithms are executed on the fragment processor of the GPU. The italic terms in the following descriptions relate to the parts of the graphics pipeline shown in Figure 3.

2.5.1 Segmentation of the input image using the GMM

The image that is to be segmented is loaded into a texture of the graphics card. Because we want to process every pixel of the image, every pixel need to become a fragment. To achieve this, we define in the *vertex processor* four vertices for the four corners of the texture. In the *rasterizer* every input pixel of the texture is rastered in one fragment. The *fragment processor* is the instance that executes on every fragment the same program, i.e., the implementation of a GMM as described in Section 2.2. The resulting segmented image is again stored in a texture.

2.5.2 Generating the filled model silhouette image

To be able to deal with all kinds of models, we use 3D mesh models as representation. The model

is loaded and the OpenGL display lists are generated. To generate the filled model silhouette image (ideal measurement), the current transformation-matrix (Eq. (7), (8)) of the hypothesis is set in the OpenGL rendering pipeline. The faces of the model are set to white and are then rendered to a texture.

2.5.3 Computing the residual

To evaluate the quality of the hypothesis, we calculate the difference of the segmented and the hypothesis texture (Eq. (12)). The procedure is the same as in the segmentation step, but the program in the *fragment processor* is exchanged to compute the binary XOR.

2.5.4 Task Division between CPU and GPU

Fig. 4 summarizes the task division between the two processors, where on the graphics card the GLSL shader language is employed, and data are exchanged through texture images.

With this implementation, only the main steps of the particle filter (prediction, resampling and output computation) are left to the CPU, together with the exponentiation in Eq. (13) and the weight normalization.

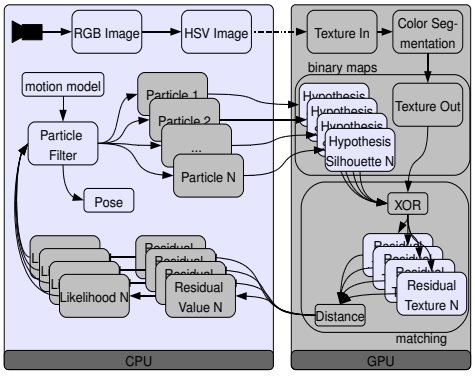

Figure 4: Scheme of the proposed tracking system, illustrating the subdivision of tasks between the CPU and the GPU.

3 Experimental Results

As already emphasized in the previous descriptions, our methodology can be applied to different visual modalities, with different object shapes and degrees of freedom.

processor	hand tracking	face tracking
CPU	1.8 fps	1.9 fps
GPU	5.4 fps	13.3 fps

Table 1: Performance comparison for the tracking the face tracking and the hand tracking task on the CPU and the GPU

In this section, we present results concerning in particular hand and face tracking tasks with skin color segmentation, using 3D mesh models and 2D similarity transformations as explained in Sec. 2.3, in order to demonstrate the speed-up of our approach with respect to the equivalent CPU-based implementation.

Concerning shape models, a simple ellipsoid was used for face tracking, approximating the shape of the head, while hand tracking was done with a complex 3D triangle mesh model. For both experiments, an Intel dual-core machine with programmable graphics card (NVidia GeForce 8800) was used, with input images of resolution (640 x 480) from a standard FireWire camera.

The GMM model was built from a training data set of labeled skin-pixels, and consists of $K = 2$ mixture components, as in [21]. The processing speed for image segmentation, including the conversion from RGB to HSV, is 20 times faster on the GPU, already providing a big advantage for tracking.

Table 1 show a comparison of the overall processing times in frames-per-second (fps) for the tracking of the ellipsoid model (head tracking) and the hand model (hand tracking), including particle filter prediction and likelihood computations, with around 100 particle hypotheses. It can be seen that the GPU outperforms the equivalent CPU implementation, reaching a framerate up to 13 frames per second for face tracking, and around 5 frames per second for the complex hand model.

Output results of the tracking can be seen in Figure 5 illustrating how the algorithm deals with all pose parameters. In both cases, after a simple initialization with a uniformly distributed particle set around the image center (left of Figure 6), the filter converges in a few frames to the target which is

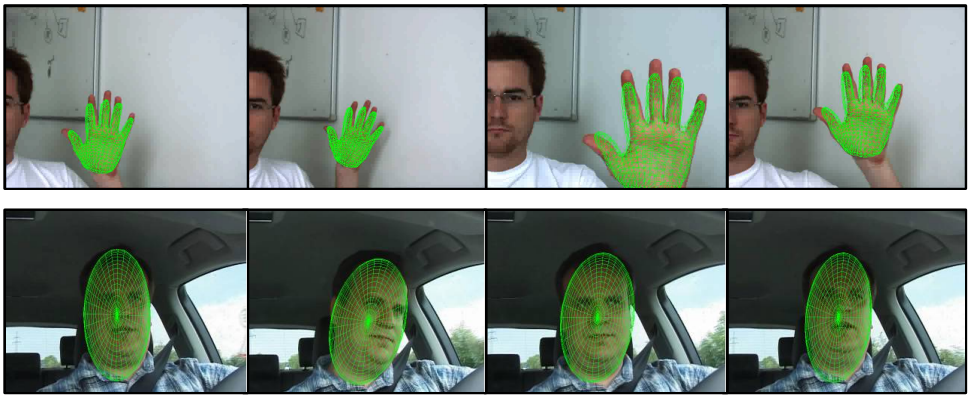

Figure 5: Hand and face tracking sequences with different mesh-models

then successfully tracked throughout the rest of the sequence.

4 Conclusions and future developments

We have presented an algorithm for GPU-accelerated particle filtering, based on fast pixel-level segmentation and matching, for real-time object tracking tasks. The proposed formulation is general with respect to the segmentation modality, object shape and degrees of freedom, and obtains a better precision of localization with respect to feature-level methods such as color histogram matching.

Our implementation achieves a higher frame rate than the equivalent CPU-based version; this result is obtained by performing many of the expensive steps (pre-processing, hypothesis computation and likelihood evaluation) on graphics hardware, through the rendering engine and the OpenGL shader language, while minimizing the back-transfer of data to the main memory.

In the experiments of this work only rigid models were used, but the algorithm could cope with articulated models as well. In future developements we want to investigate the performance of the algorithm using articulated models. Other potiential developements lie in the integration of the GMM segmentation with background subtraction on the graphics card in order to get more robust results, and multiple calibrated cameras employed for tracking 3D objects within the same framework. Finally, tracking multiple, simultaneous targets at pixel-level with multi-class segmentation is another subject of interest for this methodology.

5 Acknowledgements

This work is partly supported by the German Research Council (DFG), under the excellence initiative cluster *CoTeSys - Cognition for Technical Systems* [1].

References

[1] S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online non-linear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, February 2002.

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

[3] Andrew 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.

[4] Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer. Kernel-based object tracking. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(5):564–575, 2003.

[1] http://www.cotesys.org

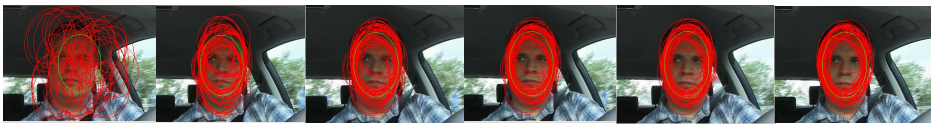

Figure 6: Condensation of the particle set after re-initialization.

[5] Nvidia CUDA. `http://www.nvidia.com/cuda`.

[6] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[7] A. Diplaros, T. Gevers, and N. Vlassis. Skin detection using the EM algorithm with spatial constraints. *IEEE International Conference on Systems, Man and Cybernetics*, 4, 2004.

[8] Nir Friedman and Stuart Russell. Image segmentation in video sequences: A probabilistic approach. In *Annual Conference on Uncertainty in Artificial Intelligence*, pages 175–181, 1997.

[9] H. Greenspan, J. Goldberger, and I. Eshet. Mixture model for face-color modeling and segmentation. *Pattern Recognition Letters*, 22(14):1525–1536, 2001.

[10] Daniel Grest, Volker Krger, and Reinhard Koch. Single view motion tracking by depth and silhouette information. In *In Scandinavian Conference on Image Analysis (SCIA07*, 2007.

[11] Eric Hayman and Jan-Olof Eklundh. Probabilistic and voting approaches to cue integration for figure-ground segmentation. In *ECCV 2002*, pages 469–486, 2002.

[12] M. Isard and A. Blake. Condensation – conditional density propagation for visual tracking. *International Journal of Computer Vision (IJCV)*, 29(1):5–28, 1998.

[13] W. R. Mark, R. S. Glanville, K. Akeley, and M. J. Kilgard. Cg: A system for programming graphics hardware in a c-like language. *ACM Transactions on Graphics*, 22(3):896–907, July 2003.

[14] Katja Nummiaro, Esther Koller-Meier, and Luc J. Van Gool. An adaptive color-based particle filter. *Image Vision Comput.*, 21(1):99–110, 2003.

[15] John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krger, Aaron E. Lefohn, and Timothy J. Purcell. A survey of general-purpose computation on graphics hardware. In *Eurographics 2005, State of the Art Reports*, pages 21–51, August 2005.

[16] Patrick Pérez, Carine Hue, Jaco Vermaak, and Michel Gangnet. Color-based probabilistic tracking. In *ECCV '02: Proceedings of the 7th European Conference on Computer Vision-Part I*, pages 661–675, London, UK, 2002. Springer-Verlag.

[17] David Reynard, Andrew Wildenberg, Andrew Blake, and John A. Marchant. Learning dynamics of complex motions from image sequences. In *ECCV '96: Proceedings of the 4th European Conference on Computer Vision-Volume I*, pages 357–368, London, UK, 1996. Springer-Verlag.

[18] B. Rosenhahn, U. G. Kersting, A. W. Smith, J. K. Gurney, and T. Brox. A system for marker-less human motion estimation. In *Page 6 [22] Rosenhahn B., Klette*, pages 45–51. Springer, 2005.

[19] Randi J. Rost. *OpenGL(R) Shading Language (2nd Edition)*. Addison-Wesley Professional, January 2006.

[20] L. D. Stone, T. L. Corwin, and C. A. Barlow. *Bayesian Multiple Target Tracking*. 1st. Artech House, Inc., 1999.

[21] M.H. Yang and N. Ahuja. Gaussian Mixture Model for Human Skin Color and Its Application in Image and Video Databases. In *Proc. SPIE: Storage and Retrieval for Image and Video Databases VII*, volume 3656, pages 458–466, 1999.

[22] Z. Zivkovic. Improved adaptive Gaussian mixture model for background subtraction. In *Proceedings of the 17th International Conference on Pattern Recognition*, volume 2, 2004.