Real Time Parallel Implementation of a Particle Filter Based Visual Tracking

Joel Falcou, Thierry Chateau, Jocelyn Serot, Jean-Thierry Lapresté

Real Time Parallel Implementation of a Particle Filter Based Visual Tracking

Joel Falcou, Thierry Chateau, Jocelyn Sérot, and Jean-Thierry Lapresté

LASMEA, UMR6602, CNRS, Blaise Pascal University,Clermont-Ferrand, France
{Surname.NAME}@lasmea.univ-bpclermont.fr

Abstract. We describe the implementation of a 3D visual tracking algorithm on a cluster architecture.Parallelisation of the algorithm makes it possible to obtain real-time execution (more than 20 FPS) even with large state vectors, which has been proven difficult on sequential architecture. Thanks to a user-friendly software development environment, this large gain in performance is not obtained at the price of programmability.

1 Introduction

Particle filtering is a widely used method to solve vision tracking problems. However realistic applications require large state vectors - and thus large particle distribution sets -to produce accurate and/or robust results. In practice, this precludes real-time execution - at more than a few frames per second, typically - of these applications on standard, sequential platforms. We therefore propose a parallel implementation of a 3D tracking algorithm operating on a stereo video stream and running in real-time on a cluster architecture. We demonstrate the efficiency of this implementation with a pedestrian tracking application.

The paper is organized as follow : in section 2, we'll introduce the general problem of probabilistic visual tracking and how to solve it using sequential Monte Carlo method. in section 3, we'll describe the parallel architecture we used for implementing a parallel version of this tracking algorithm and discuss results of this implementation in section 4.

2 Probabilistic Visual Tracking

Visual tracking can be seen as the estimation, at time t, of the posterior probability function $p(\mathbf{X}_t|\mathbf{Z}_{0:t})$ where $\mathbf{X}_t$ is the hidden state (position) of the object and $\mathbf{Z}_{0:t} \doteq (\mathbf{Z}_0, ..., \mathbf{Z}_t)$ denotes the temporal data sequence (images). In the case of a conditionally independent observation process and a Markovian prior on the hidden states, the sequence of filtering distributions $p(\mathbf{X}_t|\mathbf{Z}_{0:t})$ to be tracked are defined by the recursive equation:

$$p(\mathbf{X}_{t+1}|\mathbf{Z}_{0:t+1}) \propto p(\mathbf{Z}_{t+1}|\mathbf{X}_{t+1}) \int_{\mathbf{X}_t} p(\mathbf{X}_{t+1}|\mathbf{X}_t) p(\mathbf{X}_t|\mathbf{Z}_{0:t}) d\mathbf{X}_t \qquad (1)$$

Assuming that the distributions of probabilities are Gaussian, the Kalman filter provides an optimum analytical solution. However, visual tracking applications are highly non-linear and multi-modal problems. In this case, the posterior can be estimated by sequential Monte Carlo techniques [1].

2.1 Particle Filter

Particle filtering [2, 3] is a sequential importance sampling algorithm for estimating properties of hidden variables given observations in a hidden Markov model. Standard particle filter assumes that posterior $P(\mathbf{X}_t|\mathbf{Z}_t)$ can be approximated by a set of samples (particles). Moreover it also assumes that the observation likelihood $P(\mathbf{Z}_t|\mathbf{X}_t)$ can be easily evaluated.

A particle filter approximates the posterior using a weighted particle set $\{(\mathbf{X}_t^n, \pi_t^n) : n = 1, .., N\}$. Figure 1 describes the algorithm used here, also called CONDENSATION[3]

1. **initialize** $\{(\mathbf{X}_0^n, \pi_0^n)\}_{n=1}^N$ **from the prior distribution** $\mathbf{X}_0$
2. **for** $t > 0$
   (a) **resample** $\{(\mathbf{X}_{t-1}^n, \pi_{t-1}^n)\}_{n=1}^N$ **into** $\{(\mathbf{X}_{t-1}^{'n}, 1/N)\}_{n=1}^N$
   (b) **predict, generating** $\mathbf{X}_t^n \sim p(\mathbf{X}_t|\mathbf{X}_{t-1} = \mathbf{X}_{t-1}^{'n})$ **to give** $\{(\mathbf{X}_t^n, 1/N)\}_{n=1}^N$
   (c) **weight, setting** $\pi_t^n \propto p(\mathbf{Z}_t|\mathbf{X}_t = \mathbf{X}_t^n)$ **to give** $\{(\mathbf{X}_t^n, \pi_t^n)\}_{n=1}^N$ **normalized so** $\sum_{n=1}^N \pi_t^n = 1$
   (d) **estimate** $\hat{\mathbf{X}}_t \doteq \frac{1}{N} \sum_{n-1}^N \mathbf{X}_t^n$
3. t++, **jump to** 2.

Fig. 1. The particle filter algorithm (CONDENSATION)

2.2 State Space and Dynamics

We want to track an object in a 3D space defined in a reference frame R_w. Left and right camera projection matrices between R_w and the image plane are given by $\mathtt{C}_l$ and $\mathtt{C}_r$. At time t, the state vector is defined by $\mathbf{X}_t \doteq (\mathbf{P}_t, \mathbf{V}_t)^t$, where $\mathbf{P}_t$ is the 3D position of the center of a bounding box associated with the object to be tracked and $\mathbf{V}_t$ is the associated velocity. For a state $\mathbf{X}_t$, the corresponding 2D points $\mathbf{p}_t^l$ and $\mathbf{p}_t^r$ of the center of an image bounding box for left and right camera are given by :

$$\left((\mathbf{p}_t^{(l,r)})^t \ 1\right)^t \propto \mathtt{C}_{(l,r)} \left((\mathbf{P}_t^{(l,r)})^t \ 1\right)^t, \tag{2}$$

where $\mathtt{C}_{(l,r)}$ is the projection matrix associated to the left (right) camera, obtained by a classical calibration step. Since height and width of the 3D bounding box are assumed to be constant, the corresponding height and width of each

image bounding box is computed using projections matrices. A first order auto-regressive dynamics is chosen on these parameters:

$$\mathbf{X}_{t+1} = \mathtt{A}\mathbf{X}_t + \mathtt{B}\mathbf{v}_t \ , \ \mathbf{v}_t \sim \mathcal{N}(0, \Sigma) \tag{3}$$

Matrices $\mathtt{A}$, $\mathtt{B}$ and Σ can be estimated from a set of sequences for which the true position of the object is known.

2.3 Observation Likelihood

This section describes the tracker likelihood function $P(\mathbf{Z}|\mathbf{X})$ which is defined as the likelihood that the state of the object (position) is $\mathbf{X}$ according to an observed image $\mathbf{Z}$. Many particle filter based trackers use a likelihood function linked to a distance between the model and the current particle to be weighted like $\pi = \exp(-\lambda.d(.,.))$. However, the parameter λ must be adjusted to provide good performances. The method proposed here does not use empirical probabilities build from distances, but directly calibrated probabilities computed from the output of an Adaboost Classifier [4]. Features used are multiscale Haar wavelets [5] and a compact description of the object is selected from the Adaboost offline learning step [6]. This classifier $\mathbf{s}_t(\mathbf{X}_t) = (s_t^l(\mathbf{X}_t), s_t^r(\mathbf{X}_t))^t$ returns an uncalibrated vector[1] of values for the input 3D state $\mathbf{X}_t$. We propose to build the likelihood function used to evaluate weights of the particle filter from $\mathbf{s}_t(\mathbf{X}_t)$. Since the likelihood function used by the particle filter is a probability, $P(class|input)$ must be produced from the output of the classifier. A sigmoid is used to build calibrated probabilities from $\mathbf{s}_t(\mathbf{X}_t)$ [7].

$$P(\mathbf{Z}_t|\mathbf{X}_t) \doteq \frac{1}{1+\exp(\hat{\theta_1}.s_t^r(\mathbf{X}_t)+\hat{\theta_2})}.\frac{1}{1+\exp(\hat{\theta_1}.s_t^l(\mathbf{X}_t)+\hat{\theta_2})} \tag{4}$$

Parameters of the sigmoid $(\hat{\theta_1}, \hat{\theta_2})$ are estimated using Platt scaling method [8]. If $s_t^{(.)}(\mathbf{X}_t)$ is the output of the classifier, calibrated probabilities can be produced from the sigmoid:

$$P(\text{positive}|s_t^{(.)}(\mathbf{X}_t)) = \frac{1}{1+\exp(\hat{\theta_1}.s_t^{(.)}.(\mathbf{X}_t)+\hat{\theta_2})} \tag{5}$$

where $\hat{\theta_1}$ and $\hat{\theta_2}$ are computed using maximum likelihood estimation from a calibration set (s_i, y_i) (s_i represents output of classifier and $y_i \in \{0, 1\}$ represent negative and positive examples). $\hat{\theta_1}$ and $\hat{\theta_2}$ are computed by a non-linear optimization of the negative log likelihood of the training data, which is a cross-entropy error function:

$$\arg\min_{(\hat{\theta_1},\hat{\theta_2})}(-\sum_i y_i \log(p_i) + (1-y_i)\log(1-p_i)) \tag{6}$$

[1] Two values corresponding to classifier score associated to left and right camera.

where

$$p_i = \frac{1}{1 + \exp(\hat{\theta_1}.s_i + \hat{\theta_2})} \tag{7}$$

The easiest solution is to choose the same training set to fit the sigmoid than the training set used to train the classifier. However, Platt shows that it causes a biased estimate of the distribution of probability. A solution is to use a fraction of the training set (70% for example), to train the classifier, and to use the other fraction (30%) to estimate the parameters of the sigmoid. An other solution is to use a cross-validation method (see [9] for details).

3 Parallel Implementation

3.1 Architecture Synopsis

Our cluster architecture has been introduced in [10] . It is sketched in fig 2. It includes fourteen computing nodes. Each node is a dual-processor Apple G5 XServe Cluster Node running at 2 GHz with 1Gb of memory. Nodes are interconnected with Gigabit Ethernet and provided with digital video streams, coming from a pair of digital cameras, by a *Firewire IEEE1394a* bus. This approach allows simultaneous and synchronized broadcasting of input images to all nodes, thus removing the classical bottleneck which occurs when images are acquired on a dedicated node and then explicitly broadcasted to all other nodes.

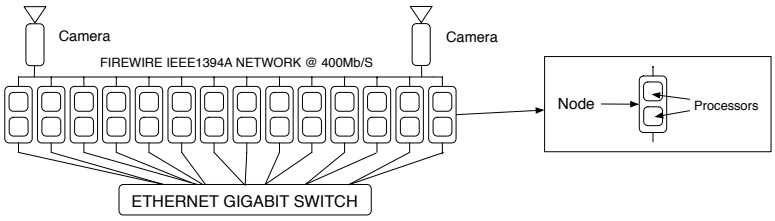

Fig. 2. Cluster architecture

Another distinctive point of this cluster is the presence of the Altivec [11] extension. This extension can provide, at a single processor level, speedups in the range of 2-12 for a large class of low- to -mid-level image processing operations [12], thus making it possible to reach global speedups up to 30 to 50 without requiring to a large, costly and power-consumming cluster.

Programming relies on a hybrid three-level parallel programming model :

1. At the lowest level are the ***Firewire* drivers** handling the transfer of images from the cameras to the processor memory (using the DMA). For the

programmer, the calls to these drivers are encapsulated within CFOX[13], a dedicated library which provides in particular functions for configuring cameras, obtaining a pointer to the most recently acquired frame in memory and to synchronizing video streams.

2. The middle layer deals with **parallelism issues**. We use a hybrid three-level parallel programming model, involving a fine-grain SIMD-type parallelism within each processor [11], a coarse grain shared-memory multi-processing between the two processors of a node and a coarse grain message passing based multi-processing between two processors of distinct nodes. The first level can be exploited using the Altivec native C API [11] or, more easily, using EVE [12], a high-level vectorization library specifically tuned for this extension. The second level is exploited by describing the process running on each node as two concurrent threads using the PTHREAD library. At the third level, the application is decomposed into a set of processes running in SPMD mode and communicating using the MPI library. The use of PTHREAD to explicitly schedule the execution of applications on the two processors of each node is due to the fact that the lowest level of the software architecture doesn't map correctly onto the dual processor architecture. Each node is viewed as a single *Firewire* node by the driver layer and as a two-processor node by the parallel layer. This duality leads to resources sharing conflicts that can't be handled correctly by MPI. We need to keep a one-for-one mapping between the *Firewire* level and the MPI level and use PTHREAD for this purpose.

3. The upper layer acts as a **high-level programming framework for implementing parallel applications**. This framework, QUAFF, is a C++, ready-to-use library of abstractions (skeletons, in the sense of [14]) for parallel decomposition, built-in functions for video i/o and mechanisms allowing run-time modification and parameterization of application-level software modules. Developing specific applications with QUAFF boils down to write independent algorithmic modules and to design an application workflow using those modules. One can develop various workflows prototype using a simple XML description and dynamically choose which one is used by the application. Once a prototype is considered finalized, the developer can use a small tool to turn this XML based applications into a strongly optimized, templates based code. This off-line, optimisation step can boost performances by a factor 2.

3.2 Parallelisation Strategy

For the visual tracking application, we use a pure data-parallel strategy in which the particle distribution $\{(\mathbf{X}_t^n, \pi_t^n) : n = 1, .., N\}$ is scattered among the compute nodes. Each node therefore performs the prediction, measure and weighting steps on a subset of the initial particule distribution. On each node the left and right projections and measures are themselves computed in parallel at the SMP level,

each by one processor. Moreover, on each processor, computations are vectorized whenever possible at the SIMD level. Once the new particules weights have been evaluated, they are merged back to the root node which perform the estimation step. The final resampling is then performed over all nodes by scattering the sum of measures scores, and evaluating an index matrix that are gathered on root node. Then the root node performs the particules swapping and replications to update $\hat{\mathbf{X}}_t$.

4 Results

Figure 3 shows a sample execution of our pedestrian tracking application. The upper part of the figure shows the two video streams and the projection of the particles distribution. Lower part is the estimated 3D path of the pedestrian projected on the ground plane.

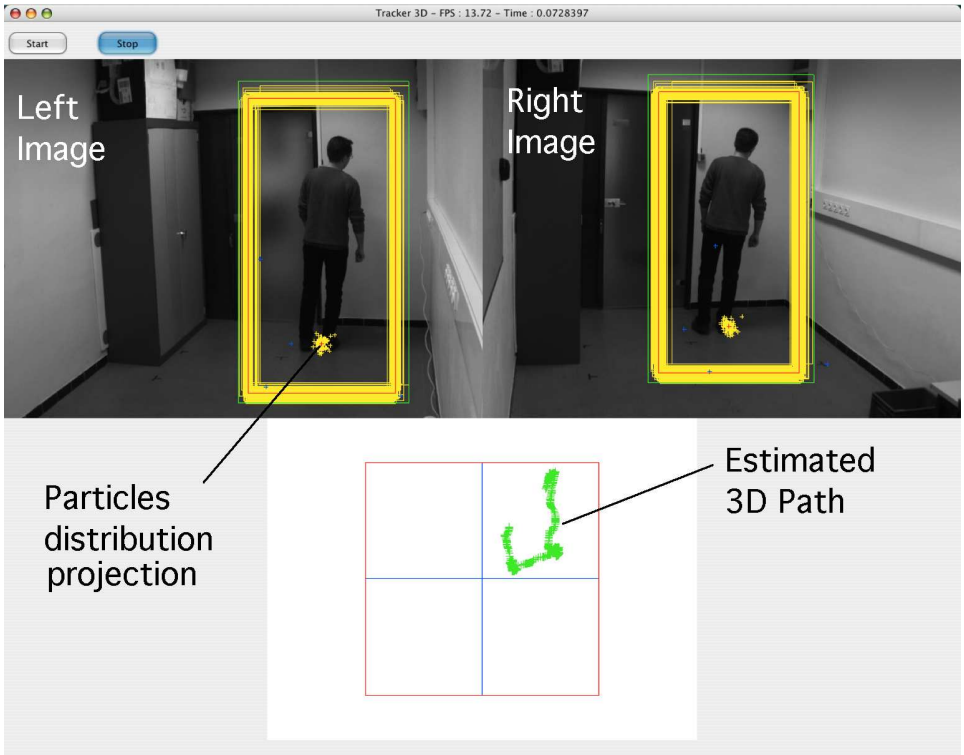

Fig. 3. Sample Tracking Session

Table 1 shows results obtained on a stereo 640×480×8bits video stream for several size of particles distribution. Execution times and frame rate are given for a single processor machine and a complete 14 nodes (28 processors) topology.

	200 part.	500 part.	1000 part.	2000 part.	5000 part.	10000 part.
Sequential	0.0609s	0.1439s	0.2874s	0.5952s	1.6393s	3.8462s
	16.40 FPS	6.95 FPS	3.48 FPS	1.68 FPS	0.61 FPS	0.26 FPS
14 Nodes	0.0231s	0.0265s	0.0313s	0.0462s	0.0858s	0.1567s
	43.31 FPS	37.72 FPS	31.9 FPS	21.64 FPS	11.66 FPS	6.38 FPS
Speed Up	×2.7	×5.42	×9.16	×12.88	×19.43	×24.5

Fig. 4. Timing results for various particles number

Near real-time performances are achievable with a low number of nodes : a 20 frames per second rate is achieved by using approximatively 2000 particles scattered on 14 nodes and a 12 frames per second rate is achieved with nearly 5000 particles. Contrary to the sequential implementation, we're able to keep up with real time constraints for more than 3000 particles. We can also see that the global speed up is linear with the number of particles.

5 Conclusion

This paper shows that real-time execution of a particle filter based visual tracker can be achieved even with large state vectors thanks to a multi-level parallelisation[2], a cluster architecture with a well-designed I/O mechanism and an user-friendly software development environment that provides large gains in performance without loss of programmability.Moreover, this approach can be used for other Monte Carlo simulations requiring real-time execution like SLAM[3]. Further works will include a multitarget 3D tracking dealing with occultations.

References

1. Doucet, A., Godsill, S., Andrieu, C.: On sequential monte carlo sampling methods for bayesian filtering. Statistics and Computing **10** (2000) 197–208
2. Arulampalam, S., Maskell, S., Gordon, N., Clapp, T.: A tutorial on particle filters for on-line non-linear/non-gaussian bayesian tracking. IEEE Transactions on Signal Processing **50** (2002) 174–188
3. M. Isard, A. Blake: Condensation – conditional density propagation for visual tracking. IJCV : International Journal of Computer Vision **29** (1998) 5–28

[2] Including SIMD, SMP and MIMD
[3] Simultaneous Localization And Mapping

4. Viola, P., Jones, M.J., Snow, D.: Detecting pedestrians using patterns of motion and appearance. In: Int. Conf. Computer Vision, Nice, France (2003) 734–741
5. C. Papageorgiou, M. Oren, T. Poggio: A general framework for object detection. In: IEEE Conference on Computer Vision. (1998) 555–562
6. Tieu, K., Viola, P.: Boosting image retrieval. International Journal of Computer Vision **56** (2004) 17–36
7. Niculescu-Mizil, A., Caruana, R.: Obtaining calibrated probabilities from boosting. In: Proc. 21st Conference on Uncertainty in Artificial Intelligence (UAI '05), AUAI Press (2005)
8. Platt, J.: Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods. In: Advances in Large Margin Classifiers. MIT Press (1999) 61–74
9. Platt, J.: Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In: Advances in Large Margin Classifiers. MIT Press (1999)
10. J. Falcou, J. Serot, T. Chateau, F. Jurie: A Parallel Implementation of a 3D Reconstruction Algorithm for Real-Time Vision. Parallel Computing 2005 (2005)
11. I. Ollman: AltiVec Velocity Engine Tutorial http://www.simdtech.org/altivec (2003)
12. J. Falcou, J. Serot: E.V.E., An Object Oriented SIMD Library. Scalable Computing: Practice and Experience **6** (2005)
13. J. Falcou: CFOX : Open Source High Performance IEEE1394 Driver for OS X http://cfox.sourceforge.net (2003)
14. Cole, M.: Algorithmic Skeletons : Structured Management of Parallel Computation. Algorithmic Skeletons : Structured Management of Parallel Computation (1989)