Fast Exact Bayesian Inference for High-Dimensional Models

João Filipe Ferreira[a], Pablo Lanillos[a], and Jorge Dias[a,b]

***Abstract*— In this text, we present the principles that allow the tractable implementation of exact inference processes concerning a group of widespread classes of Bayesian generative models, which have until recently been deemed as intractable whenever formulated using high-dimensional joint distributions. We will demonstrate the usefulness of such a principled approach with an example of real-time OpenCL implementation using GPUs of a full-fledged, computer vision-based model to estimate gaze direction in human-robot interaction (HRI).**

I. Introduction

Bayesian inference has by now found its way into all branches of science, including natural sciences, humanities and social sciences, and also applied sciences, such as medicine. In most cases, Bayesian inference has been used as an alternative to the more "traditional" frequentist inference methods for statistical analysis of data. The application of Bayesian inference to *model* cognition and reasoning, either to study the animal brain, such as in neuroscience, or to synthesise artificial brains, such as in artificial intelligence (AI) or robotics, has become very popular in the past couple of decades. In AI, numerous recent examples of the popularity of the so-called *Bayesian approach* can be found; robotics, as a consequence, follows suit, as can be seen in [1], [2], [3], [4]. However, the most important restriction to the use of the Bayesian approach has been the issue of *implementation of Bayesian inference*, especially in what concerns the development of artificial cognitive systems, which require real-time performance.

In general, Bayesian models involve many random variables, dependent on each other in arbitrarily complex fashion. Let us consider the case of a generative model for which all variables are discrete. This is not an unreasonable scenario, since inference computations will invariably be performed by computers, which imply some sort of discretisation of continuous signals. For example, modern digital computer-controlled systems, such as robots, use analog-to-digital and digital-to-analog converters, which discretise readings providing input from sensors and control commands providing output to actuators.

In this case, we arrive to a general expression of exact inference given by

$$P(S \mid o) = \frac{\sum_F P(S \wedge F) P(o \mid S \wedge F)}{P(o)}, \qquad (1)$$

where S denotes a random variable representing the conjunction of all *searched* variables (i.e. variables for which one intends to infer the probability of any given value), o denotes the instantiation of a random variable representing the conjunction of all *observed* variables (i.e. variables of which the current joint value o is known or observed), and F denotes a random variable representing the conjunction of all *free* variables (i.e. although their relation to all other variables in the model is known, variables of which the current value is not only unknown but also not searched for, and hence the influence of which one wants to factor out through marginalisation). Note that $P(o) = \sum_F \sum_S P(S \wedge F) P(o \mid S \wedge F)$ is, in fact, a normalising scalar, depending *only* on the observed data o.

In fact, obtaining this general expression for exact inference is only feasible if all variables are discrete, with the possible exception of o (e.g. as in hidden Markov models, or HMMs) [1]. Unfortunately, even if this prerequisite is met, obtaining the actual expression is known to be, in general, an NP-hard problem. In the last few decades, however, many automatic algorithms have been devised to make this problem more tractable [5], [2].

Other solutions to help remove some of the burden from the automatic algorithms is to model "intelligently" during the conceptual phase, taking advantage of conditional independence between variables and using hierarchical methods to decompose the model into submodels, such as data fusion, probabilistic subroutines, mixture models and other constructs, or by resorting to the Markov property to allow for recursive/iterative Bayesian updates – please refer to [1] for more information. Fortunately, most models found in artificial cognitive system development and research are already relatively simple in form (HMMs and their derivatives, for example, are ubiquitous), or hierarchical in nature – many examples of typical models can be found in [1], [2].

But, even if an expression of the form (1) is obtainable, the dimensionality of the model – in particular the cardinality of the involved random variables – in most cases makes computations using ordinary processing units impossible to perform in a satisfactory amount of time, and consequently researchers have turned to approximate inference methods instead [5], [2].

In this text, we will present a generalisable method to make inference tractable for a considerable amount of high-dimensional Bayesian models commonly used in robotics

This work was supported by the Portuguese Foundation for Science and Technology (FCT) and by the European Commission via the COMPETE programme [project grant number FCOMP-01-0124-FEDER-028914, FCT Ref. PTDC/EEI-AUT/3010/2012].

[a] AP4ISR team, Institute of Systems and Robotics (ISR) Dept. of Electrical & Computer Eng., University of Coimbra. Pinhal de Marrocos, Pólo II, 3030-290 COIMBRA, Portugal jfilipe@isr.uc.pt.

[b] Jorge Dias is also with Khalifa University of Science, Technology, and Research Abu Dhabi 127788, UAE.

applications, by resorting to the "single-instruction, multiple-data" paradigm (SIMD) in order to exploit the inherent *data-level parallelism* of Bayesian inference. A survey on related work is presented in a sister publication [5], also submitted to this workshop.

II. From Computability to Tractability

To illustrate the problem created by the "curse of dimensionality", let us assume that we have obtained a computable form for expression (1) and examine its computational implications.

Assuming $M = \langle F \rangle$, the set of probability values of the posterior $P(S \mid o)$ may be represented in compact form as a probability vector of size equal to the cardinality $N = \langle S \rangle$. This posterior would therefore result from *independent* computations performed on the corresponding pairs of elements of the M pairs of probability vectors of size N,

$$P(S \wedge [F=j]) = \left[\begin{array}{cccc} P_{s_1}([F=j]) & P_{s_2}([F=j]) & \cdots \end{array} \right]_{1\times N}$$
$$P(o \mid S \wedge [F=j]) = \left[\begin{array}{cccc} f(s_1,[F=j]) & f(s_2,[F=j]) & \cdots \end{array} \right]_{1\times N}$$

normalised by the scalar $P(o)$. The direct application of the non-optimised (i.e. "brute force") expression of equation 1 would imply $N \times M$ sums of products per vector element (M of which are shared by both the numerator and the denominator), which means a grand total of $N \times ((N \times M) - 1)$ sums and $N \times (N \times M) + 1$ products[1].

To begin the path from computability to tractability, general optimisations may be applied to the inference equation so as to reduce the amount of computations needed. Consider a concept popularly used to optimise inference for occupancy grids, which rely on a binary searched variable, the *odds ratio* [1], defined as

$$odds(S) = \frac{P([S=1])}{P([S=0])}. \tag{2}$$

One can adapt and generalise it to define the *probability ratio* for discrete random variables, defining it as

$$ratio(S=i) \equiv \frac{P([S=i])}{P([S=0])}, \tag{3}$$

considering, with no loss of generality, the support of the searched random variable as non-negative integers instantiated as i ranging from 0 to $N-1$. If explicitly considering the dependence on a specific value for the free variable F, this expression further generalises to

$$ratio(S=i, F=j) \equiv \frac{P([S=i] \wedge [F=j])}{P([S=0] \wedge [F=j])}. \tag{4}$$

analogously considering, again with no loss of generality, the support of the free random variable as non-negative integers instantiated as j ranging from 0 to $M-1$.

[1]Including the division by $P(o)$, represented as a product of an inverse of the scalar.

As with odds, the most important benefit reaped from using this representation is that inference now corresponds to computing, for each of the N elements of the *posterior ratio vector*, the following expression [6],

$$y_i \equiv \frac{\sum_j a_{i,j} x_{i,j}}{\sum_j a_{0,j} x_{0,j}} = \frac{\sum_j a_{i,j} x_{i,j}}{z}, \tag{5}$$

where

- $y_i = ratio(S = i \mid o)$ is the i^{th} element of posterior ratio vector $\mathbf{y}$ (conditional on the instantiation o, with $y_0 = 1$);
- $a_{i,j} = ratio(S = i, F = j)$ is the i^{th} element of the j^{th} prior ratio vector $\mathbf{a}_j$;
- $x_{i,j} = P(o \mid [S = i] \wedge [F = j])$ is the i^{th} element of the j^{th} likelihood vector $\mathbf{x}_j$; and,
- $z = \sum_j a_{0,j} x_{0,j}$ is a scalar (constant), dependent on observed data o and on vector elements 0 for each vector j, but independent of which vector element i is being computed.

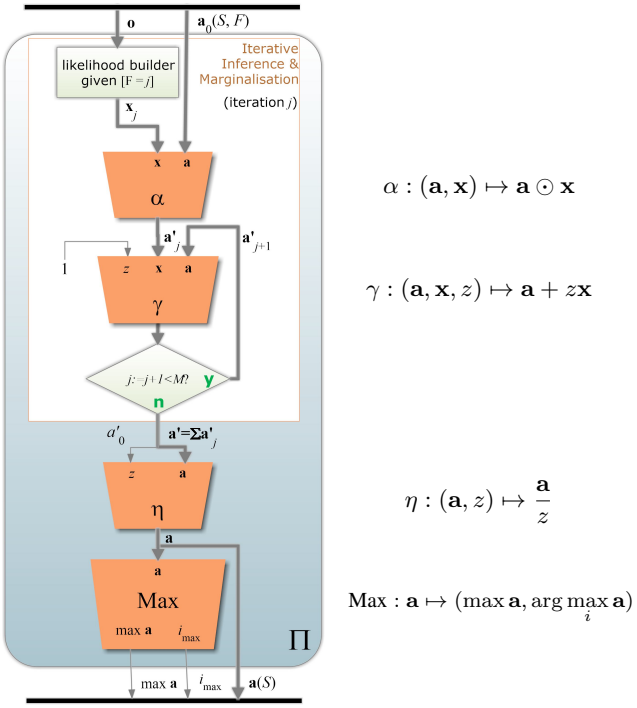
This implies that we have reduced the computations needed to compute the posterior (ratio) to the grand total of $N \times (M-1)$ sums and $N \times M + 1$ products (i.e. by a factor of N), while retaining the capability of, for example, computing a maximum a posteriori (MAP) point estimate directly as the maximum of the posterior ratio vector.

Regardless of the reductions in operations needed to perform inference, more ambitious modelling using high-dimensional variables will soon render computational times impractical, whatever the processing speed available, if computations are performed sequentially. In the last few decades, however, massively parallel computing-capable devices such as GPUs have flourished as generic APIs[2] have become available, such as CUDA [7], developed specifically for NVIDIA graphics cards, or the OpenCL framework [8], developed to execute across heterogeneous platforms consisting of central processing units (CPUs), graphics processing units (GPUs), digital signal processors (DSPs), field-programmable gate arrays (FPGAs) and other processors. This provides the opportunity of applying the "single-instruction, multiple-data" (SIMD) paradigm to make exact Bayesian inference tractable, since expression (5) is already adequately vectorised for this effect.

In a nutshell, this enables the development "inference modules", such as idealised in Figs. 1 and 2. To implement inference, these modules use as basic building blocks the elementary SIMD operations defined next.

We will start by defining an operation that receives two probability ratio vectors, or a probability ratio vector and a likelihood vector, both of size N, and performs a Hadamard

[2]Application Programming Interfaces.



$$\alpha : (\mathbf{a}, \mathbf{x}) \mapsto \mathbf{a} \odot \mathbf{x}$$

$$\gamma : (\mathbf{a}, \mathbf{x}, z) \mapsto \mathbf{a} + z\mathbf{x}$$

$$\eta : (\mathbf{a}, z) \mapsto \frac{\mathbf{a}}{z}$$

$$\text{Max} : \mathbf{a} \mapsto (\max \mathbf{a}, \arg\max_i \mathbf{a})$$

Fig. 1. Conceptual execution diagram for SIMD inference module, with respective data flow, of the general case of inference following equation (5). Note that this module requires that prior probability ratios for each value of the conjunction $[S \wedge F]$, denoted here as $\mathbf{a}(S,F)$, are available. The definition of the elementary SIMD operations used in the modules represented as blocks labelled α, γ and η can be found in the main text.

product[3],

$$\begin{aligned} \alpha : \mathbb{R}^N_{\geq 0} \times \mathbb{R}^N_{\geq 0} &\mapsto \mathbb{R}^N_{\geq 0} \\ (\mathbf{a}, \mathbf{x}) &\mapsto \mathbf{a} \odot \mathbf{x}. \end{aligned} \tag{6}$$

Another operation allows performing linear combinations between probability ratio vectors,

$$\begin{aligned} \gamma : \mathbb{R}^N_{\geq 0} \times \mathbb{R}^N_{\geq 0} \times \mathbb{R}_{\geq 0} &\mapsto \mathbb{R}^N_{\geq 0} \\ (\mathbf{a}, \mathbf{x}, z) &\mapsto \mathbf{a} + z\mathbf{x}. \end{aligned} \tag{7}$$

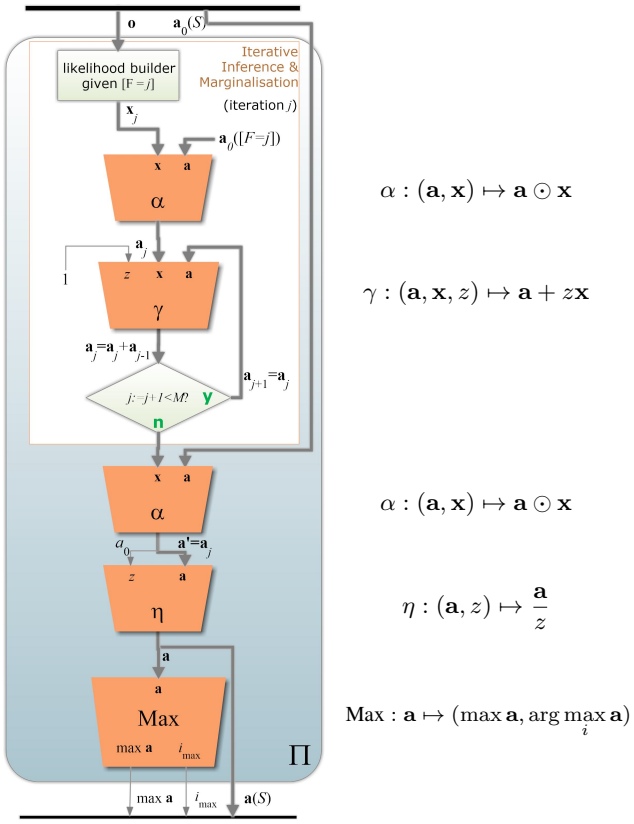
Next, the normalisation operation divides all the elements of a probability ratio vector by a scalar,

$$\begin{aligned} \eta : \mathbb{R}^N_{\geq 0} \times \mathbb{R}_{\geq 0} &\mapsto \mathbb{R}^N_{\geq 0} \\ (\mathbf{a}, z) &\mapsto \frac{\mathbf{a}}{z}. \end{aligned} \tag{8}$$

Finally, the maximum operation allows determining the largest element of a probability ratio vector and its respective index, which is the equivalent to determining the MAP point estimate of the corresponding probability distribution,

$$\begin{aligned} \text{Max} : \mathbb{R}^N_{\geq 0} &\mapsto \mathbb{R}_{\geq 0} \times \mathbb{N}_{\geq 0} \\ \mathbf{a} &\mapsto (\max \mathbf{a}, \arg\max_i \mathbf{a}). \end{aligned} \tag{9}$$

[3]An element-by-element multiplication between two vectors or matrices, denoted here as $\odot$ so as to prevent ambiguities, especially concerning function composition.



$$\alpha : (\mathbf{a}, \mathbf{x}) \mapsto \mathbf{a} \odot \mathbf{x}$$

$$\gamma : (\mathbf{a}, \mathbf{x}, z) \mapsto \mathbf{a} + z\mathbf{x}$$

$$\alpha : (\mathbf{a}, \mathbf{x}) \mapsto \mathbf{a} \odot \mathbf{x}$$

$$\eta : (\mathbf{a}, z) \mapsto \frac{\mathbf{a}}{z}$$

$$\text{Max} : \mathbf{a} \mapsto (\max \mathbf{a}, \arg\max_i \mathbf{a})$$

Fig. 2. Conceptual execution diagram for SIMD inference module, with respective data flow, assuming independent priors for S and F. Note that this module avoids the need for prior probability ratios for each value of the conjunction $[S \wedge F]$ (denoted as $\mathbf{a}(S,F)$ in Fig. 1), making Bayesian updates straightforward. This can in turn be used to implement hierarchical processes, such as data fusion, temporal filters or probabilistic subroutines by resorting to the Markov property [1]. In all cases, the "Max" operation block, which computes the maximum a posteriori (MAP) point estimate given the output posterior ratio vector $\mathbf{a}(S)$ (see main text for an explanation) is optional, and will be only used, for example, if the module is standalone, or if it is the last in a chain of modules. The definition of the elementary SIMD operations used in the modules represented as blocks labelled α, γ and η can be found in the main text.

Considering these operations, if we want to apply the SIMD paradigm to exact Bayesian inference, we should be prepared to reserve memory for *at least* $2 \times N$ vector elements and a scalar at any given moment during execution to perform a fully parallel computation of the type $\mathbf{a}_{n+1} = f(\mathbf{a}_n, \mathbf{x}, z)$. Note that we are already counting on reusing data structure $\mathbf{a}$, which makes sense given the Bayesian update notion. Fortunately, due to the complete independence of the computations involved in computing f, inference may be decomposed into several SIMD implementations, resulting in a mixed parallel-sequential computation solution.

The operations defined above are universal for any inference process performed in this fashion. However, in Figs. 1 and 2 there are additional computational blocks, each of which referred to as a "likelihood builder". These blocks are model-specific, and may be implemented in one of two ways:

1) the block computes the likelihood vector (using the

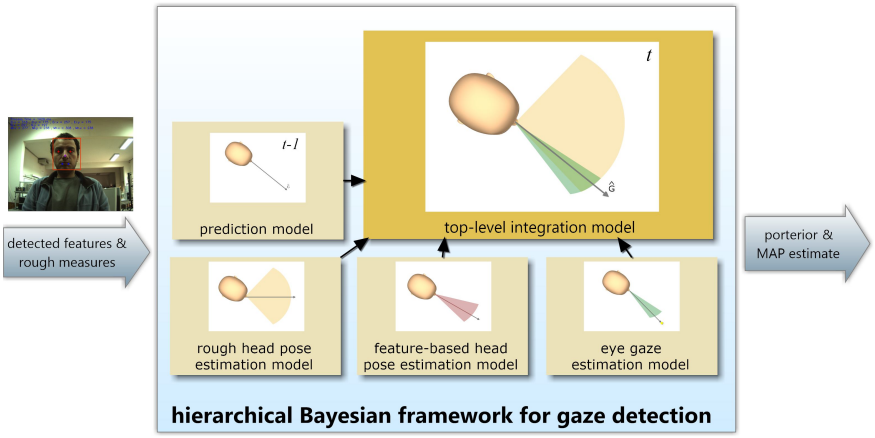


Fig. 3. Overview of the gaze estimation case-study framework.

SIMD paradigm) *at runtime* as a function of o and F;

2) the block reads a *predefined* likelihood vector from data storage (be it from a buffer in memory, or from disk) as a subset indexed by $(o, [F = j])$ of all possible vectors given all possible values for the observed data and free variables.

The former has the advantage of needing only memory space to hold the final likelihood vector of N elements, but it assumes that the likelihoods are computable as a function of o and F, which might not always be true (for example, if the likelihoods result from learnt probability tables, and not from parametric distributions), and also that the SIMD execution time of such a computation is tractable.

The latter, on the other hand, is more general in the sense that it allows using learnt probability tables as well as (pre-)computed likelihood values, and implies only the time needed for a (access-)read-write operation. However, it requires a countable set of K different possible observed data (i.e. in this case, random variable O must be discrete) and a resulting storage space sufficient to store $N \times M \times K$ elements, which might be unmanageable. Of course, a hybrid solution – for example, storing K values function of o, but allowing for manipulation of these values as a function of F and S – may be implemented, to achieve a balance between the pros and cons of both approaches.

We therefore propose that the modules introduced in Figs. 1 and 2 can be used to develop *fast, and potentially real-time, implementations of inference processes* for many popular classes of models found, for example, in a substantial amount of robotic applications. These modules will be specifically used to implement inference for the case study example presented in section III to illustrate the usefullness of this approach.

III. Case Study – A Hierarchical Bayesian Framework for Gaze Estimation

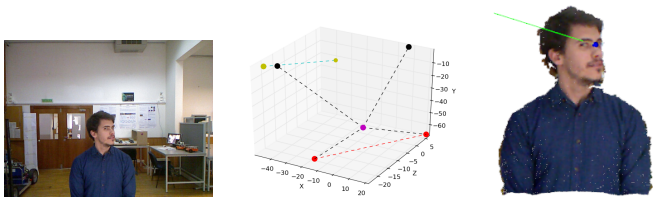
The problem of gaze estimation consists in establishing $G \equiv \{\mathbf{se}_X, \mathbf{se}_Y, \mathbf{se}_Z, \theta, \phi\}$, a conjunction of five discrete random variables composing the gaze vector, including the 3D coordinates of its initial point, the anatomical landmark named *sellion* ($\mathbf{se}$) corresponding to the midpoint between the eyes, and pitch and yaw direction angles, denoted as θ and ϕ respectively. To solve this problem we have designed a hierarchical framework comprised of one top integration level and several submodels, denoted generically as π_i, each of which representing a specific strategy assuming different levels of incompleteness in the data perceived by the sensors.

As seen in the framework overview shown in Fig. 3, at each time instant, the hierarchy receives inputs including a preliminary estimate of the general orientation of the head, a rough estimate of an origin for a head-centred frame of reference, and 3D positions of detected facial features relating to that frame, yielding as a result a posterior distribution and respective *maximum a posteriori* (MAP) estimate $\hat{G}$. It also uses the posterior of the previous time instant as prior knowledge for a prediction model under a smooth motion assumption. The fact that the framework uses 3D coordinates of facial features as inputs makes it flexible enough to be used with any combination of feature detectors of choice and image-based 3D sensors (e.g. stereo rigs, RGB-D sensors or 3D cameras).

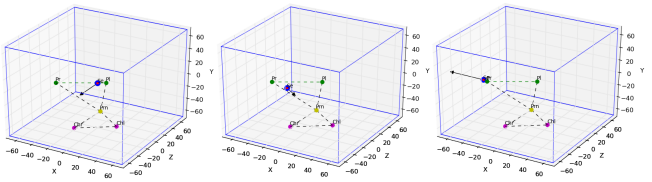
The final gaze orientation estimate is given by the "consecutive" refinement in the *top-level integration model* of the output given by the hierarchy's submodels:

- The *rough head pose estimation model*, yielding an approximate measure of head orientation as frontal or left or right profile, obtainable using the minimum amount of data sensed at a given instant.
- The *feature-based head pose estimation model*, that uses the restrictions imposed by whatever facial features are detected at a given instant, which should be available at a reasonable distance to the artificial observer.
- The *eye gaze estimation model*, that uses the relative positions of the irises within each eye, which should be available when the interlocutor is in close proximity to the artificial observer, allowing a more fine-grained estimation of a fixation point.
- The *prediction model*, that uses information from previous estimation steps.

An example of the result of the inference process in



(a) RGB scene (b) Input data (c) Final estimate



(d) Rough head model (e) Head pose model (f) Eyes gaze model

Fig. 4. Inference process. Facial features are used by the submodels to infer the best feasible gaze estimation and then the mixture model combines them to provide the final estimate. Red dots are the extremities of the lips, purple dot is the nose, black dots are the eyes and cyan dots are the extremities of the eyes. The overall estimate given by the top-level model is shown in (c) as a green vector with a blue initial point superimposed on a 3D reconstruction of the Kinect point cloud.

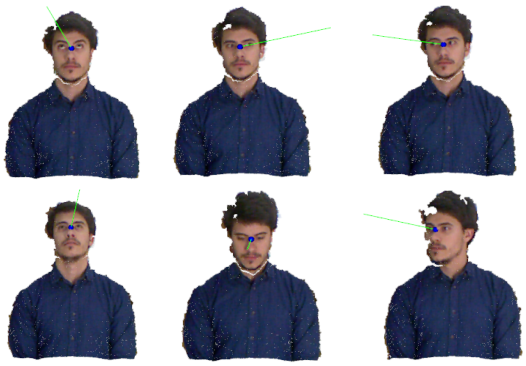


Fig. 5. Results for different combinations of eye gaze and head pose directions. In each case, a 3D reconstruction of the subject and the gaze estimate are shown, using the same convention as in Fig 4c.

a given instant is depicted in Fig. 4. The final overall estimate yielded by the top-level model is shown in Fig. 4c, and reflects how the framework outputs a solution that is consistent with the geometry of the features given as input. An additional sequence of results that attest to the correctness of the model is shown in Figure 5, demonstrating the consistency of the model for a set of very diverse situations, ranging through many different combinations of head poses and eye gaze directions. Even when, for a specific situation, head pose and gaze directions are in blatant contradiction, the framework still yields a perfectly acceptable result.

IV. Results and Discussion

The hierarchical Bayesian framework presented in section III was nearly wholly implemented using the SIMD inference module of Fig. 2 to compute partial inference results for each of its composing models. In fact, each of these was further decomposed so that each conditionally

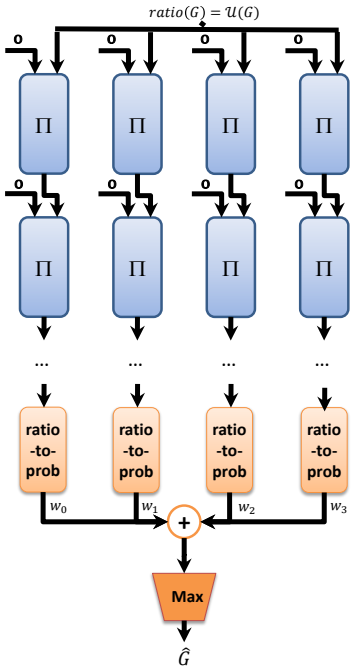


Fig. 6. Simplified diagram showing SIMD implementation of the case-study framework presented in Fig. 3. All four submodels have similar implementations, and compose the four vertical processing lanes in the diagram above the ellipsis marks. Each lane applies consecutive Bayesian updates using the SIMD inference module Π of Fig. 2, based on observations $\mathbf{o}$ that correspond to facial features or angles composing the underlying geometries. Note that each lane has a different size of Π-cascades, and that each module Π has different corresponding incoming observations and "likelihood builders", a situation that is simplified in this diagram by abuse of notation, so as to improve readability. The posterior ratios computed by the four lanes are then integrated by the top-level model (under ellipsis marks) – they are transformed into probability distributions and subsequently linearly combined in SIMD-fashion through a weighted sum, by iteratively using operation γ of equation (7). Finally, the MAP estimate of gaze direction vector $\hat{\mathbf{G}}$ is computed using equation (9).

independent likelihood function would be taken into consideration separately and then its effect integrated through consecutive Bayesian updates, therefore trading off memory usage at the expense of execution time (see Fig. 7). The SIMD implementation of the hierarchy is shown in Fig. 6. The only submodel which was not implemented using the proposed SIMD solution was the top-level integration model: since it is a mixture model of the posterior distributions of the underlying low-level models, it is a degenerated model with no observations and only searched and free variables. We found that, for this type of models, it is more efficient to convert all of the posterior distributions from ratios to probabilities, and then perform the corresponding weighted sum that the mixture model implies in SIMD fashion, instead of using the SIMD inference module of Fig. 2. The remaining models all use the SIMD inference module – all involve searched, free and known variables, and the eye gaze estimation model additionally involves an additional free variable concerning distance to fixation, which is not needed for the final gaze vector estimate.

The performance of the SIMD solution proposed in this paper was compared against a CPU implementation us-

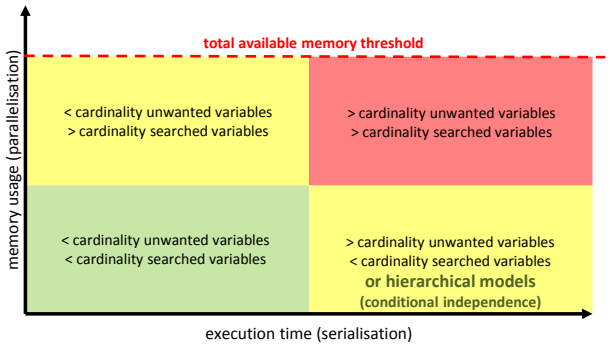


Fig. 7. Bayesian model taxonomy and its implications on a mixed parallel-sequential SIMD-based solution.

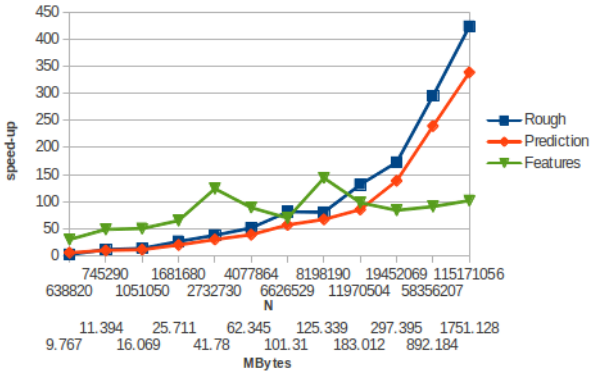


Fig. 8. Speedup according to searched variable cardinality N and corresponding GPU memory usage. All models benefit from a speedup globally proportional to N – the speedup curves are not smooth throughout, however, because $\mathbf{G}$ is a composite searched variable, and each of its components influences the "likelihood builders" for each model differently from one another. This effect is particularly visible for the feature-based head pose estimation model, which exhibits two local maxima.

ing a professional inference engine, ProBT® v2.3.0 by ProBAYES, a generic inference programming tool that facilitates the creation of Bayesian models and their reusability using the Bayesian programming formalism [2]. This library is a powerful programming tool for low-to-medium complexity models, and also allows for easy design, prototyping and testing of high complexity models, such as the framework presented in section III.

The computational resources used in the preliminary performance study presented next consist of a pair of GPUs, two Asus GTX780 3 GB DirectCU II OC GPU units, used to execute the proposed SIMD-based solution, and a four-core CPU, an INTEL Core i7-3770, used to execute its traditional, fully sequential counterpart. These resources are supported by 8 GB of RAM memory (Kingston HyperX Blu DDR3-1600 Mhz), 128 GB of SSD storage via a KINGSTON SSDNow V200, and a Seagate Barracuda 1 TB hard disk. Unfortunately, we could not yet use the eye gaze estimation model in this study due to several debugging issues regarding the definition of its "likelihood builder", which we are close to solving.

In Figs. 8 and 9, results are presented that show the remarkable speed-ups obtained by using our solution as compared to the baseline ProBT inference engine performance.

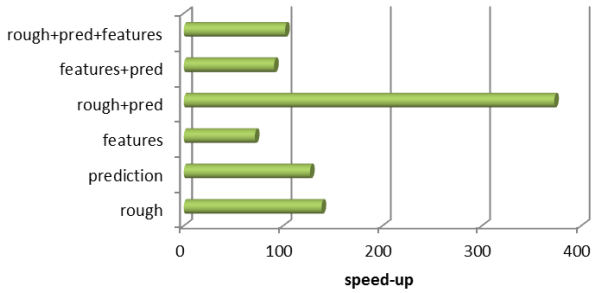


Fig. 9. Overall speedup for each model/combination of models.

These results are all the more impressive when considering the Kullback-Liebler divergence between ProBT and SIMD posterior probability distributions is in the order of 10^{-12}. In conclusion, these results clearly show that the execution times obtained using our solution will allow for estimating gaze with resolutions that are required by HRI applications at rates greater than 1 fps, and it is currently being integrated in a full-fledged system for HRI – see [9].

As suggested by these results, our proposed solution allows in general for the tractable implementation of complex, high-dimensional models such as the case-study presented herewith. However, this case-study has been decomposed in order to comply with computational resource limitations exclusively "by hand". Therefore, to be able to use the concept presented in this text in an even more generalised and useful way, it would be important to devise an automatic (or at least semi-automatic) procedure so as to determine the *computational tree* for each model one would wish to implement in a tractable fashion, in order to fully take advantage of the taxonomy presented in Fig. 7 and therefore exploit *structure-level parallelism* [5]. One possible way to do this would be to adapt state-of-the-art procedures [5], so as to deal with ratios instead of probabilities.

REFERENCES

[1] J. F. Ferreira and J. Dias, *Probabilistic Approaches for Robotic Perception*, ser. Springer Tracts in Advanced Robotics (STAR). Springer, 2014, vol. 91.

[2] P. Bessière, E. Mazer, J.-M. Ahuactzin, and K. Mekhnacha, *Bayesian Programming*, ser. Machine Learning & Pattern Recognition. Chapman & Hall/CRC Press, 2014.

[3] P. Bessière, C. Laugier, and R. Siegwart, Eds., *Probabilistic Reasoning and Decision Making in Sensory-Motor Systems*, ser. Springer Tracts in Advanced Robotics. Springer-Verlag, 2008, vol. 46.

[4] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. Cambridge: MIT Press, 2005.

[5] J. D. Alves, J. F. Ferreira, J. Lobo, and J. Dias, "Brief Survey on Computational Solutions for Bayesian Inference," in *Workshop on Unconventional computing for Bayesian inference (UCBI), IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

[6] BAMBI, "Deliverable D2.1 – Abstract probabilistic model of biochemical cascades," Bottom-Up Approaches to Machines Dedicated to Bayesian Inference (EC FP7 FET Project, Grant Agreement N° 618024), January 2014.

[7] NVIDIA, "CUDA Programming Guide ver 1.2," 2007.

[8] J. Stone, D. D. Gohara, and G. Shi, "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems," *Computing in Science & Engineering*, vol. 12, no. 3, pp. 66–73, 2010.

[9] P. Lanillos, J. F. Ferreira, and J. Dias, "Designing an Artificial Attention System for Social Robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.