Learning to Approximate Computing at Run-time

Paulo Garcia*, Mehryar Emambakhsh*, Andrew Wallace
School of Engineering and Physical Sciences, Heriot-Watt University
{p.garcia, m.emambakhsh, a.m.wallace}@hw.ac.uk

***Abstract*—Intelligent sensor/signal processing systems are increasingly constrained by tight power budgets, especially when deployed in mobile/remote environments. Approximate computing is the process of adaptively compromising the accuracy of a system's output in order to obtain higher performance for other metrics, such as power consumption or memory usage, for applications resilient to inaccurate computations. It is, however, usually statically implemented, based on heuristics and testing loops, which prevents switching between different approximations at run-time. This limits approximation versatility and results in under- or over-approximated systems for the specific input data, causing excessive power usage and/or insufficient accuracy, respectively. To avoid these issues, this paper proposes a new approximate computing approach by introducing a supervisor block embedding prior knowledge about runtime data. The target system (i.e., signal processing pipeline) is implemented with configurable levels and types of approximations [1]. Data processed by the target system is analysed by the supervisor and the approximation is updated dynamically, by using prior knowledge to establish a confidence measure on the accuracy of the computed results. Moreover, by iteratively evaluating the output, the supervisor block can learn and subsequently update tunable parameters, to improve the quality of the results. We detail and evaluate this approach for tracking problem in computer vision. Results show our approach yields promising trade-offs between accuracy and power consumption, achieving 2.54% energy saving for our case study.**

***Index Terms*—Tracking, EKF, Approximate Computing, Prior Knowledge, Field Programmable Gate Array (FPGA)**

I. Introduction

Power/performance trade offs are well established compromises in the design of all embedded systems [2]. In both hardware and software domains, there is a great deal of formal and empirical knowledge which guides system architects towards optimal design time decisions, and myriad runtime operation modes (i.e., power saving modes) controlled locally or remotely [3]. Approximate computing promises unprecedented power savings by introducing a trade off between power and another dimension: accuracy [4]. For applications resiliant to innacurate computations [5], or where there isn't a single golden result [6], approximate computing methods can improve traditional design strategies for power reduction: essential in the dark silicon era.

Despite its promise, approximate computing is still an immature technology: a formal model of the impact of approximations on other design metrics does not yet exist [7]. Hence, most approximate computing applications require two premises to be implemented successfully: (a) adequate test data are available, to correctly model the accuracy impact of approximations [8]; and, (b) approximations are performed iteratively at design time, to meet the required power/accuracy goals, and remain static throughout deployment [9].

This is in stark contrast to performance/power trade offs, where well established benchmark suites offer near total coverage of application scenarios [10]: in approximate computing, test data that allows adequate modeling of accuracy is often unavailable. In performance/power trade-offs, systems can self-tune their operation based on load and run time parameters to dynamically adjust metrics [1]. In approximate computing, approximations are static: mainly because there is no trusted method to determine if accuracy suffices, without access to ground truth [5]. In this paper, we tackle this problem: adjusting the level of approximations at run time, for signal processing applications. Our hypothesis states that prior knowledge about processed data can guide built-in approximation engines, dynamically modifying the level of approximation whilst ensuring that accuracy suffices for the required task. We define prior knowledge as any heuristics or statistical assumptions about the data, which have been empirically or formally verified.

Specifically, this paper offers the following contributions:

- We introduce the concept of prior knowledge-guided approximations. This represents a statistical measure of approximation impact, unlike test data-based empirical measures prevalent in the state of the art [9]. To be more specific, we use Kullback-Leibler (KL) divergence to evaluate if approximation error is within the acceptable bounds, changing approximation level accordingly at run-time.
- We introduce a model of run time approximations, which use prior knowledge to ensure that accuracy suffices, without access to ground truth, unlike iterative comparisons to ground truth prevalent in the state of the art [4], [8]. Our approach dynamically modifies the level of approximation in function of KL divergence value.
- We describe and evaluate a proof of concept of our approach, using an Extended Kalman Filter (EKF) for target tracking [11], where we have prior knowledge about the target's motion. Specifically, we assume that the object's motion statistics remain Gaussian.

The remainder of this paper is organized as follows: Section II describes a top level view of our methodology, explaining how prior knowledge can guide approximations dynamically. Section III describes a case study of our proposed method, where prior knowledge is used to dynamically adjust the approximations applied to an EKF for tracking. Section IV describes our experimental setup and the obtained results.

*these authors contributed equally to this work.

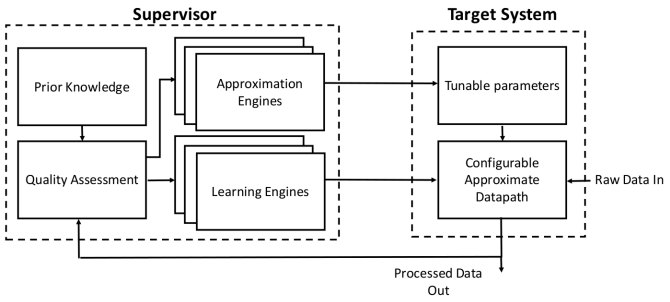


Fig. 1: Block diagram

Finally, Section V presents our conclusions and future work.

II. Approximation Methodology

Our methodology is based on equipping the processing pipeline with variable levels of approximation, i.e., configurable approximations, and an approximation engine within a supervisor block (which may contain additional optimisations such as parameter tuning): a block diagram is depicted in Fig. 1. Depending on the application and the deployment technology (e.g., CPU, ASIC, FPGA) the nature of approximations varies, but our method is applicable across the entire spectrum. Traditional approximation methods include bit width reduction [4], memoisation [12], predictive memory access [13], arithmetic re-writes [9], input-based approximations [14], etc.

Based on prior knowledge about the data, our approximation engine dynamically monitors the processing pipeline's output, and verifies whether or not the calculations still obey the assumptions about the data. If yes, then it is assumed that current levels of accuracy are still within error bounds: hence, the pipeline can be approximated further. If not, then it is assumed that accuracy has exceeded error bounds, and the level of approximation is reduced. Using this method, it is possible to converge on an approximation strategy that optimises power consumption *in situ*, without access to ground truth.

Our approach is predicated on runtime-configurable levels of approximation. In software solutions running on CPUs and GPUs, this can be achieved through different software versions [15] or through Instruction Set Architecture (ISA) level approximations [1]. In bespoke hardware solutions implemented on ASIC or FPGA, through configurable hardware versions which clock- or power-gate accurate circuitry; this is the approach we use on our experiments, which we detail in Section IV.

III. Case Study: EKF Tracking

The proposed approximate computing algorithm is evaluated for a non-linear, Gaussian, single-target tracking problem. It is assumed that the target is circularly rotating over the xy-plane. To have the paper self-contained and to simplify explaining the points at which approximation is imposed, the motion model and EKF equations are detailed in this section.

It is assumed that the sensor measures the range and bearing (azimuth) of the target in meters and radians, respectively. This

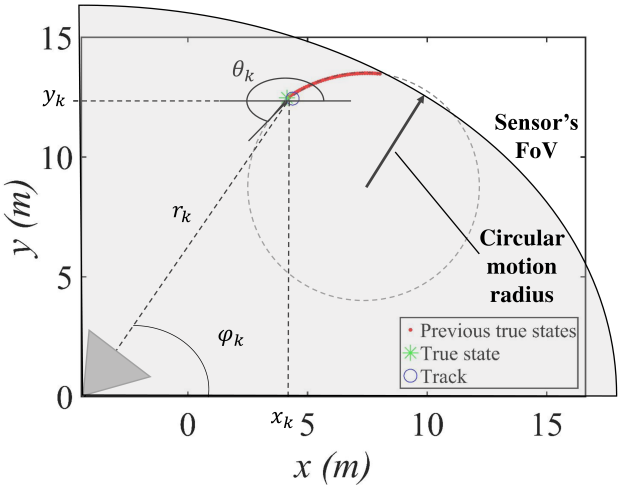


Fig. 2: The tracking scene containing the FoV of the sensor, target motion trajectory and state parameters definition.

is a typical sensing scenario in LiDAR, radar and stereo vision used in numerous applications, e.g., autonomous vehicles [16],

$$\begin{cases} r_k = \sqrt{\bar{x}_k^2 + \bar{y}_k^2} + n_{r_k} \\ \phi_k = atan2(\bar{y}_k, \bar{x}_k) + n_{\phi_k} \end{cases}, \tag{1}$$

where $\bar{x}_k$ and $\bar{y}_k$ are the true states of the target location, $\mathbf{Z}_k = [r_k, \phi_k]$ contains the measured range and bearing values, and $\mathbf{N}_k = [n_{r_k}, n_{\phi_k}]$ is a random vector sampled from a Gaussian with zero mean, and all are at the k^{th} iteration. $atan2(.)$ computes the angle between the point (vector) $[\bar{x}_k, \bar{y}_k]$ and positive x-axis.

It is assumed that at time k, the target has a three dimensional state vector $\mathbf{X}_k = [x_k, y_k, \theta_k]^\intercal$, in which $[x_k, y_k]$ and θ_k are the position and pose states of the target, respectively. This is shown in Fig. 2. The state estimation equations are,

$$\begin{cases} \hat{x}_k = x_{k-1} + (\Delta t) v_k \cos(w_k \Delta t + \theta_{k-1}) + m_{x_k} \\ \hat{y}_k = y_{k-1} + (\Delta t) v_k \sin(w_k \Delta t + \theta_{k-1}) + m_{y_k} \\ \hat{\theta}_k = \theta_{k-1} + w_k \Delta t + m_{\theta_k} \end{cases}. \tag{2}$$

$\mathbf{u}_k = [v_k, w_k]$ is the speed vector containing the radial and angular speed parameters in m/sec and rad/sec, respectively, which can be computed from previous state vectors. Δt is the time update resolution in seconds. $\hat{\mathbf{X}}_k = [\hat{x}_k, \hat{y}_k, \hat{\theta}_k]^\intercal$ is the estimated state vector for the k^{th} iteration. $\mathbf{M}_k = [m_{x_k}, m_{y_k}, m_{\theta_k}]^\intercal$ is a random vector sampled from a Gaussian distribution with zero mean.

Due to the non-linearity in (1) and (2), Gaussianity will not be preserved and therefore, a linear approximation is used based on Taylor series expansion. The Jacobian matrix for the estimation step is computed using 2 as follows:

$$\mathbf{J}_{X,k-1} = \begin{bmatrix} 1 & 0 & -(\Delta t) v_k \sin(w_k \Delta t + \theta_{k-1}) \\ 0 & 1 & (\Delta t) v_k \cos(w_k \Delta t + \theta_{k-1}) \\ 0 & 0 & 1 \end{bmatrix}. \tag{3}$$

Assuming $\mathbf{\Sigma}_{X,k-1}$ and $\mathbf{Q}_k$ are the state and noise covariance matrices at $k-1$ and k, the estimated state covariance matrix at iteration k will be: $\mathbf{\Sigma}_{\hat{X},k} = \mathbf{J}_{X,k-1} \mathbf{\Sigma}_{X,k-1} \mathbf{J}_{X,k-1}^\intercal + \mathbf{Q}_k$.

The correction step of EKF incorporates the measurement at time k. Let $\mathbf{R}_k$ be the measurement covariance matrix. Then the corrected state vector can be computed as follows,

$$\begin{aligned} \mathbf{X}_k &= \hat{\mathbf{X}}_k + \mathbf{K}_k(\mathbf{Z}_k - h(\hat{\mathbf{X}}_k)) \\ \mathbf{K}_k &= \mathbf{\Sigma}_{\hat{X},k}\mathbf{H}_k^\intercal \mathbf{S}_k^{-1} \\ \mathbf{S}_k &= \mathbf{H}_k \mathbf{\Sigma}_{\hat{X},k}\mathbf{H}_k^\intercal + \mathbf{R}_k \end{aligned} . \tag{4}$$

$\mathbf{K}_k$ is the Kalman gain, $\mathbf{S}_k$ is the innovation covariance and $\mathbf{X}_k$ is the (corrected) state vector at the k^{th} iteration. $h(.)$ is a non-linear function, which maps the current estimated target state to the coordinate frame of the sensor. $\mathbf{H}_k$ is the measurement's Jacobian matrix, which is computed as follows using 1,

$$\mathbf{H}_k = \frac{1}{\sqrt{x_k^2 + y_k^2}} \begin{bmatrix} x_k & y_k & 0 \\ -y_k & x_k & 0 \end{bmatrix}. \tag{5}$$

The corrected covariance matrix for the k^{th} iteration will then be,

$$\mathbf{\Sigma}_{X,k} = \left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \mathbf{K}_k\mathbf{H}_k \right) \mathbf{\Sigma}_{\hat{X},k}. \tag{6}$$

A. KL divergence for Gaussian motion

Comparison with the prior knowledge can be performed in various ways. For an iterative filtering, which estimates and corrects the state of the object using the obtained measurements, the prior knowledge can be defined using the target's motion. Any deviation from this known model can be used to compromise over approximation during run-time.

One solution to quantify such model is to stack state vectors for a given number of consecutive iterations. Then estimating the statistics of the stacked tracks, using either parametric or non-parametric approaches, can be used to map the result to our prior knowledge.

To be more specific, in this paper, we use a parametric approach to approximate the probability density function (PDF) of the stacked target states from iteration k to $k+N$, i.e. $\mathbf{s}_{k:k+N}$. Our prior knowledge is then compared with the computed approximated PDF. In our work, we use KL divergence to perform this task as follows,

$$D_{KL}(\mathbf{H}_{k,N}||\mathbf{H}_p) = \sum_i \mathbf{H}_{k,N} \log \frac{\mathbf{H}_{k,N}}{\mathbf{H}_p} \tag{7}$$

in which $\mathbf{H}_{k,N}$ is the approximated PDF of the k^{th} to $(k+N)^{th}$ state samples and $\mathbf{H}_p$ is the "prior knowledge" PDF of the target's motion.

For this experimental setting, we assume that the object's motion statistics remains Gaussian (our prior knowledge). This can be iteratively evaluated by (7). It can be easily shown that the KL divergence for two Gaussian distributions can be simplified as follows,

$$D_{KL}(\mathbf{H}_{k,N}||\mathbf{H}_p) = \log \frac{\sigma_p}{\sigma_{k,N}} + \frac{\sigma_{k,N}^2 + (\mu_{k,N} - \mu_p)^2}{2\sigma_p^2} - \frac{1}{2} \tag{8}$$

in which, $(\mu_{k,N},\ \sigma_{k,N}^2)$ and $(\mu_p,\ \sigma_p^2)$ are the mean vector and variance of the stacked samples and prior knowledge, respectively. For our experiments, we assume that μ_p is the radius of the target's circular rotation, which is known with a $\pm\sigma_p$ standard deviation (both in meters).

We define the stacked samples $\mathbf{s}_{k:k+N}$ as the Euclidean distance between the target state $[x_k, y_k]^\intercal$ and centre of rotation. The mean and variance $\mu_{k,N}$ and $\sigma_{k,N}^2$ are then computed as follows,

$$\begin{aligned} \mu_{k,N} &= \frac{\sum_i s_{k:k+N}^i}{N} \\ \sigma_{k,N}^2 &= \frac{\sum_i (s_{k:k+N}^i - \mu_{k,N})^2}{N} \end{aligned} \tag{9}$$

in which, $s_{k:k+N}^i$ is the i^{th} element in $\mathbf{s}_{k:k+N}$. Small values for $D_{KL}(.)$ correspond to higher similarity between the prior knowledge and tracks. Therefore, more intense approximation is imposed. On the other hand, the approximation level is reduced once the $D_{KL}(.)$ is higher than a given threshold.

IV. Experimental results

Figure 3 presents our methodology to obtain accuracy and power profiles for several degrees of approximation, and to generate a version with configurable approximations.

We begin by implementing EKF in Matlab and applying several algorithm-specific approximations (described in detail in Sub-Section IV-A). Matlab simulations are performed to obtain accuracy profiles for each approximation at this step. Matlab Coder is then used to generate C code corresponding to the exact and various approximated versions. C code generated by Matlab is not directly synthesizable to FPGA: hence, we refactor it manually to obtain synthesizable versions. Functionality and corresponding accuracy are not affected by this step. We then apply algorithm-independent approximations on this refactored C code, and use a High Level Synthesis tool (Xilinx Vivado HLS) to generate Verilog Hardware Description Language (Verilog HDL) exact and approximated versions. Matlab's HDL coder offers a direct route from Matlab to FPGA, but not all language constructs are supported and it would not enable us to perform algorithm-independent optimisations: hence the use of HLS through Xilinx's tools.

RTL simulation and FPGA synthesis (we target Virtex 7 technology) is performed using Xilinx Vivado Design Suite to obtain resource usage and power consumption. Power consumption is estimated through Xilinx Power Estimator tool embedded in Vivado, which uses resource usage information and switching rates obtained from RTL simulation to calculate a high accuracy measure of power consumption. This step enables us to obtain power profiles for each approximation.

The final step combines the various versions into one configurable solution. This is an FPGA implementation which combines accurate and approximated versions, where the level of approximations can be configured and modified at runtime. Unused logic (e.g., an exact version of some operation when running in corresponding approximate mode) is clock gated to eliminate dynamic power consumption. This version is then used to obtain results for dynamic approximations by the approximation engine using prior knowledge.

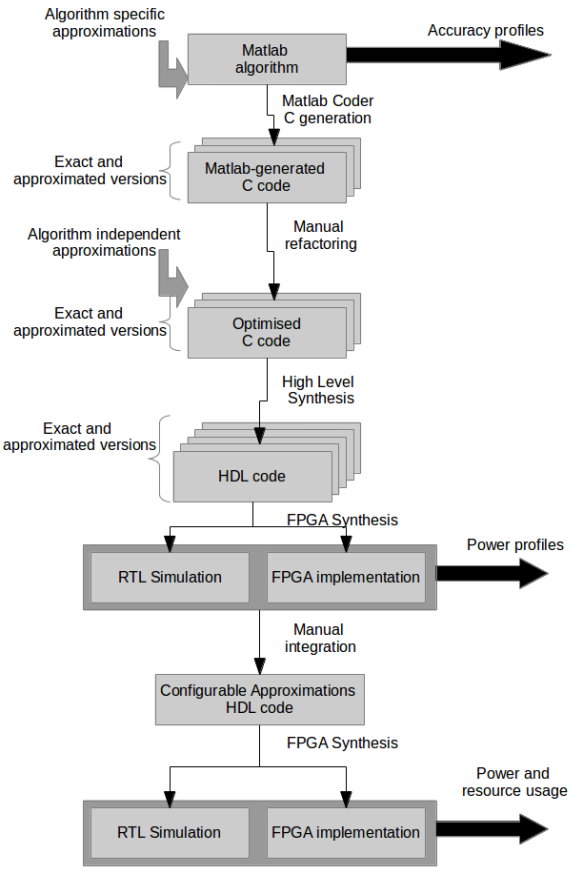


Fig. 3: Experimental design flow.

A. *Power and Accuracy Profiles*

Algorithm-dependent approximations are modifications on the implementation of certain constructs, which vary from algorithm to algorithm. These can be complete re-writes of algorithm functionality; however, since our goal is to determine the validity of prior knowledge for runtime approximations, simple arithmetic re-writes suffice. We implemented four different re-writes:

1) COS re-write: in the exact HDL version, cosine functions are implemented using look-up tables with 180 entries. In the approximated version, these are quantised to use only 90 entries, requiring fewer data than the exact version.
2) SIN re-write: identically to cosine, sine functions are re-written using a quantized loop-up table.
3) SQRT re-write: the square root of the sum of squares $\sqrt{x^2+y^2}$, which requires two DSP multipliers and a look-up table for the square root, is replaced by a sum of absolutes $|x|+|y|$, which requires only two conversions to unsigned and an adder.
4) ATAN re-write: the arc tangent function is realized in the exact version through a look up table. In the approximate version, it is approximated through the first three terms of a Taylor series, requiring only arithmetic operations, thus reducing memory requirements.

We detailed several algorithm-independent approximations in Section II. In our experiments, we implemented bit width reduction, from floating to fixed point. The default bit width is 64 bits (Matlab generates double precision floating point operations). Our approximations replaced 64 bits data with fixed point 27 bits data (15 integer and 12 fractional bits). This is one of the fixed point bit widths recommended for power reductions in the Virtex 7 family.

Version	Power (W)			% of baseline
	Static	Dynamic	Total	
Exact	0.247	0.552	0.799	100%
COS	0.247	0.523	0.770	96.37%
SIN	0.247	0.523	0.770	96.37%
SQRT	0.247	0.534	0.781	97.74%
ATAN	0.247	0.532	0.779	97.49%
27 bits Fixed Point	0.246	0.538	0.785	98.24%

TABLE I: Power profiles for exact and each individual approximation.

We ensured that during our RTL simulations, 100 iterations of the complete function are performed, using the same randomly generated input data for all versions, in order to obtain representative activity that ensures high confidence in the power estimation. Table I depicts power consumption for the exact and each individual approximated version. Static power remains constant across approximations because the contribution of on-chip memories (BRAMs) dominates, and the number remains constant except for a small reduction when reducing bit widths.

Our next set of experiments started measuring the power consumption of versions that combine bit width reduction with several of the other approximations: results are presented in Table II.

Approximations (using 27FP)	Power (W)	% of baseline
COS	0.756	94.61%
SIN	0.756	94.61%
SQRT	0.767	95.99%
ATAN	0.765	95.74%
COS & SIN	0.741	92.74%
COS & SQRT	0.767	95.99%
COS & ATAN	0.736	92.11%
SIN & SQRT	0.738	92.36%
SIN & ATAN	0.736	92.11%
SQRT & ATAN	0.747	93.49%
COS & SIN & SQRT	0.709	88.73%
COS & SIN & ATAN	0.707	88.48%
COS & SQRT & ATAN	0.718	89.86%
SIN & SQRT & ATAN	0.718	89.86%
COS & SIN & SQRT & ATAN	0.689	86.23%

TABLE II: Power profiles for combinations of different approximations.

We use Matlab simulation to determine the accuracy of each combination, compared to the baseline. Table III depicts the accuracy of each version. Results in the ”Absolute” tab are the mean and standard deviation (σ), respectively, of the absolute error (Euclidean distance in meters) compared to the ground truth (real object position) in our simulation. Results in the ”Accuracy” tab are the ratios between the exact version results and the absolute value of the difference between exact and approximate results.

Version	Absolute (m)		Accuracy (%)
	Mean	σ	
Exact	0.5633	0.3092	100
COS	0.6742	0.2805	83.55
SIN	0.6524	0.3652	86.34
SQRT	0.6968	0.3764	80.84
ATAN	3.3520	1.7953	16.80
COS & SIN	0.6044	0.3973	93.19
COS & SQRT	0.7497	0.4101	75.13
COS & ATAN	0.9771	0.6937	57.65
SIN & SQRT	0.7353	0.3711	76.60
SIN & ATAN	7.7667	2.5248	7.252
SQRT & ATAN	1.4812	0.8149	38.02
COS & SIN & SQRT	0.5870	0.3680	95.96
COS & SIN & ATAN	1.7503	1.0135	32.18
COS & SQRT & ATAN	0.9949	0.7006	56.61
SIN & SQRT & ATAN	7.7528	2.5424	7.265
COS & SIN & SQRT & ATAN	1.8295	1.0765	30.78

TABLE III: Accuracy profiles for combinations of different approximations, for 100 iterations. Exact version uses double floating point precision (64 bits); every other version uses 27 bits fixed point precision.

The final integrated version with configurable approximations, including clock gated logic, is compared against the baseline in Table IV.

	Power (W)	FPGA Resources			
	Total	LUT	FF	DSP	BRAM
Exact	0.799	46449	17666	61	55
Approx.	0.689	44995	16981	58	55

TABLE IV: Resource usage and ower consumption for baseline and final configurable approximations version.

B. Dynamic approximations using prior knowledge

The results of dynamically performing the approximation using the algorithms explained in Section II and III are detailed here. First, using the approximation results of each individual step, an "approximation level" is defined. To do this, various combinations of the micro approximations are sorted in ascending order according to their accuracy error. This results in a vector of tuples containing pairs of indexes (used as the approximation level) and strings indicating the approximation combination.

The first tuple in this vector corresponds to the approximation method, which generates the highest accuracy. Its last element, however, creates the weakest accuracy. At each filtering step, in order to use different approximation combinations, the approximation level is either increased or decreased. Determining whether to increment or decrement the approximation level is performed by evaluating the KL divergence result (8).

The result of the overlay tracking plot using dynamic approximation in various iterations is shown in Fig. 4. Red shows ground truth, green shows tracking without approximation and blue shows tracking after approximation, using KL divergence thresholds of 0.5, 1, 2, 3.

At every iteration, KL divergence is computed according to (8), with sample stack size $N = 75$. If it is less than a given threshold T (for our results in Fig. 4, $T = 0.5$, 1, 2 and 3) the approximation level is incremented; otherwise, it is

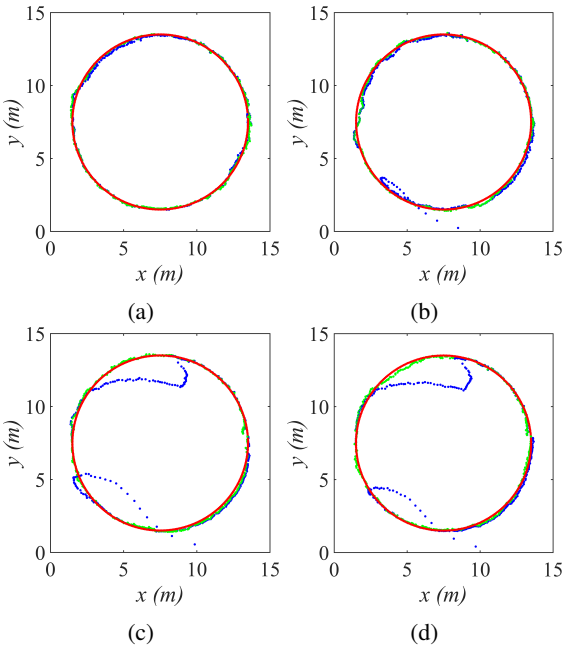


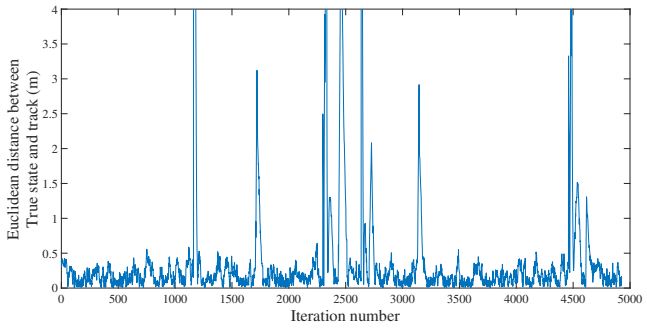
Fig. 4: Tracking results, where red, green and blue show the true state, tracking without approximation and tracking after approximation, respectively, using KL divergence threshold: (a) 0.5, (b) 1, (c) 2, (d) 3.

decremented to have a more precision tracking performance. The Euclidean distance between the found track and true state is computed for each iteration and shown in Fig. 5-a, as well as the KL divergence in Fig. 5-b. When the KL divergence goes above T in Fig. 5, the accuracy deteriorates. The approximation level is then immediately reduced in order to compensate the accuracy. Therefore, in only a few more iterations the accuracy again increases. This causes median accuracy error of 0.1661 (m) with median KL divergence 1.2098 for 5000 iterations.

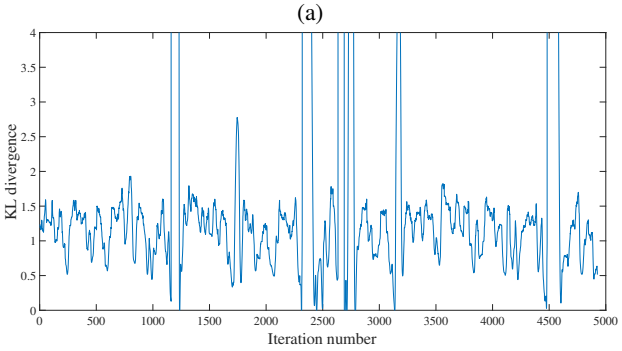
Using the power profiles in Table II, the consumed energy of each approximation can be computed by multiplying the power with the time resolution ($\Delta t = 0.1$ (s)). The accumulated energy is plotted in Fig. 6 for the last 1500 iterations. The baseline which consumes a constant power of 0.799 (W), results in a total of $\approx$ 394 (J). However, the accumulated energy consumption for the dynamic approximation case is $\approx$ 384 (J). This shows about 2.54% energy reduction. By compromising the accuracy and increasing the KL divergence threshold, more intense approximations can be obtained, which results in even lower energy consumption. This is shown in Fig. 6 as the dashed green and cyan lines for thresholds 1.5 and 4.0, respectively

V. Conclusions and Future Work

We have described the use of prior knowledge to perform runtime approximations. Our experiments, on a tracking application using EKF, demonstrate that prior knowledge can indeed be used to ensure that accuracy is within acceptable bounds, without having to perform empirical evaluations based on representative test data. Our results support the conclusion



(a)



(b)

Fig. 5: (a) Euclidean distance between the true state and tracks; (b) Corresponding KL divergence.

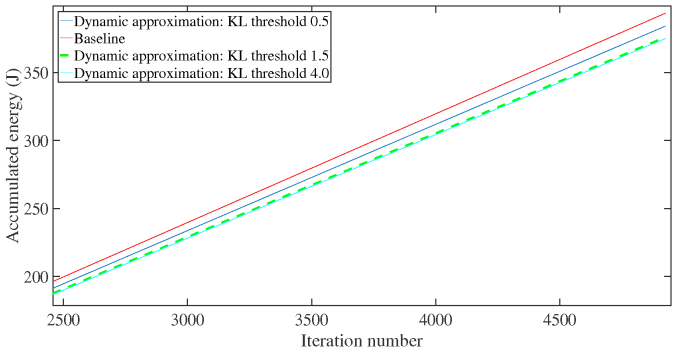


Fig. 6: Accumulated energy consumption for the baseline (no approximation) and dynamic approximations, with three different KL divergence thresholds.

that approximate computing need not be limited to design time: thus enabling power optimisations with the same degree of runtime configuration as traditional performance optimisations.

Future work will address three limitations of our study: (a) the approximation engine, which leverages prior knowledge, is external to the test system: hence, its power contribution was not modeled. We will integrate a hardware version embedded in the processing pipeline so power evaluation is completely accurate. (b) the utilized approximations are *ad hoc* heuristics, and result in a poor theoretical lower limit to power consumption (86.23%). We will adopt more sophisticated approximation methodologies, using state of the art automated tools and benchmarks, in order to obtain greater power savings. (c) We have only modeled one case study; we will experiment with several different algorithms from various domains in order to evaluate the coverage of our approach, in function of different types of application-specific prior knowledge.

Acknowledgment

We acknowledge the support of the Engineering and Physical Research Council, grant references EP/K009931/1 (Programmable embedded platforms for remote and compute intensive image processing applications), EP/K014277/1 (MOD University Defence Research Collaboration in Signal Processing) and EP/N012402/1 (TASCC: Pervasive low-TeraHz and Video Sensing for Car Autonomy and Driver Assistance (PATH CAD)).

References

[1] V. Vassiliadis, K. Parasyris, C. Chalios, C. D. Antonopoulos, S. Lalis, N. Bellas, H. Vandierendonck, and D. S. Nikolopoulos, "A programming model and runtime system for significance-aware energy-efficient computing," *SIGPLAN Not.*, vol. 50, no. 8, pp. 275–276, Jan. 2015.

[2] S. Das, J. R. Doppa, P. P. Pande, and K. Chakrabarty, "Reliability and performance trade-offs for 3d noc-enabled multicore chips," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016*. IEEE, 2016, pp. 1429–1432.

[3] S. Senni, L. Torres, G. Sassatelli, and A. Gamatie, "Non-volatile processor based on mram for ultra-low-power iot devices," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 2, p. 17, 2016.

[4] S. Mittal, "A survey of techniques for approximate computing," *ACM Computing Surveys (CSUR)*, vol. 48, no. 4, p. 62, 2016.

[5] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate computing: A survey," *IEEE Design & Test*, vol. 33, no. 1, pp. 8–22, 2016.

[6] S. Venkataramani, K. Roy, and A. Raghunathan, "Approximate computing," in *VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID), 2016 29th International Conference on*. IEEE, 2016, pp. 3–4.

[7] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing and the quest for computing efficiency," in *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 2015, p. 120.

[8] A. Yazdanbakhsh, D. Mahajan, H. Esmaeilzadeh, and P. Lotfi-Kamran, "Axbench: A multiplatform benchmark suite for approximate computing," *IEEE Design & Test*, vol. 34, no. 2, pp. 60–68, 2017.

[9] K. Nepal, S. Hashemi, H. Tann, R. I. Bahar, and S. Reda, "Automated high-level generation of low-power approximate computing circuits," *IEEE Transactions on Emerging Topics in Computing*, 2016.

[10] J. L. Henning, "Spec cpu2000: Measuring cpu performance in the new millennium," *Computer*, vol. 33, no. 7, pp. 28–35, 2000.

[11] G. Y. Kulikov and M. V. Kulikova, "The accurate continuous-discrete extended kalman filter for radar tracking," *IEEE Transactions on Signal Processing*, vol. 64, no. 4, pp. 948–958, 2016.

[12] S. Sinha and W. Zhang, "Low-power fpga design using memoization-based approximate computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 8, pp. 2665–2678, 2016.

[13] A. Yazdanbakhsh, B. Thwaites, H. Esmaeilzadeh, G. Pekhimenko, O. Mutlu, and T. C. Mowry, "Mitigating the memory bottleneck with approximate load value prediction," *IEEE Design & Test*, vol. 33, no. 1, pp. 32–42, 2016.

[14] A. Raha, H. Jayakumar, and V. Raghunathan, "Input-based dynamic reconfiguration of approximate arithmetic units for video encoding," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 3, pp. 846–857, 2016.

[15] V. Vassiliadis, K. Parasyris, C. Chalios, C. D. Antonopoulos, S. Lalis, N. Bellas, H. Vandierendonck, and D. S. Nikolopoulos, "A programming model and runtime system for significance-aware energy-efficient computing," in *ACM SIGPLAN Notices*, vol. 50, no. 8. ACM, 2015, pp. 275–276.

[16] M. B. Jensen, M. P. Philipsen, A. Mgelmose, T. B. Moeslund, and M. M. Trivedi, "Vision for looking at traffic lights: Issues, survey, and perspectives," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 7, pp. 1800–1815, July 2016.