Efficient Programmable Random Variate Generation Accelerator from Sensor Noise

James T. Meech, Phillip Stanley-Marbell, *Senior Member, IEEE*

***Abstract*—We introduce a method for non-uniform random number generation based on sampling a physical process in a controlled environment. We demonstrate one proof-of-concept implementation of the method, that doubles the speed of Monte Carlo integration of a univariate Gaussian. We show that we must measure and compensate for the supply voltage and temperature of the physical process to prevent the mean and standard deviation from drifting. The method we present and our detailed empirical hardware measurements demonstrate the feasibility of programmable non-uniform random variate generation from low-power sensors and the effect of ADC quantization on the statistical qualities of the approach.**

***Index Terms*—Sensor, Noise, Bayesian, Inference, Non-uniform, Random.**

I. Introduction

CURRENT software-based methods of non-uniform random variate generation are slow and inefficient [1], [2]. We present a programmable system capable of generating Gaussian random variates by extracting the noise properties of a MEMS sensor and demonstrate its principle and application. Sampling a random physical process that has a known distribution provides a continuous random variable with a sample rate that is only limited by the frequency of the random physical process. Table I compares thirteen state-of-the-art methods for generating non-uniform random variates. Gaussian random variate generation is typically an order of magnitude slower and less efficient than uniform random variate generation [1]. We propose a method with potential to be superior to all of the state-of-the-art methods in terms of sample rate and efficiency, consisting of a physical noise source and an ADC. In a hardware implementation, the sample rate of the ADC limits the generation rate.

A. Related Research

Uniform random numbers are widely used in cryptography [3]. The hardware *non-uniform* random number generators in Table I are fundamentally different to prior work on uniform random number generators [4], [5], [6], [7], [8], [9]. Uniform random number generators are often based on some non-uniform physical entropy source but these publications do not describe the distribution of the source. This omission makes it difficult to compare them directly to the programmable random variate accelerator (PRVA) method that we present in Section III. Table I shows that the work by Thomas et al. [1] is the fastest method for producing random variates from a normal distribution in an FPGA. This method involves transforming uniformly distributed random numbers from a pseudorandom number generator to a normal distribution. The more recent work by Guo et al. [10] achieves the same goal in an FPGA but produces approximately half the sample rate. In contrast our method generates samples with a normal distribution and uses only two operations to transform to any other normal distribution. Our method is five orders of magnitude slower than the method of Guo et al. [10]. Although we made no effort to optimize speed or energy efficiency, back-of-the-envelope calculations show room for 10^5 and 10^3 increases in speed and energy efficiency respectively.

TABLE I
COMPARISON OF STATE-OF-THE-ART PRVA METHODS. **PRVA**: PROGRAMMABLE RANDOM VARIATE ACCELERATOR. **CPU**: CENTRAL PROCESSING UNIT, **GPU**: GRAPHICS PROCESSING UNIT, **MPPA**: MASSIVELY PARALLEL PROCESSOR ARRAY, **FPGA**: FIELD PROGRAMMABLE GATE ARRAY, **MR**: MEMRISTOR, **PD**: PHOTON DETECTION, **RET**: RESONANCE ENERGY TRANSFER, **PDI**: PHOTODIODE, **EN**: ELECTRONIC NOISE, **EXP**: EXPONENTIAL, *****: THIS WORK.

Source	Speed	Efficiency	Dist(s)	PRVA	Publication
CPU	890 Mb/s	3.17 Mb/J	Normal	Yes	[1], 2009
GPU	12.9 Gb/s	108 Mb/J	Normal	Yes	[1], 2009
MPPA	860 Mb/s	403 Mb/J	Normal	Yes	[1], 2009
FPGA	12.1 Gb/s	645 Mb/J	Normal	Yes	[1], 2009
MR	6000 b/s	120 Gb/J	Unnamed	No	[11], 2017
PD	1.77 Gb/s	-	Normal	No	[12], 2017
RET	2.89 Gb/s	578 Gb/J	EXP	Yes	[13], 2018
PDI	17.4 Gb/s	-	Husumi	No	[14], 2018
PD	66.0 Mb/s	-	Arbitrary	Yes	[15], 2018
PD	320 Mb/s	-	EXP	No	[16], 2018
FPGA	6.40 Gb/s	-	Normal	Yes	[10], 2019
PD	8.25 Gb/s	-	Normal	No	[17], 2019
EN	13.8 kb/s	209 kb/J	Normal	Yes	[*], 2019

B. Generating Non-Uniform Random Variates Is Hard

Researchers in the computing systems community use the inversion and accept-reject methods for generating samples from non-uniform random variables in software [18]. Let U and X be independent uniform and non-uniform random variables respectively and F^{-1} the analytical closed-form solution for the inverse cumulative distribution function of X [18]. Algorithm 1 shows the inversion method. Which requires that F^{-1} has an analytical closed-form solution (the Gaussian distribution has no analytical closed-form solution for F^{-1}). Let g be the density of U and f be the density of X. Let $c \geq 1$ be a constant such that the condition $f(x) \leq cg(x)$ holds for all x. Algorithm 2 shows the accept-reject method for generating samples from X. When using the accept-reject method, we reject samples deviating from the desired distribution subject to probabilistic criteria [18].

C. Uses of Non-Uniform Random Variates

Non-uniform random variate generators are fundamental to applications employing Monte Carlo methods [19], such as

James T. Meech and Phillip Stanley-Marbell are with the Department of Electrical Engineering, University of Cambridge, Cambridge, CB3 0FA UK e-mail: james.t.meech@gmail.com.

Algorithm 1: Inversion method.

Result: Sample from non-uniform random variable X
Generate uniform [0, 1] random variate u
RETURN $x \leftarrow F^{-1}(u)$

Algorithm 2: Accept-reject method.

Result: Sample from non-uniform random variable X
repeat
  Generate uniform [0, 1] random variate u
  Generate uniform [0, 1] random variate x
  Set $T \leftarrow c\frac{f(x)}{g(x)}$
until $uT \leq 1$
RETURN x

population balance modeling of crystallization processes [20], ray tracing [21], and financial computing [22]. Bayesian inference evaluates Bayes' theorem to calculate $p(B|D)$ the probability that a belief B is true given new data D. This requires calculating the probability that the belief is true regardless of the data ($p(B)$), the probability that the data is true given the belief ($p(D|B)$), and the probability that the data is true regardless of the belief ($p(D)$). We will refer to $p(B)$ as the *prior*, $p(D|B)$ as the *likelihood* and $p(D)$ as the *marginal likelihood*. Bayes' theorem defines $p(B|D)$, the *posterior* as:

$$p(B|D) = \frac{p(D|B)p(B)}{p(D)}. \tag{1}$$

In practice, the analytical calculation of the marginal likelihood is impossible for all but the simplest joint distributions [23]. Implementations must instead sample from the joint distribution $p(B, D)$ to obtain summary statistics that they can use to describe it [23]. Implementations must produce these random samples from bespoke probability distributions using a non-uniform random variate generator. Low power embedded systems such as drones perform this kind of computation for particle filter localization [24].

D. Contributions

1) The observation that we can use physical noise sources such as MEMS sensors in a PRVA (Section I).
2) Estimation of speed increase and error reduction by using a PRVA for Monte Carlo integration (Section II).
3) Investigation of the impact of temperature and supply voltage on the noise distribution obtained from a commercial MEMS sensor (Section III-A).

II. Motivating Example

We performed Monte Carlo integration of a Gaussian with a mean of $\mu = 980.794$ and standard deviation of $\sigma = 7.178$ using samples from a Gaussian with the same mean and variance. We ran the experiment with a Gaussian generated by the C++ random library and repeated it with samples from the PRVA collected at 3 V and 20 °C. We saved the sensor-generated random variates in a file and then presented them to the C++ benchmark program in a 10^6 element array. We uniformly interpolate between the points in the PRVA-generated Gaussian using a [-1, 1] uniform C++ random number generator. We assume that the PRVA can produce a sample in the same amount of time that it takes to perform a read from memory. The current PRVA cannot do this but it is possible with fast ADCs sampling a physical process.

We performed the same integration using samples from a uniform distribution with a variety of ranges. Let E be the error of the integration, t be the time taken by the integration, N be the number of random samples, and D be the distribution (either uniform or Gaussian). Let S be the array of random variates, A be the area, b and h the rectangle base and height, and f the probability density function of the Gaussian for integration. Algorithm 3 shows the integration scheme that we used. We repeated each process 1000 times and calculated the average time t and error E. Figure 1 shows that the PRVA outperforms the C++ uniform random number generator for most ranges. It is only outperformed by uniform generators with a range of $\pm 10\sigma$ to $\pm 1000\sigma$. The proportion of the uniform probability density function overlapping $f(x)$ decreases as we increase the range of the uniform distribution. For a given function it is impossible to know beforehand which range of uniform random numbers will produce a sufficiently small bound on the error of integration. To avoid this problem we sample from a distribution that closely matches the distribution we integrate. The divergence between the C++ Gaussian random number generator and the PRVA is due to the lack of unique numbers in the tails of the distribution. The PRVA performs the task up to $1.4\times$ faster than the C++ Gaussian random number generator with eight threads and always $2\times$ as fast with one thread.

Algorithm 3: Monte Carlo integration. Array index starts at zero.

Result: Error E and time t
Timer start
Generate N random variates from distribution D
Sort N random variates
for $i = 1$ *to* N **do**
  b = S[i] − S[$i-1$]
  h = (f(S[i]) + f(S[$i-1$]))/2
  A += $b \times h$
end
E = abs($1 - A$)
Timer stop
t = stop − start
RETURN E, t

III. Methodology

A PRVA based on physical noise sources must have negligible drift of the mean and standard deviation over time. Drift would cause errors in calculations using the output of the PRVA. We must therefore measure and compensate for any environmental parameter that causes non-negligible drift. We sampled the z-axis of a MEMS accelerometer (the accelerometer in the Bosch BMX055) to obtain the distributions.

A. Temperature-Controlled Experiments

Figure 2 shows the experimental setup. We used a custom multi-sensor embedded system to perform the initial inves-

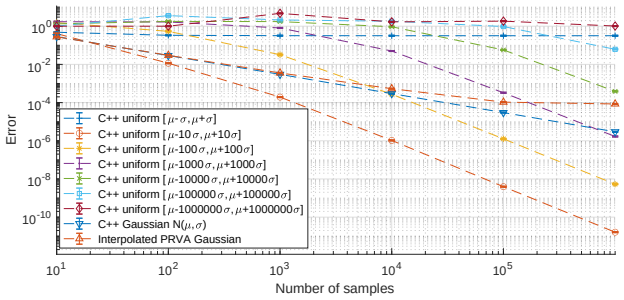

Fig. 1. The error of Monte Carlo integration depends upon the range for uniform random numbers but not for Gaussian random numbers. We plotted the error bars using a 90 % confidence interval on the mean. We compiled this code with g++-mp-7 c++11 on a 2.8 GHz Intel Core i7 CPU with 16 GB of 2133 MHz LPDDR3 RAM and OpenMP to utilize all eight threads.

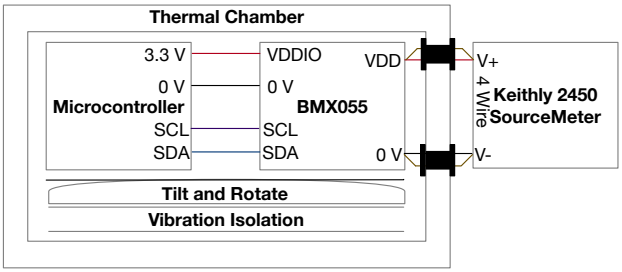

Fig. 2. Experimental setup, powering the sensor and the I2C interface separately allowed more accurate measurement of the power consumed by the sensor. The microcontroller consumed approximately 66 mW from its USB supply.

tigation into which sensors we could use to generate non-uniform random variates [25]. We placed the microcontroller, accelerometer, tilt and rotate stage, and vibration isolation platform, inside a Binder MK56 thermal chamber. We connected a microcontroller to the sensor via I2C for a 1154 Hz sample rate. Orders of magnitude higher sample rates are possible using an off-the-shelf ADC and analog-out accelerometer. We used a Keithly 2450 source measure unit to power the sensor and measure the current drawn. We set the chamber temperature to 25 °C and allowed 30 minutes for the temperature of the sensor to equilibrate whilst continuously sampling the z-axis acceleration. We then sampled 10^5 values from the BMX055 sensor at a 3.6 V supply voltage. We repeated this for all the voltages in the range of 3.6 V to 1.4 V with a 0.2 V decrement. We then repeated this process for the temperature from 25 °C down to -5 °C with a decrement of 5 °C.

B. Quantization Investigation

We investigated the effect of quantization on the Kullback–Leibler (KL) divergence between a discrete distribution and its ideal fitted curve. We used the MATLAB `normrnd` function to generate 10^5 values from a Gaussian distribution with the same mean and standard deviation as the BMX055 z-axis at 2.6 V and 10 °C. We then discretized the values into a variety of numbers of bins, fitted a Gaussian distribution to them and calculated the KL divergence between the fitted distribution and the actual distribution.

IV. Results and Discussion

Let P and Q be discrete probability distributions, x a given sample value, and χ the sample space. In our calculations Q is always the reference fitted Gaussian. The KL divergence [26] between P and Q is:

$$D_{KL}(P||Q) = -\sum_{x\in\chi} P(x)\log\left(\frac{Q(x)}{P(x)}\right). \qquad (2)$$

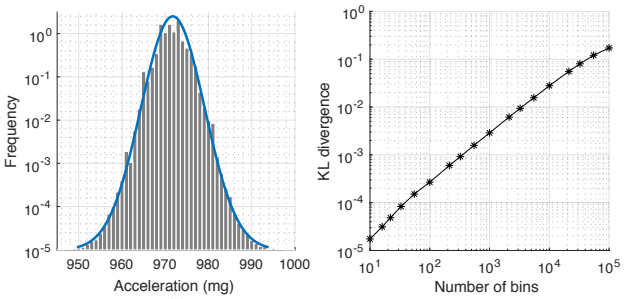

Fig. 3. **(a)** Distribution of BMX055 accelerometer z-axis noise and closest fitted distribution. The KL divergence or difference between the data and the fitted distribution is 0.00263 demonstrating that the accelerometer noise closely matches a Gaussian distribution. **(b)** The effect of increasing discretization on the KL divergence whilst keeping the total number of samples constant at 100,000. Increased quantization decreases KL divergence.

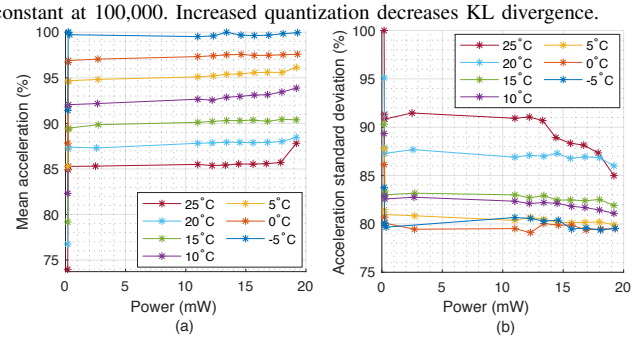

Fig. 4. **(a)** The mean of BMX055 z-axis distributions decreases with increasing temperature and increases with increasing supply power. We normalized the y-axis to the maximum observed value. The x-axis shows the power consumed by the BMX055 sensor alone. **(b)** The standard deviation of BMX055 z-axis distributions decreases with decreasing temperature and increasing supply power.

Figure 3(a) shows the BMX055 z-axis acceleration distribution at 2.6 V and 10 °C. We found that the KL divergence between the distribution from the BMX055 z-axis and its fitted Gaussian (0.00263) was more than an order of magnitude smaller than the equivalent result for a MATLAB-generated distribution (0.0392). This shows that the distribution of random variates would produce accurate results in applications such as particle filters where the distribution represents the state [27].

We rounded the MATLAB-generated floats for comparison with the BMX055-generated integers. The KL divergence between 10^5 samples of a MATLAB-generated uniform distribution with range $[\mu - 3\sigma, \mu + 3\sigma]$ and its fitted Gaussian is 0.116 for reference. We averaged the KL divergence calculations over 100 distributions to account for the random variations in the measurement. The random variates generated by the sensor are closer to an ideal Gaussian distribution than those generated by MATLAB. Figure 3(b) shows the effect of increasing the bin size on the divergence between a distribution of 10^5 values and its ideal fitted distribution. The results in the figure show that increased quantization decreases the difference between a distribution and its fitted Gaussian.

Figures 4(a) and 4(b) show how power dissipation and temperature affect the mean and standard deviation of the BMX055 z-axis acceleration measurement. Temperature has a greater effect on mean and standard deviation than voltage. Both voltage and temperature have a greater effect upon the standard deviation than the mean. A PRVA based on this phenomenon should therefore measure and compensate for both the temperature and the supply voltage of the sensor.

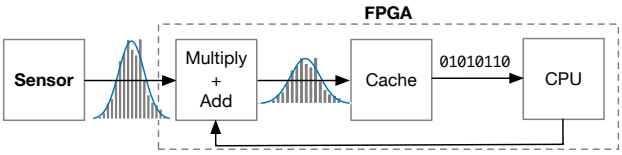

Fig. 5. Gaussian to Gaussian transform implementation. The CPU can request samples from distributions with certain parameters and the PRVA presents them to it in a cache.

V. Gaussian To Gaussian Transform

We can transform a univariate Gaussian to any other univariate Gaussian with one multiplication and one addition. This is eight fewer operations than the accept-reject method which requires at least ten operations per accept-reject test: an exponential, square, square root, subtraction, comparison and five divisions / multiplications. The accept-reject method may furthermore need to repeat this set of operations numerous times for each random variate if the generated sample is repeatedly rejected (by the condition on line 5 of Algorithm 2). The CPU that is running the application that requires random variates must generate two uniform random numbers as input for one accept-reject method sampling attempt.

We can achieve rapid addition and multiplication using fast adders and multipliers implemented in an FPGA, Figure 5 shows a block diagram of a possible implementation. The CPU requests a distribution by specifying parameters to the transform circuitry which transforms the input Gaussian to fit the requested output distribution. The FPGA transform circuitry measures and compensates for temperature and supply voltage. The transform circuitry achieves this by periodically taking temperature and voltage readings and adjusting the multiply and addition transform based on the empirically measured effects of each. The transform circuitry stores the values in a small high-speed cache that the CPU can read from. Offloading the transformation to dedicated hardware leaves the processor free to execute other instructions which will improve performance. We have not yet implemented this architecture in an FPGA but it will be the subject of future work.

VI. Conclusion

Sensors are a feasible entropy source for a PRVA with the potential to have a higher sample rate and greater efficiency than the state-of-the-art. The mean and standard deviation of the noise produced by the z-axis of a commercial accelerometer depend upon the temperature of the environment and the supply voltage. Quantizing a Gaussian distribution decreases the KL divergence between it and a fitted Gaussian. A PRVA can double the speed of Monte Carlo integration of a univariate Gaussian on a single thread. We can increase the PRVA sampling rate by substituting the BMX055 for a set of analog-out accelerometers (e.g., ADXL335s) and high speed ADCs (e.g., ADS54J60s). This method would require a customized version of the ADXL335 with the proof mass fixed in place, amplification so the noise distribution covers at least 90 % of the ADC range and no low pass filtering on its output.

VII. Acknowledgements

This research is supported by an Alan Turing Institute award TU/B/000096 under EPSRC grant EP/N510129/1, by EPSRC grant EP/R022534/1, by EPSRC grant EP/V004654/1, and by EPSRC grant EP/L015889/1.

References

[1] D. B. Thomas, L. Howes, and W. Luk, "A comparison of cpus, gpus, fpgas, and massively parallel processor arrays for random number generation," ser. FPGA'09. ACM, 2009, pp. 63–72.
[2] D. Thomas and W. Luk, "Efficient hardware generation of random variates with arbitrary distributions," ser. FCCM'06. IEEE, 2006, pp. 57–66.
[3] D. E. Eastlake, J. I. Schiller, and S. Crocker, "Randomness requirements for security," *RFC 4086*, vol. 4086, pp. 1–48, 2005.
[4] C. Hennebert, H. Hossayni, and C. Lauradoux, "Entropy harvesting from physical sensors," ser. WiSec'13. ACM, 2013, pp. 149–154.
[5] K. Wallace *et al.*, "Toward sensor-based random number generation for mobile and iot devices," *IEEE IoT*, vol. 3, no. 6, pp. 1189–1201, 2016.
[6] S.-M. Cho, E. Hong, and S.-H. Seo, "Random number generator using sensors for drone," *IEEE Access*, vol. 8, pp. 30 343–30 354, 2020.
[7] K. Lee and M. Lee, "True random number generator (trng) utilizing fm radio signals for mobile and embedded devices in multi-access edge computing," *Sensors*, vol. 19, no. 19, p. 4130, 2019.
[8] B. Perach and S. Kvatinsky, "An asynchronous and low-power true random number generator using stt-mtj," *IEEE VLSI*, vol. 27, no. 11, pp. 2473–2484, 2019.
[9] S. Srinivasan *et al.*, "2.4 ghz 7mw all-digital pvt-variation tolerant true random number generator in 45nm cmos," *IEEE VLSI*, pp. 203–204, 2010.
[10] Y. Hu *et al.*, "Gaussian random number generator: Implemented in fpga for quantum key distribution," *Int. J. Numer. Model. El.*, vol. 32, no. 3, p. 2554, 2019.
[11] H. Jiang *et al.*, "A novel true random number generator based on a stochastic diffusive memristor," *Nat. Commun.*, vol. 8, no. 1, pp. 1–9, 2017.
[12] D. G. Marangon, G. Vallone, and P. Villoresi, "Source-device-independent ultrafast quantum random number generation," *Phys. Rev. Lett.*, vol. 118, no. 6, pp. 060 503–1–060 503–5, 2017.
[13] X. Zhang *et al.*, "Architecting a stochastic computing unit with molecular optical devices," ser. ISCA'18. ACM/IEEE, 2018, pp. 301–314.
[14] M. Avesani *et al.*, "Source-device-independent heterodyne-based quantum random number generator at 17 gbps," *Nat. Commun.*, vol. 9, no. 1, pp. 1–7, 2018.
[15] L. Nguyen *et al.*, "Programmable quantum random number generator without postprocessing," *Opt. Lett.*, vol. 43, no. 4, pp. 631–634, 2018.
[16] A. Tomasi *et al.*, "Model, validation, and characterization of a robust quantum random number generator based on photon arrival time comparison," *JLT*, vol. 36, no. 18, pp. 3843–3854, 2018.
[17] X. Guo *et al.*, "Parallel real-time quantum random number generator," *Opt. Lett.*, vol. 44, no. 22, pp. 5566–5569, 2019.
[18] L. Devroye, *Non-Uniform Random Variate Generation*. Springer-Verlag, 1986, ISBN: 1461386454.
[19] D. B. Thomas and W. Luk, "Resource efficient generators for the floating-point uniform and exponential distributions," ser. ASAP'08. IEEE, 2008, pp. 102–107.
[20] J. R. van Peborgh Gooch and M. J. Hounslow, "Monte carlo simulation of size-enlargement mechanisms in crystallization," *AIChE Journal*, vol. 42, no. 7, pp. 1864–1874, 1996.
[21] R. L. Cook, "Stochastic sampling in computer graphics," *ACM Trans. Graph.*, vol. 5, no. 1, pp. 51–72, 1986.
[22] G. L. Zhang *et al.*, "Reconfigurable acceleration for monte carlo based financial simulation," ser. FPT'05. IEEE, 2005, pp. 215–222.
[23] B. Lambert, *A Student's Guide to Bayesian Statistics*. SAGE, 2018, ISBN: 1473916364.
[24] J. M. Pak *et al.*, "Improving reliability of particle filter-based localization in wireless sensor networks via hybrid particle/fir filtering," *IEEE Trans. Ind. Electron.*, vol. 11, no. 5, pp. 1089–1098, 2015.
[25] P. Stanley-Marbell and M. Rinard, "Warp: A hardware platform for efficient multimodal sensing with adaptive approximation," *IEEE Micro*, vol. 40, no. 1, pp. 57–66, 2020.
[26] S. Kullback and R. A. Leibler, "On information and sufficiency," *Ann. Math. Statist.*, vol. 22, no. 1, pp. 79–86, 03 1951.
[27] A. Doucet, N. De Freitas, and N. Gordon, *Sequential Monte Carlo methods in practice*. Springer, 2001, ch. 1, pp. 3–14.