Real-time Room Occupancy Estimation with Bayesian Machine Learning using a Single PIR Sensor and Microcontroller

Charles Leech
ARM, Cambridge
University of Southampton, UK

Yordan P. Raykov
NCRG, Aston University, UK

Emre Ozer
ARM
Cambridge, UK

Geoff V. Merrett
University of Southampton, UK

***Abstract*—This paper presents the implementation and deployment of a compute/memory intensive non-parametric Bayesian machine learning algorithm on a microcontroller unit (MCU) to estimate room occupancy in a Smart Room using a single analogue PIR sensor. We envisage an IoT device consisting of a resource-constrained MCU, PIR sensor and a battery running the occupancy estimation algorithm and operating over days or months without recharging or replacing the battery. Both hardware-independent and hardware-dependent optimizations are performed to reduce memory footprint and yet provide acceptable real-time performance while consuming less energy. We show a significant reduction in the on-chip memory usage in the MCUs by the algorithm through optimisation of the machine learning models and of the static memory footprint and dynamic memory usage. We also show that a low-end MCU does not meet the real-time requirements of the application without causing high average power consumption. However, a moderately high-performance MCU with a higher clock frequency and hardware floating-point unit provides 19x improvement in the execution time of the algorithm, better meeting the real-time specification of the application and reducing power consumption. Further, we estimate the battery lifetime of the IoT device if it operates continuously in a Smart Room. With a typical size battery, an IoT device consisting of a Cortex-M4F MCU and PIR sensor can operate for more than a month without replacement or recharging of the battery while running the compute-intensive Bayesian machine learning algorithm.**

I. Introduction

Smart Building and Smart Workplace applications are becoming increasingly of interest as the focus of technological innovation shifts from mobile to IoT devices. For these systems to be effective they must be energy efficient, low cost and non-invasive, all whilst providing useful information. We present the implementation and the deployment analysis of a novel sensing system capable of occupancy estimation using only a single passive infrared (PIR) sensor.

This novel system was first introduced in [1] where we discussed in more detail the evaluation, the training and the statistical modeling involved in the problem of occupancy estimation with a single PIR. However, as [1] points out the novel system relies on sophisticated probabilistic modeling and therefore deployment on energy constrained IoT hardware can be a great challenge. Towards that end, we extend [1] and here we develop a framework of both hardware-dependent and algorithmic optimization steps to efficiently utilize the resources available on a typical microcontroller (MCU). The undertaken optimization steps hardly affect the accuracy of the original system from [1] and we can estimate occupancy within one individual error bar with more than 80% accuracy.

The optimized model is deployed on two MCUs to estimate room occupancy natively on the devices in real time and we measure its performance, memory usage, energy consumption and battery life. To enable real-time estimations we build upon fast inference methods from [2] to derive online method for fast inference in probabilistic models. Furthermore, we demonstrate that the use of a floating point unit and hardware peripherals on the MCU leads to an almost $10\times$ reduction in execution time of the algorithm. We also show that the memory optimization that we perform enable the system to operate within 12 KB of SRAM without impacting speed.

The paper is organized as follows: Section II briefly discusses the related work. Section III provides a brief overview of the iHMM model developed in our early work. Section IV describes the hardware-independent iHMM model resource optimizations. Section V describes the algorithm implementation and porting. Section VI presents the experimental setup, and Section VII describes the hardware-dependent optimizations. Section VIII presents the memory usage, performance, power consumption and battery lifetime estimation results, and finally Section IX concludes the paper.

II. Related Work

Occupancy estimation is one of the five main tasks in human sensing and is an essential one to consider when building self-aware environments and smart building management systems [3]. There are methods that rely on data inference from cameras coupled with image processing algorithms ([4], [5], [6] and [7]), methods using multiple motion sensors at all entry and exit locations of a closed environments ([8], [9], [10], [11], [12] and [13]) and methods relying on historical patterns of movement across the environment based on data from motion or environmental sensor networks ([14], [15], [16]). PIR sensors have been dominantly used in the second of those categories. For example ([10] and [11]) showed how by placing three PIR sensors in a hallway, we can identify direction of movement and relative location of passing individuals. [12] used PIR sensors in combination with reed

978-1-5090-3202-0/17/$31.00 ©2017 IEEE

switch door sensors placed at each doorway and [13] presented a similar approach but using only PIR sensors at all entries and exits.

In contrast to these approaches, [1] proposed using a a single analogue PIR sensor as a monitoring device rather than simply counting entries or exits. The analogue PIR output is segmented using a flexible probability model and the patterns of motion which are the best descriptor of actual occupancy are identified. In effect, we have traded the complexity of image data with simpler data coupled with more complex and flexible modelling. In this way, simple single dimensional time series data generated from the PIR sensor can be used to learn some motion behaviours of interest from the data. This can be used for accurate occupancy estimation on its own, hence why [1] is able to estimate occupancy using a single sensor. However, while such an approach can reduce the cost and also the invasiveness of the system, it faces the challenge of learning a flexible Bayesian nonparametric model online, which is non-trivial on a resource-constrained MCU. To address this problem, we present several hardware-independent model optimizations in Section IV.

III. Overview of Occupancy Estimation Algorithm

In this section we present a high level overview of the pre- and post-deployment stages of our algorithm for occupancy estimation. The first stage is performed before deployment to train the model and is described in more detail in [1]. We have collected PIR data for approximately 50 hours from various office meetings. We segment the raw analogue PIR data using an infinite hidden Markov model (iHMM) [17] with Laplace components and separate the data that is most descriptive of the room occupancy. This filtered data is then used to estimate a Laplace diversity parameter which is good indicator of the levels of motion and occupancy. A regression model is fitted to the different diversity parameters for each time window of 30 seconds and the regression is used to predict occupancy.

After training the iHMM and inferring the regression parameters, we implement a modified prediction process on the MCU. The trained iHMM parameters are condensed and transfered to the MCU. On the MCU a modified online learning version of the MAP-iHMM [2] is used to fit the iHMM to incoming streams of unseen PIR data, where the method also incorporates the effect of the previously trained parameters. Once we segment the new data, we update the training iHMM parameters to incorporate the effect of the last seen data. The regression parameters are robust so they rarely need to be re-calculated.

IV. Hardware-independent Model Resource Optimizations

The iHMM used in [1] is a powerful probabilistic model that can capture arbitrarily complex time dependent patterns in entirely unsupervised way. Applications of models such as the iHMM has been limited thanks to the computationally expensive inference methods they typically require: exhaustive

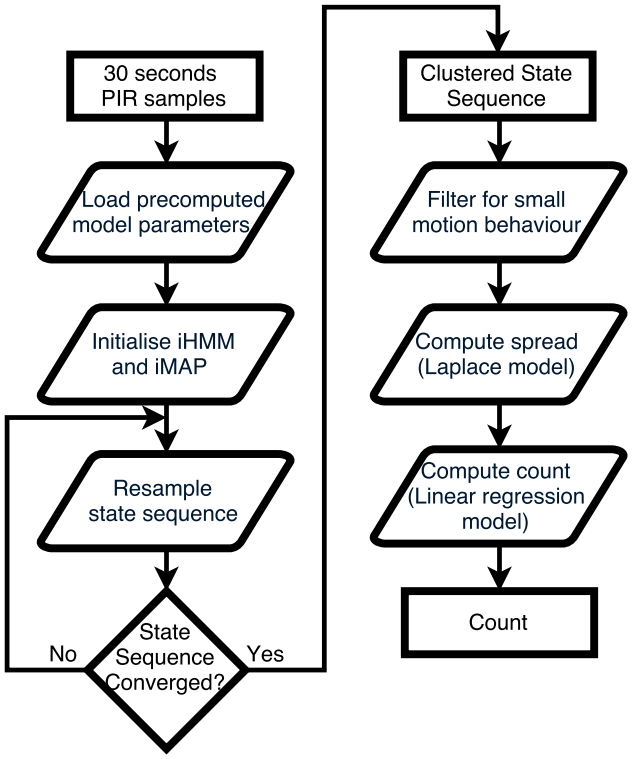

Fig. 1. Block diagram of the stages of the room occupancy estimation algorithm performed during deployment on the MCU.

Markov Chain Monte Carlo (MCMC) sampling. MCMC methods are probabilistic therefore aim to find the complete likelihood distribution of the fitted model. This involves storing orders of magnitude more parameters than often needed. For example, fitting an iHMM on 5 minutes of PIR output using MCMC would require storing approximately $N * K * I + 2K$ parameters, where: N is large (11500 for 5 minutes of data) as is the number of data points; K is the number of states found in the time series (usually $3-6$); I is the length of the chain (can vary between $120-1000$ iterations). Approximate Variational Bayes inference methods have been proposed ([18], [19]) which improve mixing drastically, but stochastic Variational methods [19] require knowledge of the size of the data to be processed a priori and storing data dependent number of variational distributions. [18] reduces the memory overhead when used on a batch of data, but in streaming applications suffers from the same problems as [19].

To fit the iHMM to streams of PIR data we build upon a recent learning algorithm called MAP-iHMM from [2], [20]. Iterative MAP methods are convenient as they converge orders of magnitude faster than MCMC methods and return only the most likely segmentation of the data rather than a full posterior distribution; this makes them quite memory efficient. Using the example from above for fitting an iHMM on 5 minutes of PIR output data, this time using iterative MAP, the number of parameters to store would be approximately $N + 2 * K$. However if the system is deployed, as we monitor more and

more PIR data, N becomes too large to store. Therefore, we derive from the MAP-iHMM an online streaming algorithm that processes batches of sensor data as it is sampled. Once converged, the method only updates the small number of parameters and discards the raw data before receiving the next batch. One challenge is that [20] and a lot of the efficient inference algorithms for Bayesian nonparametric models use a collapsed (Rao-Blackwellized) representation of the iHMM for faster and more robust convergence (standard practice in Bayesian modelling). This representation introduces dependencies between the model parameters requiring us to keep some explicit parameters for each sensor output point that influences the segmentation. To overcome this issue, after processing each window of data, we recover an approximation of the complete representation of the iHMM with an explicit distribution available for the whole parameter space. For example, the memory footprint of the collapsed representation of the model after training on 50 hours of data is approximately 228 MB, compared to few hundred KB when recovering a non-collapsed model representation.

This is possible because future data is independent of historic data given the complete model representation, while this is not the case for the collapsed representation. Therefore the complete trained model rigorously and unbiasedly can be used as a starting point for the clustering on the next window of sensor data. The trained form of the model involves updating only $(K+1)*(K+2)+2$ parameters after processing each time window, which is sufficiently more compact. Note that those few parameters still incorporate the knowledge gained by observing and segmenting many hours of PIR data and so the model is still capable of segmenting behaviours of interest despite the reduced memory footprint. This will allow the model to dynamically update the model parameters after its deployment to the microcontroller unit (MCU).

V. Algorithm Implementation and Porting

Initially, a series of Matlab functions characterise the iHMM and iMAP processes, such as the clustering of data samples and the Bayesian resampling of hyperparameters. A top-level encapsulation function connects these ML functions with the Laplace model and regression parameters to apply the algorithm to occupancy estimation.

Matlab was used to experiment with the theoretical concepts of the algorithm without exposing hardware constraints. However, many microprocessor compilers cannot interpret such high-level languages and therefore the Matlab code must be translated to C and C++ such that it can be compiled into a binary and executed directly by the MCU.

Porting can be achieved through manual programming or automatic compiler-style translation using Matlab Coder [21]. Manual programming is the simpler option however it does not scale well with increasing code size and requires detailed knowledge of the syntax and constructs of both languages. Automatic translation avoids both of these barriers, however in the case of Matlab Coder, extensive code preparation is required involving the addition of common syntactic elements from both languages in order to guide the translation tool. This negates much of the benefit that would be afforded by the process. Furthermore, due to the algorithmic optimisations described in the previous section, the algorithm occupies a relatively small code-base and as a result the porting of each function was realistic through manual programming.

Whenever possible, Matlab standard toolbox functions have been replaced by equivalents from C and C++ standard libraries, with functional testing performed to ensure that the algorithm results remain the same for the same input dataset. Moreover, there are no dependencies on libraries introduced with the C++11 standard to improve the portability of the code as support for newer C++ standards varies across toolchains and development environments. In the case where no standard library function exists as the replacement for a Matlab function, a custom function was developed from a theoretical basis or by combining other standard functions. Only one case for this was required in the algorithm in the generation of the gamma random distribution when re-sampling the values of Beta from the iHMM. Here the Ziggurat method is used, developed by Marsaglia and Tsang [22], based on the cube of scaled normal variates from a normal distribution, which is a standard library function.

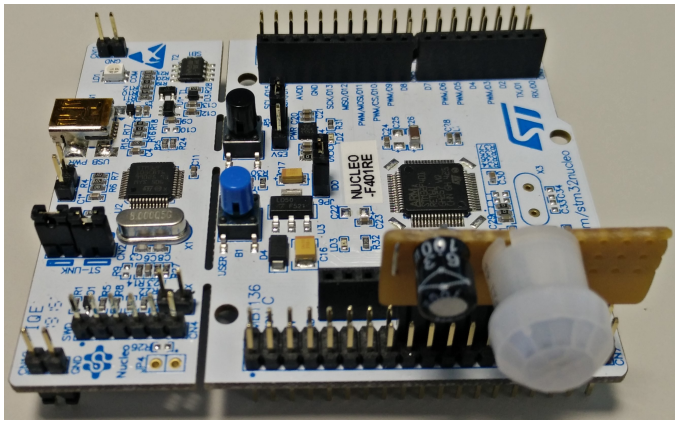

Fig. 2. Photo of the experimental setup consisting of the ST Nucleo-401RE MCU board and PIR sensor connected to the ADC through the Arduino header.

VI. Experimental Setup

We begin our experimentation with the ARM Cortex-M0 based ST Nucleo-F070RB, a highly resource-constrained MCU, to test the memory and computational boundaries of the algorithm. The specification of the MCU is shown in Table I where we highlight the critical features, including clock frequency, memory capacity and the peripherals that we will utilise. A Panasonic NaPiOn AMN21111 PIR sensor is connected to one of the ADC inputs on the Arduino header of the MCU, as shown in Figure 2. The PIR sensor continuously produces a single dimensional analogue signal which is sampled at 50 Hz by the 12-bit ADC, this produces 1500 samples per 30 second recording interval, which the algorithm then processes. Samples are recorded as integers

TABLE I
SPECIFICATIONS OF THE CORTEX-M0 AND M4 MCUS USED FOR EXPERIMENTATION. MEMORY NUMBERS SHOW SRAM SIZE WITH FLASH SIZE IN SQUARE BRACKETS.

Board name	CPU	Clock Modes (MHz)	Memory (KB)	Peripherals Used	Technology Node (nm) [23]	Operating Voltage (V)
Nucleo-F070RB	Cortex-M0	8, 24, 48	16 [128]	ADC, DMA, TIM3	180	2.4
Nucleo-F401RE	Cortex-M4F	84	96 [512]	ADC1, DMA2, TIM2	90	1.7

but converted to floating point values in the range 0.0 to 1.0 before processing.

We make extensive use of the hardware peripherals of the MCU, through the STM hardware abstraction library (HAL). A timer with a 50Hz period triggers conversion events in the ADC via an interrupt. We use DMA to transfer sampled data from the ADC to SRAM without the involvement of the CPU, allowing the CPU to sleep or perform other tasks during sampling periods.

The completion of sampling is signalled by a DMA transfer complete interrupt when the data buffer in SRAM is filled. A DMA interrupt service routine (ISR) links the interrupt to an ADC conversion complete callback function from where the count estimation algorithm is called. The configuration and interaction of the HAL components is illustrated in Figure 3.

The board is connected to a laptop via a USB cable solely for power supply and programming reasons. All data collection and processing for the algorithm is performed locally on the MCU. Furthermore, the board can be configured to receive power from a battery source connected to it and, when deployed in a Smart Room context, the program will be automatically loaded from the flash memory on start-up, removing all need for an external connection.

An LCD display is mounted to the board via the Arduino header, allowing the occupancy count estimation from the algorithm to be displayed on the device. During development, debugging information was communicated back to a PC via the USB cable and displayed via a serial terminal.

Despite the particular choice of MCU, in principle the C/C++ code for the algorithm can be compiled and executed on any microprocessor with a similar specification.

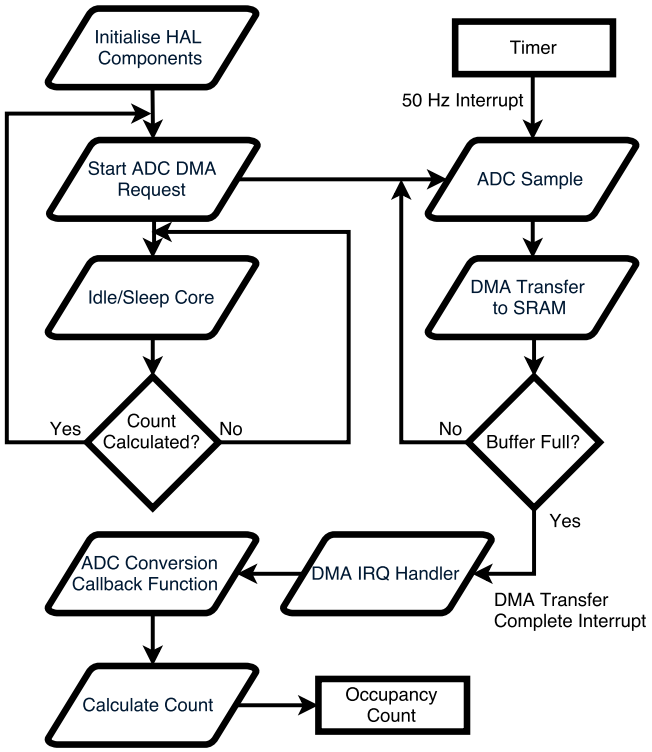

Fig. 3. The configuration of HAL components on the MCU

VII. HARDWARE-DEPENDENT OPTIMIZATIONS

The occupancy estimation algorithm is required to have low computational complexity and memory requirements to allow real-time processing and deployment on the MCU.

Strategies to manage the memory requirements of the algorithm from a theoretical perspective are described in detail in Section IV. These primarily consist of the pre-deployment construction of a non-collapsed iHMM representation to reduce the memory footprint of the algorithm from the order of MB to KB and the use of an iterative MAP inference method due to the fact that it is an order of magnitude faster than MCMC methods when fitting the iHMM to streams of PIR data, also greatly reducing the expected computation time.

In deployment of the algorithm, additional optimisations are made to further reduce the memory and computational requirements from an implementation perspective. Smaller data type sizes were used throughout the algorithm to reduce the size of memory required to hold intermediate variables. The double data type offers the greatest precision to represent real numbers but requires twice as many bytes for storage. As a result, all floating-point numbers are represented in single-precision, with negligible loss in accuracy, reducing the size of stored data from 12 to 6 KB for 1500 samples. In addition, for integers that are known to be within the range -128 to $+127$, their type is reduced to characters, reducing memory allocation by a factor of four with no effect on data values. This has the greatest affect in the storage of the cluster assignments for the hidden state sequence; from 6 to 1.5 KB.

Memory tracing was used to more accurately analyse the memory behaviour of the algorithm throughout its execution. Tracing was performed using an mbed OS API whereby calls to standard C memory management functions were wrapped and intercepted to identify when (de-)allocations were made. In addition, heap statistics were recorded online to monitor the maximum heap allocation reached in the algorithm, determining the minimum SRAM size required by the MCU. Memory reduction techniques have been used throughout the algorithm

such as lowering the scope of intermediate variables to discard temporary data and pruning data structures to pass only the minimum amount of data between functions. Updates are made directly to the hidden state sequence for each iteration so that only the final clustered states are returned by the model.

Sampling of the PIR sensor and execution of the algorithm is performed simultaneously through additional configuration of the hardware peripherals on the MCU, as outlined in the previous section, to enable the transfer of sensor data from the ADC to SRAM using the DMA module. This allows us to overlay each data processing operation with the beginning of data collection for the following estimation period, meaning that a count estimate can be obtained at regular intervals and still be based on a full 30 second sampling period. In addition, after sampling, the CPU can enter a lower power mode if has no other tasks to perform, whilst waiting for the next sampling period to complete.

VIII. Experimental Results

A. Memory Usage and Real-time Performance

To evaluate the performance of the algorithm on the Cortex-M0 MCU, we record new PIR sensor data and apply the algorithm to 30 second segments. This directly emulates the operation expected when the MCU is placed in a meeting environment. The accuracy of the translated version of the algorithm has been verified against the original Matlab program by testing with the same data sets collected from meetings of known occupancy. The percentage of time windows where the predicted number of occupants was within ± 1 and ± 2 matches those presented in the evaluation of [1]. There was no impact on the estimation accuracy of the algorithm from losses in precision due to the conversion of real numbers from double to single precision. This rounding rarely causes cluster assignments to change and the regression parameters are robust after training. All other computation and memory optimizations do not impact data values.

To analyse memory consumption, we run the program with memory tracing and heap statistic recording enabled. The peak heap allocation was recorded as 0.7 KB. To find the total memory consumption we must include statically allocated global data, including the memory blocks outlined in the previous section, which accounts for 9.63 KB in our program, giving a combined total memory consumption of 10.33 KB. This is below the 16 KB available on the Cortex-M0 MCU.

To evaluate computational demand, we use execution time as a metric, and test how long the Cortex-M0 based MCU takes to calculate the estimated occupancy count, with the algorithmic optimisations described, across the three frequency modes in table I. We measure execution time as approximately 22 seconds. The main reason for low performance is because the iHMM model uses a significant number of floating-point operations, which are emulated by software since the Cortex-M0 CPU does not have a hardware floating-point unit (FPU), and also the clock frequency of the Cortex-M0 is comparatively low, i.e. 48 MHz.

TABLE II
Results for computation time, memory consumption and current consumption of the algorithm on the Cortex-M0 and M4 based MCUs. Memory consumption is divided into SRAM and Flash, with the later shown in square brackets.

ARM Platform	Execution Time (s)	Memory Requirement (KB)
Cortex-M0	22	10.33 [11.70]
Cortex-M4	9.55	10.36 [12.03]
Cortex-M4 + FPU	1.15	10.36 [11.24]

In order to improve the execution time, we experiment with a higher performance Cortex-M4 based MCU. The Cortex-M4 CPU has a single-precision FPU that can perform floating-point calculations in hardware, dramatically increasing the speed at which our algorithm executes. This speed-up is compounded by the higher clock frequency of the Cortex-M4 (1.75x faster than Cortex-M0) which accelerates all instructions. The execution time is reduced from 22 s to 1.15 s. We disable the FPU in the Cortex-M4 to evaluate how the FPU improves execution time. We recompile the code, with the floating-point instructions being emulated, and observe that the algorithm executes in 9.55 s, which reflects the increase in clock frequency. The memory usage and execution time results are summarized in Table II. The same hardware peripherals (ADC, DMA and timer) are used in both Cortex-M0 and Cortex-M4 experiments, and the code for the algorithm remains unchanged.

B. Power Consumption and Battery Lifetime

We estimate the power consumption and battery lifetime of the platform (MCU and PIR sensor) based on current consumption and operating voltage numbers provided in the MCU and PIR datasheets and using the STM32CubeMX tool by STMicroelectronics [24]. While data is being sampled, the CPU is put into sleep mode which clock-gates the CPU and reduces the current consumption. For example, it is 9.58 mA for the Cortex-M4F MCU at 84 MHz and 7.53 mA for the Cortex-M0 MCU at 48 MHz. When data sampling is complete, the CPU is returned to run mode and, together with the continued use of the peripheral components, the current consumption of the Cortex-M4F and M0 MCUs rise to 17.18 mA and 11.50 mA, respectively. With operating voltages of 2.4 and 1.7 V, the average power consumption of the Cortex-M4F and Cortex-M0 platforms are 46.36 and 18.50 mW, respectively.

Finally, we estimate the battery lifetime of the platform if it were battery-powered and deployed in a meeting room environment to run continuously. Battery lifetimes are calculated from the average power consumption over the sampling and execution periods. We use a Lithium Polymer (LiPo) battery with a 2200 mAh capacity. The estimated lifetimes of the platform are shown in Figure 4, for the Cortex-M0 operating at three frequencies and the Cortex-M4 with its FPU enabled and disabled. Battery lifetime and execution time decrease for the Cortex-M0 platform as frequency increases. The Cortex-M4 MCU increases battery lifetime by 2.51x due to the

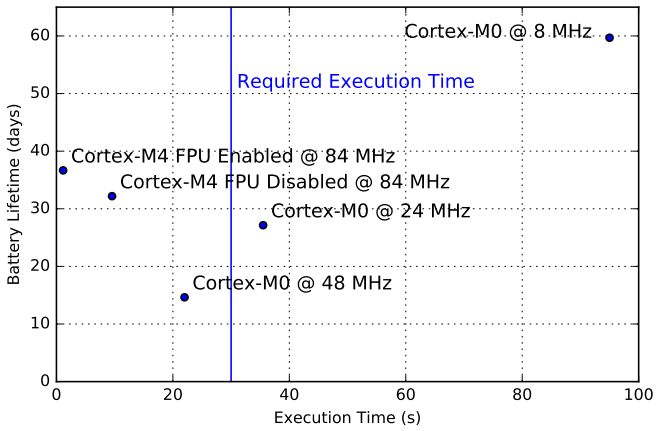

Fig. 4. Battery lifetime and execution time over a range of frequencies on the Cortex-M0 and for the FPU enabled and disabled on the Cortex-M4F. The vertical line shows the 30 second algorithm execution time requirement.

lower average power consumption whilst further reducing the execution time. The reduction in power consumption is due to the lower technology node of the Cortex-M4F MCU, which reduces dynamic current consumption, and a lower operating voltage.

IX. Conclusions

In this paper, we have investigated the deployment of a non-parametric Bayesian machine learning algorithm on a resource-constrained MCU for a room occupancy estimation application using a single analogue PIR sensor. Optimisations in several dimensions were performed to accommodate for reduced memory and computational capacity to produce an energy efficient and non-invasive solution with reduced hardware complexity. We have demonstrated a three order reduction in the memory requirement of the algorithm through hardware-independent optimisation of the machine learning models and an 8x reduction in memory utilisation on two MCUs, through analysis of the static memory footprint and dynamic memory behaviour of the implemented algorithm. We have shown that a low-end MCU did not meet the real-time service requirements for the application but a moderately high-performance MCU with a higher clock frequency and hardware floating-point unit provided 19.13x improvement in the execution time of the algorithm, meeting the real-time specification of the application. Further, we have estimated the power consumption of the IoT platform and battery lifetime if it operates indefinitely in a Smart Room application. With a typical battery size, the IoT platform consisting of a Cortex-M4F MCU and PIR sensor can operate for over 36 days without replacement or recharging the battery while running a compute-intensive self-learning algorithm.

References

[1] Y. P.Raykov *et al.*, "Predicting Room Occupancy with a Single Passive Infrared (PIR) Sensor Through Behavior Extraction," in *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, ser. UbiComp '16. New York, NY, USA: ACM, 2016, pp. 1016–1027.

[2] Y. P.Raykov, A.Boukouvalas, and M. A.Little, "Simple approximate MAP inference for Dirichlet processes mixtures," *Electron. J. Statist.*, vol. 10, no. 2, pp. 3548–3578, 2016.

[3] T.Teixeira, G.Dublon, and A.Savvides, "A survey of human-sensing: Methods for detecting presence, count, location, track," and Identity. Technical report, ENALAB, Yale University, Tech. Rep., 2010.

[4] V.Lempitsky and A.Zisserman, "Learning to count objects in images," in *Advances in Neural Information Processing Systems*, 2010, pp. 1324–1332.

[5] T.Van Oosterhout, S.Bakkes, and B. J.Kröse, "Head Detection in Stereo Data for People Counting and Segmentation." in *VISAPP*, 2011, pp. 620–625.

[6] A. B.Chan and N.Vasconcelos, "Counting People with Low-level Features and Bayesian Regression," *IEEE Transactions on Image Processing*, vol. 21, no. 4, pp. 2160–2177, 2012.

[7] D. B.Yang, H. H.González-Baños, and L. J.Guibas, "Counting people in crowds with a real-time network of simple image sensors," in *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*. IEEE, 2003, pp. 122–129.

[8] K.Hashimoto *et al.*, "People count system using multi-sensing application," in *Solid State Sensors and Actuators, 1997. TRANSDUCERS'97 Chicago., 1997 International Conference on*, vol. 2. IEEE, 1997, pp. 1291–1294.

[9] P.Zappi, E.Farella, and L.Benini, "Enhancing the spatial resolution of presence detection in a PIR based wireless surveillance network," in *Advanced Video and Signal Based Surveillance, 2007. AVSS 2007. IEEE Conference on*. IEEE, 2007, pp. 295–300.

[10] J.Yun and S.-S.Lee, "Human movement detection and identification using pyroelectric infrared sensors," *Sensors*, vol. 14, no. 5, pp. 8057–8081, 2014.

[11] P.Zappi, E.Farella, and L.Benini, "Tracking motion direction and distance with pyroelectric IR sensors," *IEEE Sensors Journal*, vol. 10, no. 9, pp. 1486–1494, 2010.

[12] Y.Agarwal *et al.*, "Occupancy-driven Energy Management for Smart Building Automation," in *Proceedings of the 2nd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building*. ACM, 2010, pp. 1–6.

[13] F.Wahl, M.Milenkovic, and O.Amft, "A distributed PIR-based approach for estimating people count in office environments," in *Computational Science and Engineering (CSE), 2012 IEEE 15th International Conference on*. IEEE, 2012, pp. 640–647.

[14] K. P.Lam *et al.*, "Occupancy detection through an extensive environmental sensor network in an open-plan office building," *IBPSA Building Simulation*, vol. 145, pp. 1452–1459, 2009.

[15] R. H.Dodier, G. P.Henze, D. K.Tiller, and X.Guo, "Building Occupancy Detection Through Sensor Belief Networks," *Energy and buildings*, vol. 38, no. 9, pp. 1033–1043, 2006.

[16] A.Khan *et al.*, "Occupancy Monitoring using Environmental & Context Sensors and a Hierarchical Analysis Framework." in *BuildSys@ SenSys*, 2014, pp. 90–99.

[17] M. J.Beal, Z.Ghahramani, and C. E.Rasmussen, "The Infinite Hidden Markov Model," in *Advances in neural information processing systems*, 2001, pp. 577–584.

[18] M. C.Hughes, W. T.Stephenson, and E.Sudderth, "Scalable Adaptation of State Complexity for Nonparametric Hidden Markov Models," in *Advances in Neural Information Processing Systems*, 2015, pp. 1198–1206.

[19] N.Foti, J.Xu, D.Laird, and E.Fox, "Stochastic variational inference for hidden Markov models," in *Advances in Neural Information Processing Systems*, 2014, pp. 3599–3607.

[20] Y. P.Raykov, A.Boukouvalas, and M. A.Little, "Iterative collapsed MAP inference for Bayesian nonparametrics," *NIPS 2015 Workshop on Bayesian Nonparametrics: The Next Generation*, 2015.

[21] T. M.Inc. Matlab coder - generate c and c++ code from matlab code. Online. The MathWorks Inc. Natick, Massachusetts.

[22] G.Marsaglia and W. W.Tsang, "A Simple Method for Generating Gamma Variables," *ACM Trans. Math. Softw.*, vol. 26, no. 3, pp. 363–372, Sep. 2000.

[23] *AN4435 Application note*, STMicroelectronics, March 2016, rev 3.

[24] Stm32cubemx - stm32cube initialization code generator. Online. STMicroelectronics. V4.18.0.