Distributed Prognostics Using Wireless Embedded Devices

Sankalita Saha, *Member, IEEE*, Bhaskar Saha, *Member, IEEE*, and Kai Goebel

Abstract—Distributed prognostics is the next step in the evolution of prognostic methodologies. It is an important enabling technology for the emerging Condition Based Management/Prognostics Health Management paradigm. This paper provides an overview of such systems with details of the system architectures and the various possible design considerations. A distributed particle filter architecture for battery health management is developed and successfully implemented using embedded smart sensor devices with wireless communication capabilities.

***Index Terms*—**Prognostics; Distributed Architecture; Sensor Network; PHM; Distributed Prognostics; Distributed Health Management; Distributed PHM.

I. Introduction

Health management is becoming more and more an enabling technology in the aerospace domain. Given a complex system, multiple sensors monitor various subsystems. The measurements taken from these sensors are then processed by suitable algorithms to determine the health of the system. Most of the system development assumes a centralized health management architecture, i.e., a central computing machine collects all the sensor data, processes them, and then runs various diagnostic and prognostic algorithms.

However, such a system architecture has several disadvantages: (a) increasingly large amounts of sensor data are being collected for more refined analysis (e.g., high frequency vibration data for structures health management or data with high sampling rate for avionics health management), requiring significant wiring for the transmission of such data; (b) these data need to be processed by increasingly more complex algorithms. A single processor with limited memory resources may not be sufficient to deal with that complexity; (c) the system becomes more vulnerable to loss of functionality if the single processor system crashes. The whole health management system might go down and it would require considerable amount of time and effort to restore the system back. In some instances such a recovery may not be possible at all.

In our vision the health management architecture combines robust functionality with low added weight. This can be realized through a *distributed* health management architecture, where the computation is no longer completely executed on a single computing machine. Multiple such smart sensor devices would monitor different parts of a system while running individual monitoring algorithms. Where heavyweight algorithms are needed, multiple nodes in the architecture collaborate to provide the answer. This can be realized by taking advantage of advances in smart sensor technology that combine the power of embedded computing devices with sensors and wireless transmission technology.

An example of such an implementation is where each embedded device would perform its local diagnostics which is assumed to be a light-weight operation (although light-weight diagnostics is not a requirement). When a fault condition has been detected, and when more computation power is required to perform remaining life calculations, the task is distributed among some of the remaining devices. Such an architecture would support an efficient as well as robust health management system. We will in the following sections continue with this illustrative example to demonstrate the operation of the distributed architecture. Results are shown for the functionality of the distributed architecture for prognostics in the context of battery health management. The algorithms were implemented on Sun Microsystem's SPOT devices.

II. Background

Although many techniques have been investigated for prognostics, the field itself is still relatively immature. Distributed prognostics is a topic that has been even less explored. Nonetheless, some of the techniques used in prognostics – such as particle filters – have been investigated in the context of distributed implementations. For example, in [2] the authors present three different distributed methods for implementing particle filter system while in [13], the authors present a parallel particle filter implementation on a shared-memory multiprocessor cluster. Distributed particle filter implementation for embedded systems have been presented in [11]. Of late, sensor networks have gained popularity and often, sensor networks employ particle filters for tracking objects. Distributed particle filters for such applications have also been explored ([3], [8], [14]).

A few efforts have been made recently in the domain of implementing a prognostics system in a distributed fashion.

Sankalita Saha is with RIACS/NASA Ames Research Center, Moffett Field, CA 94035 USA (phone: 650-604-4593; e-mail: ssaha@riacs.edu).

Bhaskar Saha is with MCT/NASA Ames Research Center, Moffett Field, CA 94035 USA. (e-mail: bhaskar.saha-1@nasa.gov).

Kai Goebel is with NASA Ames Research Center, Moffett Field, CA 94035 USA. (e-mail: kai.goebel@nasa.gov).

For example, a concept similar to our work has been outlined in [9], where the authors briefly outline a distributed prognostics system architecture. However, the distributed architecture in this case refers only to the distribution of tasks at the prognostics algorithm level, i.e., identifying the different system modules and where they fit into a given system using prognostics. In [5], the authors present a high-level CAD (Computer Aided Design) tool for designing distributed prognostics system employing varied computing platforms such as FPGAs as well as embedded processors like ARM. The design capabilities were demonstrated using small examples: a PID tank controller and an XOR gate.

III. Distributed Prognostics

A. Overview

We envision a distributed prognostics system where multiple smart sensor devices are employed that monitor various subsystems or modules and perform diagnostics operations and trigger the prognostics mode based on user defined thresholds and rules in which case the prognostics for the whole system or a part of the system is carried out in a distributed fashion.

Before going into further details, we first define the basic components of our architecture. We define a *computing element* (*CE*) as a device that contains a sensor or a set of sensors, a processing element (computing device) and a communication device i.e., a wireless transreceiver or wired communication capabilities. These devices could be placed in actual modules or subsystems where they would use the sensors as well as the *processing elements* (*PE*s). They could also be used just as a computing device in which case they could act as monitors for the rest of the system – schedule tasks, detect failures and initiate recovery, provide access to resources such as an external database etc – or act as "helpers" to offload the computation requirements from other *CE*s to maintain real-time constraints of the application. The *PE*s may not have sufficient computation capabilities for a large and complex system in which case a *central server* can be used. A *central server* is a more powerful computing device that collects data from the *CE*s – the *CE*s only preprocess the sensor data which involves minor computations such as simple filtering –, processes them and communicates the result back to the *CE*s. Note that the central server can be a multiprocessor platform and use parallel computations. Thus, the central servers can vary from a personal laptop to a cluster of microprocessors. The other important design element is the *network connection* architecture which determines how the various components are interconnected with each other. The *network connection* varies chiefly depending on the following factors:

- Wired or wireless connection capability,
- Processing power of *CE*s,
- Communication bandwidth.

Given the above definitions, a distributed prognostics system, thus, comprises of *CE*s connected using a *network connection* and may or may not contain *central servers*. There are two operating modes for a *CE*: *diagnostics* and *prognostics*. A *CE* runs in the default mode of diagnostics until a prognostics flag is raised by some *CE* in which case depending on the current state i.e., availability of its resources, it switches to prognostics mode. Thus, in the prognostics mode it is not necessary that all the *CE*s are utilized; some of them may be busy monitoring critical components or may not have enough computing power to simultaneously execute both default operations along with a part of the prognostics operation. When the prognostics mode is triggered, either *central server* or the *CE* that triggered the mode makes an estimate of how many computing resources are available and partition and delegate tasks accordingly. An overview of the system is provided in Figure 1.

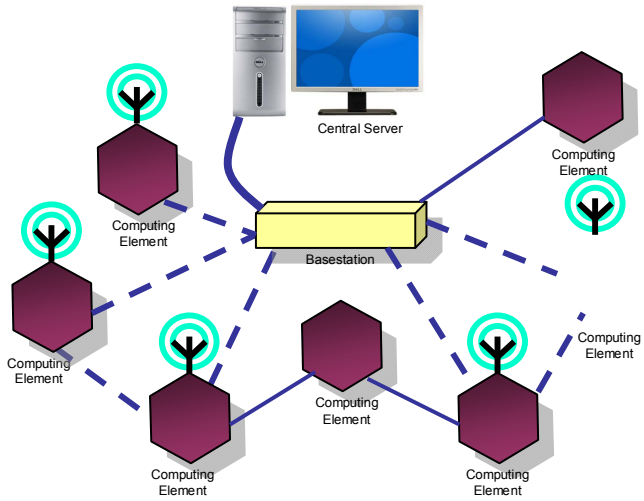

Fig. 1. Overview of distributed prognostics system architecture. Note that all the *CE*s may not have wireless connectivity.

In terms of system architecture there are three possible models:

1. Single *CE* with multiple helpers
2. Multiple *CE*s with a single helper
3. Multiple *CE*s with no helpers
4. Multiple *CE*s with multiple helpers

Note that for model 4, there are two possible scenarios: a) the *CE*s do not distribute their tasks and only share sensor data and b) the *CE*s share data and tasks. In our discussions, henceforth, we assume that the computation is always distributed and hence we do not consider case 3 for which the design considerations are much simpler.

B. Particle Filter based Prognostics Systems

Particle filters provide a powerful technique for prognostics. They are based on Bayesian learning networks and are often used to track progression of system state in order to make estimations of remaining useful life (RUL), which is at the core of system prognostics and health management. Bayesian techniques also provide a general rigorous framework for such dynamic state estimation problems. The core idea is to construct a probability density function (pdf) of the state based

on all available information.

In contrast, for the Particle Filter (PF) approach ([1], [6]) the pdf is approximated by a set of particles (points) representing sampled values from the unknown state space, and a set of associated weights denoting discrete probability masses. The particles are generated and recursively updated from a nonlinear process model that describes the evolution in time of the system under analysis, a measurement model, a set of available measurements and an a priori estimate of the state pdf. In other words, PF is a technique for implementing a recursive Bayesian filter using Monte Carlo (MC) simulations, and as such is known as a sequential MC (SMC) method.

For nonlinear systems or non-Gaussian noise, there is no general analytic (closed form) solution for the state space pdf. The extended Kalman filter (EKF) is the most popular solution to the recursive nonlinear state estimation problem [8]. In this approach the estimation problem is linearized about the predicted state so that the Kalman filter can be applied. In this case, the desired pdf is approximated by a Gaussian, which may have significant deviation from the true distribution causing the filter to diverge.

Particle filter methods assume that the state equations can be modeled as a first order Markov process with the outputs being conditionally independent which can be written as:

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}) + \omega_k$$
$$\mathbf{y}_k = h(\mathbf{x}_k) + \upsilon_k \quad (1)$$

where, k is the time index, $\mathbf{x}$ denotes the state, $\mathbf{y}$ is the output or measurements, and both ω and υ are samples from noise distributions.

In terms of computation, a particle filter based system essentially consists of the following three computational steps:

1. Sampling: In this step, samples (particles) of the unknown state are generated based on the given sampling function which provide an estimate of the current state of the system and also propagate the particles from the previous time step to the current time
2. Weight Calculation: Based on the observations, an importance weight is assigned to each particle
3. Resampling: This step involves redrawing particles from the same probability density based on some function of the particle weights such that the weights of the new particles are approximately equal.

Particle filters are expensive with respect to computation as well as memory requirements. However, they exhibit considerable amount of data parallelism to enable parallel processing. All steps enlisted above except resampling can be completely parallelized. Though, various efforts to derive parallelized versions of particle filters have been made, it has not been possible till now to formulate a complete parallel version. However, the resampling technique being used is often dictated by the application and system requirements and hence parallel versions cannot be used in all designs.

Based on the above observations, several architectures for particle filter based systems have been proposed ([2], [3]); some of which employ a central base-station/monitor while others do not. In our case, the resampling technique enforces use of a central base-station. The details of the architecture are presented in Figure 2. In this Figure, the *CE*s perform sampling and weight calculation while the central server performs all the steps in particle filtering. Thus, steps 1 and 2 in the particle filter are shared amongst the *CE*s and the central server while the resampling is done solely by the central server after collecting all the updated particle weight from the rest of the *CE*s.

The operational flow of the system is as follows:

1. When a prognostics flag is raised, the central server starts the prognostics algorithm by initializing the particle filter either based on a given fixed algorithm or reading from a file.
2. The central server collects information regarding which *CE*s are available for sharing the prognostics task.
3. The central server divides the computational load based on number of particles among the rest of the available *CE*s and sends this information to the *CE*s. It also sends the initial particle values.
4. The participating *CE*s read in the above information and read only the particle values assigned to them.
5. The *CE*s and the central server perform the sampling step.
6. If there is only one sensor, the *CE* responsible for reading that sensor value updates the central server with the reading which is then communicated to the rest of the *CE*s. If there are multiple sensors, the same is done for all the sensor readings.
7. Once the relevant sensor readings are obtained, the *CE*s and the central server perform the weight update step.
8. The *CE*s send the updated particle and their weight information to the central server which then performs resampling.
9. After resampling the central server sends the new particle information only to the *CE*s if the load distribution based on particles is constant for all the iterations. However, if the distribution varies over time – for example because any of the *CE*s notifies the central server that it cannot participate any more or its other jobs require more attention and hence this load needs to be reduced – the central server sends new load assignment as well.
10. After all the iterations are over, the central server notifies all participating *CE*s.

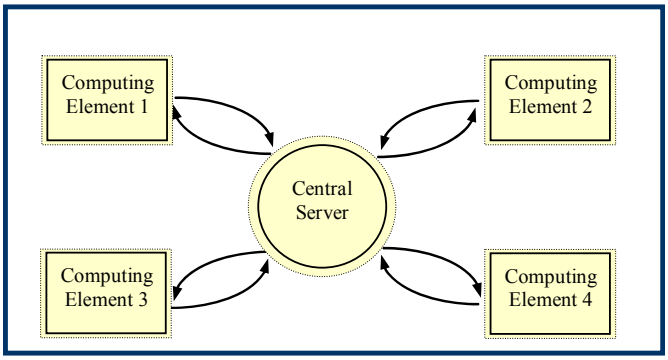

Fig. 2. System architecture for particle filter based prognostics system.

Note that all communication is acknowledgment based.

Thus, if acknowledgment(s) for a message is not obtained within a timeout period, the sender resends the message again. In the above discussions, we assume that a central server exists. However that may not be true for all applications, in which case one of the *CE*s is used as a central base station/monitor and could also be the *CE* that raised the prognostics flag.

IV. SUN SPOT (SMALL PROGRAMMABLE OBJECT TECHNOLOGY) DEVICE

A. Hardware Overview

The Sun SPOT device (Figure 3) is a small, wireless, battery powered experimental platform. It allows programming almost entirely in Java to allow regular designers to create and implement standalone systems in an easy and convenient way. The hardware platform includes a range of built-in sensors as well as the ability to easily interface to external devices.

In terms of hardware, there are two SPOT devices. One is a *base station* while the other is *free range* or *remote* device. The base station can connect to the development or host machine (a PC, or a laptop) and allows writing programs that can run on the host machine. The base station's radio can then be used to communicate with remote Sun SPOTs. Note that this does not imply that the free range SPOT devices have to be used in conjunction with the base stations always; they can be used independent of the base station and a host computing machine. The base station can also be used by the development tools to deploy and debug applications on remote Sun SPOTs. Thus the base stations mainly provide additional modes of operation as well as debugging capabilities. Note that a remote Sun SPOT can also be used as a base station, in which case its sensor board would not be used.

A full, free range Sun SPOT device is built by stacking a Sun SPOT processor board with a sensor board and battery. It is packaged in a plastic housing. The smaller base station Sun SPOT consists of just the processor board in a plastic housing.

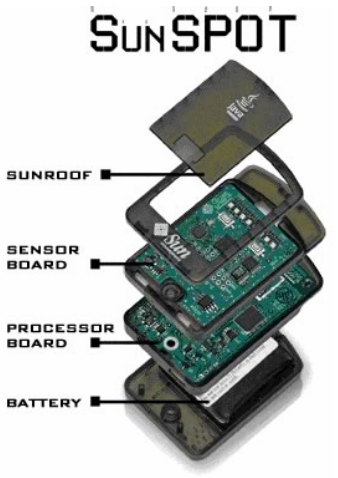

Fig. 3. Anatomy of a free ranging Sun SPOT device (courtesy of www.sunspotworld.com).

In terms of processing power, each Sun SPOT has a 180MHz 32-bit ARM920T core processor with 512K RAM and 4M Flash. The SPOT devices communicate using radio channels. The processor board has a 2.4GHz radio with an integrated antenna on the board. The radio is a TI CC2420 (formerly ChipCon) and is IEEE 802.15.4 compliant. Each processor board has a USB interface (used to connect to a PC). There are two LED's, one red and one green. Finally there is an 8-bit microcontroller Atmel Atmega88 used as a power controller. The battery used to power a SPOT device is a 3.7V rechargeable, 750 mAh Lithium-ion battery which is recharged whenever the USB interface is connected to a PC or powered USB hub. Note the base station Sun SPOT does not have a battery, getting its power via the USB connection to the host PC.

B. Software Overview

The Sun SPOTs unlike a lot of other embedded device do not require either hardware coding (using hardware description languages) or assembly language coding. It runs Java VM (Virtual Machine) on the processor directly. The VM executes directly out of flash memory. A Java Virtual Machine (JVM) is a set of computer software programs and data structures which use a virtual machine model for the execution of other computer programs and scripts. The model used by a JVM accepts a Java byte code which is a form of computer intermediate language normally but not necessarily generated from Java source code. Thus, the designer can develop the whole system in Java. The Java source code can then be compiled and the generated bytecode directly downloaded to the SPOTs.

The Sun SPOTs use a fully capable Java ME (Micro Edition) implementation called *Squawk* that provides basic OS functionality. Java ME is a specification of a subset of the Java platform aimed at providing a certified collection of Java APIs (Application Programming Interface) for the development of software for small, resource-constrained devices such as cell phones, PDAs. All the device drivers for the SPOTs are written in Java.

The software platform provided to develop Java programs for Sun SPOTs is Java Netbeans IDE. The NetBeans IDE allows applications to be developed from a set of modular software components called *modules*. Applications built on modules can be extended by adding new modules. The modules enable direct interaction with I/O (Input/Output) APIs as well.

V. EXPERIMENTS AND RESULTS

A. Application

The application domain chosen for the implementation of our proposed methodology is battery health monitoring. Batteries form a core component of the power supply system for many machines, and their degradation often leads to reduced performance, operational impairment and even catastrophic failure. Battery health monitoring has a wide variety of connotations, ranging from intermittent manual

measurements of voltage and electrolyte specific gravity to fully automated online supervision of various measured and estimated battery parameters. In our chosen application electrochemical impedance spectroscopy (EIS) is used to probe the internal electrochemical reactions of batteries. The measurement process involves the injection of a small AC voltage, swept over a large range of frequencies (e.g. 0.1Hz – 10kHz), superimposed over a fixed DC voltage at the terminals of the cells, while the impedance spectrum (Nyquist or Bode plot) is recorded.

The data collected is from second generation 18650-size lithium-ion cells (i.e., Gen 2 cells) that were cycle-life tested at the Idaho National Laboratory under the Advanced Technology Development (ATD) Program. The cells were aged at 60% state-of-charge (SOC) and various temperatures (25°C and 45°C). The full dataset is divided into a training set (25°C data) and a test set (45°C data), so as to examine the ability of the prognostic algorithm to handle the uncertainty in the different aging rates. Features extracted from the training data are used to estimate the internal parameters of the battery model (shown in Figure 4), which are subsequently used to initialize prognosis in the test case.

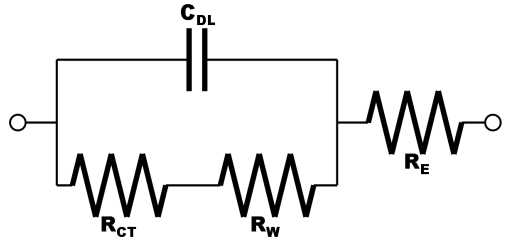

Fig. 4. Lumped Parameter Model of a Battery.

The parameters of interest are the double layer capacitance C_{DL}, the charge transfer resistance R_{CT}, the Warburg impedance R_W and the electrolyte resistance R_E, whose values change with various ageing and fault processes like plate sulfation, passivation and corrosion. From the aging data collected, R_E and R_{CT} are found to be changing significantly in value, and hence, are considered to be the state variables of interest. Exponential growth models, as shown in equation 2, are fitted onto their aging curves to identify the relevant decay parameters like C and λ:

$$\tilde{\theta} = C \cdot \exp(\lambda t) \quad (2)$$

where, $\tilde{\theta}$ is the model predicted value of R_E or R_{CT}.

The state and measurement equations that describe the battery model are given below:

$$\mathbf{z}_0 = \mathbf{C};\ \mathbf{\Lambda}_0 = \mathbf{\Lambda}$$
$$\mathbf{z}_k = \mathbf{z}_{k-1}.\exp\mathbf{\Lambda}_k + \boldsymbol{\omega}_k$$
$$\mathbf{\Lambda}_k = \mathbf{\Lambda}_{k-1} + \boldsymbol{\nu}_k$$
$$\mathbf{x}_k = [\mathbf{z}_k\ ;\ \mathbf{\Lambda}_k]$$
$$\mathbf{y}_k = \mathbf{z}_k + \boldsymbol{\upsilon}_k \quad (3)$$

where, the vector $\mathbf{z}$ comprises of R_E and R_{CT}, and $\mathbf{C}$ and $\mathbf{\Lambda}$ contain their C and λ values respectively. The $\mathbf{z}$ and $\mathbf{\Lambda}$ vectors are combined to form the state vector $\mathbf{x}$. The measurement vector $\mathbf{y}$ comprises of the battery parameters inferred from the test data. The noise samples $\boldsymbol{\omega}$, $\boldsymbol{\nu}$ and $\boldsymbol{\upsilon}$ are picked from zero mean Gaussian distributions whose standard deviations are derived from the training data. The particle filter uses the parameterized model described in equation (3) for the propagation of the particles (samples from the pdf of $\mathbf{x}_k$). Taking advantage of the highly linear correlation between R_{CT}+R_E and C/1 capacity (capacity at rated current) as derived from training data, predicted values of the internal battery model parameters are used to calculate expected charge capacities of the battery. The predictions are compared against RUL thresholds (end-of-life criteria) to derive the RUL estimates. Further details about the applied particle filtering framework can be found in [11].

B. Experimental details

The system architecture being used is the same outlined in section III.B. The *CE* in our experiments is the SunSpot and the software development for the Sun SPOTs was done using Netbeans IDE version 5.0. The base station – which can be connected to a PC – is used as a central monitor. The free ranging SPOT devices form the rest of the *CE*s. Experiments with 2 SPOTs – one free ranging and the other a base station – and 3 SPOTs – two free ranging and the other a base station – were carried out. However, since the in-built sensor capabilities of the SPOTs were not sufficient for the observations required for the batteries, sensor observations were read from an offline database via the SPOT base station. Note that a slight variation of such indirect sensor reading might be encountered commonly if some of the participating *CE*s do not make sensor readings i.e., act as helpers (as mentioned in III.A) in which case they read sensor values via the base station. The base station was connected to laptop using a USB cable. The files containing the sensor information and other initialization information were read through the USB cable and communicated to the free ranging SPOTs appropriately.

The main issue faced in the design was limitations imposed by the restrictions on the message length by the communication channel; all the state information for a single iteration could not be packed into a single message. Thus, the message had to be broken into multiple parts and sent iteratively both by the base station to send state information to the free ranging SPOT and the free ranging SPOT to communicate updated state information to the base station. The sensor values were broadcast at the beginning of every iteration of the particle filter. Since, there was only one application being executed by all the SPOTs, the availability of the SPOTs as well as the workload divisions were known a priori and hence this was not separately communicated. The initial particle values for each SPOT were sent individually using dedicated radio channels by the base station to the free ranging SPOTs.

Each state was a 2-dimensional vector. Also, since the parameter identification was done along with state estimation, 2-dimensional parameter values were also send along with each state value. The number of particles used was 100. First state tracking was performed for a few iterations followed by computation of the RUL or time-to-failure (TTF).

C. Results

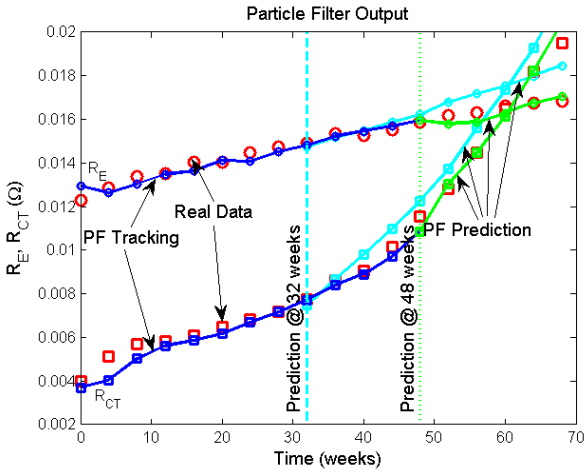

Fig. 5. Particle Filter Output using 2-SPOT configuration

Figure 6 shows both the state tracking and future state prediction plots for data collected at 45°C. RUL or TTF is used as the relevant metric for the state-of-life (SOL). This is derived by projecting out the capacity estimates into the future (Figure 6) until expected capacity hits a certain predetermined RUL threshold (end-of-life criteria). The particle distribution is used to calculate the RUL pdf by fitting a mixture of Gaussians in a least-squares sense. As shown in Figure 5, the RUL pdf improves in both accuracy (centering of the pdf over the actual failure point) and precision (spread of the pdf over time) with the inclusion of more measurements before prediction. The average (over 10 executions) RUL values computed at 32 weeks and 48 weeks for the 2-SPOT configuration are 58.06 weeks and 61.17 weeks respectively, while the corresponding values for the 3-SPOT configuration are 59.93 and 61.99 weeks.

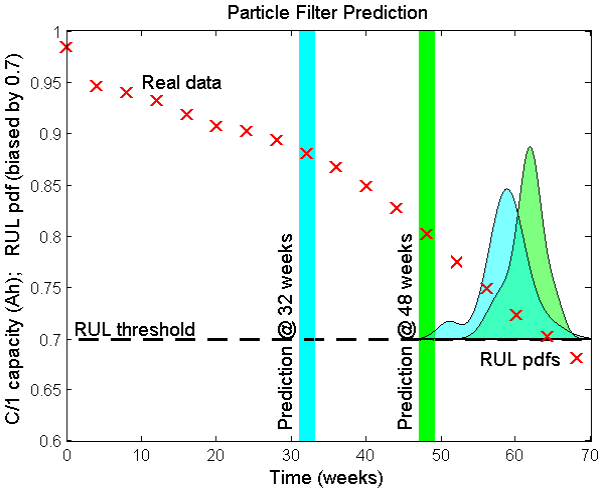

Fig. 6. Particle Filter Prediction using 3-SPOT configuration

The static program memory usages of the SPOT devices for both 2 SPOT and 3 SPOT configurations are as follows:

- Free range SPOTs: 29KB,
- Base station: 101KB.

The execution times (averaged over 10 separate executions of the whole system) for prediction after 8 weeks and prediction after 12 weeks using the 2-SPOT and 3-SPOT configurations are shown in Figure 7. An average was taken since the execution time varies – within a margin of 10-15 ms – mainly based on the wireless communication time which is dependent on the distance between the SPOTs. The execution time decreases for the 3-SPOT configuration compared to the 2-SPOT. However, a significant decrease is not observed due to the resampling step which is serial in nature and is executed completely on the base station. Also, as the number of SPOTs is increased, the amount of time spent on communication increases, which diminishes the effect of the gain in execution time obtained by distributing the computation workload.

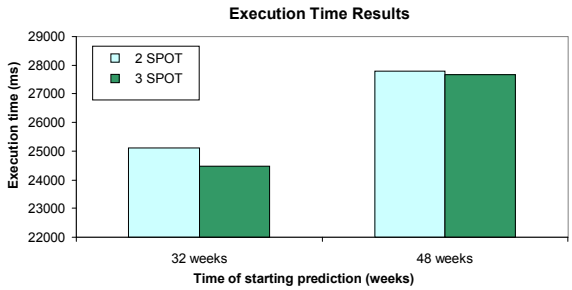

Fig. 7. Execution time results for 2SPOT and 3SPOT configurations

The low program memory utilization of the free ranging SPOTs demonstrate that more multitasking can be delegated to them; instead of executing a single application as discussed here more allocated tasks would enable more efficient use of resources. As shown by the results in Figure 9, for a multiple SPOT system, significant amount of time may be spent in communication. Thus, a free ranging SPOT can execute its own set of monitoring applications when the base station is busy communicating and collecting intermediate results from other SPOTs and processing them (in our case performing resampling).

VI. Conclusions and Future Work

The paper presents, in detail, a distributed architecture for prognostic applications. To demonstrate its functionalities and study the nature of such architectures a distributed prognostics system for batteries was successfully implemented on smart sensor devices with wireless communication capabilities. The results indicate the feasibility of using such a distributed implementation. The low execution times and program memory usage illustrate that for a complex system such a distributed architecture would enable a timely management of diagnosis and prognosis of multiple subsystems.

Future work would look into integration with direct sensor measurements using the SPOT devices and exploration of other devices with higher communication and computation power for systems with multiple diagnostic and prognostic applications running simultaneously.

References

[1] S. Arulampalam, S Maskell, N. J Gordon and T. Clapp, “A Tutorial on Particle Filters for On-line Non-linear/Non-Gaussian Bayesian

Tracking", IEEE Trans. on Signal Processing, vol. 50, no. 2, pp. 174-188, 2002.

[2] A. S. Bashi, V. P. Jilkov, X. R. Li and H. Chen, "Distributed Implementations of Particle Filters", in Proc. Of Sixth International Conference of Information Fusion, 2003. Volume: 2, pp: 1164- 1171.

[3] M. Bolic, P. M. Djuric and S. Hong, "Resampling Algorithms and Architectures for Distributed Particle Filters", IEEE Transactions on Signal Processing, Vol. 53, Issue: 7 pp. 2442- 2450 July 2005.

[4] M. Coates, "Distributed Particle Filters for Sensor Networks", in Third Intl. Symp. on Information Processing in Sensor Networks, 2004, pp. 99- 107.

[5] K. M. Goh, B. Tjahjono and A. J. R. Aendenroomer, "A Rapid Configurable Embedded Development Framework", In IEEE Conf. on Emerging Technologies & Factory Automation, 2007, pp. 135-140.

[6] N. J. Gordon, D. J. Salmond and A. F. M. Smith, "Novel Approach to Nonlinear/Non-Gaussian Bayesian State Estimation", Radar and Signal Processing, IEE Proceedings F, vol. 140, no. 2, pp. 107-113, April 1993.

[7] G. Ing and M. J. Coates, "Parallel Particle Filters for Tracking in Wireless Sensor Networks", In IEEE 6th Workshop on Signal Processing Advances in Wireless Communications, 5-8 Jun. 2005, pp. 935- 939.

[8] A. H. Jazwinski, "Stochastic Processes and Filtering Theory", Academic Press, New York, 1970.

[9] M. Roemer, C. Byington, G. Kacprzynski and G. Vachtsevanos, "An Overview of Selected Prognostic Technologies with Reference to an Integrated PHM Architecture", In Proc. of the First Intl. Forum on Integrated System Health Engineering and Management in Aerospace.

[10] M. Rosencrantz, G. Gordon and S. Thrun, "Decentralized Sensor Fusion with Distributed Particle Filters", in Proc. Conf. Uncertainty in Artificial Intelligence Acapulco, Mexico, Aug.2003.

[11] B. Saha and K. Goebel, "Uncertainty Management for Diagnostics and Prognostics of Batteries using Bayesian Techniques", in Proc. 2008 IEEE Aerospace Conference, March 2008.

[12] S. Saha, N. Bambha and S. S. Bhattacharyya, "A Parameterized Design Framework for Hardware Implementation of Particle Filters", in Proc. of the Intl. Conf. on Acoustics, Speech, and Signal Processing, pp. 1449-1452, Las Vegas, Nevada, March 2008.

[13] S. Saha, C. Shen, C. Hsu, A. Veeraraghavan, G. Aggarwal, A. Sussman and S. S. Bhattacharyya, "Model-based OpenMP Implementation of a 3D Facial Pose Tracking System", in Proc. of the Wkshp. on Parallel and Distributed Multimedia, Columbus, Ohio, Aug. 2006, pp. 66-73.

[14] X. Sheng, Y.-H. Hu and P. Ramanathan, "Distributed Particle Filter with GMM Approximation for Multiple Targets Localization and Tracking in Wireless Sensor Network", in Fourth Intl. Symp. on Information Processing in Sensor Networks, 2005, pp. 181- 188.