

REAL – TIME DATA MONITORING AND MANIPULATION
FOR WIRELESS SENSOR NETWORKS

by

SANKAR BHANU GORTHI

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

UNIVERSITY OF TEXAS AT ARLINGTON

MAY 2006

Copyright © by Sankar Bhanu Gorthi 2006

All Rights Reserved

ACKNOWLEDGEMENTS

मातृ देवो भव | पितृ देवो भव |
आचार्य देवो भव | अतिथि देवो भव ||

I venerate my Mother for She is Divine | I venerate my Father for He is Divine|

I revere my Teacher for He is Divine | I revere my Guest for the Guest is Divine||

I thank my Parents for all their encouragement and support throughout my education.

I would like to thank Dr. Frank Lewis for giving me the opportunity to pursue this thesis and for his inestimable support and guidance during the development of this thesis.

I am much indebted to Mr. Prasanna Ballal and Dr. Vincenzo Giordano, without whose input and numerable suggestions and guidance this thesis could not have been possible.

I would also like to thank Dr. Dan Popa and Dr. Harry Stephanou for graciously agreeing to be on the supervising committee for this thesis and for their helpful suggestions and feedback.

This thesis was funded by the following grants – The National Science Foundation IIS-0326505, The National Science Foundation CNS-0421282, The Army Research Office M-47928-CI-RIP-05075-1, The Army Research Office W91NF-05-1-0314 Singapore SERC TSRP grant 0421120028 and The National Instruments Lead User Grant.

March 7, 2006

ABSTRACT

REAL – TIME DATA MONITORING AND MANIPULATION FOR WIRELESS SENSOR NETWORKS

Publication No. _____

Sankar Bhanu Gorthi, M.S.

The University of Texas at Arlington, 2006

Supervising Professor: Dr. Frank Lewis

The growth of Wireless Sensor Network research has brought about the need for tools to help developers rapidly develop applications for various platforms. In the pursuit of these capabilities, this work deals with the design of a graphical approach to development of Wireless Sensor Network applications for implementation in the LabVIEW Graphical Development Environment. The work done also covers the implementation and development of the Sound and Vibration Toolkit for LabVIEW. The implementation of these tools in developing a novel test-bed at the Distributed Intelligence and Automation Laboratory (DIAL) at the ARRI at UT Arlington is also shown.

This work is supported by The National Science Foundation IIS-0326505 grant, The National Science Foundation CNS-0421282 grant, The Army Research Office grant M-47928-CI-RIP-05075-1, The Army Research Office grant W91NF-05-1-0314, Singapore SERC TSRP grant 0421120028 and a National Instruments Lead User Grant.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS.....	iv
ABSTRACT	v
LIST OF ILLUSTRATIONS.....	xii
Chapter	
1. INTRODUCTION.....	1
1.1 Introduction	1
1.2 LabVIEW	2
1.3 Objectives.....	3
1.3.1 Overview.....	3
1.3.2 The Distributed Intelligence and Autonomy Laboratory	4
1.3.3 The Sound and Vibration Toolkit for LabVIEW	5
1.3.4 List of Objectives	5
1.3.5 List of Contributions.....	6
2. WIRELESS SENSOR NETWORKS	8
2.1 Introduction	8
2.2 TinyOS	10
2.2.1 Introduction.....	10
2.2.2 OS Structure.....	11
2.3 nesC.....	12

2.3.1 Introduction.....	12
2.3.2 nesC Application/Component Structure.....	13
2.4 MICA series	15
2.4.1 MPR300 “MICA”.....	16
2.4.2 MPR400CB “MICA2”	17
2.4.3 MPR500CA “MICA2DOT”	18
2.5 Cricket series	19
2.5.1 Overview.....	19
2.5.2 Operation	19
2.6 MicroStrain® G-Link® Accelerometer Node.....	20
2.7 WSNs and the Distributed Intelligence and Autonomy Laboratory (DIAL).....	22
2.8 Conclusions	24
3. NATIONAL INSTRUMENTS™ LABVIEW™	25
3.1 Introduction	25
3.2 Programming Techniques	27
3.2.1 Basics.....	27
3.2.2 Other issues.....	28
3.3 LabVIEW and WSNs.....	29
4. THE SOUND AND VIBRATION TOOLKIT	31
4.1 Introduction	31
4.2 Toolkit Design	31
4.3 Toolkit Organization.....	32
4.3.1 Scaling	32

4.3.2	Limit Testing Analysis	33
4.3.3	Weighting Filters.....	33
4.3.4	Integration.....	34
4.3.5	Vibration-Level Measurements.....	35
4.3.6	Sound-Level Measurements.....	35
4.3.7	Fractional Octave Analysis.....	35
4.3.8	Frequency Analysis	36
4.3.9	Transient Analysis	36
4.3.10	Waterfall Display	36
4.4	Contributions to the SnV toolkit.....	37
4.4.1	Preliminary Discussion.....	37
4.4.2	Window.....	39
4.4.3	Kurtosis.....	40
4.4.4	Bollinger Band Sub-VIs	42
4.4.5	Kalman Filter	43
4.4.6	RLS Estimator.....	45
4.5	Conclusions	46
5.	THE DIAL	47
5.1	Introduction	47
5.2	TinyOS Toolkit.....	47
5.2.1	Active Message (AM) Resources.....	48
5.2.2	GetAMPacket.vi.....	48
5.2.3	GetTOSPacket.vi.....	49

5.2.4 DecipherTOSPacket.vi	50
5.2.5 GetCricketPacket.vi.....	51
5.2.6 TinyDB Resources	52
5.2.7 InitMote.vi	52
5.3 Implementation of Toolkit VIs.....	52
5.3.1 MicroStrain Acquire.....	53
5.3.2 XBow Acquire	56
5.3.3 Statistical Processing and Analysis Module (SPAM)	57
5.3.4 Cricket Demonstration.....	59
5.3.5 TinyDB Application	64
5.4 Implementation in the DIAL	66
5.5 Conclusions	67
6. CONCLUSION	68
6.1 Summary	68
6.2 Discussion of Objectives.....	70
6.3 Discussion of Contributions	70
6.4 Future work	71
6.4.1 TinyOS toolkit.....	71
6.4.2 LabVIEW and WSNs	71
6.4.3 The Distributed Intelligence and Autonomy Laboratory	72
6.4.4 Final Thoughts	73

Appendix

A. LIST OF IMPORTANT VIRTUAL INSTRUMENTS DESIGNED FOR THE TOOLKIT	74
REFERENCES	84
BIOGRAPHICAL INFORMATION.....	88

LIST OF ILLUSTRATIONS

Figure	Page
2.1 Crossbow MPR300 "Mica" (actual size)	16
2.2 Crossbow MPR400CB "Mica2" (actual size).....	17
2.3 Crossbow MPR500CA "Mica2Dot" (actual size).....	18
2.4 Crossbow MCS410CA "Cricket" (actual size)	19
2.5 MicroStrain G-Link Accelerometer (actual size).....	21
2.6 Warehouse scenario with a distributed network of sensors and mobile platforms: When an event, triggered in this case by a fire at the circled motes, occurs the robots attempt to guide the human operator whose position is localized using a localized mote platform to a safe area	22
2.7 Warehouse scenario with a distributed network of sensors and mobile platforms: Having determined the path that the human operator must traverse, the system sends robot R4 to monitor the unmonitored area near the door. Robot R2 guides the operator through safe areas in the warehouse	23
3.1 Demonstration of Race Conditions	27
3.2 Demonstration of avoiding Data Race Conditions by providing sub-VIs with duplicate outputs and error out connectors to ensure flow control.....	28
4.1 Block Diagram: Moving Window sub-VI	39
4.2 Block Diagram: Moving Window sub-VI Array Input	39
4.3 Block Diagram: Kurtosis sub-VI.....	40
4.4 Front Panel: Kurtosis Implementation.....	41

4.5	Block Diagram: Bollinger Bands	42
4.6	Block Diagram: Kalman Filter	43
4.7	Block Diagram: Kalman Filter sub-VI implemented using the custom matrix VIs	44
4.8	Block Diagram: RLS Estimator	45
5.1	Block Diagram: GetAMPacket.vi retrieves the next available TOS AM packet.....	48
5.2	Block Diagram: GetTOSPacket.vi - Synchronizes packet retrieval.....	49
5.3	Block Diagram: DecipherTOSPacket.vi Unescapes incoming Data and separates the payload and the CRC bytes	50
5.4	Block Diagram: GetCricketPacket.vi - Retrieves the next available Cricket message and retrieves the distance information and beacon ID.....	51
5.5	Block Diagram: MicroStrain Acquire: The original LabVIEW VI to handle communication and analysis and display.....	53
5.6	Block Diagram: MicroStrain Acquire modified with the SnV toolkit VIs and MicroStrain Drivers.....	54
5.7	Front Panel: MicroStrain Acquire shows a particular vibration sample	55
5.8	Front Panel: MicroStrain Acquire shows the FFT analysis of the measured vibrations on three channels	55
5.9	Front Panel: XBow Acquire shows the extension to implement Crossbow Mica2 drivers	56
5.10	Front Panel: Bollinger Band Analysis implementation in the Currency Analyzer	58
5.11	Front Panel: Cricket Driver implementation in a simple warehouse decision making scenario.....	59

5.12	Block Diagram: Cricket Demo showing the Serial communication "module"	61
5.13	Block Diagram: Cricket Demo showing the Cricket distance calculation "module"	62
5.14	Block Diagram: Cricket Demo showing the Mica decision making "module"	62
5.15	Block Diagram: Cricket Demo showing the Cricket Decision making "module"	63
5.16	Block Diagram: Cricket Demo showing the position mapping and graphic handler "module"	63
5.17	Block Diagram: TinyDB Query Builder	64
5.18	Front Panel: TinyDB Application showing the temperature distribution at four nodes	65

CHAPTER 1

INTRODUCTION

1.1 Introduction

Rapid developments in Wireless Sensor Networks (WSNs) have driven the rate of advancement in the resources and technologies available to researchers to astonishing levels. Advances in transistor design have raised transistor densities in circuit design leading to the physical size of devices shrinking exponentially. Fabrication techniques to create micro electro-mechanical systems (MEMS) are being perfected leading to low-power microscopic sensors being manufactured at very low costs. The advances in MEMS coupled with the improvements in CMOS technology have led to intelligence being embedded on tiny platforms. Together, these developments have helped to make the vision of potentially dust-size computing platforms into an inevitable reality. With low-cost CMOS-based RF radios being able to function adequately at low-power to support low data rate communication on these tiny nodes, sensor networks capable of performing wireless communications, local processing, data storage, sensing, hardware control (on mobile platforms) all within the physical size of a typical postage stamp. Future platforms will have the potential to fit within a cubic millimeter of volume.^{[1][8]}

Building smart applications to utilize the available capabilities on these platforms and building smart environments represents the next evolutionary development step for deployment in many fields, an important member of which is system automation in many warehouse scenarios. The detection of the relevant

quantities, monitoring and collection of data, evaluating, analyzing and storing the information and controlling the components of the networks through meaningful user interfaces, form major development concerns.^[2]

Industrial competition in the field of sensor network development is truly global with fragmented markets in the fields mentioned above and customers expect the best product at the best price with immediate availability. Meeting consumer demands requires a great deal of flexibility, low-cost/low-volume manufacturing skills, and short delivery times.^{[2][3]}

Success in manufacturing, and indeed survival, of the sensor network developer is increasingly more difficult to ensure and requires continuous evolution of the sensor platforms, learning to integrate new development in network topologies unleashed everyday and unlearning technologies which have become obsolete because of the increasing demands of the consumer. The developer of the sensor network should be able to adapt to constantly fluctuating consumer requests. However the development time and consequently the production time are adversely affected. This has made the availability of tools to improve manufacturing performance a strategic weapon for competition and future success.

1.2 LabVIEW

The LabVIEW Graphical Development Environment plays a major role in cutting down design, simulation, verification and manufacturing times and is an essential tool in data analysis, storage, display and deployment. The LabVIEW programming language provides an easy-to-use interface, online compilation of code as

it is “written”, concurrent error handling, inherent multi-thread processing and an extensive library of Virtual Instrument blocks (VIs). LabVIEW also provides for the simultaneous generation of user-end interfaces and the relevant code to represent the application. LabVIEW is available in various popular operating system platforms.^{[4][5][6][25]} These features come together to provide Wireless Sensor Network developers with the perfect platform to build applications to program the network and handle base-station or processor intensive calculations easily. Although LabVIEW’s set of mathematical tools are limited at present (version 7.1), newer versions can be expected to be more powerful in handling complicated mathematical analysis (in the present version, the user has to rely on a port to MATLAB placed within the LabVIEW environment to handle these calculations. The creation of VIs to handle simple matrix operations is also an objective of this thesis.

1.3 Objectives

1.3.1 Overview

The analysis of data retrieved from deployed sensor networks is a common application. However the retrieval of data is non-trivial and involves considerable expertise in being able to communicate with the sensor network and being able to address particular sections of the network for pertinent data. Care must also be taken not to overload the considerable but limited power and programming resources on the platforms. Further, the translation of the data to meaningful and consistent formats for analysis in standard tools available in various development environments is essential.^[8]

Various tools in numerous development environments are available for analysis of the data and the development of aesthetic user-interfaces for interacting with the sensor network. The majority of these tools have been developed using text-based programming languages like java, python and perl while Microsoft's suite of Visual Studio languages and MATLAB have also been used to a considerable extent. However, all of these languages take considerable time and effort for a developer to learn and implement effective solutions for a required application. National Instruments™ LabVIEW™ Graphical Development Environment is a less widely used environment in this field, but is gaining popularity. One of the main objectives of this thesis has been to develop tools which integrate into the LabVIEW environment seamlessly to provide developers tools to rapidly design Virtual Instruments for modeling, testing, remodeling and final deployment. The effectiveness of LabVIEW in the development of Wireless Sensor Network applications is also studied.

1.3.2 The Distributed Intelligence and Autonomy Laboratory

Tying the capabilities of versatile sensors, with multi-functional robotic platforms, the Distributed Intelligence and Autonomy Laboratory (DIAL) at the Automation and Robotics Research Institute (ARRI) – University of Texas at Arlington has been conceived as a powerful, adaptive, malleable and intelligent test-bed for developing technologies in Discrete event coordination, Self-localizing networks and Adaptive sampling to name a few. The goal of the laboratory has been to acquire off-the-shelf, customizable platforms and rapidly integrate them into multi-platform scenarios.^[6]

An objective of this thesis has been to develop LabVIEW tools to speed up and streamline the process of communication between the various platforms. Rudimentary decision – making systems for testing implementations of the platforms and the LabVIEW tools developed in this thesis are also to be conceptualized and implemented.

1.3.3 The Sound and Vibration Toolkit for LabVIEW

The LabVIEW platform is bundled with a variety of toolkits to perform a variety of functions. Of these, the new Sound and Vibration toolkit is studied in this thesis for its effectiveness. The VIs were to be implemented in applications that were commonly required in the test-bed at the Distributed Intelligence and Autonomy Laboratory (DIAL). Although these tools are quite extensive, LabVIEW does not provide drivers and analysis tools for the instruments which are used at the DIAL. A major objective of the thesis was to develop these tools for addition to the Sound and Vibration toolkit.

1.3.4 List of Objectives

- Identification of platforms utilized at the DIAL
- Development of tools for the programming of/communication with the various platforms
 - for LabVIEW
 - for the platform's native programming language
- Maintaining a standard mode of application/driver structure in order to sustain manageability and ease of debugging

- Development of tools to be included in the Sound and Vibration toolkit of LabVIEW
- Implementation of the tools developed in the test-bed at the DIAL

1.3.5 List of Contributions

The result of the objectives listed above is briefly listed below. The elaboration of each is given in later chapters.

- Development of drivers to interface LabVIEW to the hardware used at the DIAL – hardware supported
 - MicroStrain G-Link Accelerometer
 - Crossbow Mica
 - Crossbow Mica2
 - Crossbow Mica2Dot
 - Crossbow Cricket
- Improved existing applications for Condition Based Maintenance with the new Sound and Vibration Toolkit from LabVIEW
- Virtual Instrument Libraries developed for addition to LabVIEW's suite of toolboxes – specifically:
 - Custom Matrix multiplication VIs
 - Windowing sub-VI
 - Kurtosis VI
 - Bollinger Band Analysis
 - Kalman Filter

- RLS Estimator
- Initiated the TinyOS toolkit for LabVIEW – VIs contributed
 - Serial communication configuration VIs
 - AM message reader
 - Cricket message reader
 - Mote communication synchronizer
 - Packet deciphering
- A host of applications implemented in the test-bed
 - Statistical Processing and Analysis Module
 - MicroStrain Application for CBM
 - Crossbow Application for LabVIEW
 - TinyDB application for LabVIEW
 - Cricket navigation module

CHAPTER 2

WIRELESS SENSOR NETWORKS

2.1 Introduction

A Wireless Sensor Network (WSN) may be defined as a network composed of numerous small computers, employed in the processing of sensor data. These small computers are generally designed to be extremely basic in their functionality and in most cases are meant to be so. Designers of WSN platforms advise developers to use programming structures which are modular and run for short periods of time so as to prevent the platform from being locked into indefinite loops which burden the limited power resources available.^[8]

A major contributor to the WSN community which has been modeled on the above design philosophy is TinyOS which is “an event based operating system environment designed for use with embedded networked sensors”. TinyOS was initiated by the Electrical Engineering and Computer Sciences department of the University of California – Berkeley (UC Berkeley) around the beginning of this millennium. It is based on the Open Source philosophy wherein all the source code and documentation required for the development of TinyOS based programs is freely available to the general public. TinyOS is now developed by a world-wide consortium of developers at various educational institutions all over the world but is still lead at the helm by developers at UC Berkeley. The TinyOS platform is intended to be incorporated into the smartdust concept introduced by Kristofer Pister of UC Berkeley.^[22] Smartdust is now a

DARPA project aimed at creating “massively distributed sensor networks”. Smartdust is a general class of tiny wireless sensor systems and encompasses MEMS sensors, micro-robots and other devices equipped with wireless communications and capable of deploying various sensors (audio, light, temperature, vibration etc.).

The TinyOS environment is built using “stylized C” or a customized version of the C programming language called “network embedded system C” or nesC for short.

Although TinyOS may be ported to numerous platforms, the Wireless Sensor Network solutions provided by Crossbow Technology (XBow) have a huge developer base and as the TinyOS website notes, over 500 research groups and companies use TinyOS on the XBow “Motes”. The test-bed at the DIAL utilizes the MPR300/MPR310 “MICA”, the MPR400CB “MICA2” and the MPR560 “MICA2DOT” mote platforms developed at UC Berkeley and the MCS410 “Cricket” motes developed in a joint collaboration between XBow and the Massachusetts Institute of Technology (MIT). These platforms were all coded using nesC code in the TinyOS environment at the DIAL.

The DIAL also works with wireless sensors – the 900/868MHz G-Link® wireless accelerometer and the 900/868MHz SG-Link® wireless strain gauge – from MicroStrain®. These are programmed and handled entirely in LabVIEW.

The following sections will look at the structure and organization of the above environments and platforms.

2.2 TinyOS

2.2.1 Introduction

As noted above, TinyOS is an event based environment on top of which nesC programs are implemented. TinyOS provides a component-based architecture that supplies the drivers for the various sensors, while also providing network protocols for handling general broadcast or multihop routing, distributed services to handle parent selection in multihop, multi-threading – a standard feature in most operating systems – customized for the severe memory restrictions inherent in the miniaturized wireless sensor platforms which run many simultaneous timer algorithms to handle the numerous components handled in most TinyOS applications, and data acquisition tools which allow for black-box access to all the sensor drivers for standard operation or for easy customization to suit a custom solution. The event-based execution scheme helps to simplify power management for the available platforms which mostly still rely on galvanic or gel cells for power supply. This architecture has been, and is, an excellent solution to handle the myriad events which occur randomly in wireless sensor network applications.

All these operations are designed to work within a very small operating system footprint. The core of the operating system requires just about 400 bytes of physical memory.

The structure and design of TinyOS are constantly being upgraded as it adapts to strenuous testing by various research groups and hence, requires the developer to keep abreast of many issues when preparing applications.

2.2.2 OS Structure

The heart of the TinyOS application structure is specified as blank interface definition components, which form the template on which all components are intended to be designed. The interface definitions also allow the developer the freedom of implementing functions and events in a logical programming order which makes sense to the developer. Once the interfaces have been defined, the standard TinyOS component “provides” interfaces which a higher level application “uses” to implement the functionality. “provides” and “uses” are incidentally keywords in the nesC programming language and can be visualized as input and output pins on a chip which connect to the rest of the circuit board (which is essentially how the application is structured). Each TinyOS component consists of a declaration of interfaces which it uses and provides followed by the actual implementation of the interfaces. These are simple C-style functions which are given special keywords customized for nesC like “task”, “event” and “command”. A more detailed description is provided in the following section. ^[22]

Although this structure seems simple enough, it takes quite a bit of effort and exercise for a developer to grasp the functionality of even basic applications which access components which do not utilize networking components. Following a code tree for a mammoth TinyOS based application is cumbersome and convoluted to say the least. Although this might be said to be true for any programming language, the recursive use of implemented components at various locations in the TinyOS application structure causes the developer to be thrown off quite often without a visual code tree to follow and refer to.

As research into various networking strategies to be implemented on these miniaturized platforms proceeds to generate new implementations, the fluid change in the standards used in TinyOS programs helps only in increasing the complexity of the implementations. Although the TinyOS community does try to keep drastically new implementations to a minimum, as can be expected from any fledgling technology, constant innovations are a harsh reality until a wider implementation base is achieved. With a larger consumer base, standardized implementations will increase productivity for developers. However, as most of the platform implementations are still viewed as being in the research phase, and as research into newer platforms continues, the TinyOS programmer is resigned to keeping up with the worldwide consortium in order to develop relevant applications which are stable as well as up-to-date.

Newer versions of TinyOS (TinyOS 2.x for example) attempt to simplify this process by redefining the application structure. However these versions are still in their infancy and the world wide consortium is working on porting existing TinyOS 1.x components to the newer versions. In a later part of the thesis, it is suggested how the newer versions are easier to model using LabVIEW so that developers simply wire together components to generate the nesC code automatically. All the applications at the DIAL were built on the TinyOS 1.x version.

2.3 nesC

2.3.1 Introduction

nesC was developed by the University of California – Berkeley’s EECS department and the Intel Research Center at Berkeley. The nesC language was designed

to supplement the architecture defined and required by TinyOS. With the development of nesC, the developers of TinyOS were able to allow it to evolve into a stable and reliable environment. nesC provides an easy interface to connect and control various physical components on various platforms even when the resources for computing on the platform are limited. The data collection and processes and timing handlers on the platforms are streamlined by the use of event driven task execution rather than on interaction with a controller input or relying on batch processing routines. The nesC language was also designed to handle data race conditions which arise out of the task and data acquisition concurrency handled by TinyOS and detects them at compile time.^[22]

nesC was designed keeping in mind that the system is expected to run for very lengthy intervals of time without human intervention or interaction and should be able to operate even in the event of hardware malfunctions or issues at run-time.

2.3.2 nesC Application/Component Structure

nesC was built as a customized version of the C programming language and so keeps many of the code structures from the original, which many programmers are familiar with. As such, it is a simple step forward from C.

nesC, to aid the concept of components in TinyOS's event-based concurrency, has "wiring" conventions to wire together various components which are to be used in a bigger component or a target application. In fact, from a C programming point of view, the wiring of interfaces between different components essentially creates function pointers which address functions interfaces defined in sub-components and

implemented either in the sub-component or in the component being developed. These functions may be provided as interfaces, if required, for super-components to implement, or for the super-component to override, and so on. This essentially helps in creating black-box components which a new developer would quickly wire together for an application without having to go through the working of the functions in the sub-component.

nesC is a static language in that all memory is allocated at compile time. No dynamic memory assignments are required as the component based-architecture eliminates the need in most cases.

A nesC “task” runs to execution. It does not preempt other tasks, while a nesC “event” may. Tasks are called using the “post” keyword. A task when finished returns the program execution to its poster. Events run to execution too, but are operated in split-phase i.e. the operation request and completion signalers are separate functions. A nesC “command” is typically called to execute an operation like toggling LEDs or sending messages over a communication port. If the execution of the system is split-phase, the completion of the command results in the invocation of a completion event. Typically, when concurrent requests reach the application, the system rejects them in favor of running commands. As most command requests repeat at short intervals until acknowledged, the incoming command request need not be queued for execution. However some components do use this system. Following the former application control structure, though, helps in keeping applications uniform and free for execution even in the event of command requests being retracted. The developer would of course

have to use his/her personal judgment to determine the required structure depending upon available resources and the target application.

In general however, all task implementations are short. Command executions usually call many tasks and sub-functions and hence need careful mapping. Events usually are used to signal interfaces or call commands to execute such as sending messages via communication interfaces. nesC does support all mathematical functions supported by C, and C datatypes.

2.4 MICA series

The MICA series of motes were initially developed as upgrades to the existing rene2 platforms developed at UC Berkeley in 2001. The MICA series are macro-sized sensors compared to the micro-sensors with essentially the same features developed at UC Berkeley. The MICA series has been built to be sturdy under strenuous testing and repeated reprogramming and will eventually be replaced by cheaper, smaller motes (a working model 5 mm² has been developed at UC Berkeley) which are to be mass-produced and deployed in target locations.

2.4.1 MPR300 “MICA”

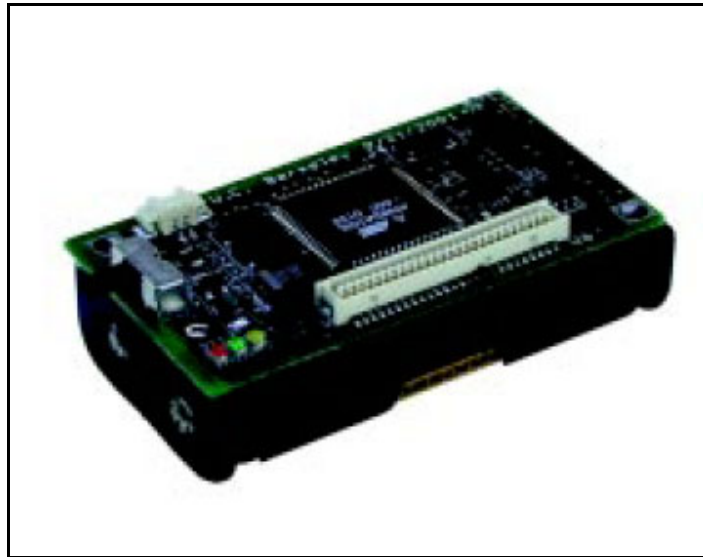


Figure 2.1 Crossbow MPR300 "Mica" (actual size)

This was the first upgrade to the rene2 and was developed in 2002. It features a 4MHz Atmel ATmega 128L microcontroller and typically features a 916MHz radio. It has a 10 bit 8 channel ADC. The MICA board also holds three programmable LEDs and a 51 pin expansion connector to interface removable sensor boards. The production of this range has been discontinued from XBow after to the development of the MPR4x0 “MICA2” series.^[27]

2.4.2 MPR400CB “MICA2”

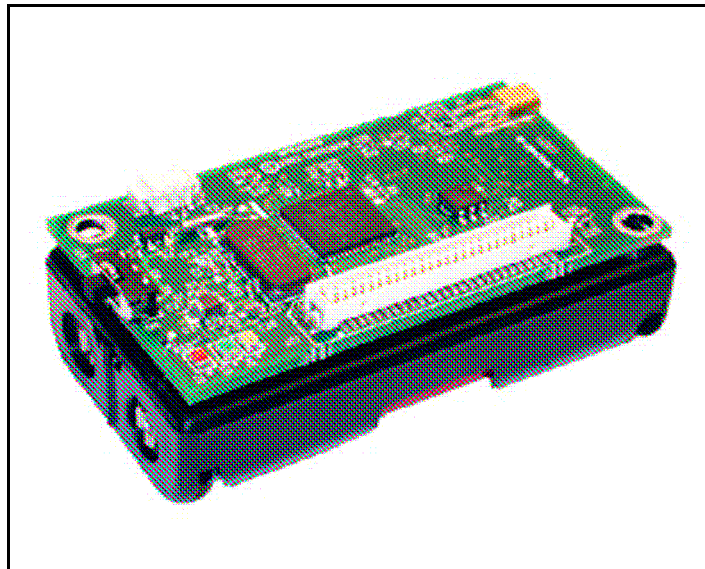


Figure 2.2 Crossbow MPR400CB “Mica2” (actual size)

The MICA2 is the second upgrade to the rene2 and features a 7.3827MHz Atmel ATmega 128L microcontroller. It features a radio with a tunable radio which typically runs at 433MHz but can be tuned to run at 315MHz, 868MHz or 916MHz. The MPR400CB has 50 radio channels which are controllable programmatically, but the MPR410CB and the MPR420CB are built with lesser channels. All these sport the same memory ratings as the MICA with the major upgrades being made in the radio circuit and the addition of I2C buses and digital input output pins to expand the board for customization. The MICA2s draw more current at run-time than the MICA but typically run longer in sleep mode than the MICA for identical programs and hence last longer on two AA (double A – 1.5V) batteries than the MICA does. During testing the two platforms at the DIAL, the MICA2 performed better at Multihop handling strategies and handling bulkier applications than the MICA.

2.4.3 MPR500CA “MICA2DOT”

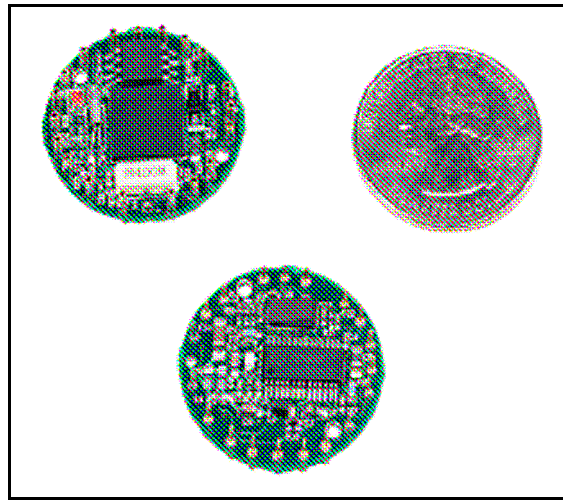


Figure 2.3 Crossbow MPR500CA “Mica2Dot” (actual size)

The MICA2DOT is essentially a miniaturized version of the MICA2 and was developed along with the MICA2. It features only one status LED and has an 18 pin expansion connector for connecting the 18 pin range of sensor boards manufactured at XBow. It comes with a 3V Coin cell holder to power it, but has a provision to be connected to a regulated power supply of 2.7V to 3.3V. It has a maximum outdoor radio range of 500ft. In addition to the above, the MICA2DOT has an onboard temperature sensor and a battery monitor as opposed to the MICA2 which needs the MTS range of sensor boards for interfacing Acoustic, Light, Temperature, Vibration and other sensors along with the output buzzer.

2.5 Cricket series

2.5.1 Overview

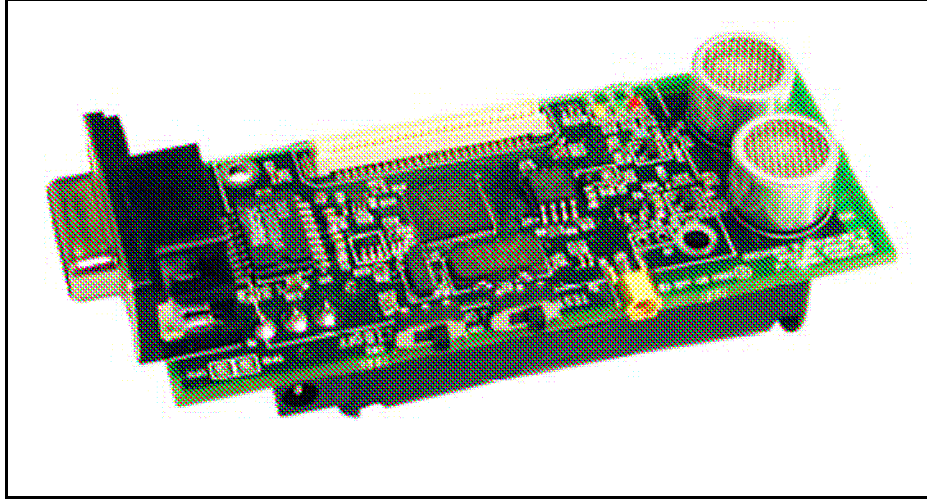


Figure 2.4 Crossbow MCS410CA "Cricket" (actual size)

The MCS410CA “Cricket” series of motes are a recent addition to the XBow product list and are prototype models for many announced upgrades. The Cricket motes are built on top of the MICA2 architecture and possess the same features except for the addition of the Ultrasonic transmitter receiver pair attached to the platform. The Computer Science and Artificial Intelligence Laboratory at MIT has been testing miniaturized versions of the Cricket motes to fit into standard memory card slots available in most handheld electronic devices. With the motes drawing power from the device they’re plugged into, the compact mote would be extremely convenient for localization of mobile users.

2.5.2 Operation

The Cricket mote is designed as an alternative to expensive indoor GPS. The Cricket network requires the installation of Cricket motes configured to run as beacons

placed in various unobstructed locations. A mote configured to run as a listener detects the radio and ultrasonic beacon signals broadcast by the different beacons and through a series of ultrasonic and radio transmission algorithms determines its distance to the beacons. It operates on line of sight however, so requires that the listener mote be pointed towards the beacons. These messages are compiled in simple string message formats and posted to the available communication interface. These messages may either be sent over the radio as listener messages (after some non-trivial manipulation of the nesC code) or via the serial port connection provided on the mote.

The code behind the operation of the cricket mote is the basic TinyOS application built using nesC code. The motes may be configured with simple string commands which the inbuilt program parses. Hence, with a simple serial port or applicable connector interfacing application running on the electronic device, a simple LabVIEW or Java application may be used to localize the moving mote.

The range of the listener beacon pair is about 25ft with an angle of attack of about 45°. Hence, ideally, in an indoor environment, the cricket beacons can be used as environment monitors as well as message routing agents.

2.6 MicroStrain® G-Link® Accelerometer Node

The G-Link® accelerometer node is a triaxial accelerometer node. It operates in the 902-928MHz radio frequency band. It consists of a coin-sized MEMS based accelerometer which operates in continuous mode of operation where it logs measured data and transmits the data in periodic radio bursts which are picked up by a transceiver connected to a base computer via a serial or USB interface. These motes have a very

strong 100m line of sight transceiver range. The nodes are used for vibration and inclination sampling. They also sport a substantial amount of memory to store up to 1,000,000 measurements. They also support simultaneous streaming from multiple nodes to the base station receiver.

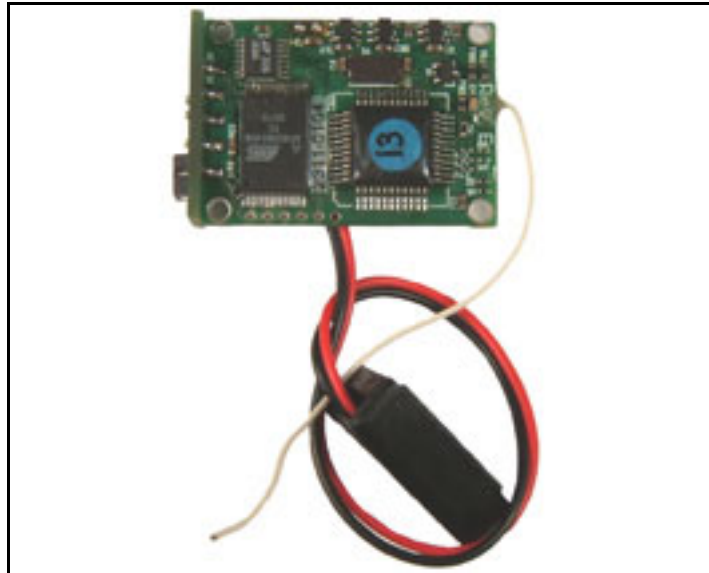


Figure 2.5 MicroStrain G-Link Accelerometer (actual size)

The programs on these nodes are hard-coded into the memory and programmers only have access to flash-registers which carry status flags which can be set to determine the execution mode of the node or to set data requests. The various flash registers can be addressed by sending simple serial string packets containing the address followed by the data to be placed in the registers. These registers are reset however on restarting the node. The nodes at the DIAL use 9V regulated power supplies which can be replaced by battery packs. The nodes sport 12-bit Analog to Digital Converters and are very accurate. These nodes were controlled entirely in LabVIEW.

2.7 WSNs and the Distributed Intelligence and Autonomy Laboratory (DIAL)

The DIAL employs various sensor platforms and mobility platforms which are used in WSN research. One example of the application scenarios and which can be instantly applicable to a well known scheme is Sensor Network and Mobile Platform assistance in a fire-escape strategy.^[6]

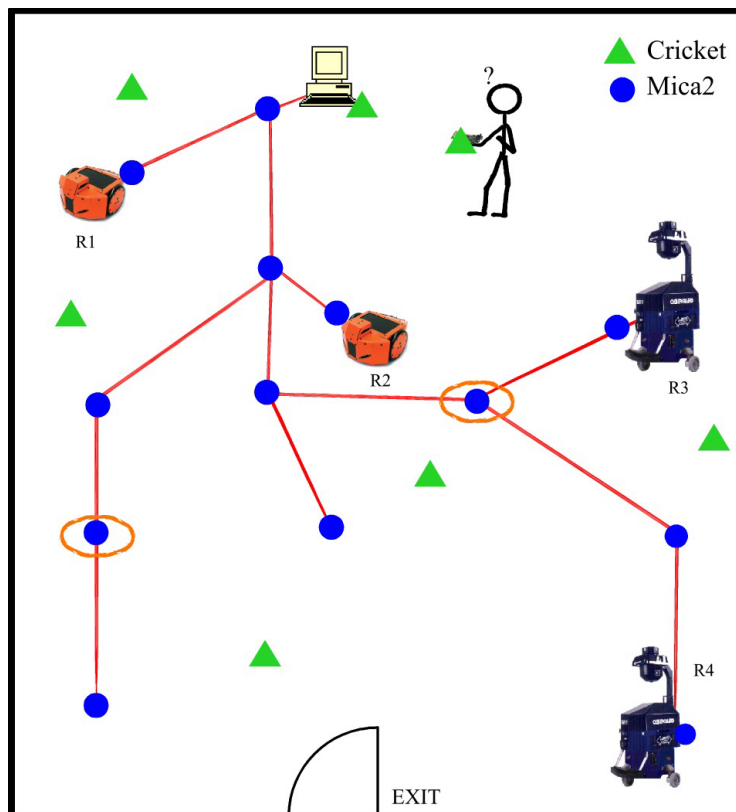


Figure 2.6 Warehouse scenario with a distributed network of sensors and mobile platforms: When an event, triggered in this case by a fire at the circled motes, occurs the robots attempt to guide the human operator whose position is localized using a localized mote platform to a safe area

The sensor networks in the scenario shown involve independent MICA2 and Cricket mote networks. The Cricket networks are used exclusively for localization of the mobile nodes, platforms and human operators. The localization algorithms used on the robots controlled by the PXI microcontrollers need discrete Kalman filters and RLS

estimators which were implemented as part of this thesis. They are described in Chapter 4. The static MICA2 network monitors the warehouse for temperature, light, etc and determines the location of the event(s). Event-triggering requires even-detection algorithms which can be implemented using Bollinger Band type of algorithms and Kurtosis, Skew and other measurements which are also implemented in Chapter 4. The system then determines the path in which to guide the human and the mobile platforms are triggered to monitor the region and the path to be followed.^{[4][14][16]}

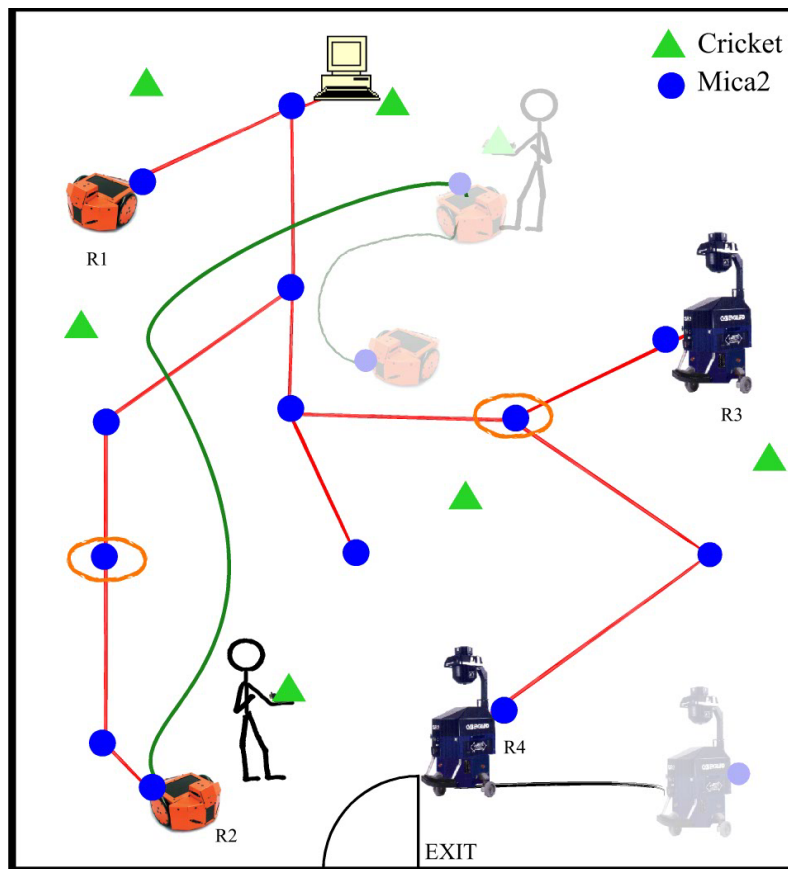


Figure 2.7 Warehouse scenario with a distributed network of sensors and mobile platforms: Having determined the path that the human operator must traverse, the system sends robot R4 to monitor the unmonitored area near the door. Robot R2 guides the operator through safe areas in the warehouse

Scenarios like these require seamless integration of the data acquisition and analysis modules and this is provided by the tools described in the succeeding chapters.

2.8 Conclusions

This chapter dealt with some of the various platforms which are available at the DIAL. The modular nature of the TinyOS application structure and wiring scheme of nesC are of particular interest as these bear many similarities to the structure of typical LabVIEW™ virtual instruments. The following chapter discusses the programming concepts of LabVIEW and their applicability to Wireless Sensor Network applications.

CHAPTER 3

NATIONAL INSTRUMENTS™ LABVIEW™

3.1 Introduction

LabVIEW (Laboratory Virtual Instrumentation Engineering Workbench) is a graphical development environment from National Instruments™ (NI). The LabVIEW distribution was first developed as a programming language for the Apple Macintosh in 1986. It has since evolved into an environment which aids scientific and engineering research and application building with various versions and flavors for different operating systems with versions available for Microsoft® Windows® UNIX, Linux and Mac OS.

LabVIEW is an extremely powerful data acquisition, analysis, storage and presentation tool. The programming language in LabVIEW called “G” represents code execution as dataflow. G is one of the first in a host of dataflow languages developed for various platforms and target applications. MathWorks’ Simulink and Agilent VEE can be considered to be prominent competitors in the scientific and engineering community to LabVIEW. However, NI has always provided simple solutions to incorporate MATLAB code and to interface Simulink code to simple evaluation VIs. With the introduction of better file/project structuring and matrix-handling strategies in the latest version of LabVIEW (LabVIEW 8.0), NI seems to be challenging the MathWorks hold on engineering community. However, further discussion in this vein is outside the scope of this thesis.

The straightforward visual dataflow and structuring of LabVIEW programs simplifies program development and debugging. The extensive libraries in LabVIEW to handle various common tasks in standard data acquisition and analysis engineering (or many other areas of use for that matter) help the developer to adhere to standard application or virtual instrument templates so as to decrease complexity while providing the ability to customize application-flow to suit the target function of the virtual instrument. LabVIEW is especially useful to non-programmers as putting together simple applications for a basic function is straightforward. A rudimentary introduction to the available tools and resources is of course needed, but the extensive online and offline help provided by LabVIEW and the many examples in the tutorials simplifies the process.

Even though critics of LabVIEW claim that debugging of larger applications running complex algorithms is complicated and takes a tremendous amount of effort even by the original programmer, it can be argued that that is true for all complex systems. The visual nature of LabVIEW and easy access to sub-VIs – the main menu in any front panel window contains the tree of sub-VIs of any VI – softens the blow a great deal, as opposed to other visual languages like Visual Basic for example. The concurrent error handling provided while preparing LabVIEW applications also ensures syntactical errors do not occur in the program.

3.2 Programming Techniques

3.2.1 Basics

The LabVIEW program is called a Virtual Instrument (VI) as the development of the code which controls data flow concurrently creates objects on a front-end display for the user to interact with the program. The program flow in G is not a linear execution of tasks and depends on the arrival of all inputs at a particular node for it to execute. This mostly requires that various nodes run simultaneously. Parallel execution of various nodes of the virtual instrument is therefore innately implemented in G.^[25] The inbuilt scheduler of most LabVIEW run-time engines automatically handle multi-threading and multi-processing at optimized levels. This is again one of the reasons why LabVIEW is a powerful programming environment.

This however raises the issue of data race conditions which may occur, for example, when nodes which are required to run sequentially execute due to the immediate availability of all inputs. This can be simply demonstrated as in the figure below.



Figure 3.1 Demonstration of Race Conditions

The initialization, write and close sub-VIs for serial communication are all connected directly to the Serial Port identifier. This causes the VI to behave unpredictably.

This issue can be handled by simply forcing the dataflow by either placing the VIs in sequence structures or by connecting the error or duplicate Serial Identifier outputs to sub-VIs which are forced to wait for the previous VI(s) to execute.

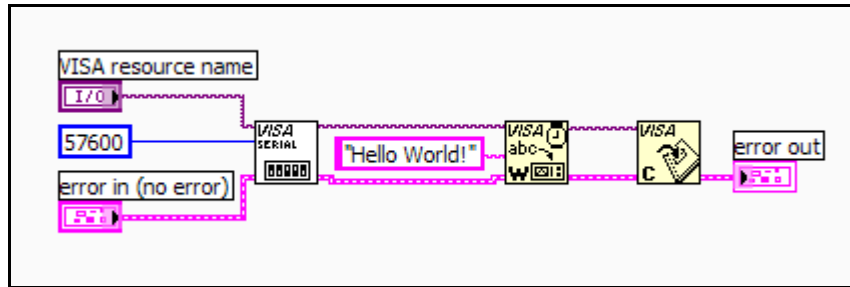


Figure 3.2 Demonstration of avoiding Data Race Conditions by providing sub-VIs with duplicate outputs and error out connectors to ensure flow control

The latter is a better alternative as it ensures consistent error handling and eliminates the need for sequence structures which aid to the visual complexity of the VI.

In addition to the above, it should be mentioned that as VIs increase in functionality and complexity, the increased visual size of the code actually encourages programmers to adopt a modular approach which ensures better organization. This architecture, along with continuous flow of data simplifies debugging and maintenance.

3.2.2 Other issues

It should also be mentioned that it is good programming practice to ensure that sub-VIs are identifiable by purpose and classification. This ensures reusability and simplifies later implementation or modification of applications.

Grouping together parallel processes is not suggested. Instead, separating them and placing them in separate while loops and interfacing them with the liberal use of local variables ensures faster performance. For example, data acquisition from serial

ports and analysis of data thus obtained, and the subsequent display of the translated data can be seen as three separate processes. Applying different time delays on each process leads to better program and hardware control.

3.3 LabVIEW and WSNs

With the above programming paradigm and the inherent need for modularity and ease of visualization of data retrieved from a WSN platform, it is but logical that a graphical approach will be a better solution for designing WSN applications (if not all data acquisition and programming applications).

As of this writing, all applications developed using the TinyOS environment rely on a text based programming language (nesC) approach to building applications on wireless platforms and preparing communication protocols where the motes periodically or on querying, send a preformatted message format to the base station and this message is interpreted by the language or environment of the developer's choice (LabVIEW at the DIAL).

However, it was observed that a lot of development time was initially taken up by the need for training with the nesC language and the LabVIEW development environment, although, the time required adapting to LabVIEW was significantly shorter. The initial goal of the DIAL as far as LabVIEW was concerned was to develop drivers to be included in the LabVIEW Sound and Vibration Toolkit. Soon, it was apparent that this was not enough. The constantly varying nature of the TinyOS applications and the desperate need for better tools to build applications led to the consideration of a Graphical Development paradigm for TinyOS development. The

similarities in the code structure of LabVIEW and TinyOS and the apparent need and imposition of modularity of code and need for parallel processing of data and hardware management made it clear that LabVIEW would be a very viable solution.

Further, with National Instruments recently embarking on a fierce campaign to garner the hardware control market and also to promote the use of their real-time processors modules (PXI microcontrollers); the need for a robust hardware control scheme arose.

LabVIEW could be a major strength in the wireless sensor community if the code structure and the compiler used by the TinyOS community were to be emulated in a customized version of LabVIEW. With the ability of users to visualize the organization of the code and with LabVIEW's superior concurrent compilation scheme, WSN applications can be developed at rapid rates and help developers concentrate on the research aspect of development. The major push towards this would also be the fact that thanks to LabVIEW's Block Diagram – Front Panel architecture of code, simulation of implementations of code on the physical platform will be radically simpler.

As of this writing, the development phase of the new TinyOS 2.0 architecture is under way and if a LabVIEW solution were available, a majority of the developers are sure to opt for the graphical approach to application design.

CHAPTER 4

THE SOUND AND VIBRATION TOOLKIT

4.1 Introduction

This section of the thesis deals with the testing of and development of VIs and tools in NI's Sound and Vibration (SnV) Toolkit. This toolkit was designed to bring together various tools to perform standard measurements on audio, acoustic and vibration measurements. Tools for concurrent display of analyzed data on the front panel are also provided, but the majority of the VIs deal with the analysis and interpretation of data.

4.2 Toolkit Design

The design philosophy of the SnV toolkit is illustrated in the figure below. As illustrated, the SnV toolkit deals with the data once it has been acquired from the acquisition device. The toolkit provides tools to perform relative calibration of devices with drivers designed by NI. These mostly have to do with NI's range of DAQ cards and devices and non-NI products aren't currently supported.

Once the data has been accumulated, it is first sent to the Scale Voltage to Engineering Units VI which is a standard VI required for all SnV analysis operations. As the name suggests, the VI applies a standard SnV format to the input data and attaches an appropriate engineering unit (chosen by the developer) along with a

timestamp if provided. The input may also be filtered by selecting an input weighting filter provided with the VI.

The scaled data is conditioned for the required application. The signal may be integrated or frequency weighted. Limit testing on the resulting data can be used to set off condition based maintenance algorithms. A host of tools available for standard sound and vibration analysis tools can be applied to the conditioned signal. A detailed discussion on the different groups of tools is given later in this section.

Limit testing is expected to be used again in typical applications to check for anomalous data. Finally, for the display of the analyzed or received data, the SnV toolkit also provides a new addition to the LabVIEW collection of display graphs. The Waterfall display and related control VIs to determine the behavior of the display are provided.

It must be mentioned that anti-aliasing filters are not provided in this toolkit as all NI DAQ devices inherently deal with aliasing. Hence, for a custom device, the developer will have to design the anti-aliasing filters along with preparing custom drivers.

4.3 Toolkit Organization

4.3.1 Scaling

In typical data acquisition applications the signal is scaled to appropriate Engineering Units (EU) before any analysis is performed. The Sound and Vibration Library (SVL) provides the Scale Voltage to EU to scale the signal to the appropriate EU. All data sent for analysis in the SVL requires that the signal be scaled in this VI

first. Along with ensuring a uniform programming practice, it also ensures that the data to be used has been assigned to the right analysis. This is again a great help while debugging complex programs.

4.3.2 Limit Testing Analysis

The SVL Limit Testing VI can be used to analyze typical scaled signals, frequency spectrum measurements and is especially useful in determining peaks to verify operation within set limits. The limit testing VI allows for an envelope to be defined around the signal to set the upper and lower limits. The VI also allows for these limits to be adjusted dynamically to suit acceptable levels. Masking the input signal to get discontinuous areas of the signal for analysis is also possible with this VI.

If the VI is placed on the block diagram, it requires that at least one limit is specified. This ensures that the VI is utilized if placed on the block diagram and does not take up memory resources. Indicators can be automatically added from the outputs of the VI.

4.3.3 Weighting Filters

The Weighting filters are designed to filter the signal according to industrial standards to implement psophometric weighting filters for acoustic signals, especially those obtained from instrument-grade microphones.

The Weighting filters contain status flags to make sure that a weighted signal is not weighted again at different stages in the application – the VIs produce an error report in the Channel Info output.

The Weighting Filters do require that the incoming data's sampling rate be within limits set by the ISO/IEC. The signal can be tested to be within maximum frequency tolerance levels using the SnV Toolkit Maximum Frequency Within Tolerances (ANSI/IEC) VI.

The weighting filters group also contains Radiocommunications and Telecommunications Weighting filters, which handle anti-aliasing if reading from a standard NI DAQ device.

4.3.4 Integration

The SnV Toolkit provides two VIs for performing integration in the time-domain or in the frequency-domain. The principle of the definite integrals within the VIs is simple enough, however, the challenges which arise when integrating Vibration data must deal with the inherent DC component which should be first removed for analysis. This is because in most vibration testing, the DC component is erroneous as it indicates that the device under test (DUT) has a net acceleration which is generally not the case, as the DUT is usually fixed to the sensor and plugged in to the data acquisition module. For Mobile sensors however, this has to be handled differently.

The other challenge arises due to the lower-frequency sampling limits on most vibration transducers. The vibration signal therefore is very close to the DC component of the signal and hence depends upon the DC noise. The integration VIs attenuate the DC noise and this leads to erroneous results. As this is a hardware issue, care must be taken to obtain accurate measurements for low frequency measurements.

4.3.5 Vibration-Level Measurements

The SnV Toolkit provides VIs to perform RMS level measurements for accelerometer readings. Like all the VIs in the SnV toolkit, the RMS VIs can be used in Single-Shot mode after the entire signal has been collected or in continuous mode with short-time samples of data being analyzed. The RMS VI output can be interfaced directly with a display tool or analyzed further in the SnV toolkit.

4.3.6 Sound-Level Measurements

The sound level VIs can be implemented in Linear, exponential or peak-hold modes depending upon the application. The scaled signal output is wired to the appropriate Sound-Level Measurement VI and analyzed and displayed accordingly. As with all other VIs the Sound-Level Measurement VIs has extensive support in the help files of MATLAB.

4.3.7 Fractional Octave Analysis

The Fractional octave analysis is a widely used technique for analyzing audio and acoustic signals. The 1/3 and the 1/12 octave, when analyzed, in particular exhibit characteristics analogous to human ear responses.

This analysis is also required as per many industrial standards such as the ANSI and the IEC.

4.3.8 Frequency Analysis

A very common application requirement is the Fast Fourier Transform (FFT) and the SnV Toolkit provides many VIs for the implementation of the FFT for standard mode of operation, Zoom mode of operation and also, handles data over various channels. The FFT VIs provided by the SnV Toolkit also provide standard windowing formats. The averaging modes provided by the FFT VIs are also discussed extensively in the help files which accompany the SnV toolkit.

4.3.9 Transient Analysis

The Short Time Fourier Transform (STFT) VIs are provided in this grouping. These VIs generate the STFT of the input signal as functions of time provided the timestamp is input. The output of the STFT VIs can directly be sent to the Waterfall display graphs.

4.3.10 Waterfall Display

This display tool is an addition to the LabVIEW set of graphical display tools. It is a visualization technique that presents the various analyses of non-stationary signals, such as machine vibrations during run-up, braking, as well as others. This VI is soon to be replaced by an Express VI grouping in order to reduce the number of blocks needed on the block diagram.

4.4 Contributions to the SnV toolkit

4.4.1 Preliminary Discussion

As part of the study of the SnV toolkit, this thesis was also aimed at increasing the number of tools provided. The main issue encountered during the development of these VIs was the lack of standard resources to handle Matrix based operations in LabVIEW. There has been a slight upgrade to this state in the new release (LabVIEW 8), but the major concern with Control System research is the development of custom matrix handlers for standard applications.

A polymorphic set of VIs to handle matrix based mathematics were developed and implemented. The following VIs were built using these tools and have been tested to work stably.

As the VIs are to be inducted into the SnV toolkit, they have been designed to always handle time-varying inputs as arrays by default. The Scale Voltage to Engineering Unit VI collects all incoming data into an array and passes them along with a timestamp if provided. The VIs thus run in an All-at-once mode. The execution doesn't end until all the data has been processed. This leads to a certain delay in program execution, but if the data processing is handled in a separate thread, this does not influence hardware handlers. Error Handling is provided to keep up with the current updates in LabVIEW code structure. In case of an error in any part of the instrument code, the execution of all VIs is skipped until that error is handled. All the VIs are designed to be modular so as to assist future updates/upgrades. No Global variables were used so as to ensure that code transfer is simplified. All VIs are executed in reentrant mode to allow for parallel processing.

Most of these VIs are integral to the development of tools used frequently in Condition Based Maintenance scenarios. The VIs for Kalman filtering, RLS estimation, Bollinger Band analysis were designed to be implemented on the real-time PXI modules developed by National Instruments. The scenarios involving the localization of motes or mobile platforms to estimate the errors in position utilize the Kalman filter which can be customized to be a distributed algorithm to run on different motes and platforms. The RLS algorithm is useful in channel estimation applications in the WSN.

The Bollinger Band and Kurtosis VIs are used in event detection algorithms as they provide measurements which can be used to determine the triggering of rapidly changing events. They are also useful in the analysis of polled statistical data from mote networks for calibrating the sensor readings and event detection algorithms.

Most application scenarios at the DIAL involve localization algorithms and event detection to trigger processes in the mobile platforms for Condition Based Maintenance.

4.4.2 Window

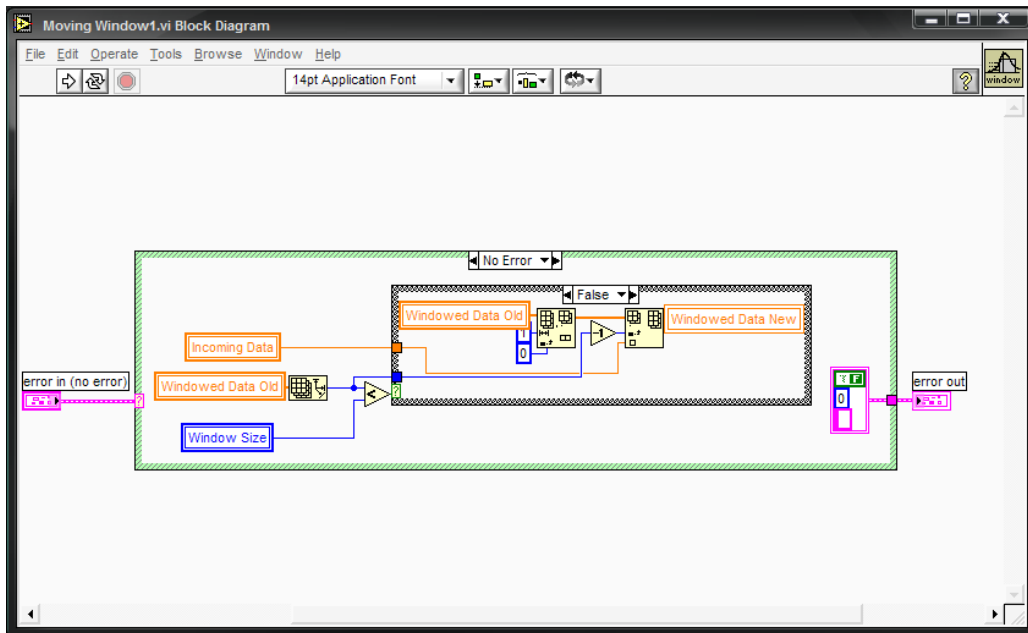


Figure 4.1 Block Diagram: Moving Window sub-VI

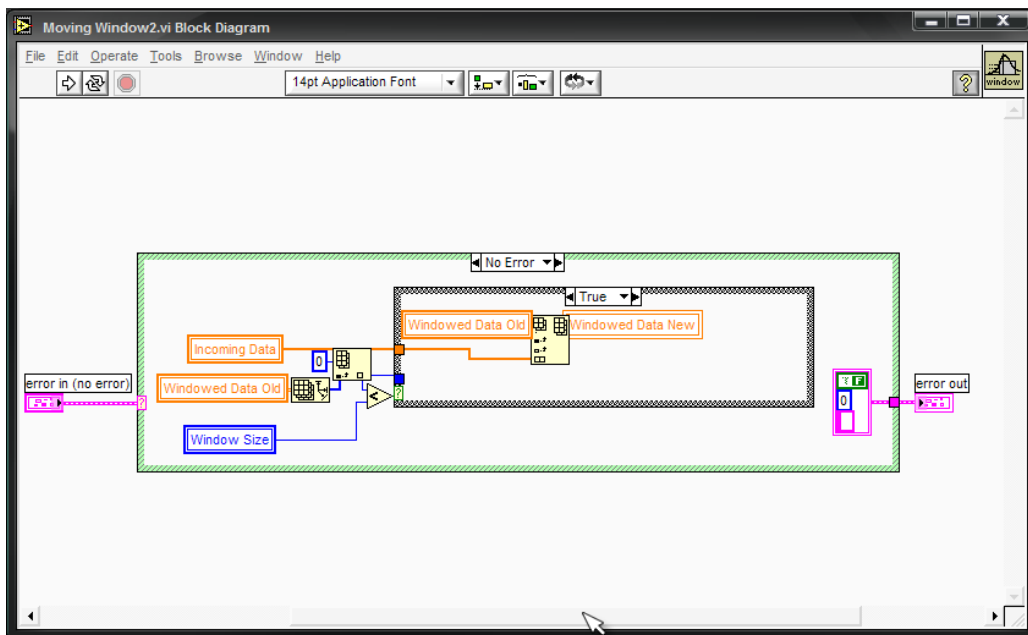


Figure 4.2 Block Diagram: Moving Window sub-VI Array Input

This is a standard VI which is again used in most SnV analysis. It is a simple VI to sample the data using a window of user specified length. This is a polymorphic VI that handles

- Scalar data in
- Array data in

The output automatically adjusts to the appropriate 1-dimensional or 2-dimensional array output.

The moving window VI is very useful as it provides a simple solution to maintaining a buffered array of data to be stored as opposed to using shift registers in conditional loops in LabVIEW. Apart from reducing clutter, it simplifies handling the length of the stored array and offers easy customization for different data types.

4.4.3 Kurtosis

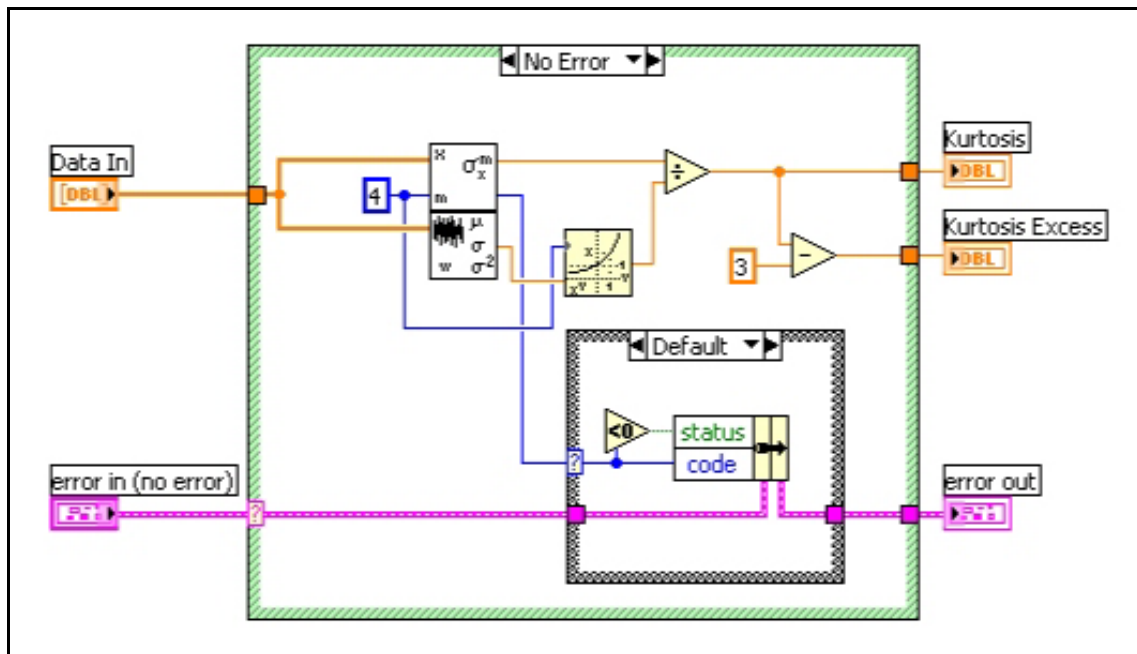


Figure 4.3 Block Diagram: Kurtosis sub-VI

The Kurtosis set of VIs are again a group of Polymorphic VIs to handle single dimensional or two dimensional data. The Kurtosis is performed across rows of the two dimensional matrix treating the various rows as channels which need to be analyzed.

The equation which governs the Kurtosis calculation is given as $\beta = \frac{\mu_4}{\mu_2^2}$ where

μ_2 and μ_4 are the second and fourth moments of the input data.

The implementation of a simple application to demonstrate the Kurtosis of a noisy sinusoidal signal is shown below.

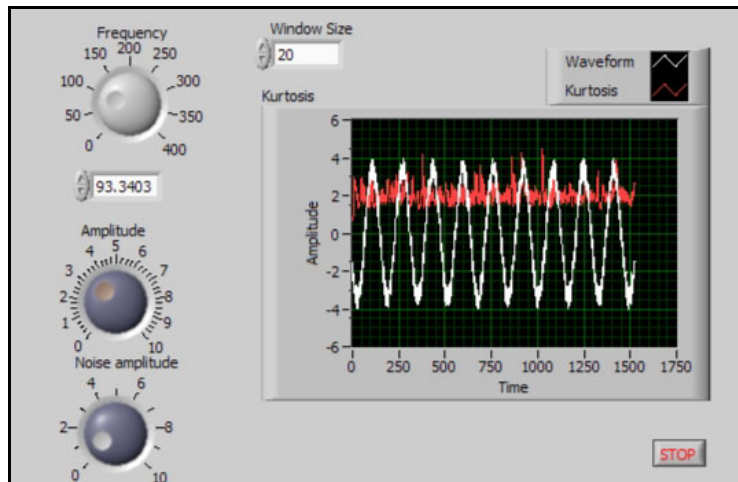


Figure 4.4 Front Panel: Kurtosis Implementation

The kurtosis VI does have an upper limit to its execution due to the exponential of the calculated standard deviation. Care must be taken not to raise the frequency above this upper limit. The limit is system specific and is usually not reached in standard applications.

The Kurtosis VI is useful in the simulation of localization algorithms and peak detection algorithms which are regularly used at the DIAL in scenarios involving localization of mobile platforms and event detection.

4.4.4 Bollinger Band Sub-VIs

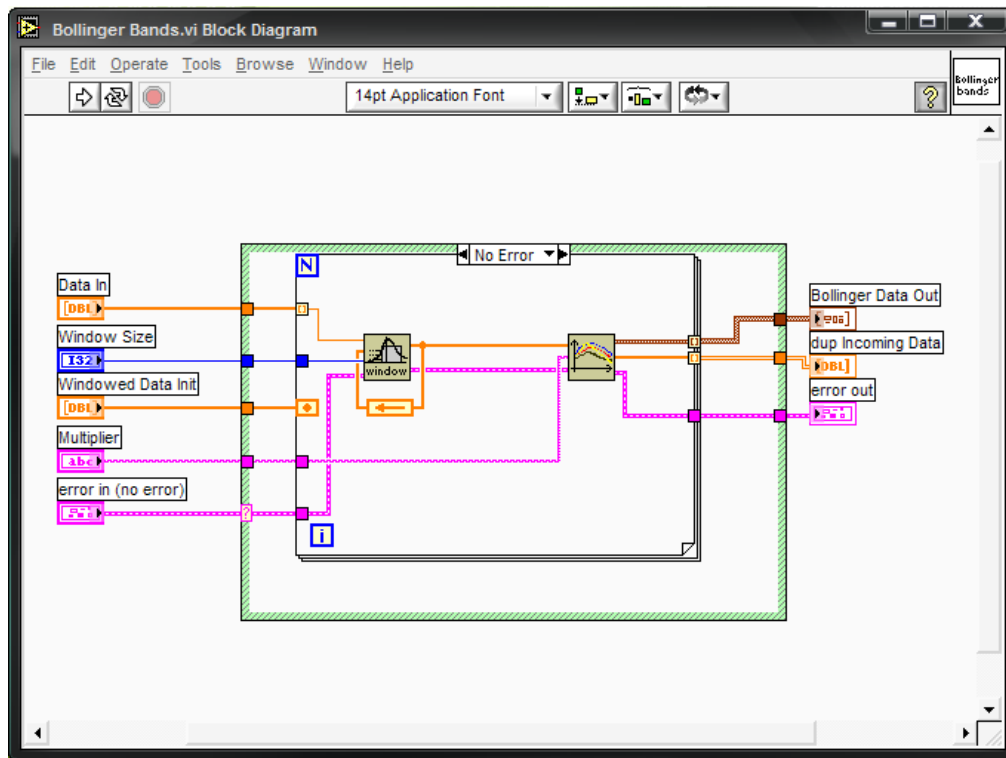


Figure 4.5 Block Diagram: Bollinger Bands

The Bollinger Band set of polymorphic Sub-VIs are developed using the moving window sub-VI and as can be observed, the output is a cluster of three arrays which contain the Bollinger Bands for the incoming data. The Windowed Data Init variable is set to initialize the band data and acts as an accumulator.

As the Bollinger Band sub-VI only takes an array input, the data is first windowed to get a user-defined series of data points which can be analyzed. The multiplier supplied for this VI is preprogrammed for three values (1, 1.5 and 2) which are the standard values in most texts and can be changed at run-time to a user defined value.

4.4.5 Kalman Filter

The Kalman Filter set of polymorphic VIs were developed using the standard Kalman Filter algorithms.

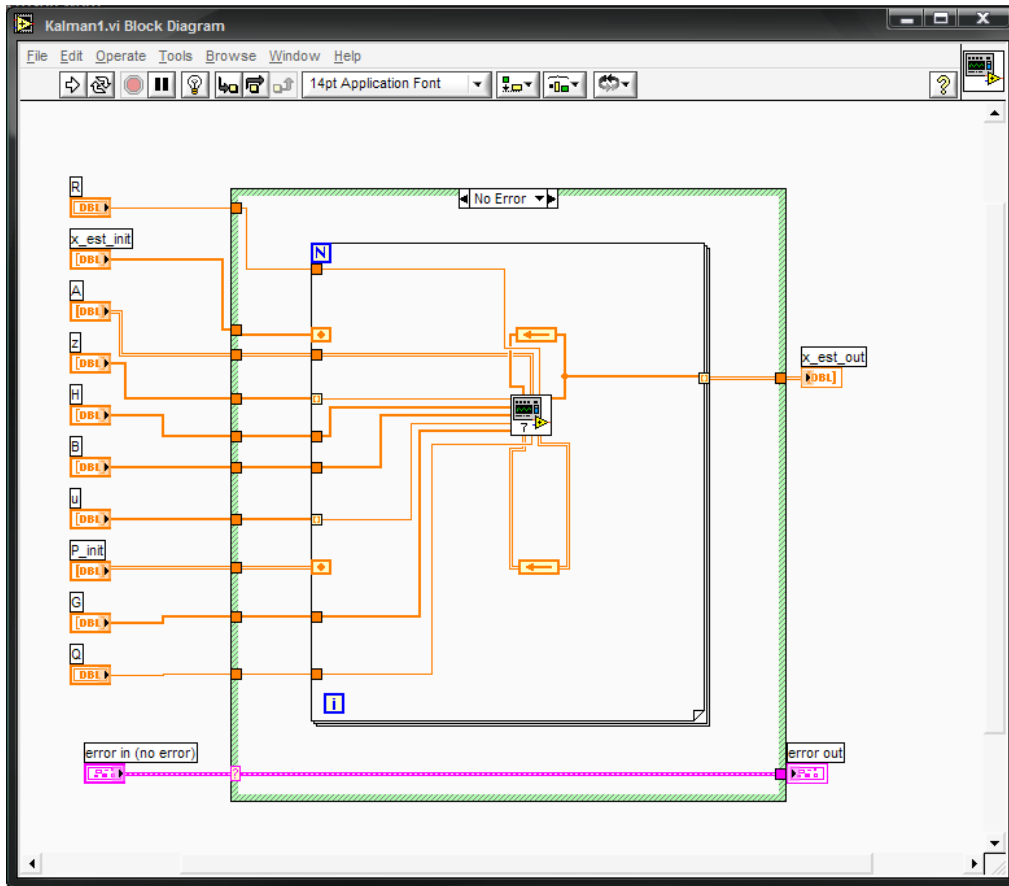


Figure 4.6 Block Diagram: Kalman Filter

The equations governing the construction of the above sub-VI are given below.

Given a system defined by the equations given below,

System model and measurement model:

$$x_{k+1} = A_k x_k + B_k u_k + G_k w_k$$

$$z_k = H_k x_k + v_k$$

$$x_0 \sim (\bar{x}_0, P_{x_0}), \quad w_k \sim (0, Q_k) \quad v_k \sim (0, R_k)$$

Time update:

error covariance:
$$P_{k+1}^- = A_k P_k A_k^T + G_k Q_k G_k^T$$

estimate:
$$\hat{x}_{k+1}^- = A_k \hat{x}_k + B_k u_k$$

Measurement Update:

error covariance:
$$P_{k+1} = \left[(P_{k+1}^-)^{-1} + H_{k+1}^T R_{k+1}^{-1} H_{k+1} \right]^{-1}$$

estimate:
$$\hat{x}_{k+1} = \hat{x}_{k+1}^- + P_{k+1} H_{k+1}^T R_{k+1}^{-1} (z_{k+1} - H_{k+1} \hat{x}_{k+1}^-)$$

It is assumed that the P and initial estimate matrices are initialized. The VI recursively estimates the states while executing the following sub-VI:

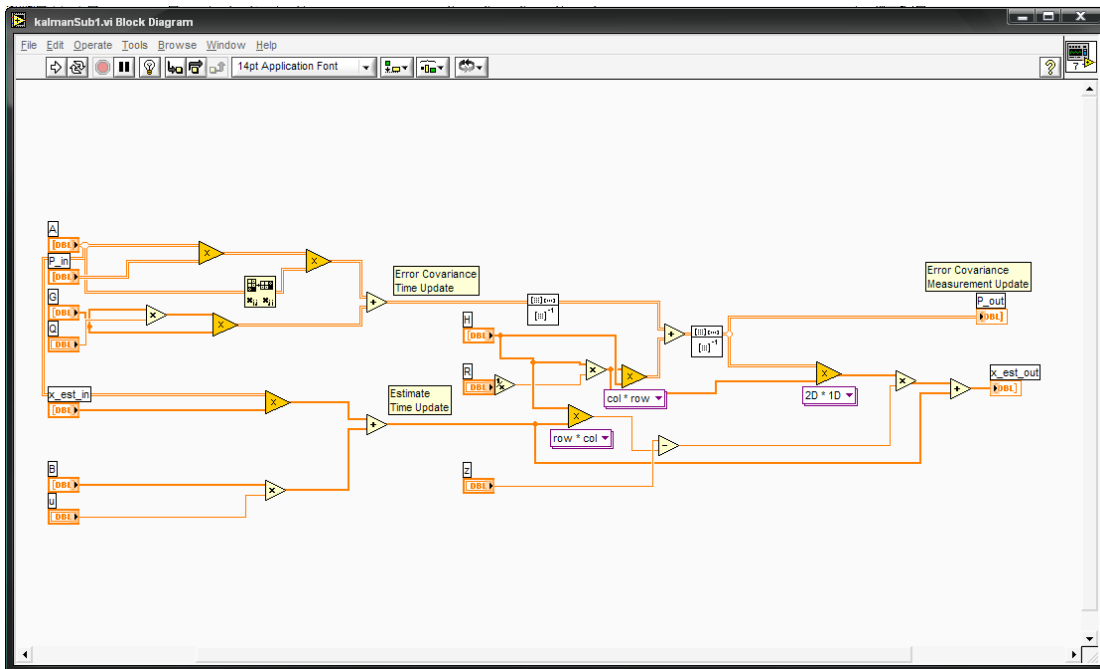


Figure 4.7 Block Diagram: Kalman Filter sub-VI implemented using the custom matrix VIs

The custom matrix handler VIs can be seen here according to the dimensions of their inputs. The polymorphic VI can be seen to adapt to the input data format and resize according to the dimensions of the input arrays. These VIs were tested using

standard Kalman Filtering problems encountered in standard texts and are observed to work faultlessly. The H-matrix however is assumed to remain static and the VI cannot analyze a time-varying system matrix.

4.4.6 RLS Estimator

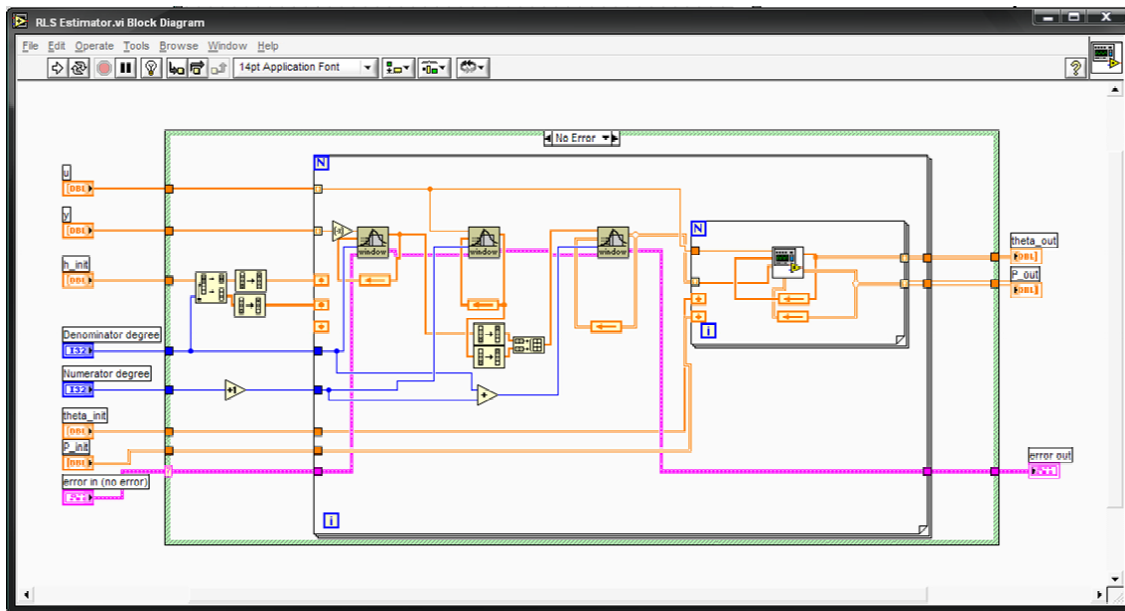


Figure 4.8 Block Diagram: RLS Estimator

This sub-VI again implements the Window and matrix VIs developed for the SnV toolkit. This is a single shot estimator which needs the entire data table to be known before processing. The “h” matrix needs to be initialized and so do the theta and “P” matrices. The degrees of the numerator and denominator polynomials need to be specified.

The equations of the system and the estimator which govern the working of the RLS estimator sub-VIs are given below.

System model

$$x_{k+1} = Ax_k + Bu_k, \quad x_0$$

$$y_k = Hx_k$$

RLS Algorithm

$$P_{k+1} = P_k - P_k h_{k+1} (h_{k+1}^T P_k h_{k+1} + \sigma^2)^{-1} h_{k+1}^T P_k$$

$$\hat{\theta}_{k+1} = \hat{\theta}_k + P_{k+1} \frac{h_{k+1}}{\sigma_{k+1}^2} (y_{k+1} - h_{k+1}^T \hat{\theta}_k)$$

This VI was also tested and seen to match results obtained from other methods.

4.5 Conclusions

The Sound and Vibration toolkit, the VIs developed to be added to the SnV Toolkit, along with the many VIs developed for the TinyOS toolkit are essential tools for developers of Wireless Sensor Networks. Standardized data analysis schemes make sure that application development and maintenance is less error-prone.

CHAPTER 5

THE DIAL

5.1 Introduction

The Distributed Intelligence and Autonomy Laboratory (DIAL) established at the ARRI of UT Arlington is an innovative test-bed aimed at studying the integration of mobile platforms and wireless sensor networks. The laboratory aims at developing a dynamic environment which is monitored using a mobile wireless sensor network implemented using off-the-shelf robotic platforms and the wireless sensor network research platforms distributed by Crossbow Technologies. Apart from the Crossbow platforms discussed in Chapter 2, the test-bed includes Acroname Inc. Garcia© robots which are highly flexible commercial platforms provided with Brainstem microcontrollers and infrared sensors for obstacle avoidance. Cybermotion sentry robots donated to the ARRI by JC Penney are also available. A LabVIEW based application to control and monitor this test-bed was developed as part of this thesis.

5.2 TinyOS Toolkit

With the increasing demand for rapid development of applications for TinyOS, the DIAL has been developing tools to streamline the process by designing a toolkit to be integrated into LabVIEW. This toolkit aims at providing tools to simplify code generation for TinyOS applications and bypassing the Cygwin environment which is

required for programming XBow motes with TinyOS applications. The contributions of this thesis to the TinyOS toolkit are listed below.

5.2.1 Active Message (AM) Resources

The standard data transfer format in TinyOS is the AM message format. It forms the template for all messages sent to and received from the TinyOS network. The AM message format has a fixed type of synch bytes which start and end all messages. However, the length of the message is not always constant. This necessitates the design of interpreter tools to extract the message packet and decipher the message specified in the format determined while programming the motes.

5.2.2 GetAMPacket.vi

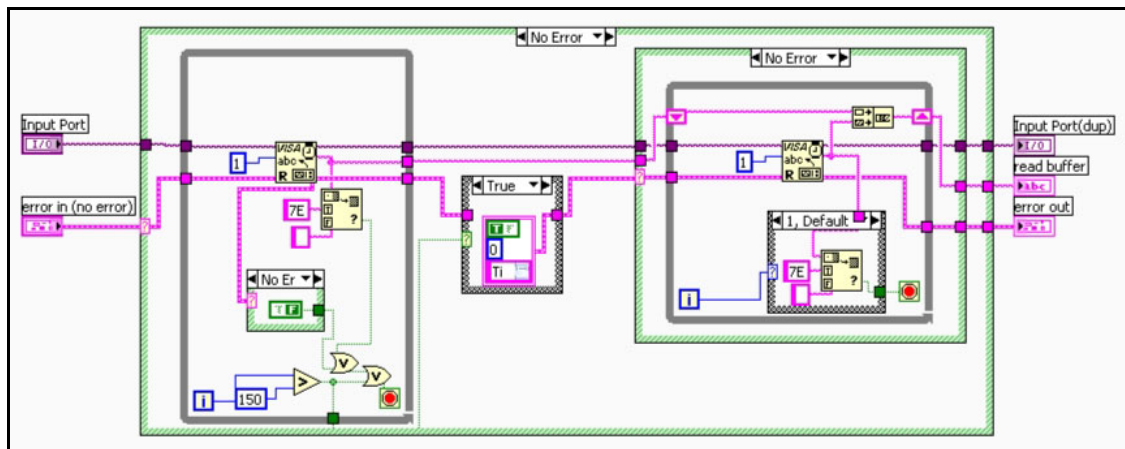


Figure 5.1 Block Diagram: GetAMPacket.vi retrieves the next available TOS AM packet

This VI is built on the standard template of all the VIs developed so far. The error in input, while helping the developer set the data flow in the top level application, also ensures that the VI will not execute as long as the error exists or until the error is

handled. The VI reads from the serial port one byte at a time and discards bytes which do not correspond to the synch byte. If the synch byte is encountered, or a time out, set at 150 iterations – indicating that 150 bytes have been read without encountering a synch byte, is not triggered, it clears the error flag and moves to the second stage of the VI. In this stage, the VI reads the next available byte – whatever it may be – stores and ignores it and continues to read the bytes until the next synch byte is reached. This ensures that at least one full message is read.

If this VI is run iteratively, it ensures that the messages read are in perfect synch and that relevant messages are read every time.

5.2.3 GetTOSPacket.vi

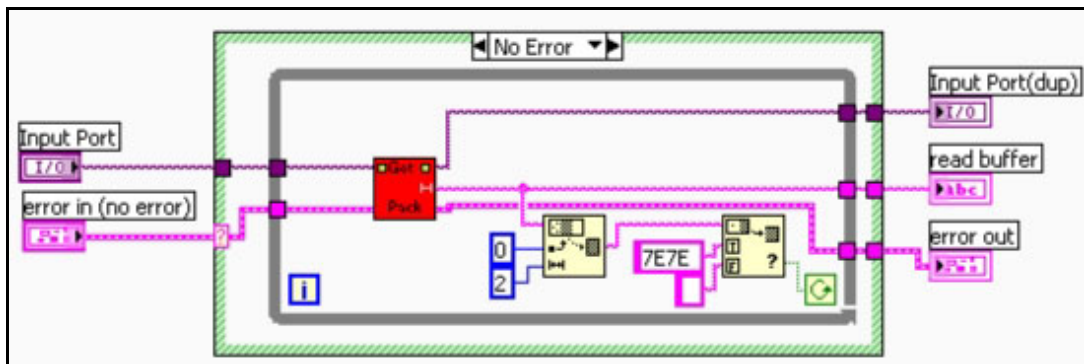


Figure 5.2 Block Diagram: GetTOSPacket.vi - Synchronizes packet retrieval

This VI implements the GetPacket.vi sub-VI. This VI has been used liberally in all the top level applications dealing with reading data from the TinyOS networks at the DIAL. It has been tested and observed to work flawlessly.

As has been seen from the working of GetPacket.vi, the output of the sub-VI might have an erroneous message packet in it. This case arises when the serial port has been out of synch and the sub-VI has synchronized the packet retrieval. Due to the sub-

VI ignoring the first packet after retrieving the synch byte, it does not rule out the case where the ignored byte might be a synch byte indicating the beginning of a message. So, the GetTOSPacket.vi checks for the presence of two synch packets at the beginning of the retrieved message and if true, discards it. Alternatively, a second version of this VI simple deletes the first synch byte and retains the message. This VI also does not execute if an error is encountered in this VI's caller.

5.2.4 DecipherTOSPacket.vi

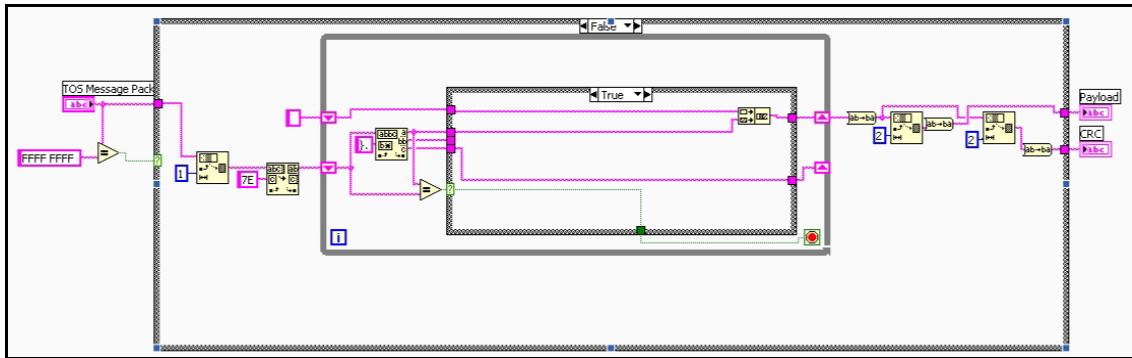


Figure 5.3 Block Diagram: DecipherTOSPacket.vi Unescapes incoming Data and separates the payload and the CRC bytes

This VI is always called after the execution of the GetTOSPacket.vi or the GetPacket.vi. This VI is used to trim the message of the synch bytes which wrap the message and “unescape” the data received which might be escaped. Escaped data occurs when the message contains bytes which correspond to standard message bytes (like the synch byte or group ID bytes). The offending byte is exclusive-or’ed with a predetermined value and suffixed to an escape byte. Whenever the VI encounters an escape byte, it discards it and exclusive or’s the next byte with the appropriate value and continues execution. It must be noted here that the CRC bytes which occupy the last

two bytes of all messages will also be escaped if they correspond to the standard formats. The deciphered packet and the retrieved CRC are sent back to the calling application.

This VI is implemented in every application which requires the interpretation of the data received in CRC packet communication with the MICA series of motes.

5.2.5 GetCricketPacket.vi

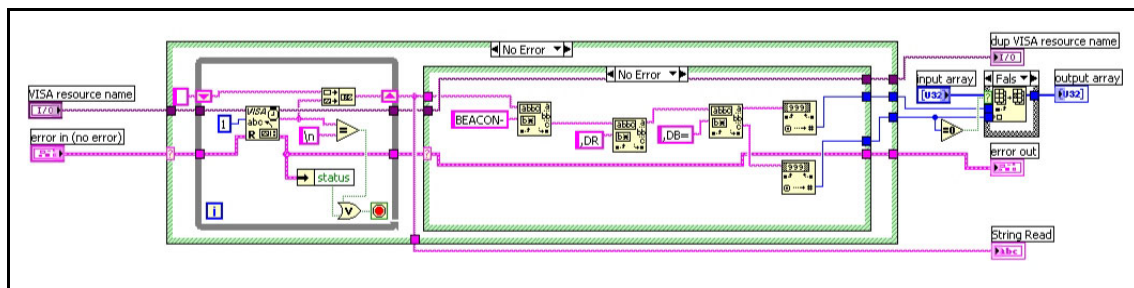


Figure 5.4 Block Diagram: GetCricketPacket.vi - Retrieves the next available Cricket message and retrieves the distance information and beacon ID

This VI is similar to the GetAMPacket.vi except that it is used to interpret data retrieved from the Cricket series of motes. The Cricket message ends with the carriage return (or the line feed) and this is used as the synch byte. The VI reads bytes – one at a time – until the new line character is received and then retrieves the rest of the message up to the next new line character. As the applications dealing with the cricket motes at the DIAL deal with the distance packets only, the distance section of the message is retrieved and the value is returned to the caller.

5.2.6 TinyDB Resources

Apart from the AM type and Cricket type of messages, TinyDB messages have also been supported in this toolkit. This collection of VIs used to build TinyDB query messages (in the AM message format) and the VIs used to retrieve the required to extract the appropriate parameter received from the network.

5.2.7 InitMote.vi

The InitMote set of VIs are used to initialize the platform under use. All platforms under test at the DIAL have diverse requirements as far as the serial port communication protocols are concerned. This series of VIs helps in automating the process by initializing the serial port to the appropriate value and sending appropriate handshake messages if necessary. If the platform returns a positive status message, these VIs do not return an error ensuring that error handling is present at all stages of the application.

The initialization VIs were developed for the Cricket, MICA2, MICA and the MICA2 motes running TinyDB. This template can be used to initialize other platforms as and when acquired. The initialization VIs are used in every application involving any of the Cricket, MICA series or the MicroStrain gauges.

5.3 Implementation of Toolkit VIs

With these resources ready to implement, the DIAL has been able to build applications implementing the TinyOS network. Some of them are described below.

5.3.1 MicroStrain Acquire

This VI is an implementation of the Sound and Vibration (SnV) toolkit on an existing application developed at the DIAL.

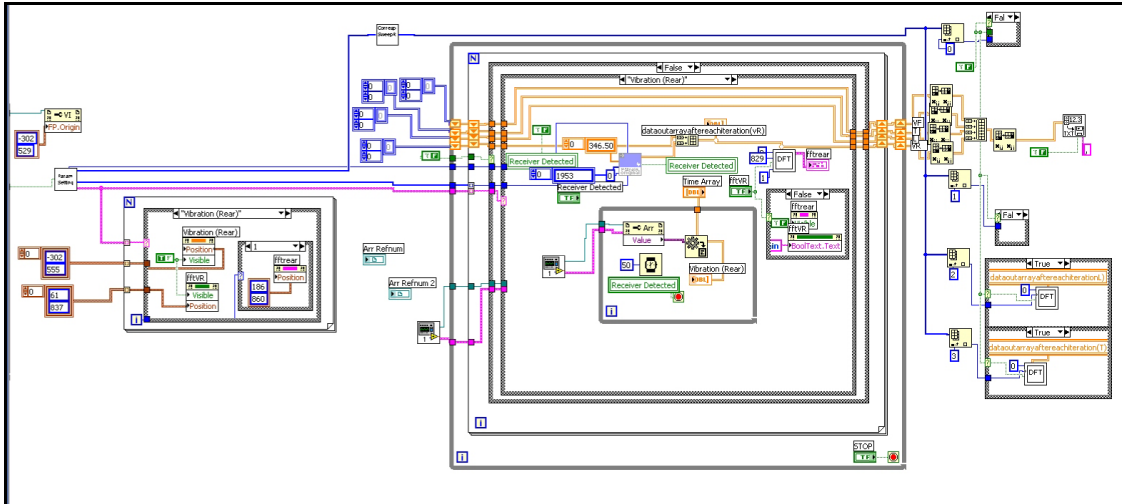


Figure 5.5 Block Diagram: MicroStrain Acquire: The original LabVIEW VI to handle communication and analysis and display

This figure shows the VI developed to use the MicroStrain G-Link accelerometer to monitor the activity on an air conditioning system at the ARRI for Condition Based Maintenance. The application runs the G-Link in periodic mode and sends request messages to the node to retrieve stored measurement values.

The first stage of implementation of the SnV toolkit to this application was to design drivers to interface the Scale Voltage to Engineering unit VI for the node. This also involved the trimming of deprecated VIs which had been removed from the LabVIEW suite of VIs. The drivers designed set the channels to retrieve messages from, the sampling period and other status values which are written into standard registers at initialization. Message retrieval drivers were then designed for sending appropriate messages to the nodes to trigger transfer of data. Once the transfer of data is complete,

the formatted data is sent back to the main application. In the main application, the retrieved data is scaled in the SVT VIs and the FFT is performed.

The process is repeated until the user input to stop the application is received. The final code (with the initialization VI trimmed due to lack of space) is shown below.

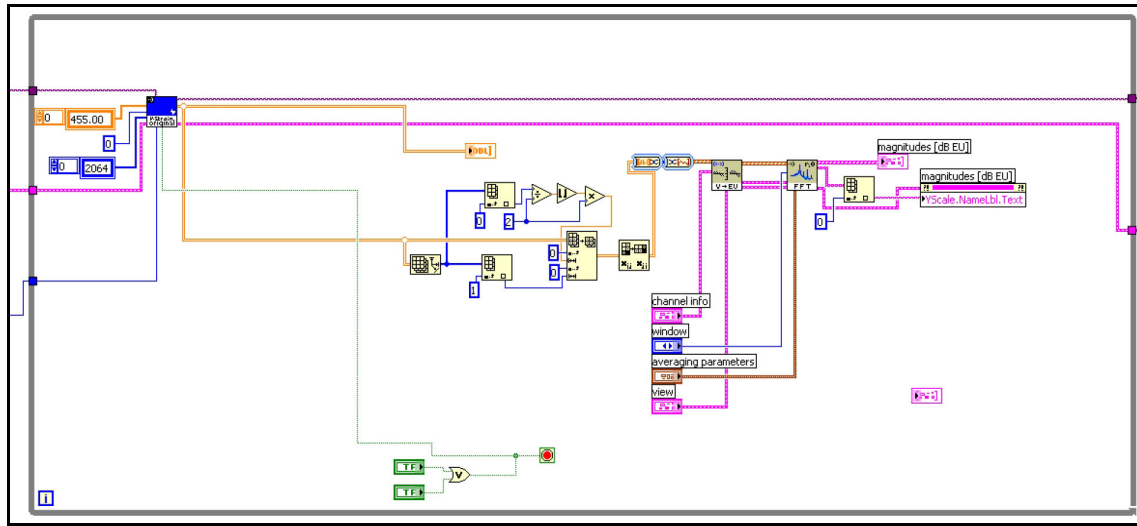


Figure 5.6 Block Diagram: MicroStrain Acquire modified with the SnV toolkit VIs and MicroStrain Drivers

It was observed that with the implementation of the new drivers for the node and the implementation of the SVT, the response of the node was greatly improved and the stability of the program increased. The front panel with some vibration results is shown in the figures below (This figure also demonstrates the various functions provided by the FFT VI in the SnV Toolkit).

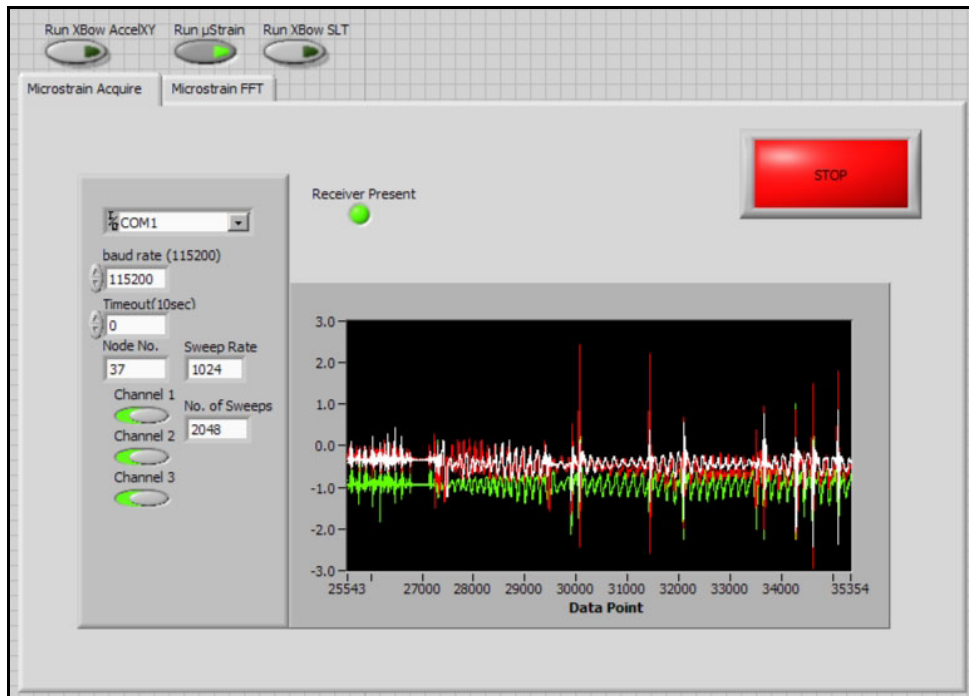


Figure 5.7 Front Panel: MicroStrain Acquire shows a particular vibration sample

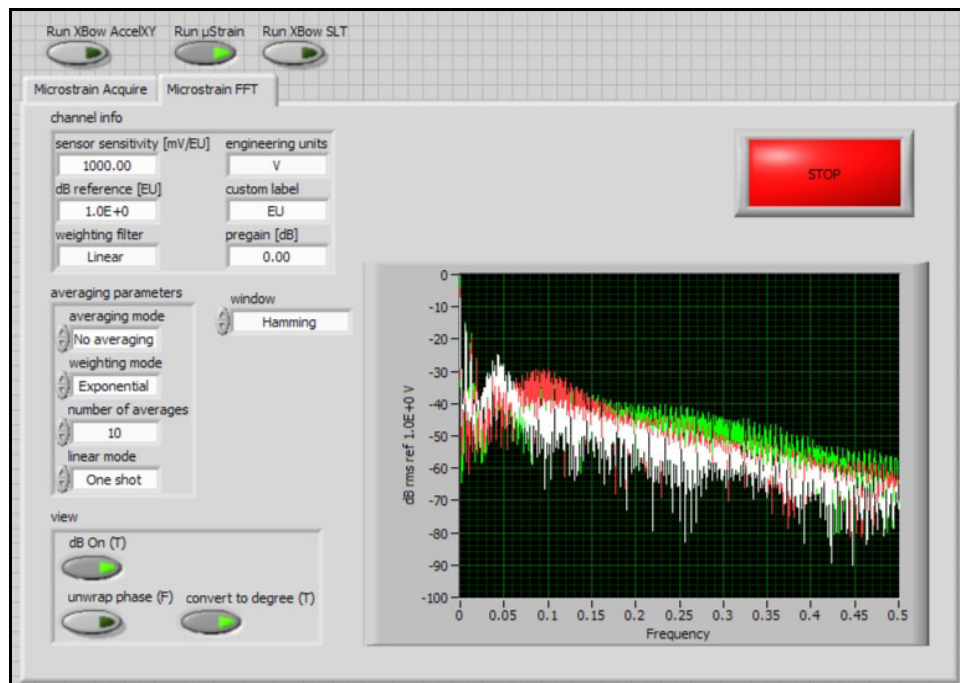


Figure 5.8 Front Panel: MicroStrain Acquire shows the FFT analysis of the measured vibrations on three channels

5.3.2 XBow Acquire

The next stage in this application was to extend it to interface the XBow mote network.

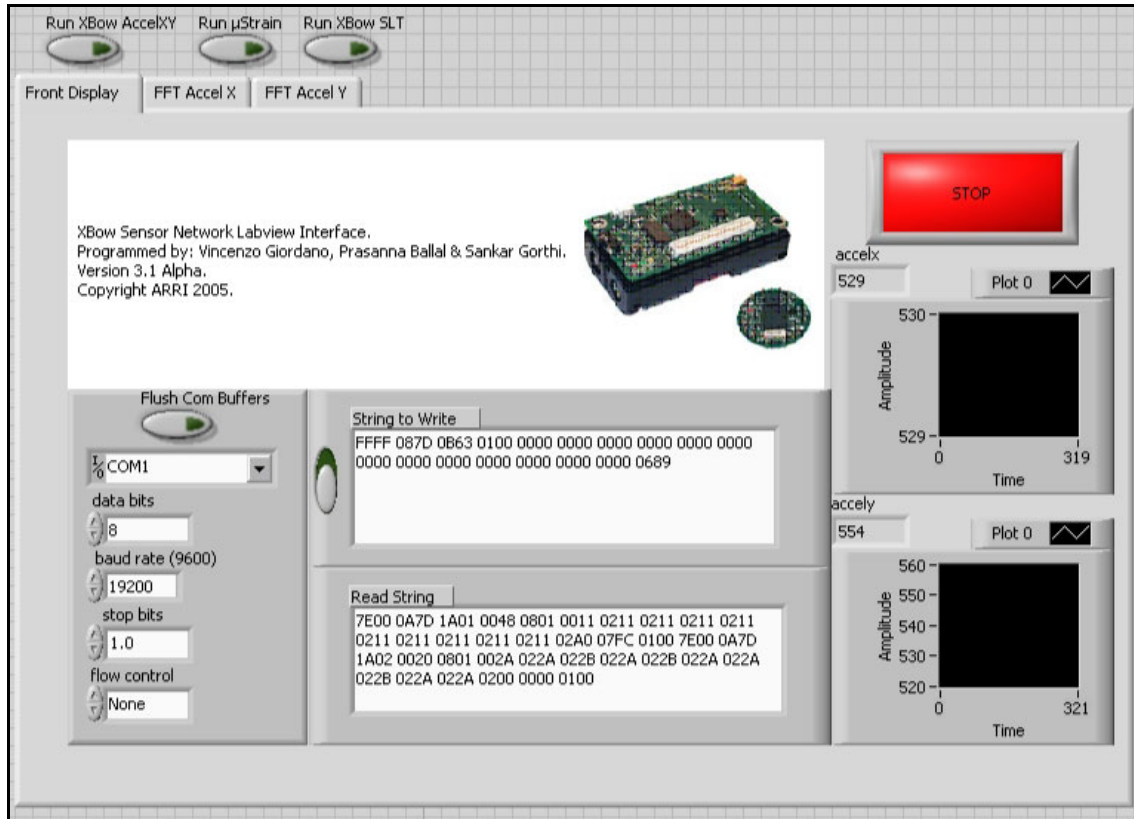


Figure 5.9 Front Panel: XBow Acquire shows the extension to implement Crossbow Mica2 drivers

The program structure of the VI is fairly similar to the MicroStrain Acquire portion of the code with the only divergence being in the interpretation of the data retrieved from XBow motes. The platform was also tested for the values of the Sound, Light and Temperature measurements.

5.3.3 Statistical Processing and Analysis Module (SPAM)

As a departure from the WSN data retrieval, the idiosyncratically named SPAM application was developed to test the robustness of a stand alone application which had implemented the various VIs that had been developed for the SnV toolkit. This application was idiosyncratically named SPAM but is a sophisticated Web Crawler used to poll data from a federal government public website. The application retrieves data relating to the values of various currencies as ratios with respect to the US Dollar from a series of websites, collects them, discards values which indicate national holidays and plots the values along with a Bollinger band analysis of the data. The Bollinger band is customizable – the window size and the Bollinger multiplier may be specified by the user while the application is being executed.

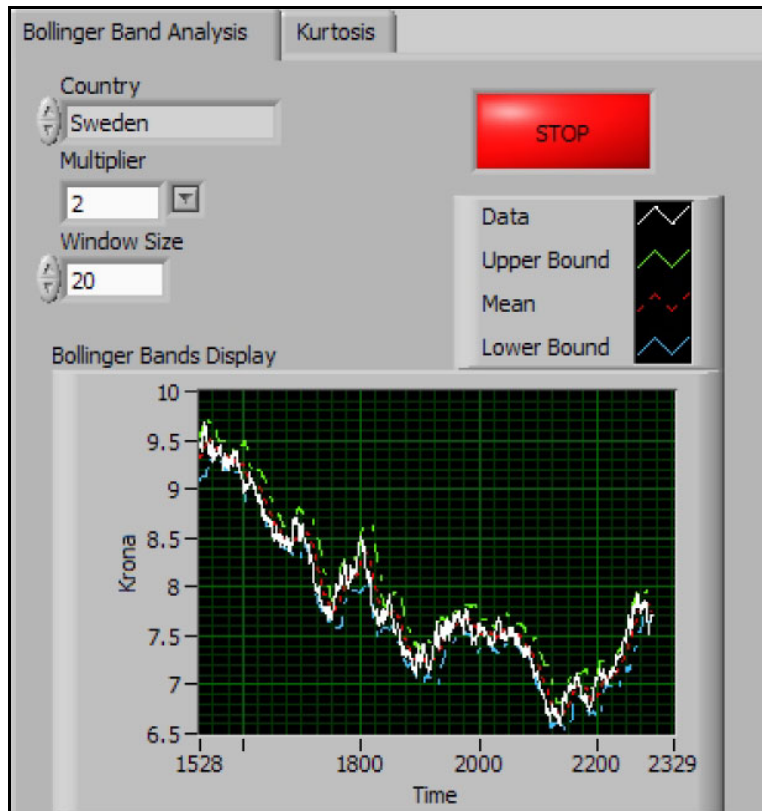


Figure 5.10 Front Panel: Bollinger Band Analysis implementation in the Currency Analyzer showing the currency analysis over about 10 years with the Bollinger bands shown

The Kurtosis analysis of the data retrieved is also provided. This application has been tested over long periods of time and has been seen to be error-free and accurate. This application has helped to increase confidence in bulky standalone applications developed on the LabVIEW environment.

5.3.4 Cricket Demonstration

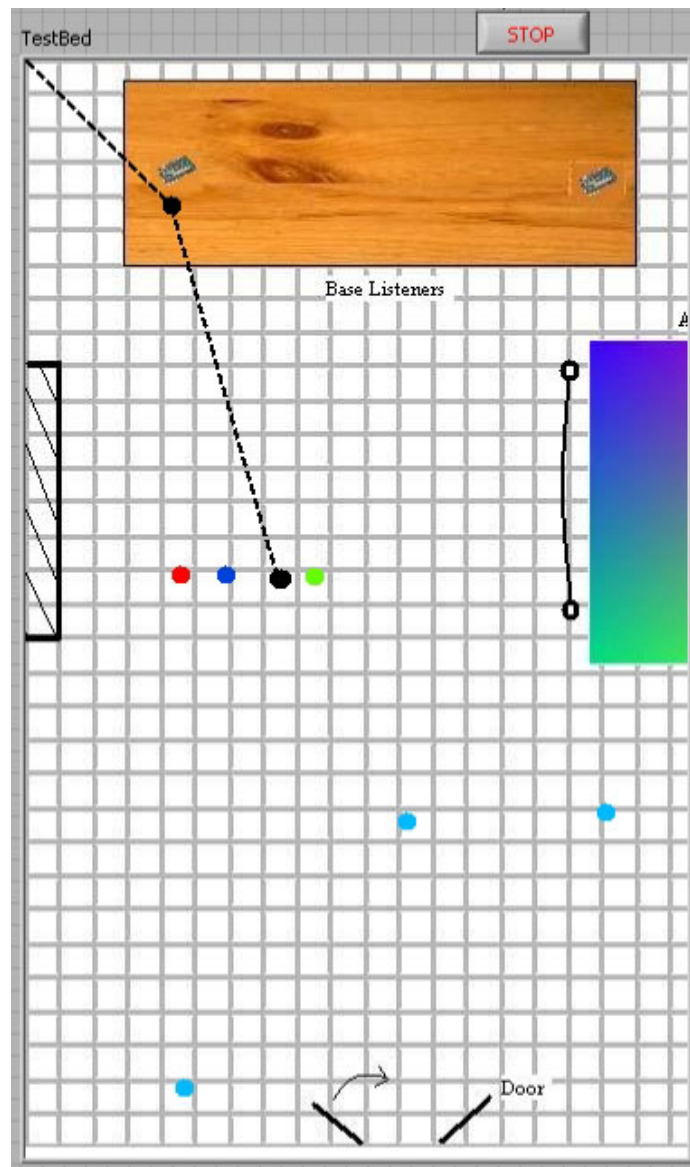


Figure 5.11 Front Panel: Cricket Driver implementation in a simple warehouse decision making scenario

An application to test the functioning and the accuracy of the Cricket series of motes was developed using the resources built for the TinyOS toolkit. A section of the Front Panel of the application is shown above.

The application featured a combination of two networked series of nodes controlled from a common base station. The two networks consisted of mobile cricket motes and the stationary XBow MICA2 sensors. The MICA2s (the blue dots on the front panel) were programmed to run an environment testing application. The motes retrieve the light readings at each mote and send the data back to the base station. The navigator (a Cricket mote represented by the black dot) starts at the bottom of the map, moves through the sensor field and the MICA2s guide the navigator through safe areas. The application was built using three MICA2s but can be extended to include any number of multihop linked motes.

The navigator – once safely through the sensor field stage – reaches the moving system (three localized Cricket motes represented by the red, blue and green dots). This stage represents a dynamic system which the navigator must traverse.

The application was also intended to test the communication between two disjointed networks and was observed to be a very stable application determined only by the line-of-sight contact between the Cricket Beacons and Listeners which are used for localization of the moving motes.

The code behind the application is interesting to note as it emulates the concurrent data acquisition and analysis paradigm implemented in TinyOS.

The first figure shows the data acquisition stage of the application. As can be observed, the three processes are separated and the retrieval of data from each of the two Cricket listener beacons and the MICA2 base station is concurrent.

The second figure shows the interpretation of the data retrieved from the Cricket motes. The id of each mote is checked and the required data of the motes under

consideration are retrieved. Then the distances of the motes with respect to the base station are measured and stored in local variable linking to the storage variable.

The third figure shows the analysis stage for the MICA2 where the light reading at each mote is retrieved and the distance of the mote with respect to the navigator mote is calculated. Using these measurements, the decision making block shows an appropriate message on the front panel.

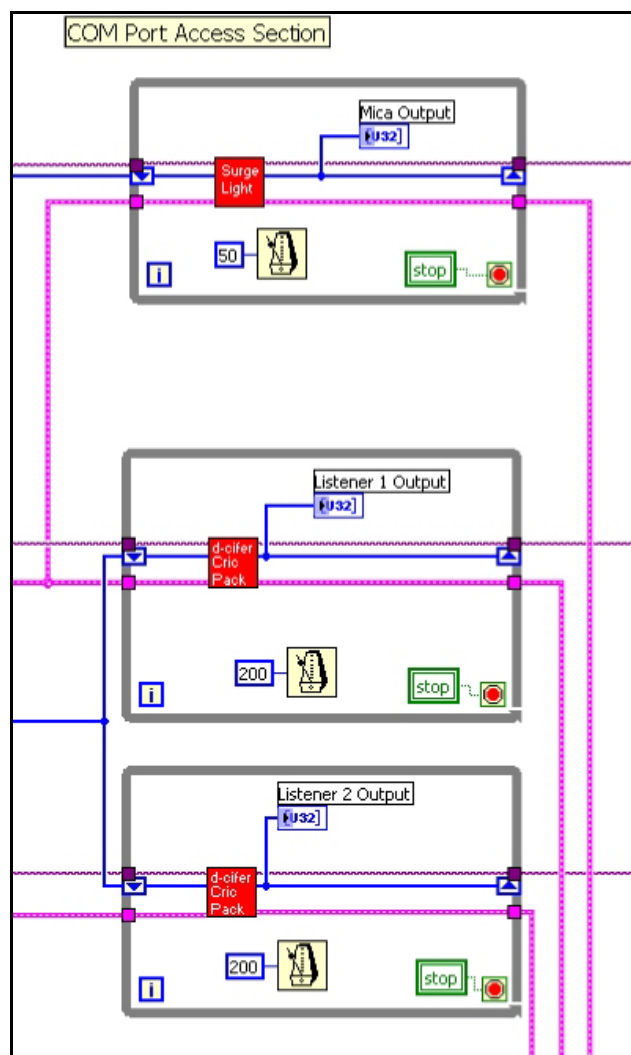


Figure 5.12 Block Diagram: Cricket Demo showing the Serial communication "module"

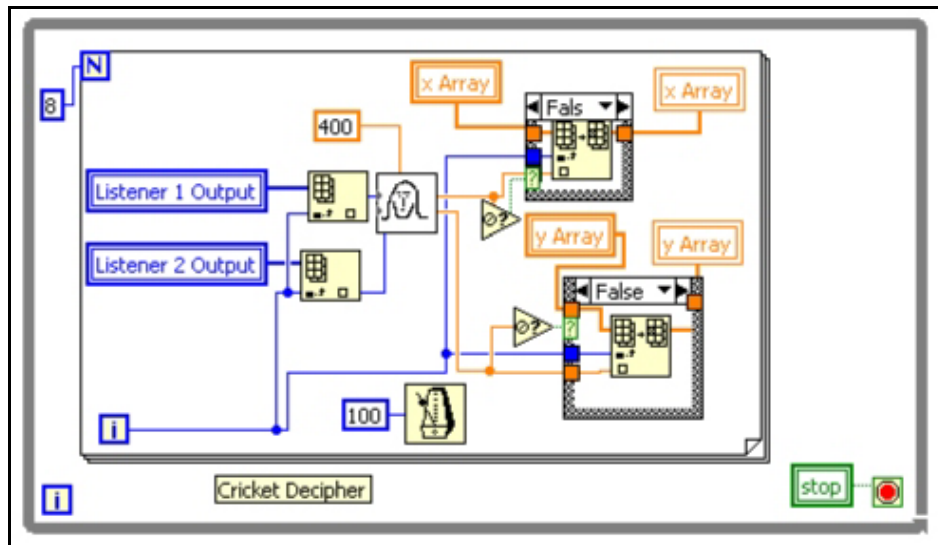


Figure 5.13 Block Diagram: Cricket Demo showing the Cricket distance calculation "module" where the readings from the two listeners is used to triangulate the coordinates of the mobile platform

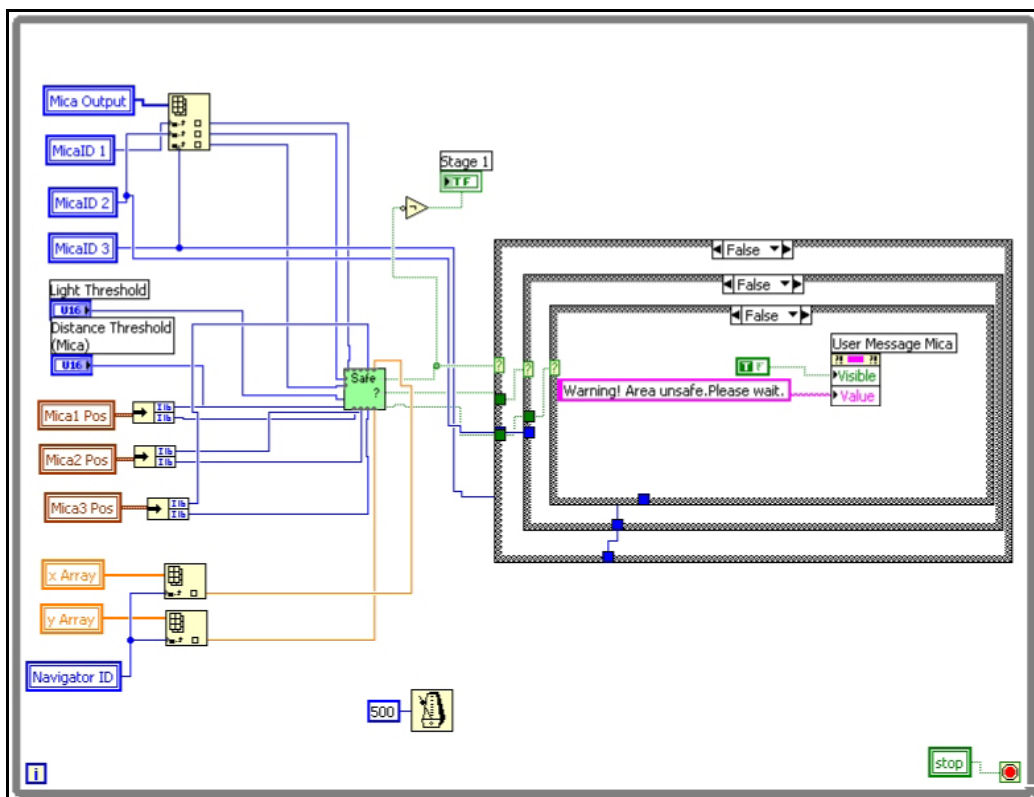


Figure 5.14 Block Diagram: Cricket Demo showing the Mica decision making "module" to guide the operator through safe zones in the network

The next figure below shows the code for the Cricket decision module and the Graphical display module.

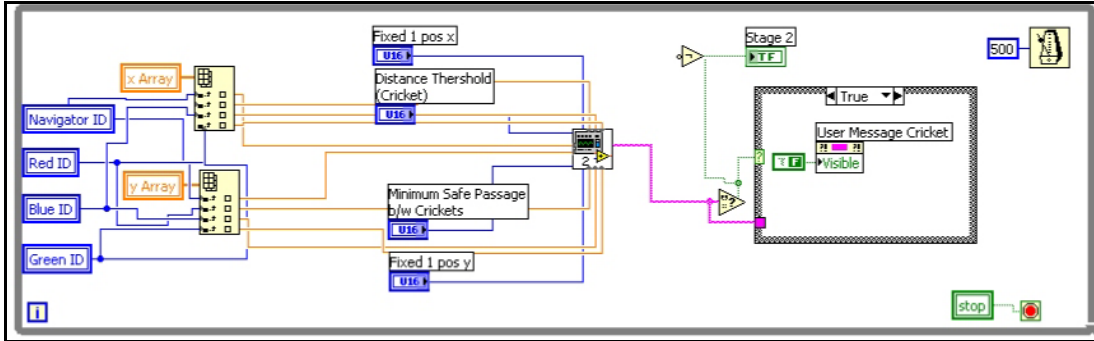


Figure 5.15 Block Diagram: Cricket Demo showing the Cricket Decision making "module" to guide the operator through the moving obstacle stage of the application

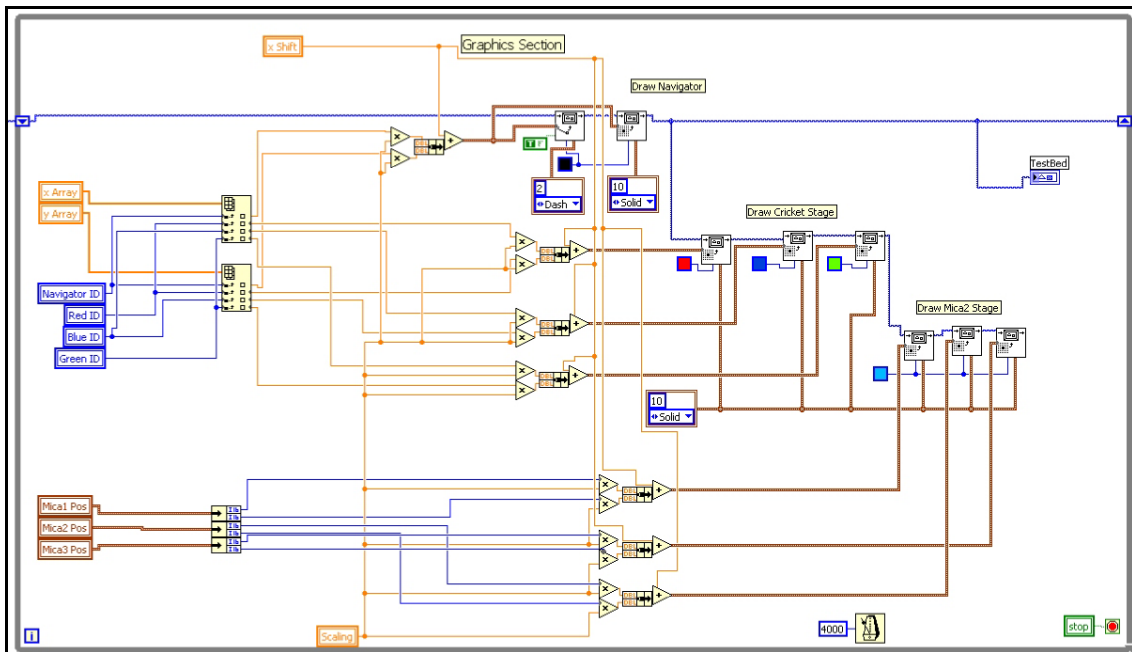


Figure 5.16 Block Diagram: Cricket Demo showing the position mapping and graphic handler "module"

This division of the code structure into modular sections all running concurrently, along with providing a great deal of stability to the application, improves performance and reliability. Debugging was also simplified as the developer only has to

deal with certain sections of the code the errors in which become apparent during execution.

5.3.5 TinyDB Application

The final and most recent addition to the TinyOS toolkit suit of VIs was the implementation of a simple query processor like interface to address particular motes in a TinyOS network. The TinyDB application sends SQL like queries to the base station. The base station running a nesC application interprets the query and configures the network to behave accordingly. The code behind the query builder portion of the application is shown below.

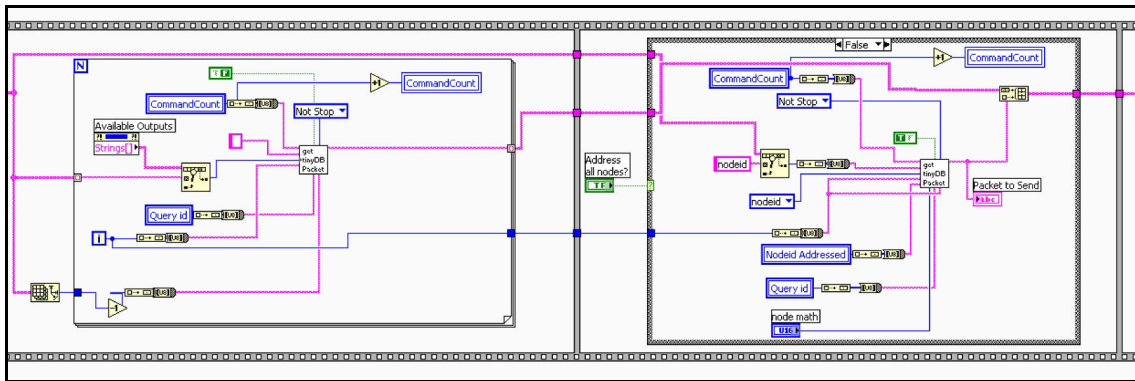


Figure 5.17 Block Diagram: TinyDB Query Builder

The generation of the query to be sent to the base station in itself is, however, not trivial. The many sub-VIs required for the implementation of the queries will be provided in the Appendices. The QueryBuilder.vi sub-VI takes an input of a set of strings representing the parameters to be retrieved like the ids of parent nodes (in a multihop network), the current node id and the corresponding light measurement at that node. Many such permutations and combinations of the data that can be retrieved from the motes are available. However, there is an upper limit to the number of parameters

which can be retrieved at a time and the application was built implementing only the parent id and temperature and light measurements. However, the application may be easily expanded to retrieve battery level readings, accelerometer, magnetometer measurements etc by adding the appropriate string to the query list.

The results from this application while querying the Light readings from a series of four motes deployed in a target area showing the light reading levels at each point are shown below.

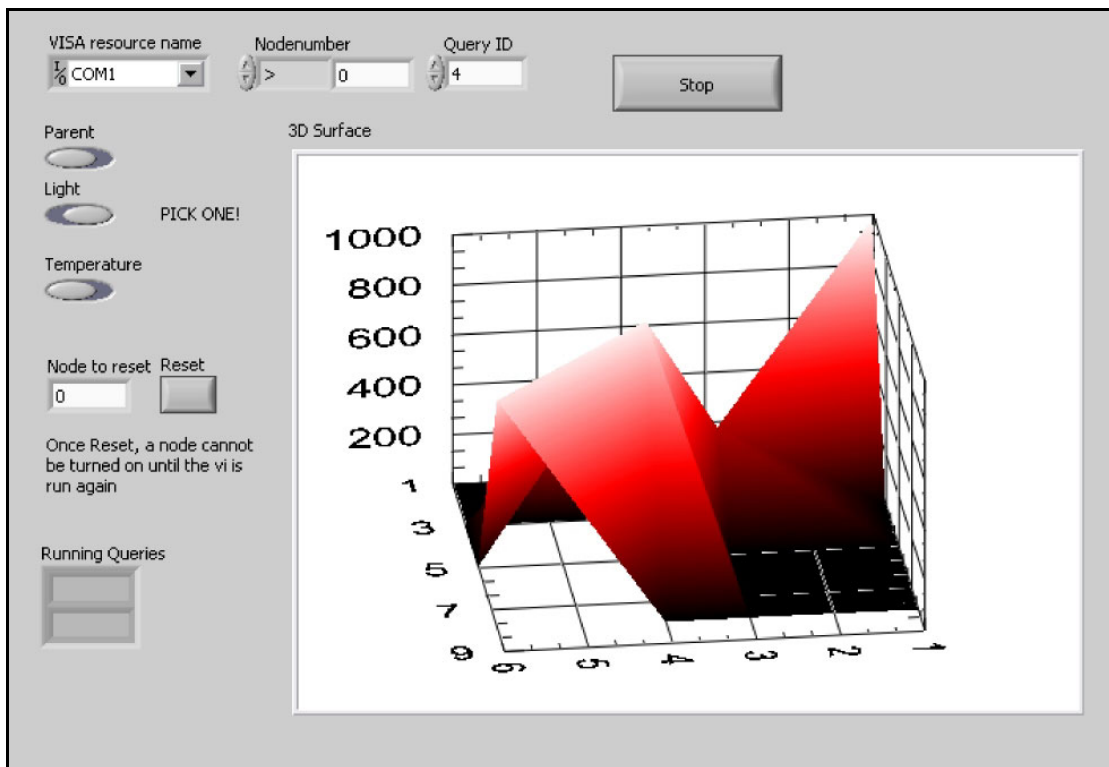


Figure 5.18 Front Panel: TinyDB Application showing the temperature distribution at four nodes

It can be observed that the light reading at coordinate (3, 3) has fallen below the rest of the motes and implies that the mote at that position is either in the dark or is malfunctioning. The query here has been sent to retrieve the light readings of all motes with node ids greater than zero (top of the front panel). Incidentally, the query can be

structured to retrieve values from motes of particular mote IDs by selecting the equal to option when sending the query.

5.4 Implementation in the DIAL

The tools described in the previous sections are integral to all of the applications involving the MICA series of motes or the Cricket series. These tools simplify interaction with the mote platforms and quickly building applications involving these platforms.

The applications developed are also very useful in demonstrating the power and customizability of the LabVIEW platform for WSN application building. The applications in themselves are extremely useful in test-bed monitoring and as examples of decision making systems. They also serve as templates for applications developed in the DIAL and as the basic building blocks themselves are customizable, they allow for fluid application design.

5.5 Conclusions

The various applications built for the development of the DIAL were implemented and tested for robustness. Apart from hardware issues and battery failures, all the applications developed were stable and repeatable endlessly.

The VIs built to be inducted into the TinyOS Toolkit were tested in various applications and have been observed to streamline application building for Wireless Sensor Networks.

CHAPTER 6

CONCLUSION

6.1 Summary

As has been the common theme in almost every section of the thesis, the need for modular structuring of the applications built to be implemented on either the WSN platforms or PC based applications for human interfaces to best utilize limited resources and to build robust applications is apparent. This is a foregone conclusion in most application structures. However, most application builders rely on text-based programming languages for building their solutions. Although the power and flexibility of the well-established text-based structure is undeniable, the developer still has to rely on third party software or utilities to visualize the same code structure graphically. The code structure or tree is extremely useful for debugging and optimization. However, with the availability of excellent graphical programming languages (like G in LabVIEW), this is obviously redundant. A graphical programming approach to building applications, especially for implementation on hardware or real-time systems, is more intuitive and encourages modularity, efficient code architecture and efficient maintenance as demonstrated by faster debugging, rapid upgrades or updates. Digressing, it is evident that a symbolic representation of any application is simpler to understand as it is not bound in the narrow confines of representative language or script.

The ease of implementation of wireless sensor program architectures using graphical programming languages is obvious. Wireless Sensor platforms rely on low

processing overhead and the need for multiple processes to run simultaneously (data acquisition, communication, power management etc.). Building such applications using a graphical programming language like G, while providing all the advantages as listed above, also provides easy access to low level architectural elements which would otherwise require the developer to methodically sift through architectural elements which are represented in text. As was evidenced by the use of the tools created at the Distributed Intelligence and Autonomy Laboratory (DIAL), the organization of the various elements which were used to build wireless sensor network applications greatly reduced modification times of low-level application parameters when compared to a similar application built in a text-based environment. As every wire in LabVIEW is representative of a variable and its dataflow, data-typing and source control is automatic in LabVIEW while it is caught only at compile time in the text-based approach.

It was observed during the growth of this thesis that low-level handling of hardware was exceptionally intuitive and rapidly customizable – especially when dealing with serial communication with the mote platforms which was a major component in the development of all the applications developed in this thesis.

However, it has to be noted that the main issue with building applications in LabVIEW is that stand-alone applications are inordinately bulky and occupy a large amount of processor memory as opposed to similar applications built on MATLAB or Visual Basic. This is mainly due to the run-time engine that comes bundled with every LabVIEW stand-alone application. Hopefully, this will change in future versions of LabVIEW.

This does not affect applications which are intended to be implemented on LabVIEW's supported platforms – like the real-time processor module (the PXI series) or Personal Digital Assistants (PDAs) which have the LabVIEW run-time engine installed on them.

Overall, LabVIEW, by far, outperforms traditional text-based programming approaches for building wireless sensor network applications.

6.2 Discussion of Objectives

The objectives of this thesis can be summarized as an attempt to bridge the gap between the developers of WSN platforms and researchers. The manifold applications of WSNs require that the tools involved in their implementation be easy to use and customize for any level of expertise with application building. LabVIEW's graphical programming platform and the TinyOS group's modular event based code structure are ideally suited for each other but have been mutually exclusive so far. With the introduction of programming and analysis tools and their demonstration in relevant scenarios should convince new users of WSNs to use a graphical programming approach.

6.3 Discussion of Contributions

In summary the contributions of this thesis have been to develop the tools envisioned in the objectives and implementing them in relevant WSN scenarios. The tools developed serve as templates on which to develop interaction and analysis tools.

Along with programming tools also being developed at the DIAL, the resulting toolkit promises to be an important factor in the WSN research at the DIAL.

Apart from the tools, the building of application scenarios and deployment of these applications for the design of a novel test-bed has been a contribution of this thesis.

6.4 Future work

6.4.1 TinyOS toolkit

With the development cycle of TinyOS 1.1.x almost reaching its end, the development of tools to encompass the various capabilities of TinyOS adapted to LabVIEW would be very well received by the Wireless Sensor Network community. With increasing interest in the implementation of LabVIEW VIs in TinyOS scenarios, the toolkit could possibly be instrumental in setting the trend for future Wireless Sensor Network application structures.

Porting the new TinyOS 2.x to be implemented in LabVIEW should be simpler than the above because of the better organization of the component architecture and a more malleable coding design for nesC. Graphically modeling the code of TinyOS 2.x would encompass the entire hardware and software component tree. This would allow the user to appreciate the entire code structure and have intuitive and easy access to all levels of hardware and software components.

6.4.2 LabVIEW and WSNs

The main issue with LabVIEW has been that it offers limited access to hardware level architecture of any component. Most of the drivers distributed by National

Instruments for their devices are copyrighted and protected in the code by encryption. LabVIEW does not allow for custom variations of their programming languages to be built using LabVIEW. The latter limits the possibilities as far as Wireless Sensor Networks –especially the TinyOS applications – are concerned. Although a limited architecture of the TinyOS operating system and component architecture can be built using LabVIEW, emulating the nesC compiler on LabVIEW would require the input of the creators of the G programming language.

Building the same structure using MATLAB’s Simulink or a custom graphical environment – designed from scratch – are also viable options. However G’s multi-threading and online compilation are far superior to any solutions which could be developed as viable options in a short time frame.

6.4.3 The Distributed Intelligence and Autonomy Laboratory

The basic framework for a truly innovative test-bed has been laid at the DIAL. Along with the construction of various platforms, the template for simple communication needs to be customized for all the available platforms. As the developed scheme is extremely malleable and easily implemented, it can be customized for a more robust and secure solution.

The multi-platform network is the perfect deployment device for the TinyDB application. The tools developed in this thesis can also be easily implemented for performing localization measurements with the various mote platforms from Crossbow.

With the addition of the LabVIEW PDA control modules, the implemented scenarios may be easily extended to include Personal Digital Assistants to realize a more interactive, facile and exciting test-bed.

6.4.4 Final Thoughts

The scenario of a graphical development platform for Wireless Sensor Network Applications was proposed and the implementation using LabVIEW was initiated in this thesis. If the philosophy behind pursuing a Graphical Development System is truly realized, the near future will see the Graphical programming paradigm at the forefront of all design.



The innovative test-bed at the Distributed Intelligence and Autonomy Laboratory (DIAL) has been initiated. With the various platforms running on a simple, yet robust, and easily adapted control algorithm, concentrating on research into various control strategies was simplified.




The various additions to the LabVIEW family of toolboxes were listed. Further developments were suggested.




APPENDIX A




LIST OF IMPORTANT VIRTUAL INSTRUMENTS DESIGNED FOR THE TOOLKIT





Table A LabVIEW VIs Designed for the Toolkit




Virtual Instrument	Description
	<p>This was the first version of the Mote communication VIs. The VI, apart from opening the connection to the serial port, optionally closes and reopens the connection to force the buffers of the serial port to reset. This scheme of communication was abandoned in favor of ensuring that the connection to the serial ports was closed upon normal completion of all applications.</p>
	<p>This was also a first version sub-VI used to translate the data read from the MICA series of motes. Specifically to extract data carried on the AM packet format by specifically extracting the data bytes located at particular offsets. This VI does not unescape the packet but instead, translates the data once extracted from the original packet. This sub-VI is re-entrant</p>





	<p>This is a top-level VI which communicates, depending upon user selection, with the MICA motes or the MicroStrain motes to read vibration measurements on all available axes. This VI is an extension of the VIs designed by Mr. Ankit Tiwari in his thesis and simplifies the structure by using the Sound and Vibration toolkit VIs.</p>
	<p>The Matrix multiplication polymorphic VI handles matrix multiplication unlike previous versions available in LabVIEW. The VI automatically selects array or vector multiplication modes depending upon the inputs wired into the VI. It supports all combinations of array and vector multiplication.</p>
	<p>The Kurtosis VI as the name suggests calculates the kurtosis and also the kurtosis excess of an array of data input.</p>

	<p>The Window VI is a polymorphic VI to handle buffering of one dimensional or two dimensional arrays or matrices. This VI is implemented in the design of the Kurtosis, Kalman Filter, Bollinger Band and RLS estimator top-level VIs. A user specified windowed buffer of the scalar or vector data input is maintained</p>
	<p>The Bollinger Band sub-VI calculates the upper and lower Bollinger bounds on the input data array and outputs the data as a cluster. The input data is also duplicated at the output so that the window sub-VI may use this data as an input in its next iteration.</p>
	<p>This is a middle layer sub-VI which implements the Bollinger band sub-VI and the window sub-VI to buffer the incoming scalar or vector data in and perform the Bollinger band analysis treating each column as a channel.</p>

 <p>SPAM.vi</p>	<p>The Statistical Processing and Analysis Module VI (SPAM) was designed using the Bollinger Band analysis VI, the Kurtosis VI and the LabVIEW DataSocket set of VIs to read the currency data from the federal reserve website and to perform the Bollinger band analysis on the currency data from every business week since 1995. It has a tabbed interface to switch between Bollinger band analysis and a kurtosis analysis of the data.</p>
	<p>The remove empty strings VI is a simple sub-VI designed for the SPAM VI to trim data string arrays which had empty strings resulting from blank data fields corresponding to holidays during weekdays.</p>
 <p>Kalman Filter.vi</p>	<p>This is a top-level VI which holds the polymorphic sub-VIs which calculate the discrete time Kalman estimates for the input system. The VI is constructed using the matrix multiplication VIs. This VI is used in localization algorithms and position estimation at the DIAL.</p>

	<p>The RLS Estimator VI is again a top-level VI and holds the RLS estimator sub-VI and the window sub-VI. The RLS sub-VI performs the discrete time RLS estimation of the input system and gives the estimates as output. This VI is used in localization algorithms in mobile platform scenarios for estimating the accurate position of the mobile platform.</p>
	<p>The Initialize Mica series of VIs are used to quickly initialize the various COM port settings associated with MICA mote communication. This VI is always the first VI used when connecting to the Motes for reading data.</p>
	<p>The GetTOSPacket sub-VI gets one TinyOS AM message packet from the serial port specified and exits. This VI has a time out of 150 cycles before exiting with an error.</p>
	<p>The GetTOSPacketSynch.VI implements the GetTOSPacket sub-VI and purges incomplete or erroneous packets read to synchronize the communication between the PC and the mote platforms.</p>

	<p>The Decipher TOS Packet VI unescapes the packet embedded between the synch bytes and separates the payload from the CRC bytes.</p>
	<p>The Surge Light VI is a generic template for designing message interpretation VIs for communicating with the MICA series of motes. This VI implements the GetTOSPacketSynch and the DecipherTOSPacket sub-VIs. The mote is assumed to be programmed with a standard application like Surge and the VI reads the light reading from the specified location in the packet. This template can be extended to any type of message which is to be read from the mote platforms.</p>
	<p>The CricketInit.vi initializes the COM port for communication with the Cricket series of motes and contains standard settings which are necessary for initializing communication with the Cricket motes. It is an extension of the InitMica2 VI in as much as it is also used to set the mode of operation of the Cricket at the time of initialization</p>

	<p>The SetCricketListen.VI can be used in the application to set the mode of operation of the Cricket mote during the execution of the program. This VI is also part of the CricketInit.vi.</p>
	<p>The DecipherCricketPack.vi is used to extract required data from the message packet read from the mote. This VI also acts as a template for reading other data from the Cricket packet.</p>
	<p>The Coordrefs.vi is a top-level application which implements the mica resources and the cricket resources to track the motion of a mobile platform carrying a cricket and guide it through a MICA network in safe areas determined by a decision making system.</p>
	<p>The InitTinyDB.vi is an initialization VI and is part of the TinyDB application interface developed for TinyDB. This VI, apart from initializing the COM port, retrieves any running queries on the network.</p>

<div data-bbox="370 306 490 428" style="border: 1px solid black; padding: 5px; text-align: center;"> get Queries </div>	<p>The getQueries.vi sub-VI writes a request query to the TinyDB network and retrieves the running queries as an array output.</p>
<div data-bbox="370 562 490 684" style="border: 1px solid black; padding: 5px; text-align: center;"> Build TinyDB payload </div>	<p>The Build TinyDB Payload VI generates the node-id query which goes at the end of every TinyDB query. This VI serves as the basic template from which other command generation VIs are developed.</p>
<div data-bbox="370 890 490 1012" style="border: 1px solid black; padding: 5px; text-align: center;"> Build tinyDB cmd </div>	<p>The Build TinyDB Command VI generates the TOS message packet which forms the query being sent. It takes various parameters which make up the TinyDB query as inputs and appropriately manipulates them for the required result.</p>
<div data-bbox="370 1260 490 1381" style="border: 1px solid black; padding: 5px; text-align: center;"> get tinyDB Packet </div>	<p>The Get TinyDB Packet VI uses the Build VIs described above to generate the appropriate packet which forms the present query request to be sent with the various parameters required to be set when generating a query.</p>
<div data-bbox="370 1629 490 1751" style="border: 1px solid black; padding: 5px; text-align: center;"> get cmd Array </div>	<p>The Get Command Array VI is a middle level VI which implements the Get TinyDB Packet sub-VIs to generate an array of queries which are to be written to the TinyDB mote and takes only an array of required parameters as an input.</p>



The TinyDB application VI is a top-level application which is used to send simple queries to the TinyDB motes (MICA series motes running the TinyDB program) and analyzing and interpreting the data retrieved for the various available parameters. The present version has implemented the monitoring of light readings, parents for determining the network topology, the battery voltage reading of each mote and the temperature reading at each mote.

REFERENCES

- [1] Alec Lik Cheun Woo, “A Holistic Approach to Multihop Routing in Sensor Networks”, December 2004
- [2] F. L. Lewis, “Wireless Sensor Networks”, Smart Environments: Technologies, Protocols and Applications, ed. D. J. Cook and S. K. Das, John Wiley, New York, 2004
- [3] Marcus Bengtsson, Eric Olsson, Peter Funk, Mats Jackson, “Technical Design of Condition Based Maintenance System – A Case Study using Sound Analysis and Case-Based Reasoning”, Maintenance and Reliability Conference, Proceedings of the 8th Congress, University of Tennessee – Maintenance and Reliability Center, Knoxville, May 2nd – 5th, 2004
- [4] Prasanna Mohan Ballal, “Structure and Decision in Mobile Wireless Sensor Networks”, August 2005
- [5] Aditya Narayan Das, “Data-Logging & Supervisory Control in Wireless Sensor Networks”, December 2005
- [6] Prasanna Ballal, Vincenzo Giordano, Pritpal Dang, Sankar Gorthi, Frank Lewis, “A LabVIEW based test-bed with off-the-shelf components for research in mobile sensor networks”, To Appear in ISIC 06, Munich, 2006

- [7] Ahmed, A.A., Shi, H and Shang, Y, “SHARP: a new approach to relative localization in wireless sensor networks”, Proc. Of Distributed Computing Systems Workshops, 2005. 25th IEEE International Conference on 6-10 June 2005 Page(s): 892- 898
- [8] Chong C., Kumar S., “Sensor Networks: Evolution, Opportunities and Challenges”, Proceedings of the IEEE, col. 91, no.8, August 2003
- [9] P. Corke, R. Peterson, and D. Rus, Coordinating aerial robots and sensor networks for localization and navigation, in: 7th International Symposium on Distributed Autonomous Robotic Systems (Toulouse, France, 2004) 281-290
- [10] Dahlberg T., Nasipuri A., Taylor C., “Explorebots: A mobile network experimentation testbed”, SIGCOMM '05 Workshops, August 22-26, 2005, Philadelphia, PA, USA
- [11] Dang P., Lewis F. L. and Popa D.O. “Dynamic Localization of Air-Ground Wireless Sensor Networks”; to appear in 14th Mediterranean Conference on Control and Automation, June 2006
- [12] Dantu K., Rahimi M., Shah H., Babel S., Dhariwal A., Sukhatme G., “Robomote: enabling mobility in sensor networks”, Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on 15 April 2005 Page(s):404 – 409
- [13] Giordano V., Ballal P., Lewis F., Turchiano B., Zhang J. B., “Supervisory control of mobile sensor networks: Matrix formulation, simulation and implementation”, to appear in IEEE Transactions on System, Man and Cybernetics part B

- [14] Girod L., Elson J., Cerpa A., Stathopoulos T., Ramanathan N., Estrin D., “Emstar: a software environment for developing and deploying wireless sensor networks”, in Proceedings of the 2004 USENIX Technical Conference, Boston, MA, 2004
- [15] King J., Pretty R., Gosine R., “Coordinated execution of tasks in multiagent environment”, IEEE transactions on Systems, man and cybernetics- Part A: Systems and Humans, vol. 33, no.5, September 2003
- [16] Johnson D., Stack T., Fish D., Ricci R., Lepreau J., “TrueMobile: A mobile robotic wireless and sensor network testbed”, University of Utah Flux Group Technical Note, April 2005
- [17] Khatib, O, “Real-time obstacle avoidance for manipulators and mobile robots”, Robotics and Automation. Proceedings. 1985 IEEE International Conference on Volume 2, Mar 1985 Page(s):500 – 505
- [18] Puccinelli D., Haenggi M., “Wireless Sensor Networks: Applications and Challenges of Ubiquitous Sensing”, IEEE Circuits and Systems Magazine, vol. 5, pp. 19-29, August 2005.
- [19] Harris, B., Cook, D., and Lewis, F.L., “Automatically generating plans for manufacturing,” J. Intelligent Systems, vol. 10, no. 3, pp. 279-319, 2000.
- [20] Howard, A.; Mataric, M.J.; Sukhatme, G.S.; “An incremental deployment algorithm for mobile robot teams”, Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and System, vol.30, Pages:2849 – 2854, October 2002

- [21] Werner-Allen G., Swieskowski P., Welsh M., “MoteLab: A wireless sensor network testbed”, Fourth International Symposium on Information Processing in Sensor Networks, 2005. IPSN 2005. 15 April 2005 Page(s):483 – 488
- [22] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer and David Culler, “The nesC Language: A Holistic Approach o Networked Embedded Systems”, Proceedings of Programming Language Design and Implementation (PLDI) 2003, June 2003
- [23] “Wireless Sensor Networks”, en.wikipedia.org/wiki/Wireless_Sensor_Network
- [24] “National Instruments” www.ni.com
- [25] “LabVIEW”, en.wikipedia.org/wiki/LabVIEW
- [26] “TinyOS Community Forum”, www.tinyos.net
- [27] “Crossbow Technologies”, www.xbow.com
- [28] “MicroStrain Miniature Sensors”, www.microstrain.com/g-link.aspx

BIOGRAPHICAL INFORMATION

Sankar Bhanu Gorthi received his Bachelor's Degree in Electrical and Electronics Engineering from the Osmania University College of Engineering, Hyderabad, India in 2003. He joined the University of Texas at Arlington in Spring 2004 registering in courses leading up to a major in Control Systems. His interest in Robotics and Control Systems led him to look for opportunities at the Automation and Robotics Research Institute at UT-Arlington. There, he worked with Wireless Sensor Networks and helped build the Distributed Intelligence and Autonomy Laboratory along with Prasanna Ballal and Vincenzo Giordano under the supervision of Dr. Frank Lewis. His interests lie in Intelligent Control Schemes and Robotics. He is also interested in working on Graphical Programming. Ultimately, he wishes to acquire his PhD in Control Systems and pursue a career in teaching which has been his life-long dream.