DISCRETE EVENT CONTROLLER: APPLICATION USING DYNAMIC RESOURCE ALLOCATION

by

ABHISHEK CHAITANYA TRIVEDI

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

THE UNIVERSITY OF TEXAS AT ARLINGTON

August 2007

ACKNOWLEDGEMENTS

I would like to thank many people who have guided and supported me for the completion of this thesis. First of all I would like to thank my Parents, Chaitanya Trivedi and Divya Trivedi, my brother Chinmay, my Grand Parents and all my relatives for the encouragement and support they have constantly provided to me throughout my education.

I sincerely thank my supervising Professor Dr. Frank Lewis for guiding and supporting me, and giving me the opportunity to pursue this thesis.

I am obligated to Mr. Prasanna Ballal without whom this thesis would not have been possible. I am grateful to him for all his inputs, suggestions and guidance during the course of this thesis.

I thank Dr. Dan Popa and Dr. Harry Stephanou for graciously agreeing to be on my thesis supervising committee and taking interest in my thesis work.

Last but not the least; I would like to thank Dr. Vincenzo Giordano who gave me the first opportunity to work at ARRI and Sankar Gorthi who was a great help during initial of stages my thesis work. I would thank all my friends especially Somil, Daxesh, Viral and Suraj who were supportive throughout my studies at UTA.

July 23, 2007.

ABSTRACT

DISCRETE EVENT CONTROLLER: APPLICATION USING

DYNAMIC RESOURCE ALLOCATION

Publication No. _____

Abhishek Chaitanya Trivedi, M.S.

The University of Texas at Arlington, 2007

Supervising Professor: Dr. Frank Lewis

This work consists of developing a Discrete Event Controller with Dynamic Resource Allocation, simulating it and then applying it on a test-bed using robots and static sensor nodes. These are the contributions made:

1. Interfacing the Garcia robot with Mica2/Cricket Sensor.
2. Implementing Obstacle Avoidance and Path planning on Garcia Robot for navigational purposes.
3. Localizing the Garcia Robot with the help of Cricket sensor and controlling it from the Base Station to perform assigned tasks.

4. Simulating a Discrete Event Controller which dynamically coordinates multiple missions and simultaneously performing dynamic resource assignment and solving any shared resource conflicts.
5. Implementing the aforementioned Discrete Event Controller on a test-bed containing the Garcia robots and Mica2 sensors. The navigational techniques mentioned above are used for mobility of Garcia in the test-bed while the Base station runs the Controller managing missions, assigning robots to sensors and resolving the conflicts to prevent a deadlock.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

Table Page

CHAPTER 1

INTRODUCTION

1.1 Introduction

Wireless Sensor Networks are a trend of the past few years and the only way to monitor the real world adequately is to use a network of devices. This means deploying a number of small sensor nodes capable of far more complex tasks in hostile environments or over large geographical areas instead of centralized, expensive, single-node platform [50]. When these stationary wireless sensor nodes collaborate with mobile autonomous robots, they form a powerful system with flexible network architecture. This system is responsible for adaptation and self-configuration since networks consist of numerous unreliable nodes, communication links and changing environment conditions [4, 30]. The mobile sensor nodes stand out for the network as they can adapt and self-configure using their mobility and additionally providing increase in network coverage, better routing performance and better connectivity [48].

A Discrete Event Controller is needed for the coordination of cooperating heterogeneous wireless sensor networks (WSN) containing both static ground sensors and mobile sensor robots. A Discrete Event System is a system whose behavior is characterized and governed by events occurring at unknown irregular moments of time. The events can be controlled (i.e. retrieving data from a sensor at particular time) or uncontrolled (i.e. the sensor data getting higher or lower than the set levels) [46]. The

next tread of the system is to control the execution of a set of tasks to be performed by a set of resources, arrange coordination of events, resource contention management, performance monitoring and optimization. Discrete-Event Systems (DES) control theory [46] has been used to control several different problems including manufacturing and assembly tasks, the coordination of mobile robots [8], a hybrid discrete event systems approach for robot control [26] and different environmental sensing applications.

A Discrete Event Control (DEC) system has to make the best choice from among set of given alternatives to perform the task. The controller's objective is to allocate a finite number of discrete resources to a set of tasks so as to achieve optimal system performance without any deadlocks. In a mobile sensor network, dynamic coordination of multiple robots [21] is required to execute different tasks along with multiple competing missions, network topology changes and addition/removal of nodes. This can be satisfied by implementing dynamic resource allocation algorithms in the DEC.

1.2 Objective

The objective of this thesis is to study and implement an efficient matrix-based Discrete Event Controller (DEC) [44] with Dynamic Resource Allocation on Mobile Wireless Sensor Network test bed at Automation and Robotics Research Institute. In this thesis the robots from Acroname Inc. called Garcia along with sensors from Crossbow called Mica2/Cricket are used for the implementation. Different techniques

such as obstacle avoidance, localization and path planning used for the robots navigation and which is subsequently used in DEC and dynamic assignments are presented in this thesis.

The underlying principle in interaction between the static sensors and robots is: the sensors serves as the communication, sensing and computation medium for the robots, whereas the robots provide actuation (mobility), which is used among other things for sensor deployment, repair, reacting to the events of the environment and other tasks [43]. Each sensor is equipped with some limited memory, processing capabilities, multiple sensing modules and communication capabilities. But the use of mobile sensors helps in saving power of the static sensors as they can transmit the data collected, at low power to the robots when they arrive near to it.

The use of mobile robots in sensor networks requires positioning themselves strategically to enhance global performance. For this the robot needs autonomous navigational capabilities [29]. This can be obtained by obstacle avoidance using which the robot can travel around in the environment and relaying or gathering data to or from the static sensor nodes. Localization can be used to send the robots at particular positions and perform the tasks assigned, while Path planning can be used to program the robots with the predefined paths and send them in hostile or toxic regions to deploy sensors or collect data from them.

Centralized and Decentralized techniques are available for task assignment and resource dispatching in mobile wireless sensor networks. In decentralized technique the robots or sensors control themselves and have no information about the network except

their neighbors while in the centralized technique the supervisor or the base station controls the coordination of the robots and sensors. The centralized control technique [24] is used as it can reschedule mission planning and allocate tasks in response to uncontrollable events with appropriate resources.

DEC was first used in manufacturing systems [32] in order to sequence the most suitable tasks for each agent according to the current perception of the environment. The matrix formulation allows fast, direct design and reconfiguration of discrete event controllers. It provides a better dynamical description, high-level interface and efficient handling of missions along with the popular tools as Petri-Nets.

Task Allocation is required for assigning available resources to tasks. There are two major subdivisions: offline and online. Offline Task Allocation is the problem of assigning resources to tasks if certain information (e.g. the distribution of task arrival times, relative task priority) is known a priori. In online Task Allocation, all information about the tasks becomes available only upon task arrival [43]. The assignment of resources to tasks must be computed in real time and greedy algorithms must provide good approximate solutions to it. If it fails, the system could result into a deadlock which could mean disaster in some scenarios. So a MAXWIP deadlock avoidance policy using [28] is implemented which checks the system after every new resource assignment whether it is regular and free from any deadlocks. The MAXWIP policy is a greedy algorithm implemented by on-line updating of the resource requirements matrix [31]. If the assignment does not result into a deadlock, the DEC [44] is implemented to assign the subsequent tasks to be performed.

1.2.1 Software's and Languages used

In this thesis Labview, NesC and Matlab are used extensively for designing and implementing various algorithms and applications required for Dynamic DEC. Labview which is a powerful graphical programming language with integrated I/O capabilities is used for retrieving and processing sensor data, developing Matrices and Petri-nets intuitively for further usage in DEC and dynamic allocation of resources. It gives ease of use, better interaction and user friendly applications, while getting the developer free of the low level architectures of I/O and complexity of implementing graphics in a text based programming language. NesC a C-like modular programming language makes it easy to control Garcia, get data from different sensors and simultaneously communicate and exchange data with the base station using Mica2s and Crickets. Matlab is used for simulating the Discrete Event Controller with Dynamic Allocation of Resources and gives the perception of the system before implementing it on the Robots using different sensors.

1.3 Applications

WSNs are finding applications in different fields and are spreading progressively as new research is done. WSNs are deployed on a global scale for environmental monitoring and habitat study, on a battle field for military surveillance and reconnaissance, in emergent environments for search and rescue, in factories for condition based maintenance, in buildings for infrastructure health monitoring, in homes to realize smart homes, or even in bodies for patient monitoring [51].

CHAPTER 2

ROBOT AND SENSOR PLATFORM

2.1 Garcia Robot Introduction

Garcia is the perfect robot if one wants to focus their efforts on developing the software and algorithms to enable the robot to interact with the environment, other robots and sensors. That is true because the mechanical and electrical challenges of robotics are already solved. Garcia solves these problems and can be directly used to run different applications. It comprises the most flexible materials, concise design, and seamless integration of software with hardware [2].

Garcia offers the platform for robotics. It is a rock-solid collection of low-level I/O processing, mechanics, and electronics that allows the designers and researchers to use precious time figuring out the big challenges in robotics. Challenges relate with mapping, localization and collaborative behavior in robots.

2.2 Garcia Features

2.2.1. Processors and Memory

Two separate 40MHz processors handle the robot's functions. A BrainStem Moto 1.0 processor handles the motion control and several sensor inputs. A BrainStem GP 2.0 processor provides a serial interface, IR communication capability, and

additional IO. The GP and Moto processors are PIC 18C252, which is an 8 bit high performance microcontroller chip and has a 32K EEPROM for code storage [2].

2.2.2. Battery

The robot is powered by a standard 6-cell 7.2V 3000mAH NiMH battery pack.

2.2.3. Serial Port

External computers may communicate with the Garcia through a TTL-level serial port. Different host applications can be used or developed to communicate with the Brainstem board using serial port to get various sensor data from Garcia or control its motion. Most of the communication with Cricket/Mica2 is done using this port.

2.2.4. Range Finders

Garcia has six IR range finders. These are Sharp GP2D12 IR sensors. They provide valid distance measurements in a range of 4 to 18 inches. These sensors enable the robot to wall-follow or detect obstacles while maneuvering. When not in use, the sensors can be disabled to save battery power. The 6 sensors are connected to the analog pins on the Brainstem board. Two sensors (rear left and rear right) are connected to the GP board and 4 sensors (side and front) to the Moto board.

2.2.5. Ledge Detectors

Under the front end, there are left and right floor proximity detectors. These sensors can tell the robot if it is about to roll over a ledge. If the robot is moving slowly toward a ledge, it can stop its wheels and skid to a stop before plummeting.

2.2.6. Motors and Extras

Garcia comes equipped with two Maxon A-max motors. The motors can be used normally or in PID Encoder mode for controlling the motion. Commands can be sent to Garcia for turning, or moving reverse and forward. The Garcia drive train produces 3648 encoder pulses per wheel revolution. In the PID Encoder mode, Garcia's movements can be defined using encoder ticks. For example, it takes 3504 ticks for each motor to move Garcia by one foot and 18 ticks for turning Garcia by one degree totaling to 6501 ticks for a full 360 degree turn [6].

The other features [2] which Garcia includes are 4 free I2C connections, an extra serial port, and 4 servo outputs. It also has an IR transceiver, thus the robot can be controlled by an IR remote and also can transmit codes through the IR transmitter.

2.3 Garcia Commands

Garcia is a customized robot and it can manage IO, do precise motion controls and run simple level programs using the processors on board. But for accessing the real power of it, there are primitives which can be used to provide great functionalities [1]. The Brainstem board on it has got memory which stores various pieces of codes in it, these codes are a sequence of simple tasks which are also known as the Primitives. Primitives are the building blocks of behaviors when programming a Garcia robot. Garcia can store over a dozen primitives. They are stored in EEPROM so they can be updated or changed when necessary. These primitives interact with the monitor to make sure that the robot acts appropriately when encountering special conditions.

The commands to run the primitives can be sent to Garcia using the serial port in it. There are two serial ports on the Brainstem board, GP serial and Moto serial. Any one of these can be used to communicate with a host. The host can be anything from a computer to a PDA, a Linux board like STARGATE or Cricket/Mica2 modules and it can be connected remotely using a wireless card or can ride with Garcia by connecting locally using the serial cable. The baud rate used for communication is 38400. One can use the COM ports on the computer to connect to Garcia serially and test various commands through the Brainstem Console. These commands can then be used in conjunction for motion control, obstacle avoidance and other functionalities.

The commands given to Garcia are small hex codes and the structure format is:

Byte 1 - the IIC address of the module that shall receive the packet.

Byte 2 - the length of the packet.

Byte 3 to n - the data in the packet.

The limit to the packet byte is 8 bytes with 1 command byte and 7 data bytes. The address byte plays an important part in forming the command. The address for the two processors on the Brainstem board is: GP processor is 2 and Moto processor is 4. The physical hardware on the Brainstem architecture is called module. These are Servo outputs, A/D channels, Digital I/O pins, GP2D02 driver etc. which are connected to either the GP or the Moto processor and they communicate with the host using the two wire IIC protocol. These modules can be accessed using the unique address given to it, but the router address is the same for all. When the router receives a command, it checks the destination address and if it matches its own, it executes the command. If it

does not match, it transmits the command via the IIC bus to another module. A command can be sent addressed to the module and the data can be identified as to which module has responded to it using the address in it. Thus the command byte is formed of the address of the processor, address of the module for which command is intended, the command, and data to be passed on to the module if required by it.

The most important of all commands is the Heartbeat. The Heartbeat is originated by the modules to indicate the link health status. A Labview interface was developed to find out the response from Garcia through which the heartbeat code was found subsequently. The heartbeat code is 0202 0000H and 0202 0001H. These commands continue to run and they alternate between 0 and 1. Garcia has two LEDs for depicting the heartbeats. These indicators offer great assistance in trying to debug the status and operation of the link.

Some other commands which were found using the Labview interface are:

USER LED: 0403 267C 01 – ON

0403 267C 00 – OFF

INFRARED: 0204 4705 XXXX (xxxx refers to data to be transmitted. For example: 0101 if 01 is to be transmitted).

ENCODER ODOMETER: 0402 4202 (4 66 2) is used to access the 32-bit encoder odometer. The motors can be configured in PID encoder mode or used without any mode. In PID mode, the PID routine adds the distance traveled during the last iteration to a 32-bit encoder accumulator. In general, each step leads to the increments or decrements of the 32-bit odometers depending on the motor's forward or reverse

motion. When Garcia gets the command, the reply would be in this format: 0404 4202 001F AB00 1FE0, where 4202 is echo for the command, the next 3 bytes contain 32-bit odometer value for right wheel (0x001FAB) and the last 3 bytes contain 32-bit odometer value for left wheel (0x001FE0). These values can be converted into decimal and used to find distance Garcia has traveled.

MOTOR CONFIGURATION: 4 63 0 0 5 0/4 63 1 0 0 5 0, these commands put the right and left motor into PID encoder mode. 4 63 0 5 13 xxxx/4 63 1 5 13 xxxx sets the PWM output limit i.e. the motors would stop after xxxx ticks (xxxx can be any decimal number). There are various other commands which are useful for motor configuration. These are setting the P, I, and D parameters for error correction, setting the period, latency, and frequency values, but are not mentioned as they are not used in this thesis. After configuring the motors the new settings can be saved to the EEPROM using the 4 64 (0404 40) command.

The most useful commands for running the motors and getting the ranger values are:

MOTOR DIRECTION AND SPEED:

LEFT: 0404 3E01 0001 (0001 is the speed of motor in forward direction).

0404 3E01 FFFF (FFFF is the speed of motor in reverse direction).

0404 3E01 0064 (Maximum speed of 100 in forward)

0404 3E01 FF9C (Maximum speed of 100 in reverse)

RIGHT: 0404 3E00 0001

0404 3E00 FFFF

STOP: 0404 3E00 0000 (RIGHT)

0404 3E01 0000 (LEFT)

Thus, the command has code 00 for RIGHT motor and 01 for LEFT motor. Both left and right commands can be given in sequence with the same speed in either direction to move Garcia straight in forward or reverse direction. Similarly for turning Garcia in left or right direction requires the speed value of one motor in positive and the other motors value to be one's complement of the positive value. As an example, for turning left with '05' speed the command would be 0404 3E01 0005 0404 3E00 FFFB and for turning right with '03' speed the command would be 0404 3E01 0003 0404 3E00 FFFD.

RANGER/SENSOR:

The sensors on Garcia are usually turned off to save battery power usage. So to turn on the sensors before polling it, 0204 1B01 00 command should be given.

The table depicts all the commands for polling Garcia sensors.

Table 2.1 Garcia Sensor Commands

Ranger	Command	
Front Left	4 25 128	0402 1980
Front Right	4 25 129	0402 1981
Left	4 25 130	0402 1982
Right	4 25 131	0402 1983
Back Left	2 25 128	0202 1980
Back Right	2 25 129	0202 1981

These commands are used to poll the Garcia rangers and find out the ranger distance values from the obstacles when it is moving. The response to these commands is in the form 0404 04SS xxxx/0204 04SS xxxx. '04' at the start of the response suggests the front rangers and '02' suggests the back rangers. SS represents the sensor id. 00 for the left and 01 for the right, and xxxx is the data depending on the distance from the obstacle. If there is no obstacle the output is 0000. The output varies from 0000-8000 approximately, 8000 means that the obstacle is at an alarming distance. These data values can be used to detect the obstacles and stop or maneuver Garcia according to the distance from the obstacle.

2.4 CRICKET/MICA2 Description

Mica2, Mica2Dot, Cricket are radio/processor boards commonly known as motes [15]. Generally a mote has memory, a power unit, a transceiver and UART unit, an ADC/DAC unit and a processor with addition of an ultrasonic transceiver for the Cricket mote. A sensor unit comprises of light sensor/photocell, acoustic sensor, tone detector, 2-axis accelerometer, 2-axis magnetometer which can be added to the mote unit. These motes with the sensors attached can be distributed over a wide area and they can transmit the data back wirelessly to the base station which can be another mote connected to the computer. The computer can display and check the data simultaneously and send a command, if needed, to the motes accordingly. The processor on the motes runs a scaled down version of UNIX operating system called TinyOS [45]. TinyOS is an event driven operating system which can handle several

data flows simultaneously, coheres hardware and application specific components with little processing and storage overhead, gives robustness to sensor networks and manages network interfaces transparent to users, radio transmission and power utilization [16]. TinyOS employs a powerful modular programming language for this called NesC [35] which is a subset of C. NesC is basically a graph of components consisting of command handlers, event handlers, frames and bundle of simple threads composed into one modular application configuration. Thus, the programmer can use whichever module he wants and wire them together in whatever configuration he wants to form an end program. The motes have other advantages such that, they can form an adhoc network in which they determine their roles and form the most efficient networks by themselves. The network also supports multi-hopping. So, if a mote is out of range or far away from the base station, it can send the data from mote to mote and hence find the best route for the data packet to reach the base station through multiple hops in the network. Thus, the motes are very useful for various applications: in Wireless Sensor Networks, security, surveillance, environmental monitoring, distributed computing platform etc.

2.4.1. Mica2 Sensor

Mica2 is Crossbow's third generation of sensor mote and boasts of some advantages over other sensors of the same as well as previous generations:

i. 433, 868/916, or 310 MHz Multi-Channel Radio Transceiver.

ii. 128 Kb program flash memory, 512 Kb serial flash.

iii. Wireless communication with every Node as Router Capability.

The major characteristics that stand out in these are the overall small size and weight of the device coupled with its small power source. Due to limited power supply, the memory and processing computations on Mica2 should be kept low and be done by the base station. This improves the sensing durability of Mica2. The Mica2 can be programmed using the MIB510 Programmer and Serial Interface board. The Mica2 sensor is connected to the Programming board via the 52 pin connector and the serial port out on the Programming board is connected to the host machine through a standard serial cable. There are no embedded sensors on the Mica2 since a complete sensor board has been designed individually that can connect to Mica2. Thus, the sensor board improves energy conservation for Mica2 and is used whenever a sensing application is required. The MTS300 sensor board is usually used with Mica2 as it has the most common sensors used in industries namely light sensor, acoustic sensor, air temperature sensor, magnetometer, accelerometer etc. The Figure 2.1 shows the Mica2, the MIB510 programming board and the sensor board.

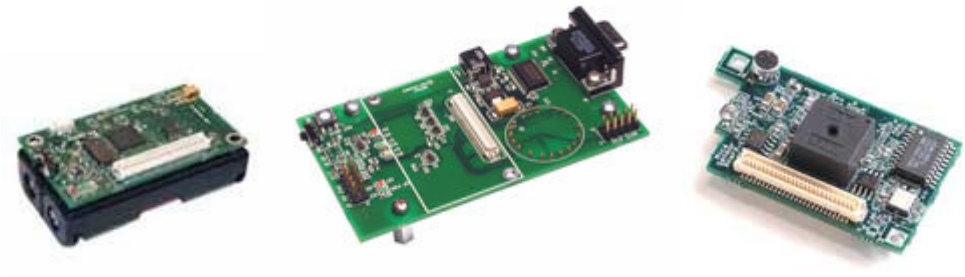

Figure 2.1: Mica2, MIB510 Programming Board and MTS300 sensor board.

2.4.2. Cricket Sensor

The Cricket Mote [14] is a location aware version of the Mica2. The Cricket Mote includes all of the standard Mica2 hardware and an ultrasound transmitter and receiver. This device uses the combination of RF and ultrasound technologies to establish differential time of arrival and hence linear range estimates. Cricket provides very precise location information for space identification, position coordinates, and orientation to applications running on handhelds, laptops, robots, and sensor nodes. It can provide distance ranging and positioning precision between 1 cm and 3 cm.

The Cricket mote sends an RF signal concurrently with the beacon which transmits an ultrasonic pulse. Listeners attached to devices, base stations and mobiles listen for RF signals, and upon receipt of the first few bits, listen for the corresponding ultrasonic pulse. When this pulse arrives, the listener obtains a distance estimate for the corresponding beacon by taking advantage of the difference in propagation speeds between RF (speed of light) and ultrasound (speed of sound). The listener runs algorithms that correlate RF and ultrasound samples and picks the best correlation. A Cricket attaches to the host device using an RS232 serial connection. Cricket can function as a Beacon, Listener or mix of both depending on the software program and a sensor board as MTS300 can be attached to the 51-pin connector alike Mica2.

Various applications that can be developed using the Cricket's location awareness are resource discovery, human/robot navigation, physical/virtual computer games, location-aware sensing, and hospital/medical applications like equipment and patient tracking/monitoring.

The Figure 2.2 shows the hardware layout of Cricket mote [14].

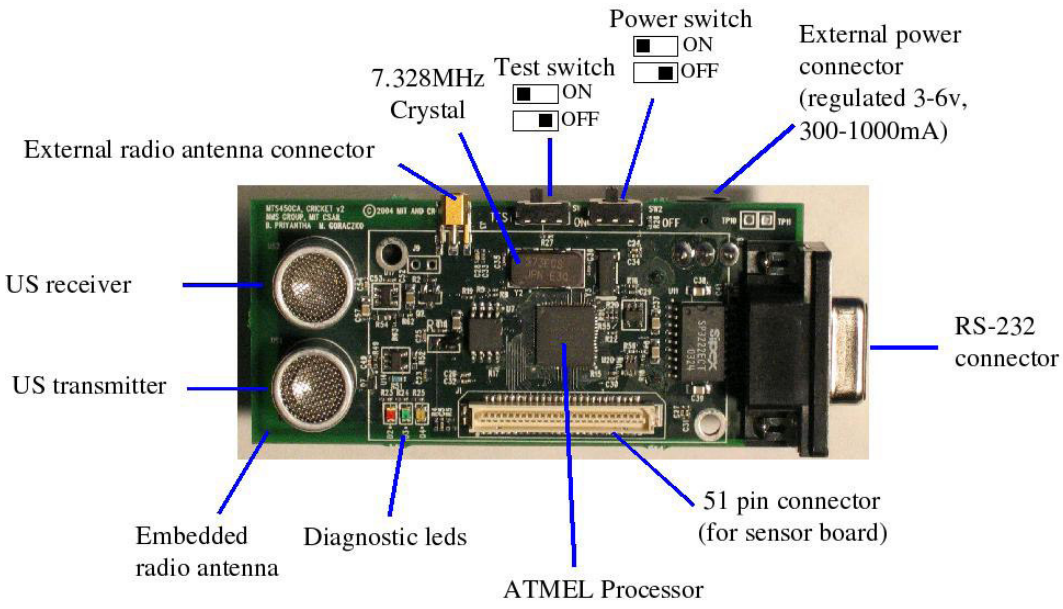

Figure 2.2: Hardware layout and Components on Cricket Mote.

The Cricket has different commands for setting up Beacon or Listener mode or to get the current configuration. For giving commands the Cricket is required to be connected to the base station through the serial port and can have an access using the HyperTerminal. For getting the current configuration of Cricket 'G CF' command is given, which would give the current mode, space id, distance from beacon, time of travel of pulse etc. For setting up the Cricket in Beacon mode the command is 'P MD 1' while for the Listener mode it is 'P MD 2'. For setting up space id the command is 'P SP "data"'. To save it to the flash memory the command is 'P SV'.

2.5 Interfacing Sensors to Garcia

The Mica2 and Cricket are the two motes used extensively with Garcia in this thesis. The motes are connected to Garcia through the serial port. Garcia has a unique connector which connects to the serial port pins on Brainstem GP or Moto board. Thus, the connection between two serial ports of Garcia and Mote is done using a user made connector. The interface of Cricket mote with Garcia is shown in Figure 2.3.

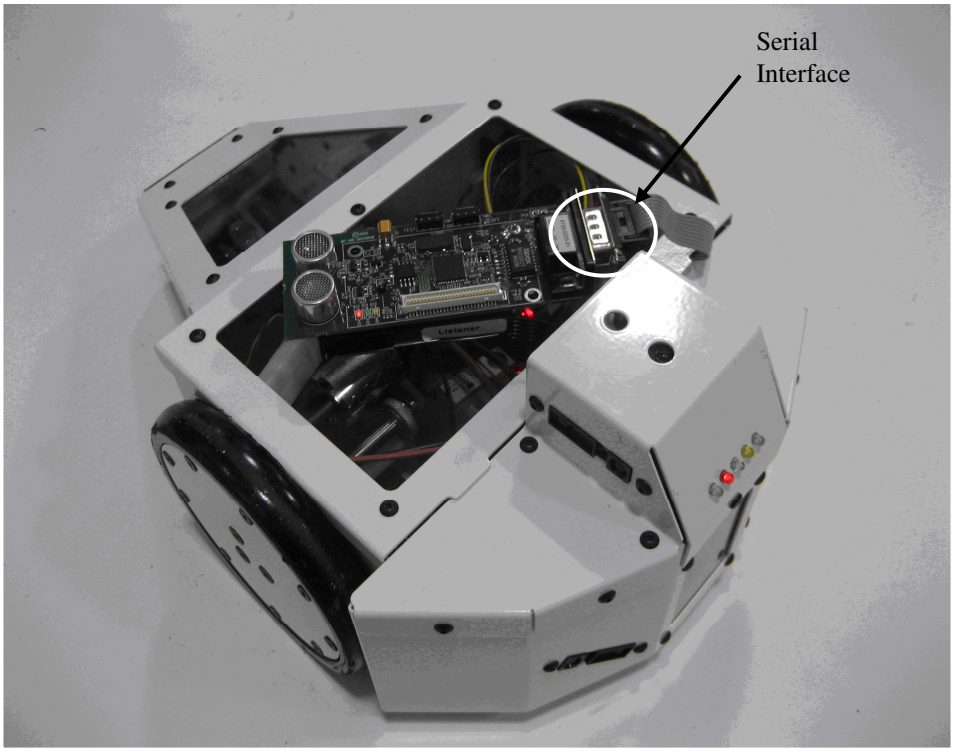

Figure 2.3: Interfacing Garcia with Cricket Mote.

The mote monitors and processes the data received by Garcia and sends appropriate commands to it. The motes are also able to communicate with the Base station which uses a similar mote. Thus, it can send the data from Garcia to the base station which can be used by the user or processed by the application running on it. The

base station can send commands to Garcia through the motes and control it according to the application needs.

The communication between Garcia and Mica2/Cricket is done using the UART/Serial interface of TinyOS. The ByteComm event is called whenever data is available or to be sent to Garcia. The data bytes available from Garcia trigger an event ByteComm; in this event the data can be stored in an array which can be further processed on mote itself or sent directly through the radio to the base station. Whenever a command is to be sent to Garcia the ByteComm is called to send the hex bytes sequentially. The wireless communication from the mote on Garcia to the base station is done using different Radio interfaces on mica which are Radioreceive and RadioSend. The Radioreceive interface is used to receive data or commands from the base station. The interface decrypts the complete packet and picks out the data sent. This data\command can then be used to call a function or pass on the data to Garcia. The RadioSend interface is used to send data back to the base station. A packet needs to be formed consisting of data before sending it to the base station. The base station mote consists of TOSBase code which can send and receive data from the PC, PDA while the mote connected to Garcia needs a code developed by the user.

Thus, the interfacing of Mica2/Cricket with Garcia can be utilized in developing many useful applications. The processing power on motes can be used with Garcia's IR rangers for obstacle avoidance, path planning etc., while the ability to communicate wirelessly with a base station and simultaneously controlling Garcia can be used in for map building, localization, and other DEC related applications.

CHAPTER 3

LOCALIZATION AND PATH PLANNING

3.1 Introduction

The most vital aspect of a mobile robot is Navigation planning. Navigation requires success of four building blocks working together. These blocks are: Perception, Localization, Cognition and Motor Control [40]. Perception means to extract data from the sensors and interpret it according to the use. Localization is that the robot should know or be informed where, in the environment, it is located. Cognition is the intelligence of robot where it should decide what steps it should take to fulfill its goal. These include finding obstacle ridden paths and then optimize that path for that particular environment. Motor Control is to move the motors according to the steps decided in the cognition block.

The environmental navigation and map planning is comprised of two behaviors which are exploration and path planning [11]. Thus, the robot has to explore through various obstacles and find a free path to reach the assigned target. Normally, a map of the environment is available with arbitrarily placed obstacles in it and the user can program the mobile robot to reach the goal using the free paths in it. But in unknown and constantly changing environments this technique cannot be used and the obstacle avoidance strategy must be developed to provide the robots with both the flexibility and the autonomy needed to cope with navigation.

3.2 Obstacle Avoidance

Obstacle avoidance technology gives mobile machines and robots life-like reflexes and allows them to navigate intelligently [40]. Obstacle avoidance is done by taking measurements from various sensors and the subsequent reaction taken from extracting meaningful information from the measurements is handled by motion planning. These two make up necessary components for any robotic task. Obstacle avoidance means planning collision-free trajectories for robotic systems using the decision making power of the processor/controller on the system while Motion planning refers to planning smooth motions for the system which is also handled by the processor/controller by giving appropriate commands to the motors.

Autonomous mobile robot like Garcia is an intelligent machine that is capable of navigating in an obstacle-cluttered environment without collision and without human intervention. Garcia uses information provided by its infrared sensors to find the distance to the obstacle. The obstacle avoidance code stored on the unused memory of Brainstem board generates a sequence of speed and steering commands for the two front wheels separately driven using two DC motors. Garcia has a built-in demo code using which it can wander around in the surroundings. It goes straight until it detects some obstacle; it will turn away from the obstacle and starts going straight again. It continuously checks the front sensors for detecting an obstacle and while turning it checks both the front and the rear sensors so that it does not collide with another obstacle. Thus, instead of using this wandering demo mode, it was simulated using a NesC code on the Cricket/Mica2 so that the Garcia can move around avoiding obstacles

and also be controlled from the Base station. Garcia can transmit back the path information to the Base station, which generates an optimal path for the next run or for other robots.

3.2.1. GP2D12 IR Rangers

As described in the second chapter, Garcia has six GP2D12 IR rangers [3] for detecting obstacles in the range of 4(10 cm) to 30(80 cm) inches. These rangers are confined in a very small package with very little current consumption and provide an analog output voltage inversely proportional to the distance. Thus these rangers are eyes to Garcia and are used in developing the obstacle avoidance application for it. The rangers have 3 pin connectors: power, ground and the output voltage. This sensor use triangulation and a small linear CCD array to compute the distance and presence of objects in the field of view. The operation is as follows: The emitter emits a small beam of IR pulse which travels straight and does not return if there is no object in its path, producing no reading at the output pin. Now, if the light reflects off an object, it returns to the detector and creates a triangle among the point of reflection, the emitter, and the detector. The CCD array detector is offset by about ¾ inch from the emitter. The angles in this triangle vary, based on the distance to the object, and from those angles, the distance to the object can be calculated. This method of ranging is almost immune to interference from ambient light and offers indifference to the color of object being detected. The Figure 3.1 shows the variation in angles for different distances and the principle that is used for the triangulation method.

The receiver portion of these sensors is actually a precision lens that transmits the reflected light onto various portions of the enclosed linear CCD array based on the angle of the triangle described above. The CCD array can then determine what angle the reflected light came back and can calculate the distance to the object.

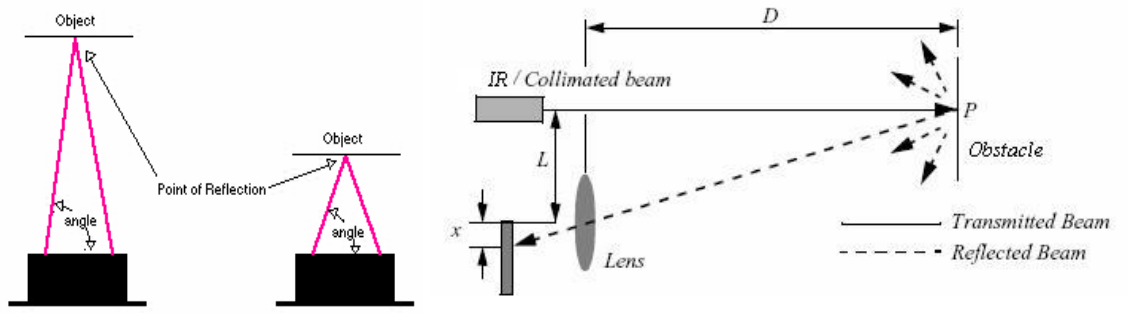

Figure 3.1 Different Angles with Different Distances and Principle of 1D IR triangulation.

The receiver here measures the position of the reflection along a single axis as shown in the figure, thus making the sensor an optical triangulation sensor in 1 dimension. The geometry of distance D is given by a formula:

$$D = f\frac{L}{x} \tag{3.1}$$

The distance is proportional to 1/x and the frequency (f) is calculated using the formula: c/λ. λ is the wavelength of the infrared light emitted. The output of the rangers is non linear (logarithmic as shown in Figure 3.2 [36]) with respect to the distance due to this kind of trigonometry, but in fact it is very accurately inversely proportional to the distance. The analog value reading can easily be converted to linear distance by dividing it into a constant i.e. Distance = Constant / (Analog reading). This

method works very well down to a distance of about 4 inches, but below this value, the sensor is unusable as it outputs voltage levels that make the results ambiguous. Similar thing happens at larger distances too.

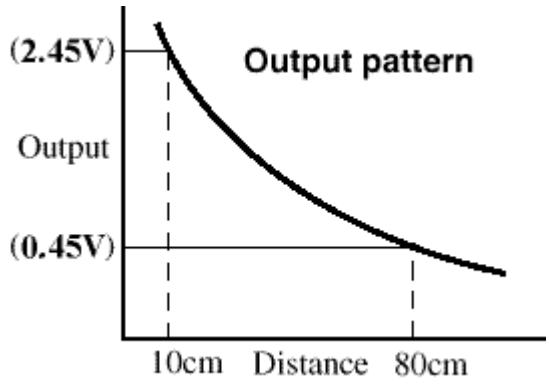

Figure 3.2 GP2D12 Output Voltage to Distance Curve.

3.2.2. Obstacle Avoidance Code Explanation

The obstacle avoidance code is developed on the basis of the distance obtained from the GP2D12 sensors. The rule based behaviors similar to fuzzy behaviors are used in building the obstacle avoidance code. The different behaviors are: Free-run, Obstacle Avoidance, Stop and Back-up. All these behaviors have control rules: For the Free-run behavior, if the Garcia senses no obstacle in its path, increase the speed gradually and move straight. For Obstacle Avoidance behavior, if there is an obstacle on the left side turn to the right side and proceed and vice versa. For Stop and Back up behavior, if the obstacle is too near to the Garcia, stop immediately and move backwards for a certain distance and then run the obstacle avoidance behavior. Thus the fuzzy logic keeps the code simple and gives the robot maximum safety and flexibility to recover from unexpected obstacles.

As stated in the 2nd chapter, the Garcia has different commands to poll individual sensors. The six sensors are connected to the Brainstem GP and Moto GP and have unique addresses. As the Garcia mostly moves forward, the front rangers are constantly polled using the commands 0402 1980 (4 25 128) and 0402 1981 (4 25 129) for the left and right rangers respectively. Thus, the wandering mode is started using an 'S' Start command through the radio. The NesC code on the Mica2/Cricket which is connected to Garcia starts a timer which constantly gives a forward command 0404 3E01 0005 and 0404 3E00 0005 to the left and right motors respectively. Another timer, which has already been started as the code initializes polls the front rangers continuously. The response from the Garcia is constantly checked in the code, which looks for 0404 04SS xxxx. The SS to be 00 or 01 depending on the sensor id response and the data value xxxx can be 0000 to 8000, where 0000 equals to no obstacle and 8000 meaning obstacle is too close. The code checks for the distance value to the obstacle and if it is greater than 3500 it calls a function which gives commands to the motors to turn away from it, i.e. if the obstacle is on right, then turn left and vice versa. The code also has provision for unexpected obstacles, in which the Garcia backs off a small distance and then turns away from the obstacle. Here a data value greater than 7000 is watched in the response. The code also transmits 'L' or 'R' whenever it turns Left or Right, which would be further used in path planning. The important functions used in the code are shown in Appendix A.

Thus instead of using the demo wandering code in which the Garcia cannot be controlled, an obstacle avoidance code is developed using the control rules, in which the

Garcia can be controlled by using the Mica2/Cricket making it autonomous, giving it capability to move around in obstacle cluttered environment with maximum security and simultaneously sending information about its movements to the base station.

3.3 Localization

Localization is the most important thing for a mobile robot, as it provides information about its exact position and orientation. This information can be further used for autonomously navigating and performing different tasks assigned to it. The localization problem would be solved if one can attach a GPS sensor to the robot which would inform the robot about its whereabouts. But GPS cannot be used for a robot as its accuracy is limited to few meters, which is too large for small robots and moreover GPS cannot function in indoor or obstructed environments. Thus a GPS kind of system is developed using Beacons and Listeners and the robot is localized from the base station using simple triangulation formulas.

There are two methods of localization: Absolute localization and Relative localization [7]. Absolute localization uses different approaches which are heading sensors, beacons and map matching. Relative localization has a main technique called the Dead Reckoning, in which the encoder connected to the individual wheel is used for localizing. The encoder counts of both the wheels are used for calculating the distance and angle of the robot in it. Extended Kalman filtering is also one of the modern methods used in localization. There are basically two ways of tracking a moving device in an indoor location: Active Mobile and Passive Mobile Architecture [42]. In the

Active Mobile architecture the receivers are at known locations and they estimate distances to a mobile device based on an active transmission from the device. In Passive Mobile architecture there are active beacons that periodically transmit signals to a passively listening mobile device, which in turn estimates distances to the beacons. As the active mobile architecture receives simultaneous distance estimates at multiple receivers from the mobile device, it performs better localization than the passive mobile system in which the device obtains only one distance estimate at a time and might have moved between successive estimates. Hence the active mobile architecture which is similar to Positioning Beacon System Localization is used in this thesis.

3.3.1. Active Mobile Architecture for Garcia using Cricket

Active Mobile architecture can be considered similar to GPS. But instead of receiving signals it sends the signal to the receiver which would eventually compute the distance. The active Transmitter on each robot broadcasts a RF message coupled with the ultrasonic pulse periodically on a wireless channel. The Receivers connected to the base station listen to such broadcasts and estimate the distance to the mobile. Actually, the receiver propagates this distance information to the base station which then updates the location of each mobile device. This system can be used with multiple robots as the transmitter can send its identification with the transmitted beacon pulse to the receivers. The figure 3.3 shows such an arrangement for multiple robots. This system mainly depends on geometric principles for localizing and tracking the robot. The Crickets described in the second chapter are used over here as they have ultrasonic beacon and listener capabilities along with transmitting/receiving a RF signal.

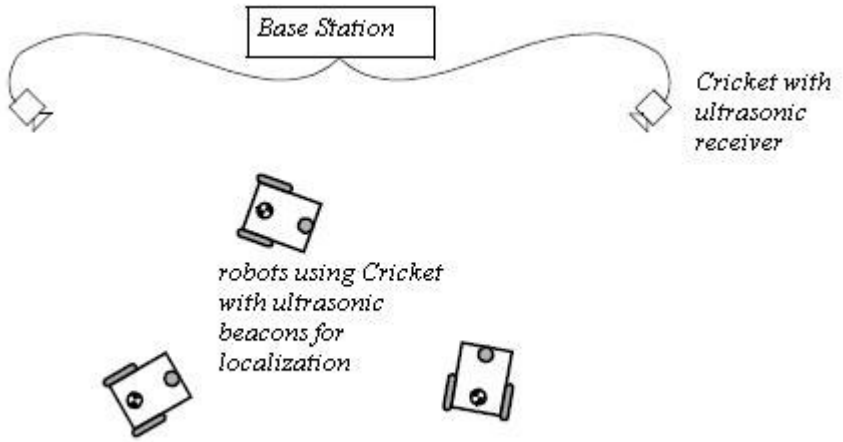

Figure 3.3 Different robots using the Active mobile architecture for localization.

The Cricket is connected to Garcia serially and the obstacle avoidance code along with a function for beacon transmission is developed. The obstacle avoidance code (Appendix A) gives the control to the beacon function for few hundred milliseconds when the cricket transmits the beacon pulse with a RF signal. The test bed is setup similar to the figure shown above. The receivers run a demo cricket code in the 'Listener' mode and are connected to the computer serially. Whenever the receivers receive the data packet, they route it to the computer. The computer runs a Labview application which takes the useful distance and ID information from the packet and uses it to localize and identify the robot. The front panel of the application is showed in Figure 3.4. The Labview application uses different modules for the steps involved in localizing the robot. The first module is used for initializing the Cricket receivers at 38400 baud rate as the Garcia Crickets run at the same baud rate. The second module is used for Deciphering the Message and ID which is received by the receiver from the robot. The Figure 3.5 shows the graphical code for the module.

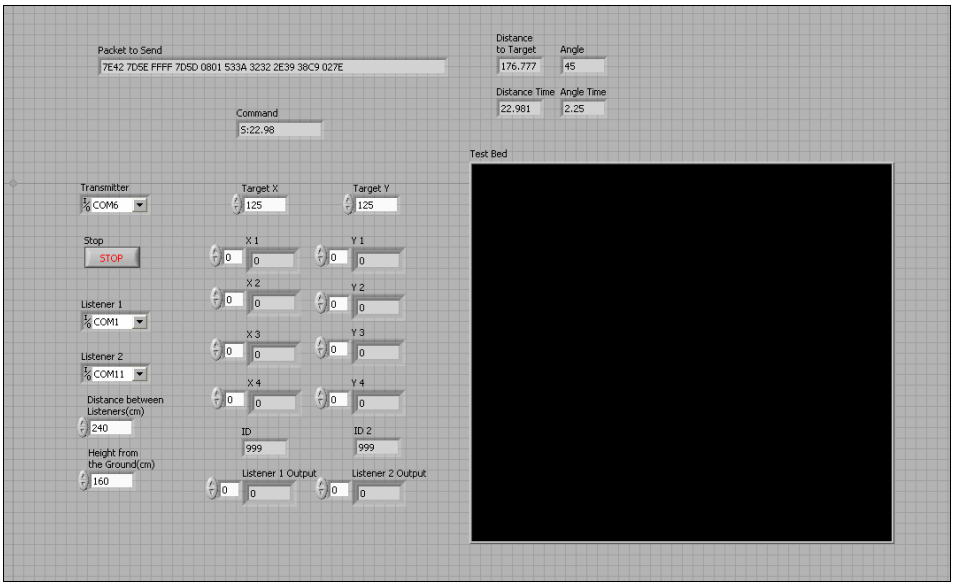

Figure 3.4 Front Panel for Localization and Path Planning Application.

3.3.1.1 Deciphering Data Packet Module

The beacon message which is received at the receiver is in the format: VR=2.0,ID=01:dd:be:be:09:00:00:95,SP=11111111,DB=224,DR=6479,TM=6789,TS=455424. The Space ID is shown as SP which can be modified for the robots as 11111111, 22222222 etc. Thus all the robots can be programmed to have a unique space id. The DB represents the distance to the beacon which is denoted in centimeters. These two values the SPID and the DB are useful for localizing and are stripped out from the message using the module showed in Figure 3.5. The triangulation method is used in this application, thus two such deciphering modules (shown in Figure 3.6) are used for the two receivers making the beacon/transmitter the third point of the triangle. The position is calculated when both the receivers receive the message from the same robot. This is because, there are many beacon messages from the multiple robots

simultaneously and if the IDs are not checked and matched; it may result in a wrong position calculation. These ID and Distance readings from both the receiver modules are stored and transferred to the next module which calculates the coordinates of the robot.

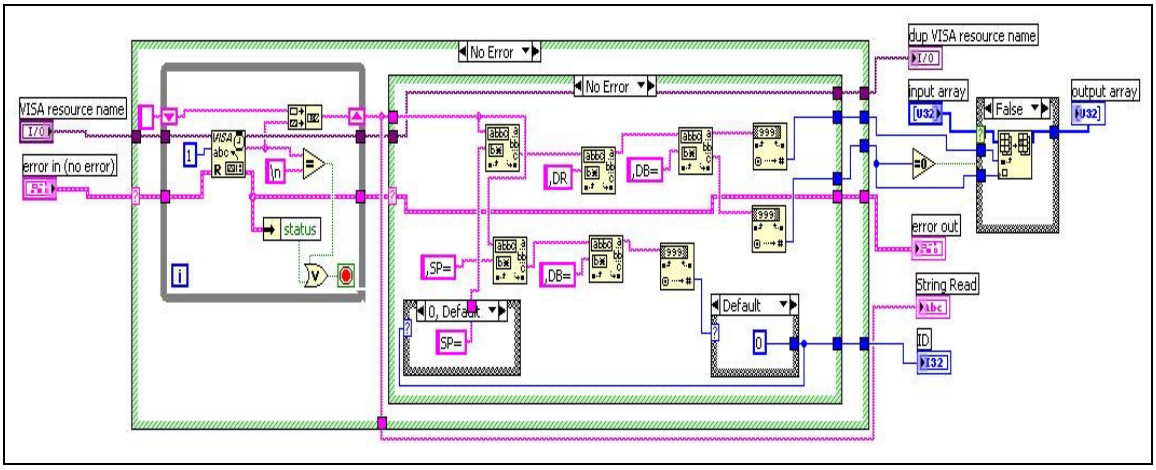

Figure 3.5 Labview code for Deciphering the Message and ID of the robot.

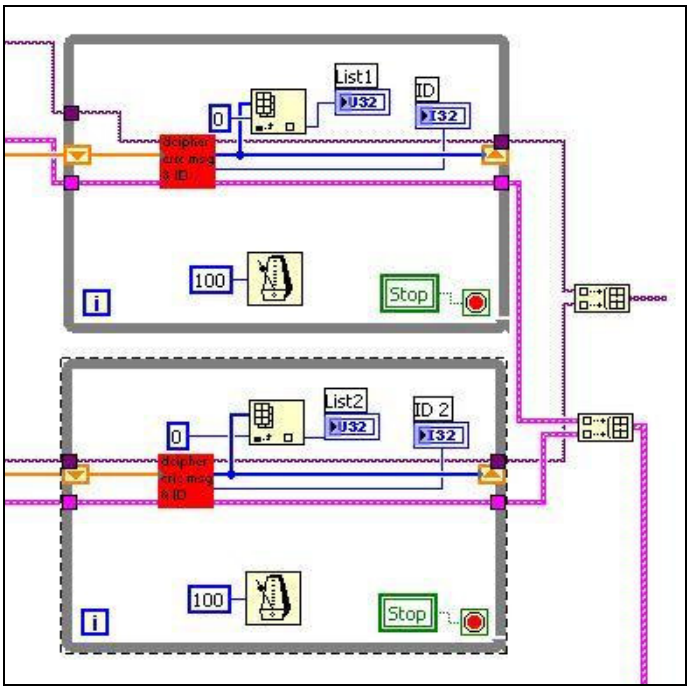

Figure 3.6 Labview code for two receivers which gives the respective IDs and Distance from beacons in List 1 and List 2.

3.3.1.2 Coordinates Calculation Module

The coordinates of the robot are calculated using the triangulation method. The setup of the test bed is shown in Figure 3.7. Here the calculation is done only in the positive quadrant. Thus, the listeners are assumed to be situated at coordinates (0,0) and (x,0) where x represents the distance between the two listeners in centimeters. The distance d1 and d2 are the distance from the beacon on the robot for both the receivers respectively. These distance values are obtained from the decipher module as explained above. Thus the coordinates can be found out using the distances from equation 3.2.

$$x_t = \frac{d_1^{\,2} - d_2^{\,2} + x^2}{2x}, \quad y_t = \left| \sqrt{d_1^{\,2} - x_t^{\,2}} \right| \tag{3.2}$$

Thus (x_t, y_t) forms the current coordinates of the robot.

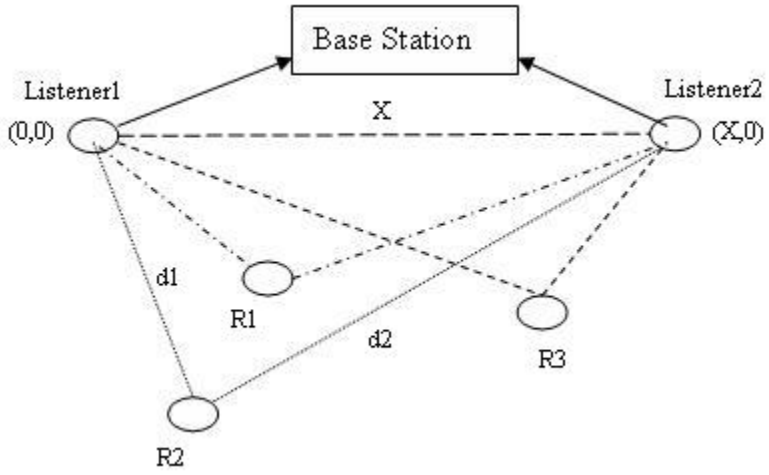

Figure 3.7 Test bed setup for the coordinates calculation of multiple robots.

The module calculates the coordinate's only if the ID from both the receivers matches, thus making the distances valid. In this way multiple robots with unique IDs

can be localized using the same module. The module gives out coordinates for all the robots respectively. The Figure 3.8 shows the module 'Cricket X, Y' that takes in the distance values with the IDs and gives out the coordinates for the robots respectively. The module is placed in a while loop that executes every 50 ms, thus updating the coordinates of the robots continuously.

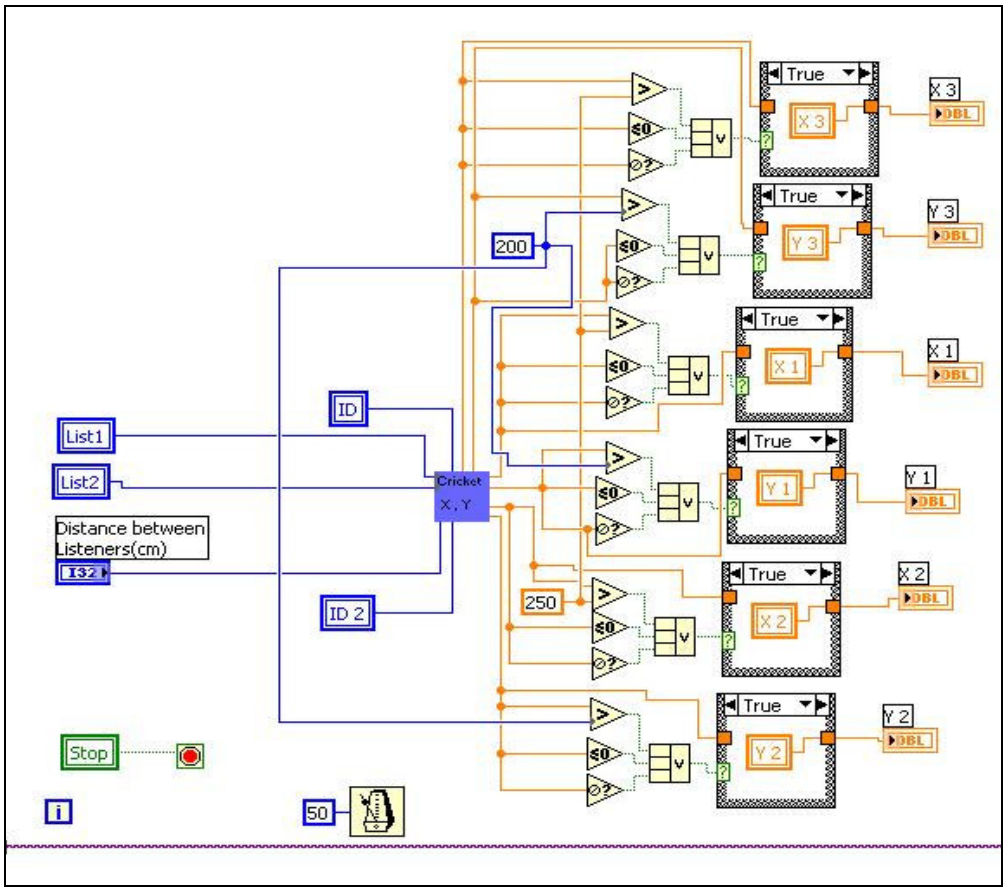

Figure 3.8 Module Cricket that calculates the coordinates of multiple robots.

3.3.2. Blimp Localization

Blimp is an indoor helium balloon which can be used for overhead surveillance, communication relay and many other applications. The controlling of Blimp wirelessly without using the remote has already been done using the Mica2 at ARRI DIAL lab.

The localization of Blimp is also an important process as it can be used for controlling and directing Blimp to do different tasks without human intervention. The same principle which is used for Garcia robot is used for localization of the Blimp.

The Blimp is localized using the same active mobile architecture and the same modules with a slight difference in the calculation of the coordinates. Here the two receivers are rested on a ground at a distance 'x' centimeters where the first receiver is at (0,0) coordinate while the second at (x,0) coordinate, thus localizing it in the positive quadrant only. The Blimp can be localized using a single Cricket Beacon as used for Garcia, but it consists of 2 Crickets instead. This is required because the orientation of Blimp cannot be determined and controlled using a single cricket. The orientation plays an important role in controlling the Blimp since without knowing the direction in which it is headed the base station cannot give a turning or a movement command to it. The two crickets on the Blimp are placed in a straight line and this line forms an angle with the X-axis depicting the orientation of the Blimp. The two crickets are localized and the midpoint of the coordinates is the location of the Blimp. It is assumed that the Blimp maintains a constant height (h). The formula to calculate the coordinates of the two crickets is the same as equation (3.2) with the only difference that the distance (d1, d2) is calculated by projecting the points on the surface using the Pythagoras theorem.

$$d_1x = \left| \sqrt{{d_1}^2 - h^2} \right|,\ d_2x = \left| \sqrt{{d_2}^2 - h^2} \right|,\quad x_1 = \frac{{d_1x}^2 - {d_2x}^2 + x^2}{2x},\ y_1 = \left| \sqrt{{d_1x}^2 - {x_1}^2} \right| \tag{3.3}$$

Thus, (x_1, y_1) gives the coordinates for the first cricket on Blimp. Similarly (x_2, y_2) is calculated for the second cricket. The midpoint (x_m, y_m) of these coordinates gives us the exact location of the Blimp.

$$x_m = \frac{x_1 + x_2}{2},\ y_m = \frac{y_1 + y_2}{2} \tag{3.4}$$

Thus, the localization of Garcia robots and Blimp is done using the Labview application. Though the coordinates are calculated only in the positive quadrant it covers a lot of area as the ultrasonic pulses can travel long distances if there are no high obstacles. This localization is further used for Path planning, for controlling and sending the Garcia's to a particular position in Discrete Event Controller for doing a particular task.

3.4 Path Planning

Path planning generates an optimal robot path using which the robot can move freely without fearing to sense any obstacles or crash into any of them [40]. Thus, if the environment remains unchanged and as expected, instead of using obstacle avoidance the robot can be programmed to follow the path or be sent the path from the base station. This has the advantage of saving power, because the sensors are not used and in addition, because the robot follows the path instead of wandering around and wasting power.

Here the path planning is short and kept simple with no use of sophisticated algorithms. As explained in obstacle avoidance, the Garcia sends a left or right

command to the base station whenever it turns left or right. This data is stored at the base station with the time stamp from the start to the end. Simultaneously the Garcia is localized and the localization points are plotted on a graph every second using a Labview code. This code module is showed in Figure 3.9.

In this code the robots are identified using the IDs from the 'Decipher' module. The calculated coordinates from the localization module are transferred to the plotting module. The coordinates of the respective robots are selected and are plotted on to a picture file. The plotting is differentiated using different color pens for different robots. The path of the robot is determined by connecting the lines between the two point coordinates of the same robot.

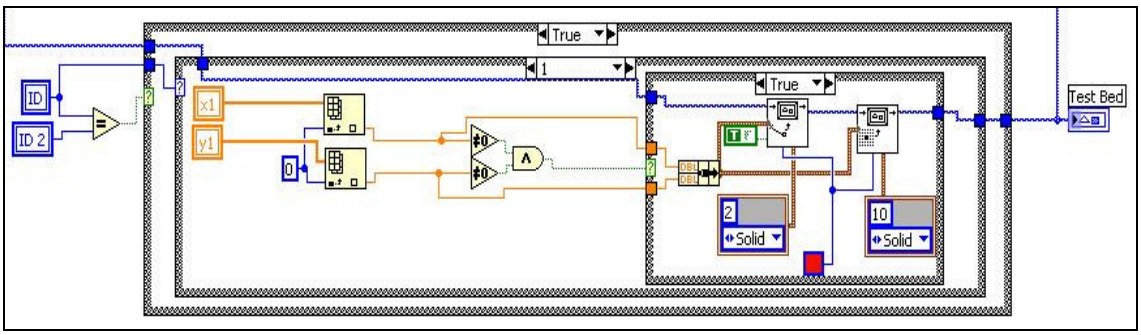

Figure 3.9 Labview code for plotting the localized points on the graph.

Thus using the data sent to the base station from the obstacle avoidance code, the coordinates found using localization and the plot formed from the coordinates an optimal path is determined. The optimal path in this case is a set of commands that the Garcia robot is given to follow. The commands dispatched are generally to go straight for a particular distance and turn left or right for a particular angle. The distances and angles are converted into time as the Cricket or Mica2 connected to the Garcia can give

a command to it and run a timer with that particular time before stopping. Garcia's forward and turning speed is kept constant according to which the path times are calculated. Thus after reaching the end goal from the starting position using the obstacle avoidance code and localization, the base station calculates and stores the path information and sends it to the same robot if the task is to be implemented again or to other robots if they require following the same path.

This kind of path planning is used in discrete event controller in case a robot comes into an environment and requires a path from the entry to exit. A robot already having the path information would travel to the intruder robot and verify if it is allowed an entry into its environment. If the verification is done the robot would send the complete path information for traversing from or to move into the environment. Also by using heat, light or chemical determining sensors and path planning, a robot path can be formed which would avoid the heat, dark or dangerous chemicals. These paths then can be used by the robot for human guidance in industries or buildings where caution is required against such types of hazards.

CHAPTER 4

TASK PLANNING AND DYNAMIC RESOURCE ALLOCATION

4.1 Introduction

Mobile sensing robots (Garcia) when used in conjunction with stationary sensing nodes (Cricket/Mica2) can greatly extend the application domains of static sensor networks [10, 39, 41]. In particular the ability to automatically adapt the topology of the network to the new operating conditions can be used to perform optimal node deployment, adaptive sampling, network repair and event detection [17]. In this perspective a mobile sensor network can be considered as a multi-robot system with heterogeneous resources. The coordination of a multi-robot system for the execution of cooperative tasks is a very active research field [9, 13, 21]. In particular, the related literature has extensively tackled the problem of dynamic resource assignment, to on-line allocate a resource to a task according to predefined criteria. However in most cases the robot team is in charge of executing one mission at a time so that no shared resource conflicts arise. If multiple missions are executed simultaneously, then system deadlock or conflicts might arise if the resource assignment is not properly made [18, 34].

The main objective of this chapter is to implement dynamic coordination of a mobile sensor network in presence of multiple missions. Therefore it is necessary to

develop a control strategy in charge of simultaneously performing dynamic resource assignment and solving on-line shared resource conflicts.

In [22, 23], a matrix-based Discrete Event Controller (DEC) [44] has been used for sequencing multiple missions in simple mobile sensor networks (MSN) (which can be considered as a particular multi-robot system). However, since the peculiarities of MSN require multiple competing missions and network topology changes, including mobility and addition/removal of nodes, dynamic resource assignment algorithms should be included in the framework of the DEC.

In this chapter, a novel approach is presented to implement on-line deadlock-free resource assignment for multi-robot systems with multiple missions. At each event occurrence, when resource changes are required, a greedy algorithm is first implemented by on-line updating of the resource requirements matrix [28]. The new resource assignment is accepted if it is compatible with a certain MAXWIP deadlock avoidance policy specified herein. Specifically, the discrete event system representing the new mission plan has to satisfy two conditions: After the implementation of the new resource assignment, it is necessary to guarantee that (1) the system is not already in deadlock and (2) the new system is regular in a sense described herein. In order to check the latter condition the regularity test [25] is launched every time a new resource assignment is proposed. Having produced an allowable assignment of resources, the Discrete Event Controller (DEC) is implemented to assign the next tasks in the multiple missions based on priority assignment policies.

4.2 Resource Assignment and Deadlock Avoidance: Matrix Formulation

The matrix approach in the discrete event controller [31, 44] provides a rigorous, yet an intuitive mathematical framework to represent the dynamic evolution of DE systems according to linguistic if-then rules:

Rule i: If <*conditions*^{i} hold > then <*consequences*^{i}>

For a mobile sensor network, mission planning can be done as follows:

Rule i: If <*robotic sensor 1* has completed *task 3* of *mission 5* and *robotic sensor 2* is available> then <*robotic sensor 2* starts *task 4* of *mission 5* and *robotic sensor 1* starts *task 3* of *mission 2*>.

The DEC describes if-then rules using two different sets of logical equations. One for checking the conditions for the activation of rule *i* (matrix controller state equation), and one for defining the consequences of the activation of rule *i* (matrix controller output equation). All the matrix operations in this case are defined to be in or/and algebra, where + denotes logical 'or' and 'times' denotes logical 'and', the over-bar denotes logical negation. The matrix controller state equation is:

$$\overline{x} = F_v \overline{v} + F_r \overline{r} + F_u \overline{u} + F_{ud} \overline{u}_d \tag{4.1}$$

where F_v is the task sequencing matrix, F_r is the resource requirements matrix, F_u is the input matrix, F_{ud} is the conflict resolution matrix and u_d is the conflict resolution vector. The current status of the DE system includes the rule vector *x*, whose entries of '1' represent rule currently activated, task vector *v*, whose entries of '1' represent 'completed task', resource vector *r*, whose entries of '1' represent 'resource currently available', and the input vector *u*, whose entries of 1 represent the occurrence

of a certain predefined event which triggers the corresponding mission. The over-bar in equation (4.1) denotes logical negation so that tasks completed or resources released are represented by '0' entries.

The activated rules determine the commands to the multi-robot system that the DEC has to sequence in the next iteration, according to the matrix controller output equations:

$$v = S_v x \tag{4.2}$$

$$r = S_r x \tag{4.3}$$

$$y = S_y x \tag{4.4}$$

S_v is the task start matrix, S_r is the resource release matrix and S_y is the output matrix, y is the vector of outputs (completed missions). The task start equation (4.2) computes which tasks are activated and may be started, the resource release equation (4.3) computes which resources should be released (due to completed tasks) and the mission completion equation (4.4) computes which missions have been successfully completed. The complexity (size and operability) of the DEC is dependent on the number of nodes and tasks. The dimension of the matrix involved in the DEC is linear in the number of nodes and quadratic in the number of jobs.

The implementation of dynamic resource assignment algorithms is equivalent to updating matrices F_r and S_r [22, 23] to have '1's in the entries corresponding to the selected task-resource assignment which selects the most efficient resource to perform each task. Then, the DEC equations (4.1)-(4.4) are computed to determine which tasks are to be started and which resources are to be reset.

4.2.1. Complete Dynamical Description

To provide a complete dynamical description of the DE system, we define the following quantities (equivalent to the marking vector), the output incidence matrix and input incidence matrix of a PN, $m(t) = [u(t)', v(t)', r(t)', u_d(t)']'$, $S = [S_u', S_v', S_r', S_{u_d}', S_y']'$, $F = [F_u', F_v', F_r', F_{u_d}', F_y']'$

where *t* represents time. Then, in order to take into account the time durations of the tasks and the time required for resource releases, we can split *m(t)* into two vectors, one representing available resources and current finished tasks ($m_a(t)$) and the other representing the tasks in progress and busy resources ($m_p(t)$)

$$m(t) = m_a(t) + m_p(t) \tag{4.5}$$

As a consequence, considering equations 4.1-4.4 which represent the rule-base of our DE supervisory controller, we have

$$m_a(t+1) = m_a(t) - F' \cdot x(t) \tag{4.6}$$

$$m_p(t+1) = m_p(t) + S \cdot x(t) \tag{4.7}$$

When a rule is activated (equation 4.1) some tasks end and some resources become available (equation 4.6), whereas some other tasks start and some other resources become busy (equation 4.7). When a transition fires, a token is moved from $m_p(t)$ to $m_a(t)$ where it may be used to fire subsequent transitions. Splitting the marking vector *m(t)* into $m_a(t)$ and $m_p(t)$ holds the key to dynamic resource assignment. Equations (4.1), (4.6) and (4.7) represent a complete description of the dynamical

behavior of the discrete event system and can be implemented for the purposes of computer simulations using any programming language (e.g. MATLAB® or C).

4.2.2. One Step Look-Ahead Deadlock Avoidance Policy

The presented matrix constructions can be efficiently used to implement deadlock avoidance policies for discrete event multi-robot systems. The following assumptions are generally made:

1. No resource fails during a mission.
2. A resource always completes its current task before starting a new one.
3. Every resource performs one task at a time.
4. After the task is completed, the resource is immediately available for a new task.
5. Each task requires one resource to be executed.
6. The system is regular.

For any two resources r_i and r_j, r_i is said to wait for r_j, denoted $r_i \rightarrow r_j$, if the availability of r_j is an immediate requirements for the release of r_i. Circular waits (CW) among resources are a set of resources r_a, r_b,…, r_w whose wait relationship among them are $r_a \rightarrow r_b \rightarrow \ldots \rightarrow r_w$ and $r_w \rightarrow r_a$. The simple circular waits (sCW) are primitive CWs which do not contain other CWs. For a complete analysis of the deadlock structures, not only the sCWs but all the CWs need to be identified.

In order to avoid deadlocks, we have to monitor those tasks of the MSN whose completion activate rules which consume resources in a CW. The task set of a CW C, $J(C)$, is the set of tasks which need at least one of the resources of C to be started.

Under the assumptions previously presented, a deadlock condition occurs if and only if there is an empty circular wait [31, 47]. For these systems, an empty CW can only be caused by activation of tasks of the corresponding critical subsystem. Using the matrix formulation of (4.1)–(4.4) and some matrix manipulations, we can come up with a compact matrix representation of critical subsystems J_o as follows [25]:

$$J_o = ({}_dCF_v) \wedge (C_d F_v) = (C_d S_v^T) \wedge (C_d F_v) \tag{4.8}$$

where each entry of '1' in position (i, j) means that task j is included in the critical subsystem of CW i. ${}_dC$ and C_d are called the input and output rules of a CW and have '1' entry in position (i, j) if the j^{th} rule increases or reduces the number of available resources in the i^{th} CW respectively.

A simple deadlock avoidance strategy consists in not allowing the number of activated tasks of the critical subsystem to become equal to or greater than the number of available resources in the i^{th} CW C_i (MAXWIP policy [31, 33]).

$$m(J_o(C_i)) < m_o(C_i) \tag{4.9}$$

Therefore, we can conveniently update the conflict resolution input u_d to inhibit rules which, if activated, would violate condition (4.9) and lead to deadlock conditions.

Our dispatching policy follows three main steps:

(i). Based on the structure of the system defined by matrices F and S, we calculate the CWs, their corresponding critical subsystems and the number of available resources $m_o(C_i)$ in the i^{th} CW C_i.

(ii). For every DE-iteration, we calculate from the current marking vector, $m_{current}$, the corresponding possible successor-marking vector, $m_{possible}$. Equation (4.6)

provides this possible successor $m_a(t+1)=m_{possible}$; $m_a(t)=m_{current}$; $m_{possible}$ is readjusted keeping into account the possible shared resource conflicts (on-line computation).

(iii). If the selected $m_{possible}$ does not satisfy condition (4.9), then it is necessary to eliminate the task that is attempting to cause a deadlock, inhibiting the corresponding rule. This is done by conveniently updating vector u_d. Then the algorithm restarts from step 2 (on-line computation).

4.3 Deadlock-Free Dynamic Resource Assignments

The concept of dynamic resource assignment is highly substantial in multi-robot systems to adapt to unstructured and dynamic environments [21]. In other words, to improve the coordination of a mobile sensor network, it is necessary to continuously update the set of matrices defined in the DEC based on new environmental conditions. To cast the dynamic selection of resources most appropriate for a task into the DEC format, one may use the Greedy activity/resource selector algorithm [12] to on-line modify the resource assignment matrix F_r as follows.

For each task that has a choice of resources to use, a Dynamic Priority Assignment Matrix (DPAM) is defined according to the example:

$$D_c = \begin{array}{r} \\ task\,1 \\ task\,2 \\ task\,3 \end{array} \begin{array}{ccc} res.1 & res.2 & res.3 \\ 0 & 1 & 0.7 \\ 1 & 0 & 0.5 \\ 0.2 & 0 & 1 \end{array}$$

which indicates that task 1 may be efficiently performed by resource 2, or less efficiently by resource 3. The numerical entry in position *(i, j)* is between '0' and '1',

and it indicates the efficiency with which resource *j* performs task *i*, with '0' indicating that resource *j* cannot perform task *i*, and '1' indicating that resource *j* performs task *i* with maximum efficiency. Note that this matrix indicates that task 1 may be performed with either resource 2 or resource 3, in contrast to the matrix F_r, where multiple entries of '1' in a row indicate that all those resources are required for that task.

According to greedy dispatching policies [21], one selects the resource to perform a given task according to the immediate 1-step look ahead maximum payoff. The algorithm looks for an ideal resource for a particular task. If it does not find a resource, it waits for a resource that is most suitable and available for that particular task. The task is not started till the resource is found. Therefore, depending on the DPAM, at each event step, for the free-choice tasks, the resource matrix F_r is modified to have 1's in the entries corresponding to the maximum values of the DPAM in each row. This effectively selects the current most efficient resource to perform each task. Then, the DEC equations (4.1)-(4.4) are computed to determine which tasks need to be started and which resources need to be reset.

After the tasks have been performed, the DPAM is dynamically updated based on the evaluation information from the task on how well the assigned resource had performed. Thus, the resources that perform well would be assigned next time to that task. The implementation of this resource assignment policy optimizes the single association resource/task without taking into account the global effect of all the associations. In multi-mission systems this way of proceeding may lead to the shared resource conflicts and deadlocks.

In [22], the implementation of the deadlock avoidance policy assumed a fixed structure of the matrices *F* and *S*. However the MAXWIP deadlock avoidance policy presented in Section 4.2 is based only on the current status of the system, i.e. on the current configuration of matrices *F* and *S*. Therefore it is possible after every iteration, to update the resource requirement matrices of the DEC using the DPAM and then check if the resource assignment conflicts with the MAXWIP deadlock avoidance policy. If a conflict arises, then a new resource assignment is presented until the requirements of the MAXWIP policy are met. Figure 4.1 shows a flowchart representing the procedure to update the DEC to adapt to the changing operating conditions while guaranteeing a deadlock free dispatching.

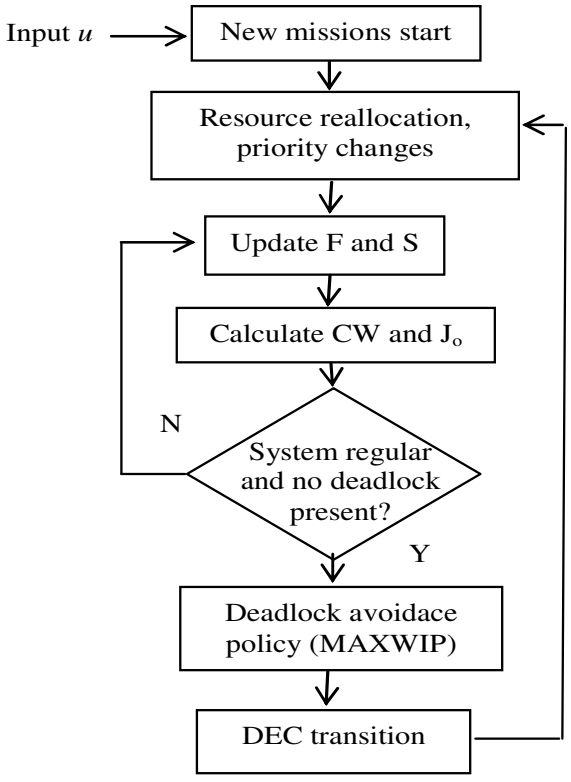

Figure 4.1 Flow chart representation of the deadlock-free dynamic resource assignment algorithm.

The procedure is as follows:

1) After a new transition of the DEC, if a resource reallocation or a priority change is needed, the mission plans can be accordingly updated by redefining matrices F_r, S_r and F_{ud} through a human operator or an automatic decision making algorithm (DPAM). The only constraint to be observed is that it is not possible to reassign the resource to a task currently in progress.

2) The new set of circular waits CW and the new sets of critical subsystem J_o are calculated. At this point, before applying the MAXWIP policy, two conditions must be met.

First of all, Gurel's regularity test [25] is applied, in order to be sure that no key resources [33] are present in the system after applying the new resource assignment. The output of the Gurel's algorithm is a matrix Res_{cw} which is a matrix of critical resources. There is a certain pathological case that requires extreme care in the process of deadlock avoidance and dispatching. This situation is called second level deadlock (SLD) [19, 20]. SLD is not a circular wait even if it necessarily evolves into a deadlock in the near future. SLD exists on the presence of critical resources known as bottlenecks and key resources. Bottleneck resources are identified by analyzing interconnectivities in circular wait relationships.

Secondly, it has to be ensured that the system is not already in a deadlock situation.

If at least one of these two conditions is not satisfied then another resource assignment attempt is made and algorithm restarts from step 1.

Thirdly, if the system is regular and not currently in deadlock, the new resource assignment is actually implemented. The MAXWIP policy is compatible with the new system.

This procedure can be seen as a constrained optimization in resource assignment. It ensures that all the tasks are performed using the best resources available which do not cause the occurrences of deadlocks.

CHAPTER 5

SIMULATION AND IMPLEMENTATION

5.1 Controlling the Localized multiple Garcia Robots

Chapter 3 discussed the method to use Crickets for localizing multiple Garcia robots from the Base station. Here, in this chapter, these Garcia's are controlled and directed from the Base station to reach a particular position and do various tasks. The Crickets used as Beacons for localization are used for receiving the commands from the base station. These commands are then given to Garcia to execute.

The Garcia robots need to be controlled in order to perform various tasks at different positions. The current coordinates and the desired or target coordinates need to be known for performing a task at that particular position. The current location of the robots can be known using the 'Cricket Coordinates' module discussed in the Chapter 3. The target coordinates where a task has to be executed can be entered by the user beforehand or can be known dynamically by the application through the same 'Beacon/Listener' principle using the 'Cricket Coordinates' module. Here, the Labview application uses another code module which calculates the distance and angle from the current position to the target position. The module uses the distance formula (Equation 5.1) to calculate the distance between the two coordinates and the angle is calculated by implementing the code shown in Appendix B in Labview. This module is named 'Distance and Angle'.

$$d = |\sqrt{(x - x_t)^2 + (y - y_t)^2}| \quad (5.1)$$

Where'd' is the distance between current coordinates *(x,y)* and target coordinates *(x_t, y_t)*.

5.1.1. Garcia Control Module

The 'Distance and Angle' module is called in another module 'Garcia Control' which sends maneuvering commands to Garcia robots according to the calculations of distance and angle. The code used in the module is showed in Figure 5.1. The module functions as: The target coordinates for all the robots are identified before running the module. The ID is checked and the current coordinate of that robot with the specified target coordinate for that robot is passed in the module. This helps in using the same module for multiple numbers of robots. The module first checks whether the current and the target coordinates are within 10 centimeters of each other. If it is so, then no maneuvering commands need to be sent and only a Beacon command in format 'ID B' is broadcast. The ID number cycles for all the robots. Now, at the Garcia end, the Cricket code has a provision for the Beacon command. The robot with the matching ID number executes the command and transmits the Beacon pulse which eventually is used to calculate its current coordinates. Thus a continuous check is kept on current coordinates of all the robots.

If the current and target coordinates are not in the 10 centimeter range, the 'Distance and Angle' code module is called to calculate the exact distance and the angle between the two coordinates. The straight line speed and the turning speed are kept

constant at ’07’ and ‘03’ respectively for all Garcia robots. This helps in finding a constant which would convert the distance and angle numbers into corresponding time.

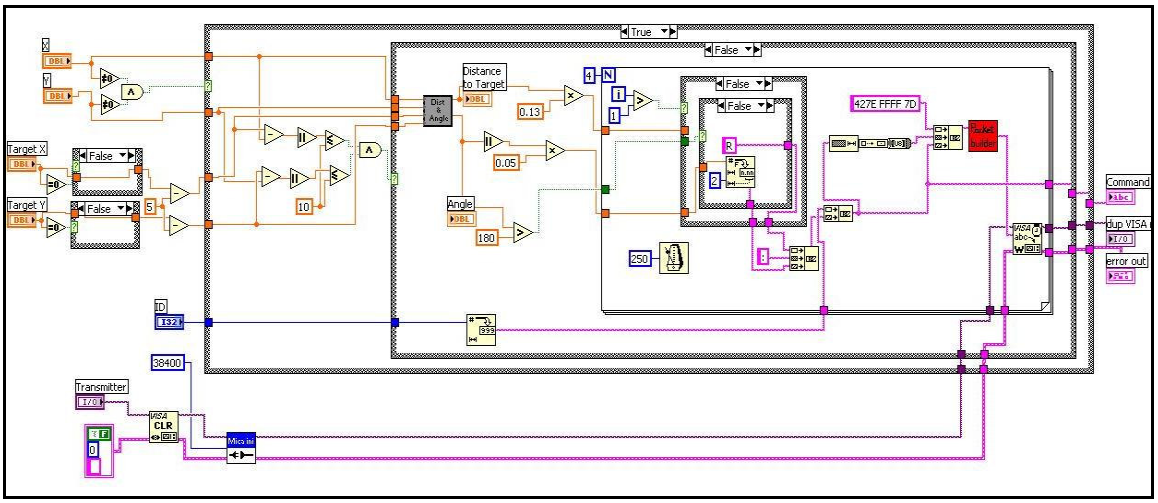

Figure 5.1 Garcia Control module which gives maneuvering commands to the robot to reach the target coordinates.

As discussed in the Chapter 3, the Cricket/Mica2 runs timers for giving the maneuvering commands to Garcia. Thus, for this reason the distance and angle output numbers need to be converted into time. The constant numbers are ‘0.13’ for ‘07’ straight line speed and ‘0.05’ for ‘03’ turning speed. These constants are obtained by calculating the time Garcia takes to reach 50, 100, 200 and 250 centimeters distance at the given speed and 90, 180, 270 and 360 degrees rotation at the turning speed. So, the distance and angle numbers are converted into the time it would take for Garcia robot to reach the target from the current position.

The calculated times are embedded into commands to be sent to the Garcia. Firstly, the turning command is sent which is in the format: ‘ID L/R:Time’. The ID is for the robot which has to execute the command. ‘L/R’ specifies whether the robot has

to turn Left or Right depending on the current orientation and 'Time' specifies the time calculated to turn for the calculated angle. The angle calculated from the code is from right to left, so instead of using only the 'right' command the 'left' command is generated if the angle is greater than 180 degrees (18 seconds). After the turning command the 'Straight' command is given to the Garcia robot. The format of this command is: 'ID S:Time', where ID is for the specific robot, 'S:Time' for going straight for specified time to reach the target. The commands formed are embedded into a packet which is suitable for transmission. The packet is formed using the 'Packet Builder' module which takes care of the header bytes, payload (command) and CRC calculation required for communication between Crickets or Mica2s. The required information for building a packet suitable for communication is given in Appendix C. The Base station transmits these commands repeatedly for every robot through a Cricket which is used as a transmitter until the robot reaches the specified destination.

At the Garcia end, the Cricket connected to it receives the command and processes it. If the ID specified in the command matches to its own, it identifies the turning command and calls the particular function for it. The Garcia starts turning and simultaneously the timer starts for the particular time sent in the command from the base station. As the timer expires the stop command is sent to Garcia, thus leaving the Garcia pointing in the direction of the target coordinate. Next, on receiving the move straight command the same procedure as explained above is followed with a straight command instead of the turning command. After reaching the destination the Garcia turns back to its original orientation by using the same time used for turning, before

performing the task it is meant to do and receiving other commands from the base station. The orientation has to be maintained as the Beacons and Listeners work on line of sight principle. The receive function of the code is shown in Appendix D.

Thus, the controlling of Garcia from the Base station using Crickets is designed for the Discrete Event Controller which is explained further in the chapter. The Dynamic Allocation of Resources is also done using the current robot coordinates and the nearby target coordinates. The controlling can also be used for independent applications where a user can specify the location to do certain task as retrieving data from a remote sensor or getting a visual with a camera installed on the robot for security purpose.

5.2 Simulation Results for Dynamic Resource Allocation

Matlab simulation results are shown here to illustrate the proposed control approach in Chapter 4. Two missions have been implemented for a sensor network composed of 7 mobile robotic sensors. The two missions encompass a sequence of 4 and 7 tasks respectively. The proposed algorithm can be easily extended to networks with several resources.

5.2.1. Resource Assignment – Attempt 1

Consider Figure 5.2. Suppose that at a certain instant of time, a new resource assignment is performed and the resource requirement matrix is accordingly updated. Also, suppose that the resources *R1*, *R2*, *R3* and *R4* are currently busy, i.e. the resource vector is r = [0 0 0 0 1 1 1].

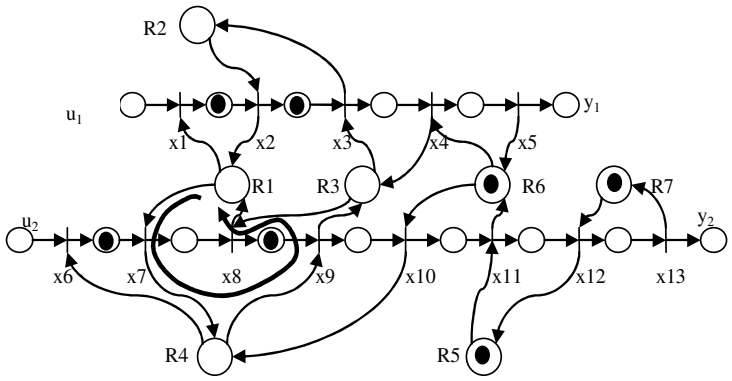

Figure 5.2 Petri net representation of the system after attempt 1.

The new resource matrix F_r would then be:

$$F_r' = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Using the equation presented in previous chapter one can calculate the matrices of circular waits, critical subsystems and critical resources respectively corresponding to the new resource assignment:

$$CW = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \quad J_o = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathrm{Re}\, s_{cw} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Analyzing vector *r* and matrix *CW*, it results that resources R1, R3, and R4, which compose the first circular wait (first row of matrix CW), are all busy. If this resource assignment was accepted, the circular wait would be empty and the system

would be in a deadlock (Figure 5.3). For the sake of clarity Figure 5.2 reports the Petri Net corresponding to this first candidate resource assignment highlighting the empty circular wait.

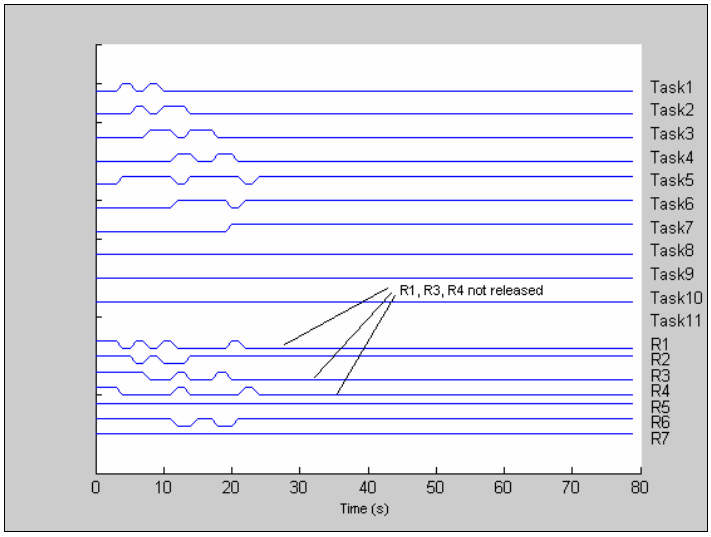

Figure 5.3 Event time trace resource assignment attempt 1: system is in deadlock.

5.2.2. Resource Assignment – Attempt 2

The greedy algorithm makes a second attempt for a new resource assignment. The resource requirement matrix, the circular waits, critical subsystems and critical resources become:

$$F_r' = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$CW = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 \end{bmatrix} \quad J_o = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

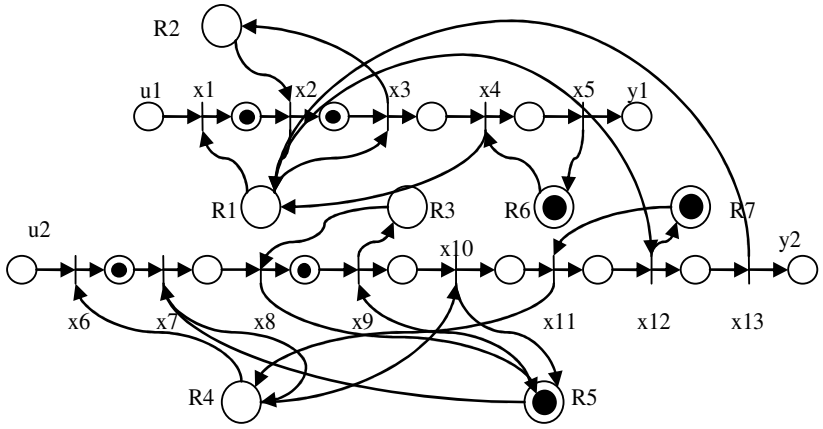

Figure 5.4 Petri net after second attempt of resource assignment: System is not regular.

Analyzing vector *r* and matrix *CW*, it is possible to note that this assignment would result in a deadlock (comparing with the corresponding Petri Net in Figure 5.4 for tasks 1 and 2). Also, by applying the Gurel's test for regularity to calculate the matrix of critical resources Res_{cw} for the new candidate system, it results that R_4 is a critical resource.

$$\mathrm{Re}\, s_{cw} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The system is therefore irregular and a second order deadlock is present: the MAXWIP policy cannot be applied.

5.2.3. Resource Assignment – Attempt 3

Therefore the proposed resource assignment is discarded and the greedy algorithm comes up with a new assignment (Figure 5.5).

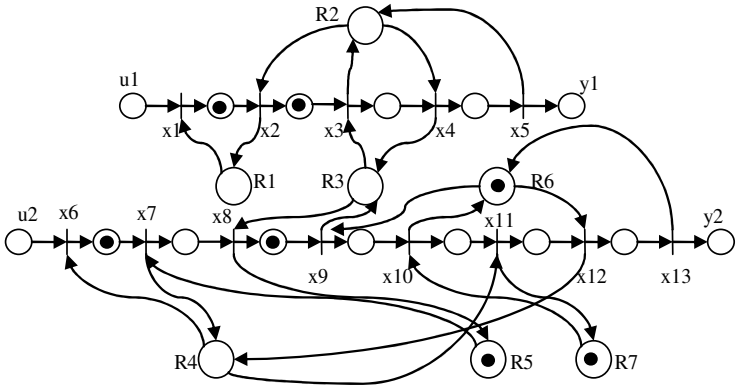

Figure 5.5 Resource assignment attempt 3: all requirements are met, the new configuration is accepted.

$$F_r' = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

The circular waits, critical subsystems and critical resources in this case are:

$$CW = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad J_o = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

$$\mathrm{Re}\, s_{cw} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The obtained system is regular (Res_{cw} matrix) and does not present empty circular waits (compare vector r and matrix C_w). This configuration is accepted and a successful implementation of the tasks with the new resource assignment can finally take place (Figure 5.6).

A good application for this technique is monitoring of a warehouse using ground sensors and mobile robots. Appropriate allocation of resources is critical in such applications.

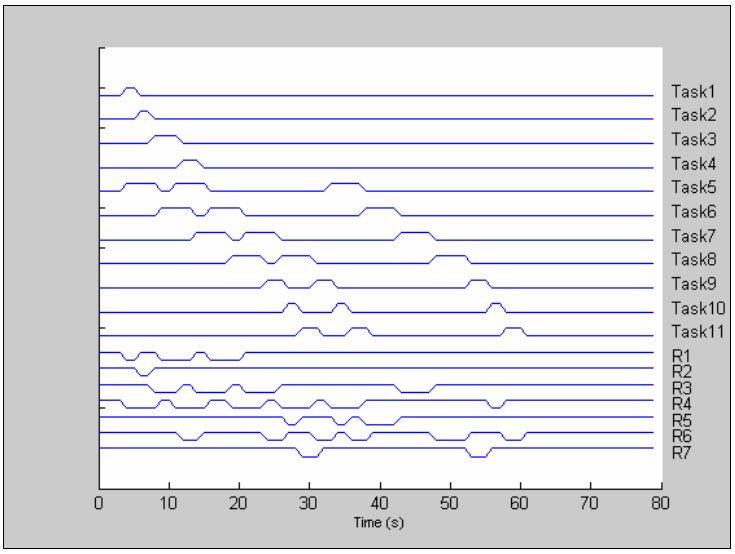

Figure 5.6 Event time trace resource assignment attempt 3: deadlock free dynamic resource assignment.

5.3 Matrix Formulation and Petri-Nets in Labview

The Matrix formulation as described in Chapter 4 is a novel approach for implementing a Discrete Event Controller for sequencing the jobs and assigning resources. The important features of matrix formulation are that it uses logical algebra and it can describe resource requirements and task sequencing. These can be modified if there are changes in task requirements and resource availability. This makes matrix formulations flexible and reconfigurable. A Petri net is a graphical and mathematical modeling tool which can be used as a supervisory control [27]. It consists of tasks, transitions, and arcs that connect both. Input arcs connect the tasks with transitions, while output arcs start at a transition and end at a task. Thus, using the matrices and the Petri nets, one can describe a complete Discrete Event System [20, 34, 38].

The matrix formulation and drawing a PN diagram is an arduous task and can contain human error in it. To avoid this, a toolkit is developed using the Labview software that would formulate the required matrices and draw a corresponding Petri-net for the complete system. The toolkit designed is capable of formulating the matrices for the MRF (Multi Reentrant Flow line Systems) [5, 18, 31, 41, 49] and FMRF (Free-choice MRF) systems. The MRF systems are capable of using shared resources for multiple tasks but do not have a decision to make, while the FMRF [37] systems have multiple choices of resources for that particular task and have to decide on a resource or multiple resources that can be used to perform the task. The matrices formulated for the system use Equations (4.1-4.4) and are: F_v - the task sequencing matrix, F_r - the resource requirements matrix, S_v - the task start matrix, S_r - the resource release matrix

and F_{ud} - the conflict resolution matrix. The input matrix F_u and the output matrix S_y form the complete F and S matrix vector which are further used in Discrete Event Controller.

The Figure 5.7 shows the front panel of the Matrix formulation code for MRF systems. It shows the inputs required and the matrix outputs' formed using it. The user has to input: The number of missions, number of tasks per missions, number of resources and resources assigned for each task. The F and S matrices are also formed using the input and output matrices, but are not showed in the figure.

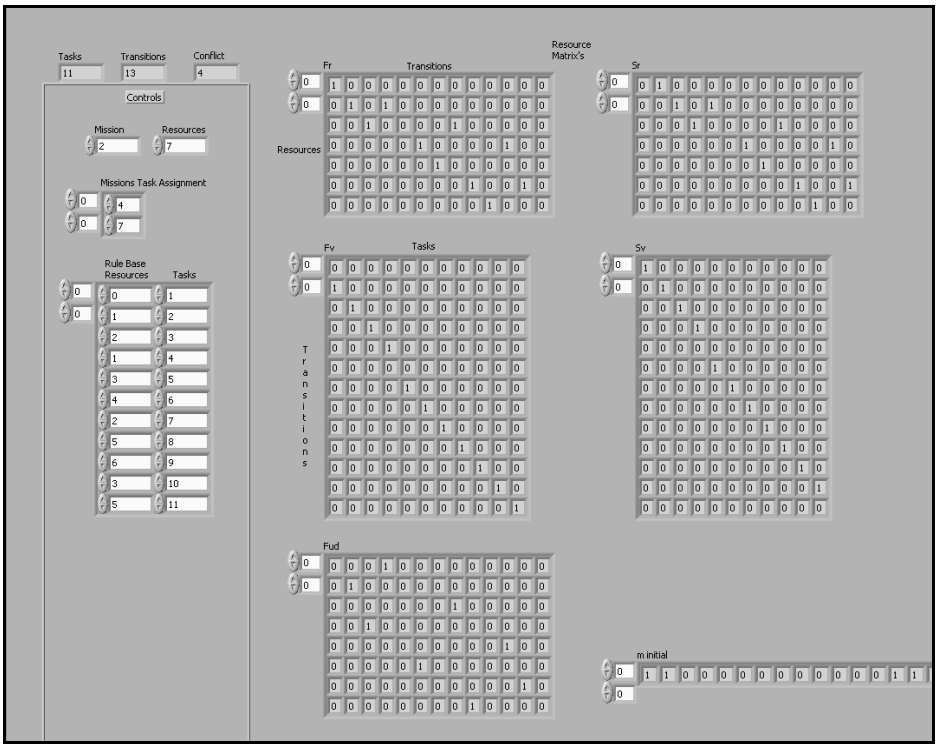

Figure 5.7 Front Panel: Matrix Formulation Toolkit.

The FMRF system's front panel is similar to Figure 5.7 with additional inputs required for decision at various tasks and the resources used for those tasks. The code

required for developing the matrices is shown in Figure 5.8. Here the code has provision for calculating the total tasks, transitions, conflicts in the system and the matrix size along with the matrix formation.

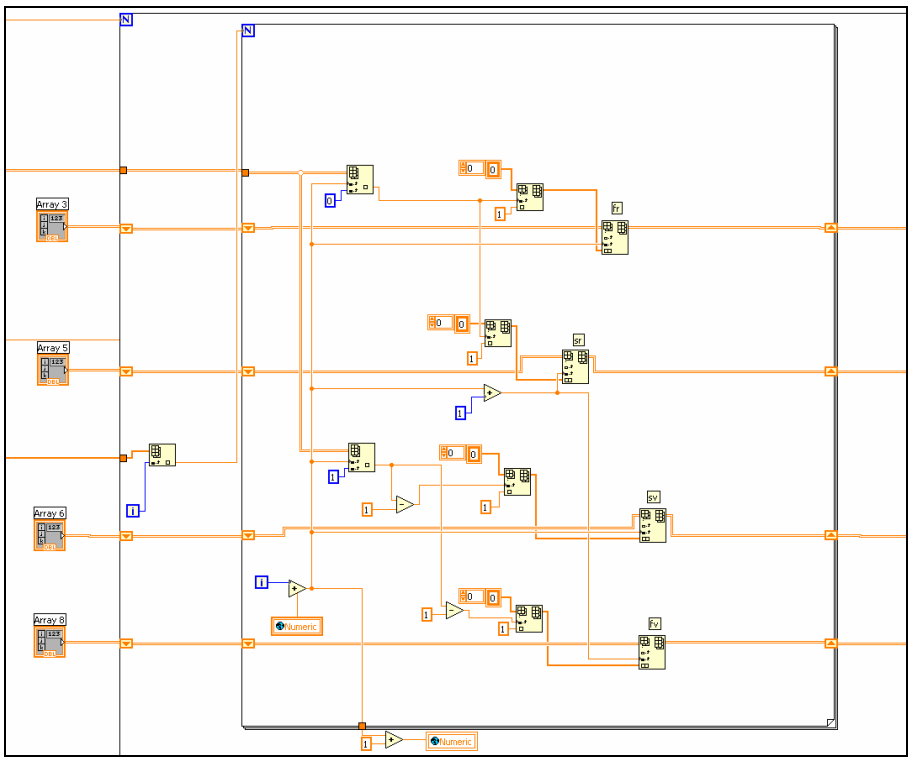

Figure 5.8: Matrix Formulation Code for the MRF system.

The FMRF Matrix formulation code and the Petri-Net formation code are too long and complicated to be discussed and displayed in this thesis. The Petri-nets formed for the MRF and FMRF systems are showed in Figure 5.9. The Petri-net toolkit uses the same inputs as used for matrix formulation and draws the graphical representation of the system. It shows various missions and the tasks for that mission respectively, the resource allocation and release for the tasks, the inputs-outputs and transition numbering according to the system.

The toolkit thus developed makes it easy for a user to design various discrete event systems to find out the respective matrices and conflicts for the system without writing and drawing it and use it further wherever it is needed.

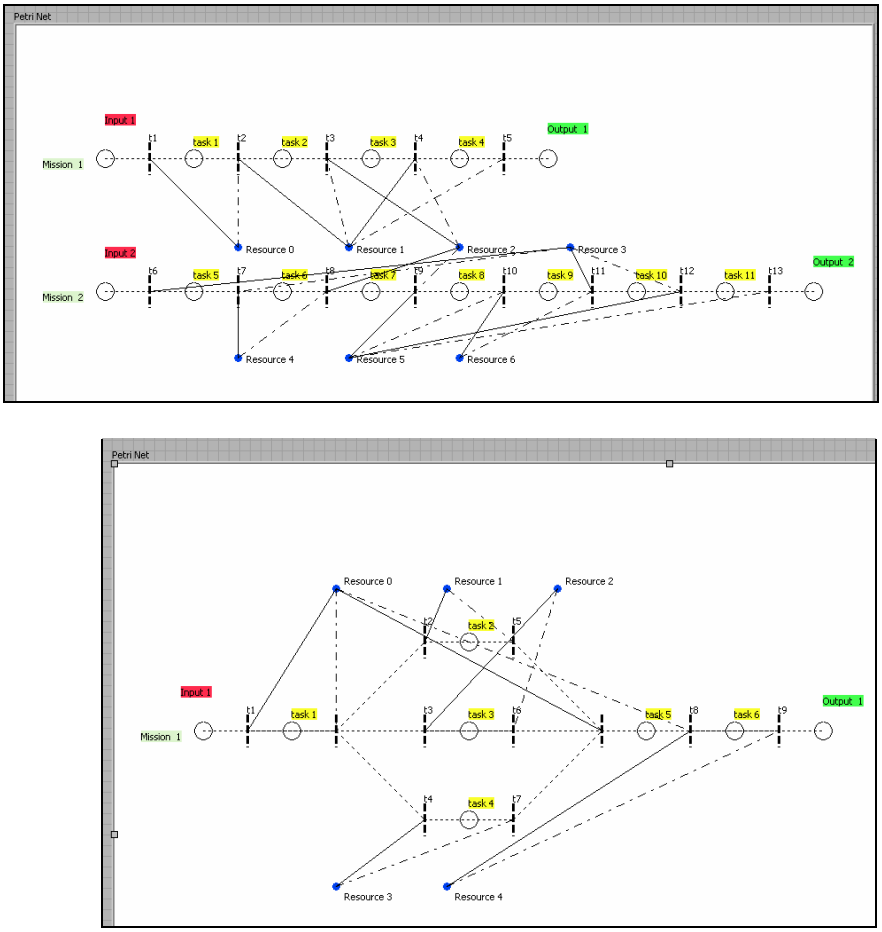

Figure 5.9: Petri-Nets for the MRF and FMRF systems.

5.4 Dynamic Discrete Event Controller

This section discusses the implementation of Discrete Event Controller and Dynamic Allocation of the Resources. The theory for Discrete Event Controller with deadlock avoidance and dynamic allocation of resources was explained in Chapter 4 and the simulation of DEC with dynamic allocation of resources using Matlab was

explained in Section 5.2. Here a simple example is implemented where if an event is triggered; the Garcia robots move to a specified location of a sensor and retrieve the data from it. The example works as follows: There are six ground sensors and three resources in the system; the setup is shown in the Figure 5.10. The sensors are placed as shown in the figure and thus forming two missions with three tasks each.

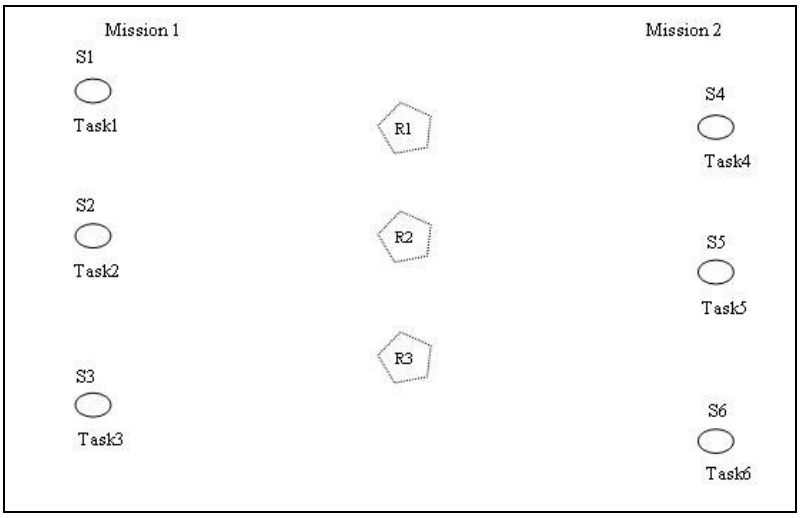

Figure 5.10 Test bed setup for Dynamic DEC implementation.

The resources used are Garcia Robots which can be localized and controlled using the Labview application discussed in Section 5.1. The position of ground sensors is assumed to be known beforehand. The allocation of the mobile resources is done by checking the coordinates of the ground sensors and the robot closest to it. One such allocation is showed for each task and the corresponding Matrix formulation and Petri-net diagram formed by the Labview code is shown in the Figure 5.11.

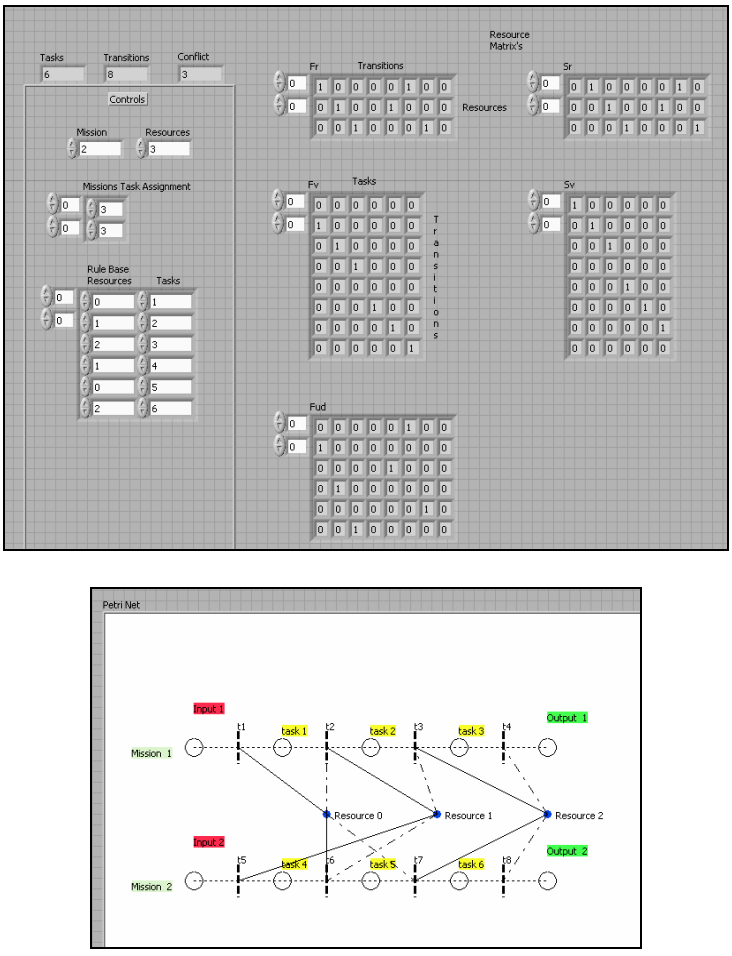

Figure 5.11 Matrices and Petri-Net Diagram for the given setup.

A possible deadlock situation that can occur in this system is shown in Figure 5.12. If Mission 1 and Mission 2 are triggered simultaneously executing task 1 and task 4, the tokens would be at positions shown in the Figure. As in their respective missions 'Task 1' is using resource 'R0' and 'Task 4' uses resource 'R1'. Now, for the token to move forward from 'Task 1' to 'Task 2' it requires the release of resource 'R1' to use it for execution of 'Task 2', and simultaneously, for token to move from 'Task 4' to 'Task 5' it requires the resource 'R0' to execute that task. As no resources are released and

made available for the subsequent tasks to use, the system results into a deadlock.

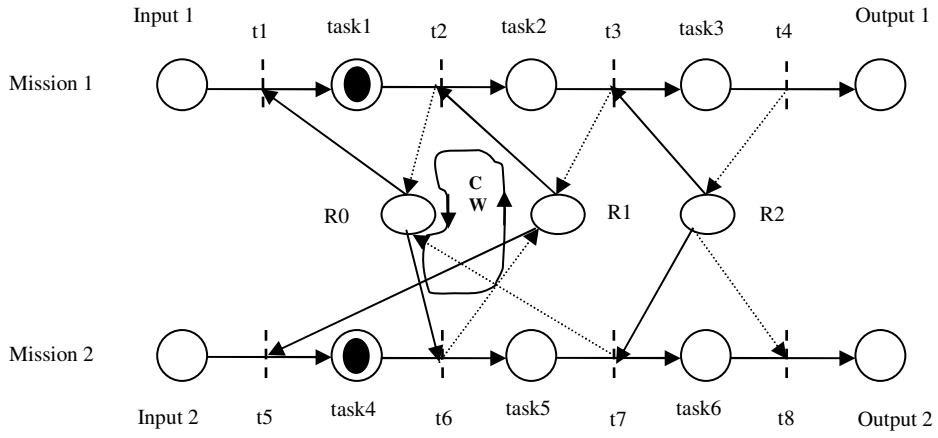

Figure 5.12 Possible Deadlock situation in the implementation.

But this deadlock can be avoided as the Equation 4.9 states that the number of activated tasks cannot be equal to or greater than number of resources in a circular wait. The activated tasks in this situation are equal to number of resources in the circular wait shown in the figure; this kind of situation is prevented from occurring. In the implementation it is taken care of by using the 'Conflict Resolution' module. If the missions are triggered simultaneously only one mission is allowed to use the shared resource and the task execution of other mission is blocked until the resource is released and made available for further use. These conflicts for resources are calculated from the matrices developed for the system and the conflict resolution code module uses it to resolve it and prevent the deadlocks. The code module is explained later in this section.

The implementation starts from the Sensor data module which constantly receives and checks the Light sensor data from the Ground sensors. Whenever the light gets less than a threshold value (1700) on a sensor, the corresponding mission gets

triggered and an input matrix is formed and sent to the Discrete Event controller application. The sensor checking and input matrix formation module is shown in Figure 5.13. The 'BasicserialRW' module is used to receive the sensor readings and 'Matrix formulation' module is used to generate the input matrix.

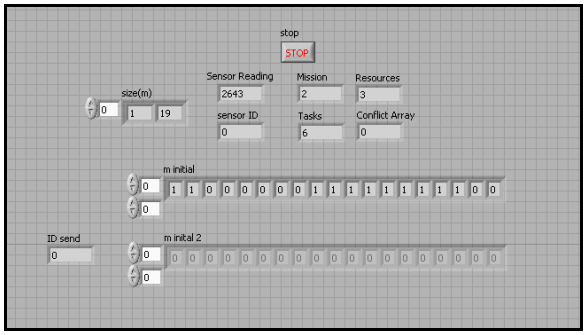

Figure 5.13 Mission Triggering: Light Sensor Detection and Input Matrix formulation.

The input matrix and the matrices F and S formed from the toolkit (Figure 5.11) are sent to the 'DEC' module. The Discrete Event Controller implements the Matlab simulation code explained in Section 5.2 with conflict resolution. The front panel of the module is shown in Figure 5.14. The tasks are shown as '1' when running and the Mission complete LED glows whenever the mission gets completed.

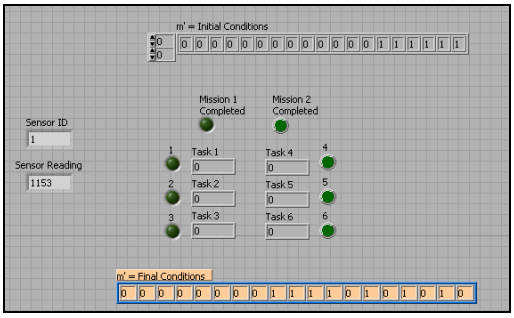

Figure 5.14 Discrete Event Controller: Front Panel.

The Discrete Event controller gets triggered when there is an input from the Sensor module discussed above. The module checks the input matrix and the F and S matrices for determining the mission triggered. The code showed in Figure 5.15 starts executing the tasks of the corresponding mission that is triggered. Suppose that, Mission 1 is triggered through an input. Then tasks 1, 2 and 3 are executed and similarly for Mission 2, tasks 4, 5 and 6 are executed. The execution of tasks means the Garcia robot moves to the assigned sensor and retrieves the data from it. The tasks implement sequentially i.e. only one task at a time. Thus depending on the number of tasks and the goal assigned the robots move to the ground sensors retrieves the data and completes the mission.

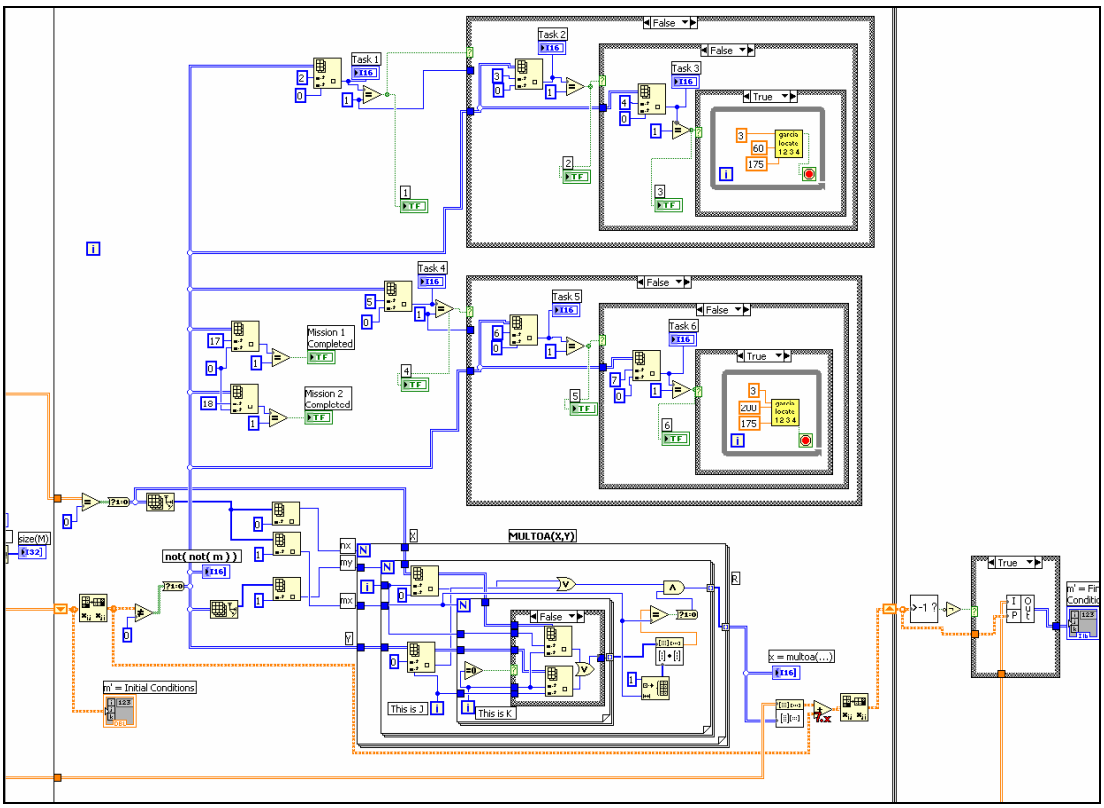

Figure 5.15 Discrete Event Controller and Task execution code.

Now if both the missions are triggered simultaneously or triggered in a way where two tasks from different missions require the same resource at the same time, conflicts would arise which would result in a deadlock because of the shared resources. But to prevent this, a 'Conflict Resolution' module is developed which resolves the conflicts and avoids any deadlock to occur. The concept of conflict resolution is to allow one task to run while blocking the other task that uses the same resource. The Labview code for conflict resolution is showed in Figure 5.16. All the conflict conditions are taken into consideration so that a deadlock cannot occur in the system.

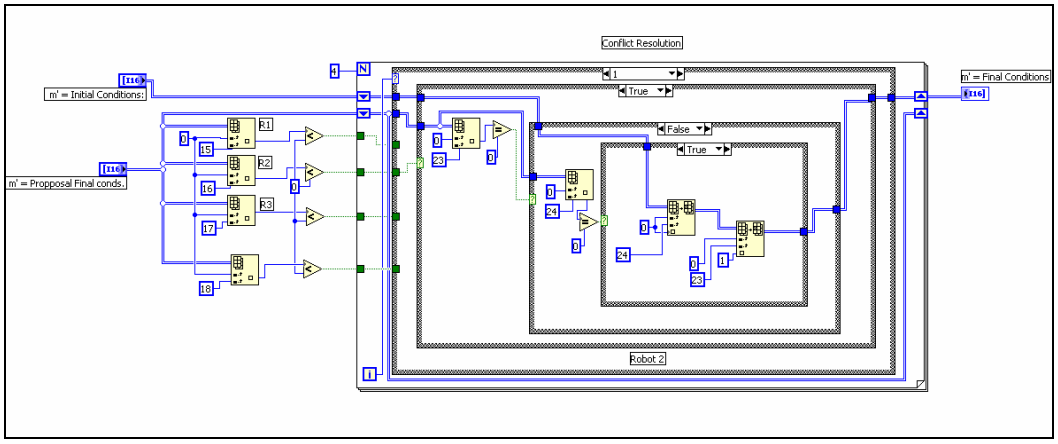

Figure 5.16 Labview code for Conflict Resolution Module.

Thus the Discrete Event Controller implemented here has many applications in real world [51] where limited resources are available and dynamic allocation is required for performing every task. The controller is completely autonomous and all the conflicts that could arise are avoided, thus making it reliable. The system can be triggered at particular times or using any kind of sensor data making it useful for data collection, security, monitoring and manufacturing purposes.

CHAPTER 6

CONCLUSION

The goal of this thesis was to implement a Discrete Event Controller (DEC) with the use of Dynamic Resource Allocation. For this, Garcia robots were used with powerful wireless sensors like Mica2 and Crickets. The Garcia robot and the wireless sensors used are completely off the shelf systems with no modifications done for tasks used for implementing the Dynamic DEC: Obstacle avoidance, Localization, Path planning and Controlling of Garcia.

Obstacle Avoidance allows the Garcia robot to navigate in any kind of environment without collision and is used in DEC for preventing the robots from colliding with each other. Localization provides information about the robots' current positions and for dynamic allocations when used as resources to perform tasks at positions that are in close proximity to it. Path planning is advantageous for robots that navigate in a static environment and do not have sensors to implement obstacle avoidance or for saving battery power instead of using it on obstacle avoidance. Path planning can also be used in DEC when localization is not possible or not required for performing missions or tasks. One such security application of DEC was discussed in Chapter 3 path planning section. Thus the most important concepts of robotics are implemented for use in Discrete Event Controller and various tool kits are designed that can be used for different robots or for stand alone applications.

Mobile Sensor Networks comprising mobile and static sensor nodes require multiple missions running simultaneously and network topology changes. A matrix based Discrete Event Controller with capability of dynamic resource allocation is implemented to tackle the shared resource conflicts and deadlocks occurring due to it. The implementation of dynamic resource assignment algorithms is complicated by the presence of shared resource conflicts and therefore on-line optimization of the task-resource assignments by using a combination of a greedy algorithm and a MAXWIP deadlock avoidance policy is done to guarantee deadlock-free dispatching. The implemented coordination control strategy is intuitive, effective and prone to be used, according to the application, with different dynamic resource assignment algorithms.

The application discussed in Section 5.4 backs the theory and the simulation discussed in Chapter 4 and Section 5.2 respectively. With the use of Garcia (mobile) nodes and Cricket/Mica2 (static) nodes a simple DEC is implemented where the mobile nodes have to collect data from the static nodes, as an event triggers the mission. The matrix formulation for DEC along with the deadlock free dynamic allocation of mobile nodes to the sensor nodes is done online and the commands are dispatched to perform the tasks of the mission. Thus the implementation completely justifies the theory proposed.

6.1 Future Work

In this thesis it is shown that, all the implementations are handled by the base station. The calculations for localization and path planning are done by the base station.

For the future work the Encoder values from the Garcia wheels can be used for localizing the Garcia. The controlling of Garcia using the wheel encoder values has already been done and the localization using it would allow the Garcia to venture far away from the base station to perform different tasks with no orientation and line of sight issues. The next important work is to implement the Matrix based DEC on the Garcia itself instead of depending on the base station to do the computations and dispatching commands. Here all the robots can form a swarm by communicating with each other and maneuver around in an environment. The Dynamic allocation in conjunction with DEC allows the robots to assume leader and follower positions in the swarm and change according to the interactions to the environment and to other robots.

APPENDIX A

OBSTACLE AVOIDANCE CODE WITH LOCALIZATION

Initialize the Beacon message and the Transmission Buffer

```
for(i=0;i<8;i++)
{CricketConfig.spaceid[i] = '1';
   beacondata->Space[i] = CricketConfig.spaceid[i];}
   beaconmsg.length = 23;
   beaconmsg.type = SYN1;
   beaconmsg.addr = TOS_BCAST_ADDR;
   beacondata->x = 0;
   beacondata->y = 0;
   beacondata->z = 0;
   beacondata->temp = 0;
   beacondata->tb = FALSE;

   mTxBuffer.length = 5;
   mTxBuffer.type = SYN3;
   mTxBuffer.addr = TOS_BCAST_ADDR;
   pTxBuffer = &mTxBuffer;
```

Start with Mode Beacon or Mode Radio depending on the application

```
command result_t StdControl.start()
{
  if (CricketConfig.run_mode == MODE_BEACON)
   {
         call BeaconTimer.start(TIMER_ONE_SHOT, 1000);
         call Timer2.start(TIMER_ONE_SHOT, 50);
   }

   if (CricketConfig.run_mode == MODE_RADIO)
   {
                  call RadioControl.start();
   }

   call UARTControl.start();

   call Leds.yellowOff();
   call Leds.greenOff();
   call Leds.redOff();
   return SUCCESS;
}
```

Beacon Code used for Localization

```
 event result_t BeaconTimer.fired()
 {
  int result = 0;
  int count = 0;

  uint8_t i,z;
  x2 = 0;
  call Timer1.stop();
  call Timer2.stop();
  call RadioControl.start();
  TOSH_uwait(2000);

  if (CricketConfig.run_mode == MODE_BEACON) {

        while ( count < CricketConfig.max_beacon_number - 1)
         {
                atomic
                 {
                    ReadyToSend = 0;
                 }
       // Add desync to help with hidden terminal
       DesyncDelay = call Random.rand();
       DesyncDelay = DesyncDelay % DESYNC_DELAY;

       // Listen for MAX_US_TRAVEL_TIME ms + extra time

        z = (CricketConfig.max_us_travel_time + DesyncDelay) / 5000;
                        for (i = 0; i < z; i++)
                        {
                                TOSH_uwait(5000);
                        }
TOSH_uwait(CricketConfig.max_us_travel_time + DesyncDelay - (5000 * z));

       if (ReadyToSend == 0)
        {
         // Send the location
                        if (CricketConfig.use_temp_sensor)
                                beacondata->temp = CricketConfig.local_temp;

                                beacondata->tb = readTestSwitch();
                                result = call RadioSend.send(&beaconmsg);
                                        if (result == SUCCESS)
```

```
                                        {
                                                call Leds.redToggle();
                                                break;
                                        }
        }

         count++;
                }

    // reschedule for next beacon

    DesyncDelay = call Random.rand();
    DesyncDelay = DesyncDelay %  CricketConfig.delta_beacon_interval_time;

  // call BeaconTimer.start(TIMER_ONE_SHOT,
CricketConfig.min_beacon_interval_time + DesyncDelay);


     if (xx == 2)        //xx =0 for autonomous mode in timer1 fired, else xx=0 in beac()
     {

        call RadioControl.stop();
        call BeaconTimer.stop();
        CricketConfig.run_mode = MODE_RADIO;
        TOSH_SET_US_IN_EN_PIN();
        TOSH_CLR_BAT_MON_PIN();
        TOSH_uwait(500);
        TOSH_CLR_US_IN_EN_PIN();
        TOSH_SET_BAT_MON_PIN();
        call RadioControl.start();
        call Timer3.start(TIMER_ONE_SHOT, 50);

     }
     else
     {
        call BeaconTimer.start(TIMER_ONE_SHOT, 300);
        call Leds.redOff();
     }
        xx++;

  }
```

```
Heartbeat, Motion and Sensing Commands
task void initHB()
        {

                call ByteComm.txByte(0x02);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x02);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x00);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x00);
                TOSH_uwait(1000);
                //0202 0000

                call ByteComm.txByte(0x02);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x04);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x1B);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x01);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x00);
                TOSH_uwait(1000);

                call ByteComm.txByte(0x04);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x04);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x3E);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x00);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x00);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x07);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x04);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x04);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x3E);
                TOSH_uwait(1000);
```

```
		call ByteComm.txByte(0x01);
		TOSH_uwait(1000);
		call ByteComm.txByte(0x00);
		TOSH_uwait(1000);
		call ByteComm.txByte(0x07);
		TOSH_uwait(1000);

		call Timer3.start(TIMER_ONE_SHOT, 50);
}

task void sensl()
{
		call Timer3.stop();
		call ByteComm.txByte(0x02);
		TOSH_uwait(1000);
		call ByteComm.txByte(0x02);
		TOSH_uwait(1000);
		call ByteComm.txByte(0x00);
		TOSH_uwait(1000);
		call ByteComm.txByte(0x02);
		TOSH_uwait(1000);

		call ByteComm.txByte(0x02);
		TOSH_uwait(1000);
		call ByteComm.txByte(0x02);
		TOSH_uwait(1000);
		call ByteComm.txByte(0x19);
		TOSH_uwait(1000);
		call ByteComm.txByte(0x82);
		TOSH_uwait(1000);
		call ByteComm.txByte(0x04);
		TOSH_uwait(1000);
		call ByteComm.txByte(0x02);
		TOSH_uwait(1000);
		call ByteComm.txByte(0x19);
		TOSH_uwait(1000);
		call ByteComm.txByte(0x80);
		TOSH_uwait(1000);
		call Leds.yellowToggle();
		call Timer4.start(TIMER_ONE_SHOT, 100);
}

task void sensr()
{
```

```
                call Timer4.stop();
                call ByteComm.txByte(0x02);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x02);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x00);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x02);
                TOSH_uwait(1000);

                call ByteComm.txByte(0x02);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x02);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x19);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x82);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x04);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x02);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x19);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x81);
                TOSH_uwait(1000);
                x2++;
                call Leds.yellowToggle();
                if (x2 == 10)
                {
                CricketConfig.run_mode = MODE_BEACON;
                call BeaconTimer.start(TIMER_ONE_SHOT, 100);
                }
                else
                {
                call Timer1.start(TIMER_ONE_SHOT, 100);
                }
}

task void right()
{
                call ByteComm.txByte(0x04);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x04);
```

```
                TOSH_uwait(1000);
                call ByteComm.txByte(0x3E);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x00);
                TOSH_uwait(1000);
                call ByteComm.txByte(0xFF);
                TOSH_uwait(1000);
                call ByteComm.txByte(0xFD);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x04);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x04);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x3E);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x01);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x00);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x03);
                TOSH_uwait(1000);

                call Timer1.stop();
                call Timer3.stop();
                call Timer4.stop();
                call Timer5.start(TIMER_ONE_SHOT, 1000);
}

task void left()
{

                call ByteComm.txByte(0x04);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x04);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x3E);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x00);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x00);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x03);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x04);
```

```
                TOSH_uwait(1000);
                call ByteComm.txByte(0x04);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x3E);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x01);
                TOSH_uwait(1000);
                call ByteComm.txByte(0xFF);
                TOSH_uwait(1000);
                call ByteComm.txByte(0xFD);
                TOSH_uwait(1000);

                call Timer1.stop();
                call Timer3.stop();
                call Timer4.stop();
                call Timer5.start(TIMER_ONE_SHOT, 1000);
}

task void reverse1()
{

                call ByteComm.txByte(0x04);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x04);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x3E);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x00);
                TOSH_uwait(1000);
                call ByteComm.txByte(0xFF);
                TOSH_uwait(1000);
                call ByteComm.txByte(0xFA);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x04);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x04);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x3E);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x01);
                TOSH_uwait(1000);
                call ByteComm.txByte(0xFF);
                TOSH_uwait(1000);
                call ByteComm.txByte(0xFA);
```

```
                TOSH_uwait(1000);

                call Timer1.stop();
                call Timer3.stop();
                call Timer4.stop();
                call Timer5.start(TIMER_ONE_SHOT, 1000);
        }
```

Receive Commands from the Radio

```
        task void beac()
        {
                call ByteComm.txByte(0x04);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x04);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x3E);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x00);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x00);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x00);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x04);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x04);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x3E);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x01);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x00);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x00);
                TOSH_uwait(1000);
                if (xx > 2)
                        {xx=0;}

            call BeaconsControl.stop();
            TOSH_SET_US_IN_EN_PIN();
            TOSH_CLR_BAT_MON_PIN();
            TOSH_uwait(500);
```

```
                TOSH_CLR_US_IN_EN_PIN();
                TOSH_SET_BAT_MON_PIN();
                CricketConfig.run_mode = MODE_BEACON;

                call Timer1.stop();
                call Timer2.stop();
                call Timer3.stop();
                call Timer4.stop();
                call Timer5.stop();
                call BeaconTimer.start(TIMER_ONE_SHOT, 1000);

        }

        task void gcstart()
        {
                call BeaconsControl.stop();
                CricketConfig.run_mode = MODE_RADIO;
                TOSH_SET_US_IN_EN_PIN();
                TOSH_CLR_BAT_MON_PIN();
                TOSH_uwait(500);
                TOSH_CLR_US_IN_EN_PIN();
                TOSH_SET_BAT_MON_PIN();
                call RadioControl.start();
                call BeaconTimer.stop();

                call Timer1.start(TIMER_ONE_SHOT, 100);
                call Timer2.start(TIMER_ONE_SHOT, 50);
        }

  event TOS_MsgPtr RadioReceive.receive(TOS_MsgPtr data)
  {

    if (!data->crc)
      return data;
      /* TODO: check CRC -- if it fails, no beacons will be trusted
        for a while and we will not be clear to send our own */
  {
         TOS_MsgPtr pBuf;
         IntGarciaMsg *message = (IntGarciaMsg *)data->data;

if (message->id == TOS_LOCAL_ADDRESS && message->cmdGType == 0x42)
                //Beacon
                {
```

```
                CricketConfig.run_mode = MODE_BEACON;
                flag = 0;
                call Timer1.stop();
                call Timer2.stop();
                call Timer3.stop();
                call Timer4.stop();
                call Timer5.stop();
                post beac();
                pBuf = data;
                call Leds.greenToggle();

                }
if (message->id == TOS_LOCAL_ADDRESS && message->cmdGType == 0x53 &&
flag == 0)              //Start
                {
                        flag = 1;
                   call BeaconTimer.stop();
                        post gcstart();
                        pBuf = data;
                        call Leds.greenOn();
                }
                else
                {
                        pBuf = NULL;
                }

                return pBuf;

        }

  return data;
 }
```

Obstacle Avoidance Function

```
 task void taskInterpretSerial()
        {

                IntGarciaIRMsg2 *message2 = (IntGarciaIRMsg2 *) pTxBuffer->data;
                int i = 0;
                int k = 0;
                int m = 0;
                while(i <= lengthb)
```

```
{
        if (buffer2[i] == 0x04)
        {
        if (buffer2[i+1] == 0x04)
        {
        if (buffer2[i+2] == 0x04)
        {
                atomic {
                                pTxBuffer->length = 3;
                                message2->bytes[0] = buffer2[i+3];
                                message2->bytes[1] = buffer2[i+4];
                                message2->bytes[2] = buffer2[i+5];
                                call Leds.greenToggle();
                        }

        call RadioSend.send(&mTxBuffer);

        TOSH_uwait(100);
        if (buffer2[i+3] == 0x01)
        {
                if(buffer2[i+4] > 0x25 && buffer2[i+4] < 0x65)
                        {post left();}
                else if (buffer2[i+4] > 0x65)
                        {post reverse1();}
        }
        else if (buffer2[i+3] == 0x00)
        {
                if (buffer2[i+4] > 0x27 && buffer2[i+4] < 0x65)
                        {post right();}
                else if (buffer2[i+4] > 0x65)
                        {post reverse1();}
        }

        for (m = i; m <= i+5; m++)
        {buffer2[m] = 0;}
        i = i + 5;
        }
        }
        }
        i++;
}
for(k = 0; k <= lengthb; k++)
        buffer2[k] = 0;
        lengthb = 0;
```

```
        call Timer2.start(TIMER_REPEAT, 50);
}

event result_t Timer1.fired()
{ if (CricketConfig.run_mode == MODE_RADIO)
{
        call Timer1.stop();
        xx =0;
        post initHB(); //sends bytes to the UART
        call Leds.redOn();
        return SUCCESS;
        }
}

event result_t Timer3.fired()
{ if (CricketConfig.run_mode == MODE_RADIO) {
        post sensl();
        return SUCCESS;
        }
}
event result_t Timer4.fired()
{
 if (CricketConfig.run_mode == MODE_RADIO){
        post sensr();
        return SUCCESS;
        }
}
event result_t Timer5.fired()
{ if (CricketConfig.run_mode == MODE_RADIO) {
        call Timer5.stop();

                call ByteComm.txByte(0x04);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x04);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x3E);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x00);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x00);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x00);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x04);
```

```
                TOSH_uwait(1000);
                call ByteComm.txByte(0x04);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x3E);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x01);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x00);
                TOSH_uwait(1000);
                call ByteComm.txByte(0x00);
                TOSH_uwait(1000);

        call Timer3.start(TIMER_ONE_SHOT, 100);
                return SUCCESS;
                }
        }


        /* Store received bytes in a buffer */
        async event result_t ByteComm.rxByteReady(uint8_t data, bool error, uint16_t
strength)
        {
                atomic
                {

                        buffer1[countd] = data;
                        countd++;
                }

                return SUCCESS;
        }

        /* Save received bytes in buffer2 every second for interpretation */
        event result_t Timer2.fired()
        {
                int i = 0;

                atomic
                {
                        for (i = 0; i < length_buffer3; i++)
                                buffer2[i] = buffer3[i];
                        for (i = length_buffer3; i <= countd+length_buffer3; i++)
                                buffer2[i] = buffer1[i];
                        lengthb = countd + length_buffer3;
```

```
                countd = 0;
            }

            post taskInterpretSerial();

            return SUCCESS;
        }
}
```

APPENDIX B

ANGLE CALCULATION CODE FOR ‘DISTANCE AND ANGLE’ MODULE

```
public static double calcAngle(float x1, float y1, float x2, float y2)
   {
      float dx = x2-x1;
      float dy = y2-y1;
      double angle=0.0d;
  // Calculate angle
      if (dx == 0.0)
      {
         if (dy == 0.0)
            angle = 0.0;
         else if (dy > 0.0)
            angle = Math.PI / 2.0;
         else
            angle = Math.PI * 3.0 / 2.0;
      }
      else if (dy == 0.0)
      {
         if  (dx > 0.0)
            angle = 0.0;
         else
            angle = Math.PI;
      }
```

```
        else
        {
           if  (dx < 0.0)
              angle = Math.atan(dy/dx) + Math.PI;
           else if (dy < 0.0)
              angle = Math.atan(dy/dx) + (2*Math.PI);
           else
              angle = Math.atan(dy/dx);
        }
         // Convert to degrees
        angle = angle * 180 / Math.PI;
         // Return
        return angle;
     }
}
```

APPENDIX C

BUILDING TINYOS SERIAL PACKETS
[OCTAVE TECHNOLOGY]

Some points that are critical to understanding the makeup of the serial data packet are listed below.

- A TinyOS data packet has a maximum length of 255 bytes.
- The raw data packet is wrapped on both ends by a frame synchronization byte of 0x7E. This is used to detect the start and end of a packet from the stream.
- The raw data packet uses an escape byte of 0x7D. This is needed in case a byte of payload data is the same as a reserved byte code, such as the frame synch byte 0x7E. In such a case, the payload data will be preceded by the escape byte and the payload data itself will be exclusively OR'ed with 0x20. For example, a payload data byte of 0x7E would appear in the data packet as 0x7D 0x5E.

Raw Data Packet

The following diagram and table describes the raw data packet

SYNC_BYTE	Packet Type	Payload Data	SYNC_BYTE
0	1	2...n-1	n

Byte #	Field	Description
0	Packet frame synch byte	Always **0x7E**
1	Packet Type	There are 5 known packet types: • P_PACKET_NO_ACK (0x42) - User packet with no ACK required. • P_PACKET_ACK (0x41) – User packet. ACK required. Includes a prefix byte. Receiver must send a P_ACK response with prefix byte as contents. • P_ACK (0x40) – The ACK response to a P_PACKET_ACK packet. Includes the prefix byte as its contents. • P_UNKNOWN (0xFF) – An unknown packet type.
2…n-1	Payload Data	In most cases will be a TinyOS Message of varying length, which is described below.
n	SYNC_BYTE	Always 0x7E

The TOS_Msg data packet is described in the following diagram and table:

Address		Message Type	Group ID	Data Length	Data	CRC	
0	1	2	3	4	5...n-2	n-1	n

Byte #	Field	Description
0 - 1	Message Address	One of 3 possible value types: • Broadcast Address (0xFFFF) – message to all nodes. • UART Address (0x007e)– message from a node to the gateway serial port. All incoming messages will have this address. • Node Address – the unique ID of a node to receive message.
2	Message Type	Active Message (AM) unique identifier for the type of message it is. Typically each application will have its own message type. Examples include: • AMTYPE_XUART = 0x00 • AMTYPE_MHOP_DEBUG = 0x03 • AMTYPE_SURGE_MSG = 0x11 • AMTYPE_XSENSOR = 0x32 • AMTYPE_XMULTIHOP = 0x33 • AMTYPE_MHOP_MSG = 0xFA
3	Group ID	Unique identified for the group of motes participating in the network. The default value is 125 (0x7d). Only motes with the same group id will talk to each other.
4	Data Length	The length (*l*) in bytes of the data payload. This does not include the CRC or frame synch bytes.
5…n-2	Payload data	The actual message content. The data resides at byte 5 through byte 5 plus the length of the data (*l* from above). The data will be specific to the message type. Specific message types are discussed in the next section.
n-1, n	CRC	Two byte code that ensures the integrity of the message. The CRC includes the Packet Type plus the entire unescaped TinyOS message. A discussion on how the CRC is computed is included in the Appendix.

APPENDIX D

RECEIVE FUNCTION FOR GARCIA CONTROL

```
 event TOS_MsgPtr RadioReceive.receive(TOS_MsgPtr data)
 {
        uint8_t i,k=0;
        uint8_t d=0,u=0,ph,pl;

   if (!data->crc)
      return data;

        {
        TOS_MsgPtr pBuf;
        IntGarciaMsg *message = (IntGarciaMsg *)data->data;

if (message->id == '3' && message->cmdGType == 'L' && flag == 0 && t == 0 && re
== 0)   //L = Left Turn
{
                        call BeaconTimer.stop();

                        flag = 0x4C;

                        if(message->dbytes[0] == ':');
                        {

                                for(i=1;i<11;i++)
                                {
                                        if(message->dbytes[i] == '.')
                                                {k=i;}
                                }
                                call Leds.greenOn();
                                if(k == 3)
                                {d = (message->dbytes[1]);
                                u = (message->dbytes[2]);
                                ph = (message->dbytes[4]);
                                pl = (message->dbytes[5]);
                                d = d - 48;
                                u = u - 48;
                                ph = ph - 48;
                                pl = pl - 48;
                                ti1 = (d*10) + u + (ph*0.1) + (pl*0.01);}
                                else if(k == 2)
                                {

                                u = (message->dbytes[1]);
                                ph = (message->dbytes[3]);
                                pl = (message->dbytes[4]);
```

```
                    u = u - 48;
                    ph = ph - 48;
                    pl = pl - 48;
                    ti1 = u + (ph*0.1) + (pl*0.01);}
                    else
                    {ti1 = 0;}

                    }

                ti1 = ti1 * 1000;
                post target();
                call Leds.redOn();
                pBuf = data;

}
else if (message->id == '3' && message->cmdGType == 'R' && flag == 0 && t == 0
&& re == 0)    //R = Right Turn
            {
                call BeaconTimer.stop();

                flag = 0x52;

                if(message->dbytes[0] == ':');
                {

                    for(i=1;i<11;i++)
                    {
                        if(message->dbytes[i] == '.')
                            {k=i;}
                    }
                call Leds.greenOn();
                    if(k == 3)
                    {d = (message->dbytes[1]);
                    u = (message->dbytes[2]);
                    ph = (message->dbytes[4]);
                    pl = (message->dbytes[5]);
                    d = d - 48;
                    u = u - 48;
                    ph = ph - 48;
                    pl = pl - 48;
                    ti1 = (d*10) + u + (ph*0.1) + (pl*0.01);}
                    else if(k == 2)
                    {
```

```
                    u = (message->dbytes[1]);
                    ph = (message->dbytes[3]);
                    pl = (message->dbytes[4]);
                    u = u - 48;
                    ph = ph - 48;
                    pl = pl - 48;
                    ti1 = u + (ph*0.1) + (pl*0.01);}
                    else
                    {ti1 = 0;}

                    }

                ti1 = ti1 * 1000;
                post target();
                call Leds.redOn();
                pBuf = data;

}
else if (message->id == '3' && message->cmdGType == 'S' && flag == 0 && t == 1)
        //S = Straight
{
                call BeaconTimer.stop();

                flag = 0x53;

                if(message->dbytes[0] == ':');
                {

                    for(i=1;i<11;i++)
                    {
                        if(message->dbytes[i] == '.')
                            {k=i;}
                    }

                    if(k == 3)
                    {d = (message->dbytes[1]);
                    u = (message->dbytes[2]);
                    ph = (message->dbytes[4]);
                    pl = (message->dbytes[5]);
                    d = d - 48;
                    u = u - 48;
                    ph = ph - 48;
                    pl = pl - 48;
                    ti2 = (d*10) + u + (ph*0.1) + (pl*0.01);}
```

```
                                else if(k == 2)
                                {
                                u = (message->dbytes[1]);
                                ph = (message->dbytes[3]);
                                pl = (message->dbytes[4]);
                                u = u - 48;
                                ph = ph - 48;
                                pl = pl - 48;
                                ti2 = u + (ph*0.1) + (pl*0.01);}
                                else
                                {ti2 = 0;}
                                }

                        ti2 = ti2 * 1000;
                        post target();
                        call Leds.redOn();
                        pBuf = data;

}
else if (message->id == '3' && message->cmdGType == 0x42 && flag == 0)
        //Beacon
{
                        call BeaconTimer.stop();

                        flag = 0x42;
                        post beac();
                        pBuf = data;
                        call Leds.redOn();
}
else
{
                pBuf = NULL;
}

                return pBuf;

}

   return data;
}
```

REFERENCES

[1] Acroname Inc., http://www.acroname.com/brainstem/ref/ref.html, Brainstem Manual.

[2] Acroname Inc., http://www.acroname.com/garcia/man/man.html, Garcia Manual.

[3] Acroname Inc., http://www.acroname.com/robotics/info/articles/sharp/sharp.html, Sharp IR Rangers information.

[4] Akyldiz I., Su W., Sankarasubramaniam Y, Cayirci E., "A survey on sensor networks", IEEE Communications Magazine, Vol. 40, Issue 8, pp. 102–114, August 2002.

[5] Ballal P., Giordano V., Lewis F. (2006), "Deadlock Free Dynamic Resource Assignment in Multi-Robot Systems with Multiple Missions: A Matrix-Based Approach," Proceedings of Mediterranean Conference on Control & Automation, Ancona, Italy, June 2006.

[6] Barnes B., "A Brainstem Tutorial using Palm Vx and Garcia", July 2005.

[7] Barry J., "Map Creation and Position Correction for an Off the Shelf Mobile Robotic System", Thesis, May 2004.

[8] Burgard, W., Moors M., Fox D., Simmons, R., Thrun S., "Collaborative multi-robot exploration", Proceedings of the IEEE International Conference on Robotics and Automation, Vol. 1, pp. 476–481.

[9] Butler Z., Rus D., “Event-based motion control for mobile-sensor network”, IEEE Transactions on Pervasive Computing, vol.2 issue 4, October-December 2003.

[10] Chong C., Kumar S., “Sensor Networks: Evolution, Opportunities and Challenges”, Proceedings of the IEEE, col. 91, no.8, August 2003.

[11] Clark A., El-Osery A., Wedeward K., Bruder S., “A Navigation and Obstacle Avoidance Algorithm for Mobile Robots Operating in Unknown, Maze-Type Environments”, Proc. International Test and Evaluation Association Workshop on Modeling and Simulation, December 2004.

[12] Corman T., Leisenson C., and Rivest R., Introduction to Algorithms, Prentice Hall of India, 2001.

[13] Cortes J., Martinez S., Karatas T., Bullo F., “Coverage control for mobile sensing network”, IEEE Transactions on Robotics and Automation, Vol. 20, Issue 2, pp. 243–255, April 2004.

[14] Cricket Sensor Resource, http://cricket.csail.mit.edu/.

[15] CrossBow Technology, http://www.xbow.com/Products/.

[16] Culler D., Hill J., “A Network-Centric Approach to Embedded Software for Tiny Devices”, University of California at Berkeley, Intel Research at Berkeley.

[17] 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.

[18] Ezpeleta, J.; Colom, J, Martinez, J., “A Petri net based deadlock prevention policy for flexible manufacturing systems”, IEEE Transaction on Robotics and Automation, Volume 11, Issue 2, April 1995 Page(s):173 – 184.

[19] Fanti M.P., Maione B., Mascolo S., Turchiano B., Event-based feedback control for deadlock avoidance in flexible production systems IEEE Transactions on Robotics and Automation, Volume 13, Issue 3, June 1997 Page(s):347 – 363.

[20] Fanti M., Maione B., Turchiano B., “Comparing digraph and Petri net approaches to deadlock avoidance in FMS”, IEEE Transactions on Systems, Man and Cybernetics, Part B, Volume 30, Issue 5, Oct. 2000 Page(s):783 – 798.

[21] Gerkey B., Mataric M., “Sold!: Auction methods for multirobot coordination”, IEEE Transactions on Robotics and Automation, vol. 18, no. 5, October 2002.

[22] Giordano V., Lewis F., Mireles J., Turchiano B., “Coordination control policy for mobile sensor networks with shared heterogeneous resources”, Proceedings of the IEEE International Conference on Control and Automation, Budapest, June 2005.

[23] Giordano V., Ballal P., Lewis F., Turchiano B., Zhang J. B., “Supervisory control of mobile sensor networks: math formulation, simulation, implementation,” IEEE Transactions on Systems, Man and Cybernetics, Part B, Volume 36, Issue 4, Aug. 2006 Page(s):806 – 819.

[24] Gordon-Spears A., Kiriakidis K., “Reconfigurable robot teams: modeling and supervisory control”, IEEE Transactions on control system technology, vol. 12, no. 5, September 2004.

[25] Gurel A., Bogdan S., Lewis F., "Matrix approach to deadlock-free dispatching in multi-class finite buffer flowlines", IEEE Transactions on Automatic Control, Vol. 45, Issue 11, November 2000, pp.2086-2090.

[26] Huber M. and Grupen R., "A Hybrid Discrete Event Dynamic System Approach to Robot Control", Laboratory for Perceptual Robotics, Department of Computer Science, Univ. of Massachusetts, Technical Report #96-43, October, 1996.

[27] 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, Issue 5, pp. 615–619, September 2003.

[28] Kusiak A. Intelligent scheduling of automated machining systems in Intelligent design and Manufacturing. A. Kusiak (ed.) Wiley, New York (1992).

[29] Laumond J., Sekhavat S., Lamiraux F., "Guidelines in Non-holonomic Motion Planning for Mobile Robots" in J.P. Laumond, editor, "Robot Motion Planning and Control", pages 1-53. Springer-Verlag, Berlin, 1998.

[30] Lewis F., "Wireless sensor networks", Smart environments: Technologies, protocols, and applications, ed. D. J. Cook and S. K. Das, John Wiley, New York, 2004.

[31] Lewis F., Gurel A., Bogdan S., Doganalp A., Pastravanu O., "Analysis of deadlock and circular waits using a matrix model for flexible manufacturing systems," Automatica, vol. 34, no. 9, pp. 1083-1100, 1998.

[32] Mireles J., Lewis F., "Intelligent material handling: development and implementation of a matrix-based discrete event controller IEEE Transactions on Industrial Electronics, vol. 48, Issue: 6, Dec. 2001 Pages: 1087 – 1097.

[33] Mireles J., Lewis F., Gurel A., "Implementation of a deadlock avoidance policy for multipart reentrant flow lines using a matrix-based discrete event controller", Proceedings of the International symposium on advances in robot dynamics and control, New Orleans, November 2002.

[34] Murata, T. "Petri Nets: Properties, Analysis and Applications." Proceedings of IEEE, vol.77, no.4, April 1989, pp.541-80.

[35] NesC Resource, http://nescc.sourceforge.net/.

[36] OOPIC, http://www.oopic.com/gp2d12.htm, Connecting an IR Distance Detector.

[37] Ozmutlu, S. And Harmonosky, C.M., "A Real Time Methodology for Minimizing Mean Flowtime in FMSs With Routing Flexibility: Threshold Based Alternate Routing," European Journal of Operational Research, 2005, Pages(s): 369-384.

[38] Peterson J., "Petri Net Theory and the Modeling of Systems," Prentice-Hall, Englewood Cliffs, NJ, 1981.

[39] 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.

[40] Siegwart R., Nourbakhsh I., "Introduction to Autonomous Mobile Robots", 2004.

[41] Sinopoli B., Sharp C., Schenato L., Schaffert S., Sastry S., "Distributed Control Applications Within Sensor Networks", Proceedings of the IEEE, vol. 91, no.8, August 2003.

[42] Smith A., Balakrishnan H., Goraczko M., Priyantha N., "Tracking Moving Devices with the Cricket Location System", Proc. 2nd Usenix/ACM Mobisys Conf., June 2004.

[43] Sukhatme G., "Robot Navigation, Task Allocation using a Sensor Network", CENS Research Project.

[44] Tacconi D., Lewis F., "A new matrix model for discrete event systems: application to simulation", IEEE Control System Magazine, vol.17 October 1997.

[45] TinyOS, http://www.tinyos.net/tinyos-1.x/doc/, TinyOS Tutorial.

[46] Wonham W., "Supervisory Control of Discrete Event Systems: An Introduction", Systems Control Group, Dept. of Electrical & Computer Engineering, University of Toronto, June 1999.

[47] Wysk, R., Yang, N., Joshi, S., "Detection of deadlocks in flexible manufacturing cells", IEEE Transactions on Robotics and Automation, vol.:7, Issue: 6, December 1991.

[48] Xiaojiang D., Fengjing L., "Improving Sensor Network Performance by Deploying Mobile Sensors", IEEE International Performance Computing and Communications Conference, 2005.

[49] Xing K.Y., Hu B.S., Chen H.X. (1991), “Deadlock Avoidance Policy for Petri-net Modeling of Flexible Manufacturing Systems With Shared Resources,” IEEE Transactions on Automatic Control, Volume 41, Feb 1991 Page(s): 289-295.

[50] Yan L., "Wireless Sensor Networks: Past, Present, and Future," IEEE Distributed Systems Online, vol. 7, no. 5, 2006, art. no. 0605-o5007.

[51] Yu Y., Prasanna V., Krishnamachari B., “Information Processing and Routing in Wireless Sensor Networks”, World Scientific, Dec 2006.

BIOGRAPHICAL INFORMATION

Abhishek Trivedi received his Bachelor of Engineering degree in Electronics and Communications Engineering from Saurashtra University. He also got a Diploma in Electronics and Communication Engineering from Government Polytechnic, Gandhinagar. He worked as a Junior Engineer at Shubham Automation Ltd. He then started his Masters in Electrical Engineering at University of Texas, Arlington in 2005. Due to his interest in Embedded programming and Wireless Systems, he started working on research projects involving Mobile Wireless Sensor Networks at Automation and Robotics Research Institute (ARRI) under Dr. Frank Lewis. He has been working as a Graduate Research Assistant at ARRI.