Development of a wireless sensor network system for target tracking and surveillance

Toh, Yue Khing

2008

Toh, Y. K. (2008). Development of a wireless sensor network system for target tracking and surveillance. Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/43658>

<https://doi.org/10.32657/10356/43658>

Chapter 5 Energy Monitoring

To improve the efficiency of sensor node selection and to increase the lifespan of the WSN, each sensor node will need to calculate its own energy consumption accurately. Each node contains a number of components and we need to measure the energy consumption of every component to accurately predict the energy remaining on the node. With the ability to evaluate the WSN energy level, it is possible to know which part of the network is actually more taxed due to intrusion.

5.1 Calculation for Energy Monitoring

With these improvements in mind, we profile the energy model of each of the component in the node and use the data to calculate the remaining energy of the node. The calculation of energy consumption in the node can be found by the following equation:

$$\text{Energy (Joules)} = \text{Voltage (V)} \times \text{Current (I)} \times \text{Time (s)} \quad (5-1)$$

However, there are some disadvantages using Joules as the measurement unit to estimate the life span of the sensor node. For example in the MICA2 hardware, it only contains an ADC to measure the battery voltage levels, extra electronics are needed just to measure the current thus making the circuit complex. It is also impossible to measure the transient current during the short transmission of radio packet or firing of sonar as the current consumption lasts only a few μs or ms. To calculate the remaining energy of the node, we also need to know the energy of the batteries in Joules. Unfortunately most batteries does not provide the energy information in Joules, instead the unit of electric charge in milliamperehours (mAh) is used for these batteries. The discharge curve of the batteries [1] as shown in Figure 49 meant that the above equation does not apply well to compute the Joules as the voltage does not remain constant but varies with both time and load.

Chapter 5: Energy Monitoring

Figure 49: Battery discharge curve and rechargeable battery rating in 2700 mAH

Since the objective of energy monitoring is to estimate the energy left in a sensor node, using the electric charge unit milliampere-hour (mAh), ampere-second (As) and milliampere-second (mAs) would make the calculation much simpler than the unit Joules. To estimate the charge left in a sensor node the following equation can be used:

$$\text{Sensor Node remaining charge (mAs)} = \text{Battery Charge (mAh)} \times 3600 - \text{Total charge consume on power up (As)} \times 1000 \quad (5-2)$$

The major challenge to energy monitoring by sensor node is accurately measure the transient current of each component. Since it is difficult to measure the current directly, a simple way is to indirectly measure the voltage supply to each component and use the I-V characteristic of the component to calculate the current consumption. As the microprocessor is able to monitor the turn on time of each component and the voltage of the supply, the total charge in mAs can be calculated by linear extrapolation.

Chapter 5: Energy Monitoring

5.1.1 Energy Profile Setup

To model I-V characteristic of the component in a sensor node, an experiment setup is needed to monitoring the current usage at different voltage levels. Figure 50 shows the hardware setup to measure and capture the transient current.

Figure 50: Hardware setup for device measurement

This setup is made up of a PC equipped with high precision DAC/ADC and voltage regulator with feedback to provide precise voltage to the DUT and a high speed NI 4071 DMM capable of capturing the transient current at ns speed and with μA accuracy. With this setup, the computer is able to vary the voltage to the DUT and captured the current waveform using the DMM.

5.1.2 Device Under Test

Since it is difficult to measure every individual component in the sensor node, the modeling of the node is broken up into 5 major components as shown in Figure 51. The 5 major components in the sensor nodes are the Radio, Processor and other peripherals, the 3 LEDs, SRF08 and PIR. Each component will be connected as DUT to the measurement setup from the previous section.

Chapter 5: Energy Monitoring

Figure 52: CC2420 ZigBee radio connection

Figure 52 shows the connection of the ZigBee radio to the experimental setup and Figure 53 shows the current waveform captured for transmitting a radio packet of 10 bytes at the maximum power setting. From the graph, the receiving current is higher than the transmission current and there are Rx-Tx / Tx-Rx transitions before and after transmitting a packet.

Figure 53: Waveform capture for transmitting 10 bytes at maximum power setting

When the radio is not transmitting, it remains at receiving mode and the current consumption remains constant. As for the Rx-Tx and Tx-Rx, the current does not vary with transmission power and thus remains constant for each packet transmitted.

Chapter 5: Energy Monitoring

Figure 54: Radio transmission per byte vs. power setting

Figure 55: Radio transmission vs. byte length

Figure 54 shows the radio transmission per byte vs. power setting and Figure 55 shows the radio transmission vs. byte length. The following equation to calculate the transmission charge can be derived from the above charts:

$$\begin{aligned} \text{Transmission Charge}(As) = & \left((3.1107 \times 10^{-7}) \times (\text{powersetting}) + 4.845 \times 10^{-6} \right) \\ & \times \left((5.2465 \times 10^{-7}) \times \text{bytelength} + 9.0542 \times 10^{-6} \right) + R_x - T_x + T_x - R_x \end{aligned} \quad (5-3)$$

where

$$R_x - T_x = 6.377 \times 10^{-7} As$$

$$T_x - R_x = 5.899 \times 10^{-7} As$$

When the radio is in receiving or idle mode, the current consumption remains constant and thus the total radio charge can be expressed in the following equation:

Chapter 5: Energy Monitoring

$$\text{Receiving Charge (As)} = 2.449 \times 10^{-5} \times t(s)$$

where

$$t(s) = \text{Receive or Idle time} \quad (5-4)$$

$$\text{Total Radio Charge(As)} = \text{Transmission(As)} + \text{Receiving(As)}$$

5.1.4 Energy Profile for Processor & Other Peripherals

The power consumption of the processor and other peripheral such as the resistors, etc are computed as a unit to simplify the energy calculation. The processor of MICAZ support both active and idle mode. Figure 56 shows the processor running a 20,000 loop in active mode and 100ms idle mode with the radio, LEDs and other components switch off.

Figure 56: Processor running 20K empty loop

Table 3 shows the charge consumed by the processor to perform multiplication, division and addition. From this table, the current consumed to perform different arithmetic depends on the length of time to perform these calculations.

Type	As	Time (s)
Loop 10k	0.000115436	0.01786568
Loop 10k Plus Addition	0.000151250	0.02331672
Loop 10k Plus Subtraction	0.000152443	0.02331672
Loop 10k Plus Division	0.000163985	0.02607808

Chapter 5: Energy Monitoring

Loop 10k Plus Multiplication	0.000165238	0.02605232
------------------------------	-------------	------------

Table 3 : Charge consumption for different arithmetic calculation **Figure 57: Processor charge consumption vs. supply voltage**

Figure 57 shows the processor charge consumption vs. voltage supply and the processor charge can be expressed with the following equations:

$$\text{Active Charge (As)} = (4.6629 \times 10^{-3} \times V + 6.7758 \times 10^{-3}) \times t_A(s)$$

$$\text{Idle Charge (As)} = (2.6295 \times 10^{-3} \times V + 4.4713 \times 10^{-3}) \times t_I(s)$$

where

V = Supply Voltage

$t_A(s)$ = Processor active time

$t_I(s)$ = Processor idle time

(5-5)

5.1.5 Energy Profile for LEDs

Figure 58 shows the three LEDs charge consumption, as each of the LED current consumption is similar, the LED charge can be expressed in the following equation:

Chapter 5: Energy Monitoring

$$\text{LED Charge (As)} = (1.982 \times 10^{-3} \times V - 3.616 \times 10^{-3}) \times t_{on}(s)$$

where

V = Supply Voltage

$t_{on}(s)$ = ON time for the LED

(5-6)

Figure 58: LEDs charge consumption vs. supply voltage

5.1.6 Energy Profile for SRF08 Sonar

Figure 59 shows the waveform capture for SRF08 sonar with 3 segments. Before firing of the sonar, the SRF08 uses a charge-pump circuitry to store a significant amount of energy into the capacitors. During this time, the current consumption peaks at 0.475mA. Although the current is very high during the charging process, the time taken is in the order of ns and therefore the charge consumption compared to listening mode which lasts for a longer 70ms is very small. When the SRF08 is not activated, the sonar goes into idle mode.

Chapter 5: Energy Monitoring

Figure 59: Waveform captured for SRF08 sonar

Figure 60: Firing charge consumption vs. supply voltage

Chapter 5: Energy Monitoring

Figure 61: Listening charge consumption vs. supply voltage

Figure 62: Idle charge consumption vs. source voltage

Chapter 5: Energy Monitoring

Figure 60, Figure 61 and Figure 62 show the charge consumption for Firing, Listening and Idle mode vs. the supply voltage. Although there is some irregularity for Idle mode, the charge consumption is assumed to be approximately linear as the irregularity is very small. The 3 modes charge consumption can be expressed in the following equations for each sonar:

$$\begin{aligned}
 \text{Firing Charge (As)} &= (7.1162 \times 10^{-6} \times V_s + 9.5459 \times 10^{-6}) \\
 \text{Listening Charge (As)} &= (2.11398 \times 10^{-4} \times V_s + 3.94113 \times 10^{-4}) \\
 \text{Idle Charge (As)} &= (1.21861 \times 10^{-5} \times V + 3.867338 \times 10^{-5}) \times t_I(s)
 \end{aligned}$$

where (5-7)

V_s = Supply Voltage to SRF08
 $t_I(s)$ = SRF08 idle time

5.1.7 Energy Profile for Passive Infra-Red

Figure 63 shows the current consumption of the PIR. The PIR operates in toggling mode, when triggered by moving objects e.g the PIR state toggles between logic "0" and logic "1" when no motion is detected.

Figure 63: PIR logic '1' to '0' and '0' to '1'

Figure 64 shows the PIR charge consumption vs. supply voltage. As the charge consumption between logic "0" or logic "1" is very small, the average charge consumption can be expressed with the following equation:

$$\begin{aligned}
 \text{PIR Charge (As)} &= (3.33333 \times 10^{-5} \times V + 2.233345 \times 10^{-5}) \times t_{on}(s) \\
 \text{where} & \\
 V &= \text{Supply Voltage} \\
 t_{on}(s) &= \text{On time for the sensor node}
 \end{aligned}$$

(5-8)

Chapter 5: Energy Monitoring

Figure 64: PIR charge consumption vs. Supply Voltage

5.1.8 Energy Monitoring Verification

Figure 65 shows the experimental setup for the energy monitoring verification setup. 2 sets of 4 x 1300mAh and 4 x 2700mAh rechargeable batteries are used to power the sensor node. An ADC card in the PC monitor the voltage level of the batteries and the DMM monitor the current consumption. This setup monitors the real total charge consumption by the sensor nodes. Using the equations from the previous section and based on the run-time energy monitoring [1], the sensor node is programmed to continuously emulate EKF calculations, fire the sonar, blinks the LEDs and transmits the calculated total charge consumption back to the PC until the batteries run out. These results are then compared with the real values monitored by the PC.

Chapter 5: Energy Monitoring

Figure 65: Power monitoring verification setup

Figure 66 shows the power monitoring verification graph for 2700mAh and 1300mAh respectively, the x-axis is time in seconds and y-axis is the charge consumption in mAs. The red line shows the calculated charge consumption by the sensor node and the white line is the actual data measured by the setup.

Figure 66: Power Monitoring Verification with 2700 mAh and 1300 mAh batteries

Chapter 5: Energy Monitoring

Figure 67 shows the percentage error for 2700 mAh and 1300 mAh batteries with a maximum error of 8.5% and 7% respectively. These results show that the sensor node is able to calculate its own power usage accurately.

Figure 67: Percentage Error with 2700 mAh and 1300 mAh batteries

5.2 Chapter Conclusion

In this chapter, we proposed using the charge unit in mAh or mAs as a more feasible mean for energy monitoring than using Joules. To monitor the energy consumption, the current information is needed but it is impossible to monitor the transient current in the node. We therefore proposed using the I-V characteristic to model the energy profile of each component and extrapolation to derive the current information. Since the node is able to monitor the operating time of every component, the node would be able to accurately calculate the total energy usage. We demonstrated that the proposed energy monitoring scheme has a maximum error of only 8.5% by the end of the experiment.

Chapter 6: Firmware and Software Architecture

Chapter 6 Firmware and Software Architecture

The system architecture of the WSN target tracking system is shown in Figure 68. Both static and mobile sensor nodes operate in an operating system called TinyOS. For the GUI, both PDA and computer run LABVIEW which provides a graphical programming interface. On the PC end, a Serial Forwarder program provides a TCP/IP connection for the GUI to read and write packet data to the MIB520 which communicates wirelessly with the WSN.

Figure 68: System architecture of the WSN target tracking system

Figure 69 shows the firmware and software overview of the WSN target tracking system. Each sensor node and the base station contain the application layer and a middleware to link the application layer with the operating system (OS) and protocol stack. In the sensor node, the application layer consists of target tracking and sensor selection blocks. The target tracking block uses information in Token Key and the EKF to estimate the state of the target. Information of the target state will be passed to the sensor selection block and the sensor selection scheme will elect the next tasking sensor node. The middleware consists of wireless configuration, ultrasonic communication, packet coding and decoding, data

Chapter 6: Firmware and Software Architecture

packaging, data reporting and Δt_k calculation blocks. Node configurations sent down from the base station are processed by the wireless configuration block. The configuration information consists of ultrasonic parameters, EKF parameters, sensor selection scheme, etc. The ultrasonic communication block handles the communication protocol between MICA2 and SR08 sonar arrays. The packet coding and decoding block decodes the packet received from the base station or neighboring sensor nodes for execution. The Δt_k calculation block calculates the sampling time Δt_k used in both EKF and sensor selection scheme. The Data Packaging block encapsulates information such as ranging information of the sonar, Token Key, etc for transmission. The OS and communication protocol stack are handled by the TinyOS.

Figure 69: Firmware and software overview of the WSN target tracking system

On the base station, the application layer consists of visualization and manual control functions. The visualization block displays the target information and sensor nodes position received from the WSN. The manual control block is used to control the motion of the target robot by a joystick connected to the base station to simulate an intruder. The middleware consists of wireless configuration, sensor node monitoring, packet injection and packet monitoring blocks. The user can send the configuration command such as the sensor gain to individual or all the nodes in the network. Packet injecting is used for troubleshooting such as

Chapter 6: Firmware and Software Architecture

testing the stability of the test bed when a corrupted radio packet is injected into the system. Packet monitoring is used to eavesdrop communications within the test bed and to store information such as passing of Token Key between nodes and transmission retries for analysis.

6.1 Firmware Overview for Sensor Nodes

6.1.1 TinyOS

TinyOS is an embedded open source operating system targeting at low power microcontroller such as MICAZ. TinyOS is written in the nesC programming language as a set of cooperating tasks and processes. NesC has a C-like syntax, but supports the concurrency model, as well as mechanisms for structuring, naming, and linking together software components into robust network embedded systems. TinyOS structure is highly modular which means that it can be easily extended. Basically TinyOS can be divided into 2 parts, namely module and configuration. Modules are where all the application codes are written, and configurations are used to assemble other components together with the modules written by the designers. This enables the designers to use other modules that are readily available in the TinyOS library.

Unfortunately TinyOS is not yet a simple language to master, the lack of good documentation, difficulty in accessing the low level hardware I/O and the lack of good debugging tools increases the development time for the entire project.

6.1.2 Network Programming – Deluge

As the number of nodes increases, collecting and programming every individual node become a tedious and time consuming task. Deluge [1] provides an efficient method for disseminating program binaries to many nodes within a WSN through wireless programming. Deluge makes use of the external EEPROM in MICAZ to store user program.

Chapter 6: Firmware and Software Architecture

Golden Image (boot loader)
User Program 1
User Program 2

Figure 70: Deluge usage of EEPROM

As shown in Figure 70, the EEPROM is divided into 3 sectors of 128kB each with the 1st sector storing the Golden image or boot loader. On CPU boot up, it will check if the program in the MICAZ internal flash is valid, or empty. If there is no program in the flash, the Golden image will automatically copy itself into the flash. This program will sit and wait for instructions such as to receive user program from the base station. The next 2 sectors store the user program but at any one time only one of the firmware is allowed to be copied and executed in the internal flash of MICAZ. Another feature of Deluge is its automatic update of old firmware with a new one if a node with a latest firmware is within its radio range. This happens because each node will broadcast the version of its firmware information at random interval and the node with the latest firmware will automatically perform the update to the WSN. This causes problem during the early development stage of this project, as some node will mistakenly execute the deluge radio packet as node command and crashes the node. This problem was solved by identifying and ignoring the radio packet meant for Deluge. Another issue which we currently face in our setup is that the frequent radio update causes the network to die after some time due to frequent packets collisions between nodes. Currently there is no solution to solve this problem except to physically reset the nodes.

6.1.3 Communication Protocol for WSN

Figure 71 shows the communication protocol [1] for the WSN. Each ZigBee radio data packet has a maximum length of 128 bytes and 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 packet type defines the type of

Chapter 6: Firmware and Software Architecture

packet for the Payload Data and by default 0x42 is used to represent user packet. The Payload Data in the ZigBee Radio Packet contains the predefined TinyOS communication protocol. The Address defines the destination of a node to receive message and the Message Type identifies the type of message. Each application will define its own message type, for example the message type for Deluge is 0xA1 and in our WSN application 0x0A is used. To differentiate Deluge package from WSN package, the WSN firmware will only respond to packet containing 0x0A in the message type and reject others with different message type. The Group ID uniquely identifies a group of nodes that uses the same radio frequency. Only nodes with the same group ID will communicate to each other. Data Length defines the length of the data payload and does not include the CRC and frame synch bytes. The communication protocol for our WSN is encapsulated in TinyOS' data.

Figure 71: Communication protocol for WSN

The communication protocol for our WSN is represented by 2 bytes **Source address**, this identifies the source node that transmits the radio packet since it is not included in the standard TinyOS communication protocol. The next byte **CMD** is the command byte which tells the node or base station what sort of command to execute followed by n bytes of data.

Chapter 6: Firmware and Software Architecture

Figure 72: Data packet for Token Key

Figure 72 shows the data packet of the Token Key used for selecting the tasking sensor. Each data packet consists 16 bytes for target state $\hat{x}(k+1|k+1)$, 64 bytes for covariance matrix $P(k+1|k+1)$, 4 bytes for T_4 parameter, and 12 bytes for distance between the target and its energy usage.

6.1.4 Firmware for WSN nodes

Figure 73 shows the detail structure of the firmware in the sensor nodes. Due to the complexity of the coding involved, this section will give a brief description of the selected modules and components.

TestSensorM – This is the main module that connects to all the components and sub modules in the sensor node. It contains coding for both the application and middleware layers as described in Figure 69.

MultilocationC – This component handles the localization of the mobile sensor node in the WSN.

PirC – This component handles the low level I/O to acquire data from the PIR sensors. Its purpose is to wake up the sensor node from low power mode when it detects a moving target within its sensor range

Sonar – This component handles the low level I/O and provides the low-level access to the SR08 sonar arrays. It is also the middle layer to configure the sonar setting such as its gain and sampling time.

Chapter 6: Firmware and Software Architecture

AmigoC – This component handles the low level communication with the amigo robot in both mobile and target robot.

ekfC – This component handles the complicate EKF computation for target trajectory estimation and prediction.

EnergyMonitorM – This module handles the computation for the energy consumption in every hardware components such as sonar, PIR, radio, CPU, LEDs, etc on the sensor node.

Figure 73: Firmware of the sensor node

6.2 Software Overview for Base Station and PDA

The application developed in LabVIEW not only provides visualization information of the network and target, it also enables us to remotely configure the sensor

Chapter 6: Firmware and Software Architecture

nodes, to eavesdrop on the communication between sensor nodes and to store these data for analysis. We have implemented the following features on both base station and PDA:

Visualization and Control:

- Plot static and mobile sensor nodes position
- Plot target velocity, position and trajectory
- Deployment of mobile sensor nodes into areas not covered by static sensors
- Control of the intruder target via joystick or buttons on PDA

Configuration and Inspection:

- Bluetooth and robot configuration
- Wireless uploading of firmware into sensor nodes
- Configure sensor nodes and EKF parameters
- Sensor node queries and monitoring
- Radio packet injection, monitoring and capture to storage

6.2.1 GUI for PDA

Figure 74 shows the screen capture of the PDA, it allows us to remotely control the target, display target information, configure and query static nodes, as well as inject and monitor radio packets. PDA helps us to perform troubleshooting during the early development stage of the system such as fine tuning the gain that causes erratic reading from individual ultrasonic sensor due to the reflected echo. Radio packets inspections are done in the network level where communication between sensor nodes has to be monitored for erratic data. Raw data from the WSN are captured, decoded and stored for further analysis. We can also inject

Chapter 6: Firmware and Software Architecture

radio packets into the WSN to test the stabilities of the network with corrupted data or to set parameters for certain group of nodes.

Figure 74: Screen capture from PDA

6.2.2 GUI for Base Station

The GUI on the base station provides more functionality than the PDA and this section will give a brief introduction.

Figure 75: Main plot front panel

The front panel in Figure 75 allows the user to control the target using a joystick, modify the speed of the target, selects the sensor selections and to deploy mobile nodes into the WSN. It also plots the trajectory of the target and the node that is currently activated with its cumulative energy consumption. Table 4 summarizes the controls and indicators on the GUI.

Chapter 6: Firmware and Software Architecture

Items	Descriptions
Graph	
Blue circle	Tasking node
Large black circle	Target ranging distance from tasking node
Text above blue circle	Cumulative energy consumption of node
Small white circle	Inactive node
Red circle	Predicted target position
Green dots & black Lines	Trajectory of the target
Indicators	
X-Position, X-Velocity, Y-Position, X-Velocity	Current position and velocity of the target from tasking sensor nodes
Amigo bot	Displays the position of the mobile sensor node
Target Robot Controls	
Robot Speed/Rotation Speed	Set speed of target
Robot Address	Target address
Enable Robot	Enable target robot
Mobile Node Controls	
Mobile Node Address	Mobile node address
Locate Mobile Node	Plots position of mobile node on plot
Disable Mobile Node	Changes mobile node to static node state
Enable Mobile Node	Changes mobile node back to mobile state
Node Controls	
Start EKF	Start Extended Kalman Filter
Sensor Selection	Select the sensor selection algorithms

Chapter 6: Firmware and Software Architecture

Retrieve Node Info	Request all sensor nodes to return it positions and plot onto graph
--------------------	---

Table 4 : Controls and Indicator for Main Plot **Figure 76: Radio command front panel**

The Radio Command Front Panel in Figure 76 is used to inject, eavesdrop and decode radio packet from the WSN. All data received is in hexadecimal and it allow user to filter only the required radio packet.

Items	Descriptions
Filter: Message Type	Type of message to filter from raw data
Filter: Group ID	Group ID to filter from raw data
Message Address	Sensor node address to send the message to
Message Type	TinyOS message type
Group ID	TinyOS Group ID
Data	Radio packet to inject into the network
Send Cmd	Execute the command to send into the WSN
Send Complete	Indicate that the radio packet is transmitted successfully
Output Cluster – Decoding the received data from WSN	
Decoded Raw Data	Raw data from WSN

Chapter 6: Firmware and Software Architecture

No. of Byte	Bytes of data minus addresses, etc.
Source add	Indicate the source of the radio packet
Destination add	Indicate the destination of the radio packet
Cmd	Command from WSN
Data	Data from WSN

Table 5 : Controls and indicator for radio command

These controls on the configuration front panel in Figure 77 sets the EKF parameters and SRF08 sonar gain for the WSN. The gain of the sonar affects the tracking performance as higher gain will cause false positive reading from the reflection of the floor, while lower gain will reduce the detection range of the sensor nodes.

Figure 77: Configuration front panel

The Query Front Panel in Figure 78 is used to query the sensor nodes on the sonar and PIR status. It will trigger the node to continuously perform sonar ranging and return all information for debugging.

Chapter 6: Firmware and Software Architecture

Figure 78: Query front panel

6.3 Deployment of Mobile Sensor Node and Target Tracking

Figure 79: Deployment of mobile node

Figure 79 shows the test bed set up for demonstrating the deployment of mobile node into a sensor network with version 2 of the test bed. On the left side of the screen is the physical test bed and on the right side is the Labview GUI program. The black circle in the GUI shows the positions of the static nodes and the green dot shows the position of the mobile node. Once powered on, all static nodes will initialize the sonar sensor and set its receiver gain. It will also broadcast its localization information to the neighboring nodes within a random timing of 2 seconds. Upon receiving the localization information, each node will first check its database to see if the neighboring nodes exist and update its location, if not it will

Chapter 6: Firmware and Software Architecture

create a new entry. Due to the limited memory of the node, only neighbors within its sensor scanning range will be stored. The GUI will also store and display the node localization information when it receives localization information from each node. The user will enable the self-localization of the mobile node by pressing the "locate mobile node button". This instructs the GUI to send an instruction to the mobile node and to execute network localization scheme. The mobile node will send its localization information back to the base station and display the location as a green dot. The user can click on the screen and the mobile node will be deployed to the location not covered by the static sensor node. Once the mobile node is in place, the user will disengage both motors of the mobile robot to save power. The mobile robot will then become a static node and it will rebroadcast its final position to the entire neighboring nodes.

Videos at <http://www.ntu.edu.sg/home/yktoh/thesis.htm>

Figure 80: Deployment of mobile node

The right side of Figure 81 shows the LabVIEW GUI program tracking a moving target. LabVIEW Toolkit [1] is used for data decoding and encoding. The black circle shows the position of each sensor node. When the target is within the sensing range, the tasking sensor node will perform measurement, estimate the location of the target and report estimated results back to the base station. The target is represented by a red dot with its location and velocity highlighted in the red box. As the target moves, the estimated trajectory of the target is plotted by a

Chapter 6: Firmware and Software Architecture

green line. The selected tasking sensor is represented by a blue dot with a larger circle showing the ranging information and total energy consumption in mAs. The user can select the type of sensor selection scheme to track the target. When a target is within scanning range, the tasking sensor will track the target and send the target status with its energy information back to the base station. Depending on the sensor selection scheme, the tasking sensor will also select the next sensor node to become the tasking node.

Figure 81: Target tracking in test bed version 2

6.4 Chapter Conclusion

In this chapter, we gave a firmware overview of the sensor nodes, the TinyOS operating system and the communication protocol for the WSN. The software architecture for both the base station and PDA is also presented in this section. We also demonstrated the capabilities of deploying mobile sensor nodes into the test bed and performing target tracking with the target status displayed on both base station and PDA.

Chapter 7 Implementation of the Tracking Algorithms and Experimental Results

In this chapter, we discussed the implementation of the tracking algorithms and evaluate factors that affect the accuracy of the target tracking system with ways to improve the tracking performance. The factors that affect tracking accuracy are slow computation speed of the nodes, different sensor selection schemes, transmission time and variation in sampling time.

7.1 Factors Affecting Tracking Performance

7.1.1 Computation Time for MICAZ in TinyOS

The computation time for each sensor scheduling operation is tested firstly by the test-bed for the EKF tracking algorithm and sensor selection scheme in [1]. Unlike [1] where the computations are performed solely by the base station, each sensor node in our test-bed will perform both EKF and sensor selection computation and return the estimation results back to base.

Since these sensor nodes have very low computation power, the computation speed affects the overall response and tracking accuracy of the sensor network. During the initial development of this test bed, the long computation time to calculate the target trajectory and sensor selection caused the network to lose track of the target when it suddenly changed its direction. The EKF function needed to estimate the state of the target and the sensor selection scheme used up most of the processing time and more than half of the available RAM. Practical solutions are needed to overcome these limitations.

To reduce the computation time, we investigated the time taken by the processor to perform different arithmetic calculation tasks. From Table 6, the computation

Chapter 7: Implementation of the Tracking Algorithms and Experimental Results

for power operator takes more than 1ms and this computation time varies with the variable. By replacing power with multiplication, it significantly reduces the computation speed. For example "pow(float_a, 2)" can be replaced with "float_a * float_a", resulting in a saving of 1225 μ s. With this simple method, a few millisecond of processing time is saved for the EKF computation.

Type of calculation	Example	Time (μ s)
<i>Global Variables</i>	int_b=5; float_a=5.5; float_b=-2; int32_c=100;	-
16-bits Integer		
Constant	int_b=55*55	< 1
Addition /subtraction	int_b=int_b+15	1 to 2
Multiplication	int_b=int_b*55	2 to 3
Division	int_b=int_b/55	30
Power	int_b=pow(int_b,2) int_b=pow(int_b,1.5)	1344 1354
Square root	int_b=sqrt(int_b);	280
32-bits Integer		
Addition /subtraction	int32_c=int32_c+55	2 to 3
Multiplication	int32_c=int32_c*55	9
Division	int32_c=int32_c/55	79 to 80
Power	int32_c=pow(int32_c,2) int32_c=pow(int32_c,1.5)	1336 1331
Square root	int32_c=sqrt(int32_c)	267
Floating point		
Addition	float_a=float_a+15 float_a=float_a+0.1215151 float_a=float_a+2256 float_a=float_a+22e5 float_a=float_a+22e-5	17 20 24 34 28
Multiplication	float_a=float_a*float_a float_a=float_a*15 float_a=float_a*0.1215151 8 float_a=float_a*22e5 float_a=float_a*22e-5	45 to 46 45 45 8 45 to 46 45 to 46
Division	float_a=float_a/15 float_a=float_a/22e32 float_a=float_a/22e-32 float_a=float_a/0.12151518	67 to 68 64 to 65 64 to 65 63
Power	float_a=pow(float_a,2) float_a=pow(float_a,1.5) float_a=pow(float_b,2)	1270 to 1271 1281 to 1282 TinyOS CRASH
Square root	float_a=sqrt(float_a)	203

Chapter 7: Implementation of the Tracking Algorithms and Experimental Results

Convert 16-bits integer to floating point	float_a=(float) int_b	46 to 47
---	-----------------------	----------

Table 6 : Computation speed in TinyOS

One peculiar observation was the addition for floating number. Unlike integers; its timing varies with the complexity of the variables. This variation is around 17 μ s per calculation. But due to the frequent usages of additions in EKF, we observed that calculation speed varies in a few ms. This would affect the accuracy of the filter as the sampling interval Δt_k should be determined with consideration of the computation speed.

7.1.2 Improving Computation Time for EKF

The complexity of calculation in EKF is mainly contributed by the matrices involved in almost every step of EKF. Matrix summation/subtraction/multiplication are carried out in a coefficient-wise manner. For a certain linear stochastic dynamic model, such as the constant velocity model, some matrices in EKF derivation are sparse, i.e., many of their coefficients are zeros. We can therefore make use of sparse computation to release the heavy burden of the matrix based computation method. The case studied below elucidates the idea for one of the equation in EKF

$$\hat{X}(k+1|k) = F(\Delta t_k) \hat{X}(k|k) \quad (7-1)$$

Rearranging equation (6 -1) in a coefficient-wise way, we can obtain the four coefficients for

$$\hat{X}(k+1|k) \Rightarrow \begin{cases} \hat{x}(k+1|k) = \hat{x}(k|k) + t \cdot \hat{x}_v(k|k) \\ \hat{x}_v(k+1|k) = \hat{x}_v(k|k) \\ \hat{y}(k+1|k) = \hat{y}(k|k) + t \cdot \hat{y}_v(k|k) \\ \hat{y}_v(k+1|k) = \hat{y}_v(k|k) \end{cases} \quad (7-2)$$

Since most of the state variables in EKF computation contain zeros, applying this method and replacing power operation with multiplication in EKF calculation cut down the EKF computation time in nesC from 30ms to around 8.5ms. The computation time for periodic sensor scheduling is shown in the columns (a) and

Chapter 7: Implementation of the Tracking Algorithms and Experimental Results

(b) of Table 7 where the sampling interval Δ_k is derived based on matrix computation and sparse computation respectively.

7.1.3 Computation Time for Covariance Based Scheduling Sensor Selection

Another factor which greatly affects the computation speed is the computation time for selecting the next tasking sensor node. Under the covariance based sensor selection scheme [1], we define a tracking error $\Phi(k+1)$ at time step k based on the trace of the covariance matrix $P(k+1|k)$,

$$\Phi(k+1) = \text{Trace}(P(k+1|k)) \quad (7-3)$$

where $\Phi(k+1)$ reflects the tracking accuracy, i.e., the smaller the tracking error, the higher the tracking accuracy. In this scheme the node with highest tracking accuracy is selected as the next tasking sensor node. This scheme requires a heavy computation load as the tasking sensor node needs to calculate accuracy for all candidate nodes in the WSN. To reduce memory usage, the same EKF function is reused to calculate $\Phi(k+1)$. As shown in column (c) of Table 7, to select the tasking sensor from 9 nodes, the computation time is around 169ms to 173ms using the sparse computation for EKF. As the number of node increases so does the computation time for sensor selection. Thus if node density is high in the network, there is always a possibility of losing the target if it travels too fast or suddenly changes its direction due to the large sampling time introduced by the computation time.

7.1.4 Transmission Time and Backoff Delay in TinyOS

The Zigbee radio has a transmission rate of 250kbps and it will take $32\mu\text{s}$ to transfer 1 byte of data. Transmission of data in each packet containing sync bytes, destination address, packet type, group ID, data length, CRC and this overhead is about 10 bytes. Since the maximum packet size is 128 bytes, the time taken to transmit is around 4ms.

Chapter 7: Implementation of the Tracking Algorithms and Experimental Results

In the TinyOS radio stack [5], there are 2 types of random backoff delays; the first is the initial delay which is introduced when checking channel availability before a packet is sent. The second is the congestion backoff delay which is introduced due to retransmission when the channel is busy. The random backoff delays for both are from 320 μ s to 4.8ms and 2.72ms to 8.16ms respectively. In this case, both transmission time and random backoff time will affect the accuracy of the filter if not accurately accounted for.

7.1.5 Calculating the Sampling Interval

The sampling interval (i.e., the time difference between the measurement times of 2 successive tasking sensors) affects the tracking performance greatly. Using Δt_k to stand for the k-th sampling interval, a smaller Δt_k can be translated to a faster response time for the filter to estimate the state of the target and to select and actuate the next tasking sensor node. During the development of our real tracking system, we found that length of Δt_k should be determined according to the following factors:

- computation time for EKF calculation;
- computation time for different sensor selection scheme;
- random delay for channel access at the MAC layer;
- communication bus delay with the ultrasonic sensors in TinyOS;
- retransmission time if there is packet loss;
- measurement time used by the tasking sensor to obtain the measurement, for example, the measurement of an ultrasonic sensor is derived from the time of flight between it emits the signal and gets the reflected ultrasound signal, which introduces the (big) measurement time.

During the development of the system, precision time stamp was not implemented due to lack of memory in the nodes. Thus to solve this problem, the

Chapter 7: Implementation of the Tracking Algorithms and Experimental Results

nodes will need to calculate Δt_k as accurately as possible. The firmware flow chart to calculate Δt_k is shown in Figure 82.

Figure 82: Firmware flow chart for calculating Δt_k

Assuming that node 1 is the tasking sensor and node 2 is the next tasking sensor. Node 1 will measure the duration for EKF calculation, sensor selection, transmission time, random backoff delay and retransmission (if any) of node 1. This timing denoted as T4 will be encapsulated into the Token Key and transmitted to node 2. Node 2 will measure the sampling time and add it with T4 to get the next Δt_k for EKF and sensor selection scheme. By having the node

Chapter 7: Implementation of the Tracking Algorithms and Experimental Results

calculating it owns Δt_k solves the problem of random and deterministic computation and transmission delays in the WSN. The results of the Δt_k for different sensor selection schemes are found in Table 7.

7.2 Test Results for Sensor Selections

7.2.1 Nearest-neighboring Based Sensor Selection

The Nearest-neighboring based scheduling is demonstrated using test bed version 1. The tracking is started with node 1 and node 2 measuring the distance from the target and triangulation is used to compute XY coordinates of the target for the initial condition in EKF. At the next time step, node 1 (current tasking sensor) will perform EKF to estimate the state of the target and transmit back to base station for display. Node 1 will continue to perform ranging and EKF for 5 times and pass Token Key to the nearest neighboring node N_j to become the next tasking node. In this scheme the fixed 0.15s sampling time Δt_k is used in the EKF computation. The following is a summary of the sequences for the target tracking in this test bed.

1. N_1 receives command from base station to start tracking
 2. N_1 performs ranging d_1
 3. N_1 commands N_2 to perform ranging d_2
 4. N_2 returns d_2
 5. N_1 computes initial XY position of the target
 6. N_1 performs ranging d_1 , compute EKF to estimate the Target State and transmit back to base
 7. Repeat step 6 for 5 times
 8. N_1 searches internal database for nearest node
 9. N_1 transmits Token Key to N_j
 10. N_j perform ranging d_j , EKF to estimate the Target State and transmit back to base
-

Chapter 7: Implementation of the Tracking Algorithms and Experimental Results

11. Repeat step 10 for 5 times
12. N_j searches internal database for nearest node to pass Token Key
13. Repeat Step 10

Figure 83: Target tracking with Nearest-neighboring based scheduling scheme

The Nearest-neighboring based scheduling scheme shown in Videos at <http://www.ntu.edu.sg/home/yktoh/thesis.htm>

Figure 83 can accurately track a target provided the target moves with near constant velocity or don't suddenly changes its direction. Unfortunately this scheme suffers from "Mirror Effect", this happens when the target suddenly stops in the centre of the 2 adjacent nodes and suddenly moves in the opposite direction. Since EKF is based on the constant velocity model, the filter is unable to keep up with the sudden changes of target trajectory and estimate that the target is moving in the opposite direction.

7.2.2 Covariance Based Sensor Selection

Chapter 7: Implementation of the Tracking Algorithms and Experimental Results

The covariance based sensor selection scheme is demonstrated using test bed version 2. In this WSN tracking system, only one sensor node is tasked to activate its ultrasonic sensors to measure the distance from the target and the Token Key identifies the tasking sensor. After measurement, this sensor node will perform the computation task of estimating the position and velocity of the target. The tasking sensor will report the target information back to the base station and select the next best tasking sensor node for continuous target monitoring. The sensor selection process is very computationally intensive if the complex calculation for new tasking node selection from all the candidate neighboring nodes is performed solely by the current tasking sensor. To prevent the loss of target due to the long computation time, the tasking sensor node is elected from the candidate tasking sensors in a distributed way via the accuracy based random competition. A candidate sensor node with highest predicted tracking accuracy will be elected as the new tasking sensor.

Columns (c), (d) and (e) in Table 7 show the results for covariance based sensor selection scheme before and after distributing calculations among nodes. As described in section 7.1.3, the computation time increases with number of nodes. In Table 7 (d), to improve the computation speed, the tasking sensor can broadcast its Token Key to all neighboring nodes. All sensor nodes will calculate $\Phi(k+1)$ with equation (7-3) and send back to the tasking sensor. The tasking sensor will select the next tasking sensor node with the smallest $\Phi(k+1)$. This method might seem to be an efficient way to reduce computation time compared to the non-distributed scheme in Table 7(c), but transmission time and backoff delay will still need to be factored in. Having all nodes replying back to the tasking sensor will not only decrease the bandwidth available to the WSN, the radio backoff delay for each node will also increase with congested radio channel.

Instead of all nodes transmitting their $\Phi(k+1)$, in the proposed competition based sensor scheduling scheme, only node with the smallest $\Phi(k+1)$ will declare itself

Chapter 7: Implementation of the Tracking Algorithms and Experimental Results

as the next tasking sensor node. Using the equation (7-4), we can calculate the random backoff time to reply to tasking sensor. Assume

$$\text{max_backoff_timer} = \frac{\Phi(k+1)}{\Phi(k+1)_{\max}} \times \text{backoff_threshold} \quad (7-4)$$

where max_backoff_timer is the upper bound for backoff time in millisecond to trigger a transmission back to tasking sensor, $\Phi(k+1)_{\max}$ is the maximum trace value in our WSN and backoff_threshold is the maximum backoff threshold which is selected to be 10ms for our WSN system. The true backoff time for the sensor node to declare itself as the tasking sensor node is a random value between 0 and max_backoff_time. Figure 85 shows the protocol for tasking sensor node competition.

Figure 85: Protocol for tasking sensor node election

The procedure is summarized below:

- Tasking node discovery:* The tasking sensor node will broadcast the NREQ (Tasking Node Request) message to sensor nodes in its neighbourhood. Using the target information in $X(k|k)$, each node will know whether the target is possibly within its sensing region. If so, it will use the equation (6-4) to calculate the max_backoff_timer and its random backoff time. A 15ms timer is

Chapter 7: Implementation of the Tracking Algorithms and Experimental Results

Initial random backoff delay	0.32 to 4.8				
Congested random backoff delay		-	-	Min : 2.72×7 = 19.04 Max : 8.16×7 = 57.12	-
NAGN & overhead	-	-	-	.48	.48
$\Phi(k+1)$ or NREP & overhead	-	-	-	2.464	2.464
NREQ+ Token Key & overhead	3				
Sampling time Δt_k	Min: 112 Max: 117	Min: 94 Max: 98	Min :169 Max :173	Min : 124 Max : 158	Min: 105 Max :118

Table 7 : Computation time for sensor selection schemes

Experiments were carried out to demonstrate the tracking system performances with different sensor selection schemes. We program the Amigo robot to move at a speed of 250mm/s and Figure 86 shows the experimental results for a benchmark circular target motion pattern. In the top graph, the yellow triangle shows the position of the 9 sensor nodes, the blue trajectory is the ground truth and the pink the estimated trajectory. The bottom graph shows the position errors for x and y. Figure 86 (a) shows the result for periodic sensor scheduling using matrix computation where the tracking error increases as the WSN had difficulty to keep up with the changing path. With Sparse Computation, the periodic sensor scheduling in Figure 86 (b) achieves a much better tracking accuracy due to the shortening of sample time. The number of nodes in this experiment is small; however periodic sensor scheduling does not work well for a large scale network.

Chapter 7: Implementation of the Tracking Algorithms and Experimental Results

Figure 86: Comparison between different sensor selection schemes
a) Periodic scheduling with EKF by Matrix Computation, b) Periodic scheduling with EKF by Sparse Computation, c) Covariance based scheduling with EKF by Sparse Computation, d) Distributed competition scheduling with EKF by Sparse Computation

For covariance based tracking shown in Figure 86 (c) even with Sparse Computation, WSN loses the target due to the large computation time needed to calculate the next tasking sensor. Figure 86 (d) shows the superior performance of our proposed distributed competition based sensor scheduling scheme. It successfully tracks the target with the smallest tracking error. Comparing column (b) with (e) in Table 2, even with the longer sampling time, we have a much better tracking accuracy compared to the periodic scheduling with improved EKF.

7.3 Nearest-neighboring Round Robin Sensor Selection

Nearest-neighboring round robin scheme is demonstrated with version 3 of the test bed. This scheme is similar to the Nearest-neighboring based scheduling scheme from 1st test bed except that token is passed between 3 or more sensor nodes depending on the trajectory of the target. The following is a summary of the sequences for the target tracking for this test bed.

1. N_{26} receives command from base station to start tracking

Chapter 7: Implementation of the Tracking Algorithms and Experimental Results

2. N_{26} performs ranging d_{26}
 3. N_{26} commands N_{27} to perform ranging d_{27}
 4. N_{27} returns d_{27}
 5. N_{26} computes initial XY position of the target
 6. N_{26} performs ranging d_{26} , compute EKF to estimate the Target State and transmit back to base
 7. N_{26} predicts the trajectory of the target
 8. N_{26} searches internal database and select nearest node N_i base on the predicted trajectory of target
 9. N_{26} transmits Token Key to N_i
 10. N_i performs ranging, compute EKF to estimate the Target State and transmit back to base
 11. N_i predicts the trajectory of the target
 12. N_i searches internal database for nearest node N_j base on the predicted trajectory of target
 13. **IF** N_j is the previous node, select the next nearest sensor node N_k , **else** select N_j
 14. Repeat step 10
-

Figure 87 demonstrate that the 20 sensor nodes WSN is able to successfully track a human moving target.

Chapter 7: Implementation of the Tracking Algorithms and Experimental Results

Videos at <http://www.ntu.edu.sg/home/yktoh/thesis.htm>

Figure 87: Human target tracking with Nearest-neighboring Round Robin Scheme

7.4 Chapter Conclusion

In this chapter, we discuss factors affecting the tracking performance caused by computation and communication delays. We implemented both the sparse computation algorithm and delay compensation to improve the EKF tracking performance. We also demonstrated the superior tracking performances of our distributed competition scheme with experimental comparison with existing sensor scheduling schemes.

Chapter 8: Conclusion and Future work

Chapter 8 Conclusion and Future work

8.1 Conclusion

In this thesis, the development of a wireless sensor network for target tracking was discussed. Problems affecting the accuracy of the target state estimation such as limited processing power of sensor nodes, sampling time and different sensor selection schemes were also studied and further analyzed. Solutions were then devised to overcome these problems in order to improve the tracking performance in a distributed wireless sensor network. A novel competition-based sensor node election scheme was also proposed and implemented in a real-time WSN target tracking system. The performance of our competitive distributed sensor scheduling scheme was evaluated and found to be better than other conventional schemes.

8.2 Further Research

The following are topics for possible future research.

1) Node election with energy consideration.

In the existing platform, a node energy monitoring scheme has been developed and implemented. However, node energy has not been taken into consideration in the selection of tasking sensor node.

2) Competition scheme for multiple sensor nodes selection.

In the existing platform, only a single sensor is selected at each time step. In the future, we shall consider the selection of multiple sensor nodes to improve tracking performance.

Chapter 8: Conclusion and Future work

3) Evader/pursuer games.

In the existing platform, the tasking sensor node forwards the target trajectory to the base station. In the future, the target (evader) trajectory will be routed to a pursuer and the pursuer will perform path planning to intercept the evader.

4) Multi-agent robots.

In multi-agent robots, tasks are to be efficiently allocated to each robot in order to cooperatively achieve a global goal such as determination of the optimal solution to intercept the evader. In the future, these robots will use the existing platform to localize their positions and to extend their range of communication by using the sensor nodes to route information between robots.

Videos at <http://www.ntu.edu.sg/home/yktoth/thesis.htm>

Figure 88: Multi-agent robots interception in WSN

Bibliography

Author's Publications

- [1] T. Yang, Y. Toh, L. Xie, "Run-time Monitoring of Energy Consumption in Wireless Sensor Networks", In *Control and Automation, ICCA 2007, IEEE International Conference*, Page(s): 1360-1365, Guangzhou, China, May 2007.
- [2] Y. Toh, W. Xiao, L. Xie, "A Wireless Sensor Network Target Tracking System with Distributed Competition based Sensor Scheduling", In *Intelligent Sensors, Sensor Networks and Information, 3rd International Conference*, Page(s):257-262, Melbourne, Australia, Dec 2007.

Bibliography

Bibliography

- [1] P. Dutta, J. Hui, J. Jeong, S. Kim, C. Sharp, J. Taneja, G. Tolle, K. Whitehouse, D. Culler, "Trio: enabling sustainable and scalable outdoor wireless sensor network deployments.", In *Information Processing in Sensor Networks, The Fifth International Conference*, page(s): 407 - 415, Nashville, TN, USA, April 2006.
- [2] C. Sharp, S. Schaffert, A. Woo, N. Sastry, C. Karlof, S. Sastry, D. Culler, "Design and implementation of a sensor network system for vehicle tracking and autonomous interception.", In *Wireless Sensor Networks, Proceedings of the Second European Workshop*, page(s): 93 - 107, Istanbul, Turkey, Jan 2005.
- [3] P. Dutta, A. Arora, S. Bibyk, "Towards radar-enabled sensor networks." In *Information Processing in Sensor Networks, The Fifth International Conference*, page(s): 467 - 474, Nashville, TN, USA, April 2006.
- [4] W. Kim , K. Mechitov , J. Choi , S. Ham, "On target tracking with binary proximity sensors", In *Information Processing in Sensor Networks, Fourth International Symposium*, Page(s): 301 - 308, Los Angeles, California, USA, April 2005.
- [5] L. Parker, B. Birch, C. Reardon, "Indoor Target Intercept Using an Acoustic Sensor Network and Dual Wavefront Path Planning", In *Intelligent Robots and Systems, Intelligent Robots and Systems, IEEE/RSJ International Conference*, Page(s): 278 - 283, Las Vegas, USA, Oct 2003.
- [6] J. Liu, J. Reich, F. Zhao, "Collaborative in-network processing for target tracking," In *Journal of Applied Signal Processing*, Page(s): 378 - 391, April 2003.
- [7] W. Xiao, J. Wu, L. Shue, Y. Li, L. Xie, "A Prototype Ultrasonic Sensor Network for Tracking of Moving Targets", In *Industrial Electronics and Applications, 1ST IEEE Conference*, Page(s): 1511-1516, Singapore, May 2006

Bibliography

- [8] W. Xiao, J. Kang Wu, L. Xie, "Adaptive Sensor Scheduling for Target Tracking in Wireless Sensor Network", In *Proceedings of the SPIE Optics and Photonics: Advanced Signal Processing Algorithms, Architectures, and Implementations*, Page(s): 59100B-1 - 59100B-9, San Diego, California, USA, July 2005.
- [9] P. Ballal and S. Gorthi, "LabVIEW Toolkit", Website <http://arri.uta.edu/acs/WSN/toolkit/>.
- [10] J. Polastre, TinyOS source code for CC2420RadioM.nc v1.42 and TimerJiffyAsyncM.nc v 1.1.
- [11] Crossbow Technology, Wireless sensor networks manual rev. b, April 2005.
- [12] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. , J. Hill, M. Welsh, E. Brewer, D. Culler, User manual "TinyOS: An Operating System for Sensor Networks", Ambient Intelligence.
- [13] D. Gay, P. Levis, R. Behren, M. Welsh, E. Brewer, D. Culler, "The nesC language: A holistic approach to networked embedded systems", In *ACM Special Interest Group on Programming Languages*, Volume 38 issue 5 , May 2003.
- [14] SRF08 Ultra Sonic Range Finder Technical Specification. Website <http://www.robotelectronics.co.uk/html/srf08tech.shtml>.
- [15] J. Hui, User manual "Deluge 2.0 - TinyOS Network Programming".
- [16] J. Thorn, "Deciphering TinyOS Serial Packets", Octave Tech Brief #5-01.
- [17] W. Xiao, J. Wu, L. Xie, L. Dong, "Sensor Scheduling for Target Tracking in Networks of Active Sensors", In *ACTA AUTOMATICA SINICA*, Vol.32 No. 6, Page(s): 922-928, Dec 2006.
- [18] T. Yang, Y. Toh, L. Xie, "Run-time Monitoring of Energy Consumption in Wireless Sensor Networks", In *Control and Automation, ICCA 2007, IEEE International Conference*, Page(s): 1360-1365, Guangzhou, China, May 2007.
- [19] S. Henion, "Discharge Curves for NiCad and NiMH Batteries", Website <http://shdesigns.org/batts/battcyc.html>