Development of a Portable XBee C Library and RSSI Triangulation Localization Framework

Samuel L. Shue and James M. Conrad
Dept. of Electrical and Computer Engineering
University of North Carolina at Charlotte
Charlotte, NC, USA
slshue@uncc.edu, jmconrad@uncc.edu

***Abstract*— Wireless Sensor Networks is a growing research field and researchers use a variety of wireless devices to perform radio communications. The XBee is a common wireless module that has become very popular amongst hobbyists and researchers alike due to its simple and efficient design. However, to use the XBee in complex networks, the user must construct specific packets to communicate within an XBee network. Here we will introduce an XBee C library which provides a modular, hardware-independent C library for XBee. Also using this library, we will explore establishing a framework for performing RSSI triangulation localization without the use of anchor nodes.**

Keywords—WSN; XBee; RSSI; Triangulation; Localization

I. INTRODUCTION

Wireless Sensor Networks (WSN) is a growing research field, and as a result, there is a growing need for a variety of development platforms. Development platforms allow researchers to quickly implement experimental ideas, such as routing protocols, sleep cycles, localization, and sensor data collection. These features are very useful for getting a project up and running quickly, but they come at a high price, such as proprietary software, limited support, and limited hardware selection. These reasons have motivated the creation of a modular development board designed to interface with the popular Digi XBee wireless modules.

The XBee has become very popular amongst hobbyists and researches because of its simple interface, yet efficient design. The XBee has two modes of operation: Transparency mode and API mode. Transparency mode allows the XBee to serve as a "cable replacement", where all packets are broadcasts to all modules on the network. API mode requires the user to create specific packets for interfacing with module and sending messages throughout the network. The API mode provides many features useful in a complex research project, however, despite the XBee's popularity, the only complete library for abstracting the API functions is written for the Arduino development environment [1].

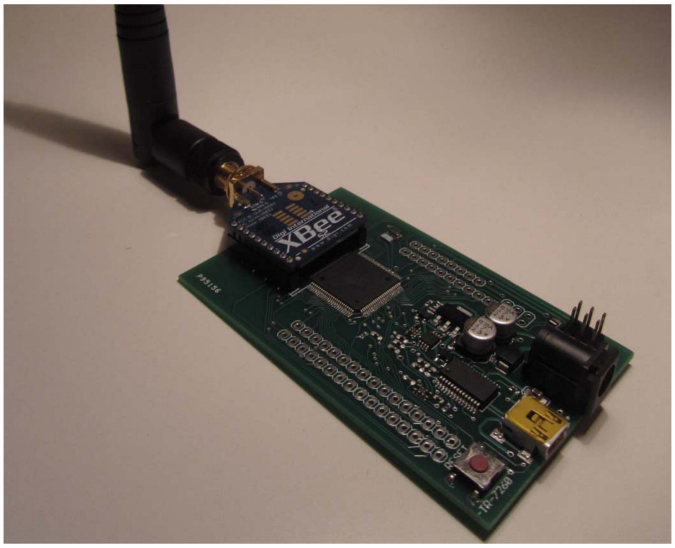

Fig. 1. Image of the Motesquito development board.

II. BACKGROUND

A. Identifying the Need

The need for a light, wireless interface was identified while trying to integrate a wireless sensor networks into a robotics project, where custom sensors needed to be integrated with the development platform. The writing custom drivers for other WSN development platforms was not practical, so a custom solution was created: a development board with a socket for connecting an XBee. The XBee comes in a variety of different modules, such as Wi-Fi, ZigBee, and 800 MHz Zigbee implementations, and the XBee footprint has become a standard that other manufacturers have begun creating wireless modules to adhere to [2]. The XBee also keep the same packet structure between the different modules, so a library could be created to allow the user to easily change wireless modules without heavily modifying the application code.

978-1-4799-6940-1/14/$31.00 ©2014 IEEE

B. Triangulation Localization

The XBee also provides access to RSSI values, which can be used to localize each sensor node through triangulation techniques. This can be especially useful within a swarm of robots providing a relative distance between each other and sharing mapping information.

An advantage of using sensor networks in a mobile robotic swarm, is the access to the robot's sensor data and motion. Using the robot's odometry readings, the sensor network can use the movement of the robot to eliminate the need for anchor nodes in a triangulation localization scheme. Typically, when performing localization, the location of at least three nodes needs to be known a priori. Using the received signal strength indicator (RSSI) distance approximation from three nodes with known locations, the location of a fourth node can be determined.

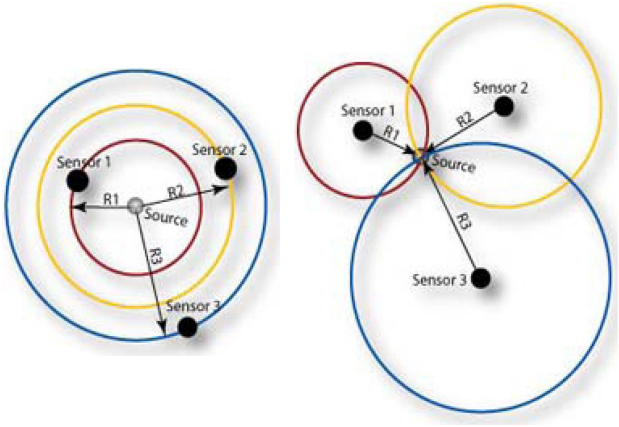

Fig. 2. Localization of a fourth node using distances from three nodes of known location. Distances from each node become the radius of a circle, and the overlap between each circle determines the location of the fourth node [1].

When establishing triangulation without anchor nodes, the primary problem is avoiding creating an inverted map. If you create the map around the first node deployed, calling it point (0,0), and calling the next node (0,Y) where Y is the estimated distance from the first node, the third node can have 2 possible locations, -X or +X, as illustrated in Fig. 3.

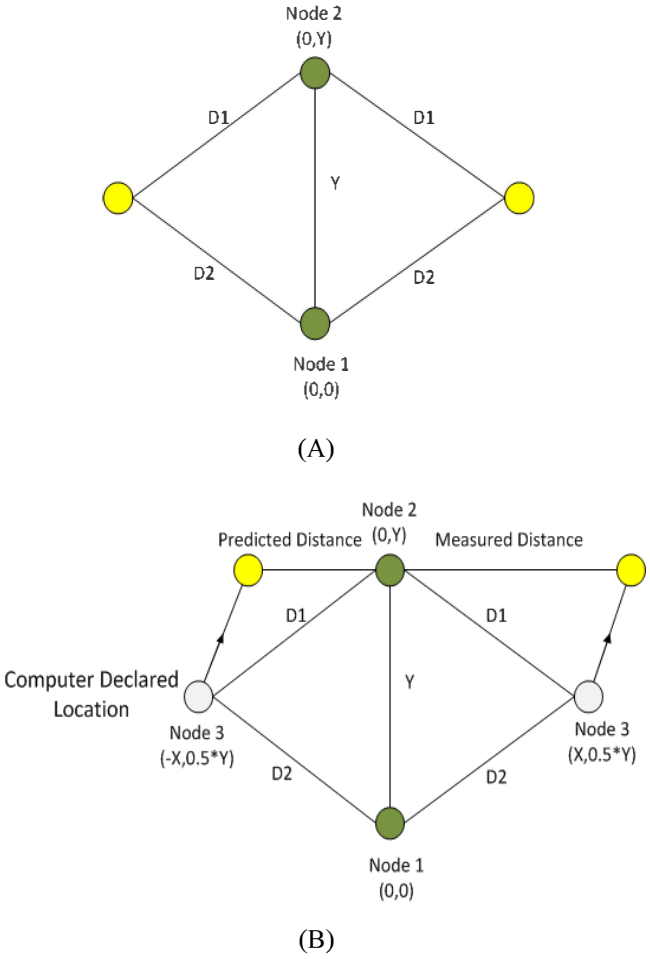

Fig. 3. Establishing the frame of reference for anchor-free triangulation. (A) shows two localized nodes (green circles) and the two possible locations for a third node (yellow circle) given distances D1 and D2 to node 1 and 2, respectively. (B) indicates a possible solution through displacement of the third localized node along the X axis. If the estimated location was correct (yellow circles), the new location based on a known displacement should match the new location based on triangulation values. If the triangulated location is changing inversely to the displacement values, then the X-axis needs to be inverted.

III. XBee Library

A. Interfacing

XBee modules are available in several different models, but the most common are the series 1 and series 2. The series 1 implements the IEEE 802.15.4 standard, which covers the MAC and Physical Layers of the OSI networking model. As a result, the series 1 only perform peer to peer communications and cannot support multihop alone because of the lack of a networking protocol. The series 2 implement the complete Zigbee protocol, which supports multihop communication. To cover a large area, multihop communication is preferred, so XBee series 2 is used.

All XBee modules are interfaced via USART serial communication. The series 2 XBee can operate using several different versions of firmware. The ZNET firmware runs a modified form of Zigbee, with several different function sets of the firmware for different roles in the network. The function sets with "coordinator" in the name causes the XBee to function as the Zigbee coordinator for the network of its specified PAN (Personal Area Network) ID. Other functions sets with "Router/End Device" allow the XBee to function as either a router or an end device, depending on its position within the network topology. There are also "API" and "AT" function sets. The AT function sets allow the XBee to function as a "cable replacement", which passes each incoming byte to the set destination address. The API function set allows the

user to create custom packets to specify destination addresses, limit number of hops a message can travel, request RSSI value, and set module configurations.

B. Xbee API Packet Structure

The XBee API function set requires communication with the module to be performed through a structured interface. This interface specifies how messages and commands are sent to the module.

API has two modes, with escape characters and without escape characters. Escape characters allow the user to exclude certain bytes from the structure of a packet, and is currently not used in this software library. The packet structure for normal API operation is shown in Fig 4. Each packet begins with a start delimiter and length, followed by the packet frame data, and ending with a checksum. The length bytes indicate the length of the frame data, not including itself, the start delimiter, or the checksum. The structure of the frame data can change depending on the structure of the API type.

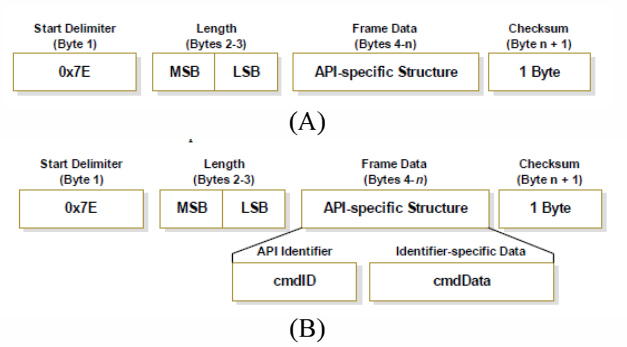

Fig. 4. (A) API mode basic packet structure. Each packet is preceded by a start delimiter (0x7E), length and suffixed with a checksum byte [XBee S2 Documentation Citation]. (B) Data frame structure. Each frame begins with an API identifier, which indicates the structure of the cmdData frame.[4]

Within the frame data the first byte identifies the API type, which is illustrated in Fig. 4. Each API type contains a specific structure depending on the function. In this library, functions have been created to support transmitting the "AT Command" and "Zigbee Transmit Request" and receiving the "AT Command Response" and "Zigbee Receive Packet."

Constructing packets for AT Commands have been abstracted even further. AT commands are how settings and internal values of the module are accessed and assigned. Each AT command has a 2 character identifier, and possibly followed by a value. For instance, to request the 16 bit address of the XBee, the user can send "MY" in an AT Command packet. The user can then wait for the AT Command Response packet to receive the address value. Rather than constructing the packet and waiting for the response, a single function has been created, get16BitAddress(), to allow the user to request the value without constructing the packets individually.

C. Hardware Abstraction Layer

A Hardware Abstraction Layer (HAL) has also been created for the Library, to help keep the library portable. The hardware abstraction layer contains macros which abstract microcontroller specific registers, interrupt service routines, and peripheral configurations. Hardware specific functions are also contained within the HAL, such as the initialization for the USART module which the XBee is attached. Each set of definitions and functions is contained with a #ifdef macro, where the user simply defines the microcontroller name that they wish to use, and the correct set is automatically selected.

If a user wants to use an unsupported microcontroller, they would only have to write macros for the USART data register, checking if the transmit buffer is empty, checking if the receive buffer is empty, initializing the USART, and the USART receive interrupt service routine.

IV. Triangulation Localization

A. RSSI Distance Estimation

To perform triangulation localization through RSSI, a way to relate signal strength to distance must be defined. RSSI distance approximation is known to only provide ~1 meter of accuracy. To approximate distance from the RSSI value, the free space propagation model is used in equation (1):

$$RSSI = -(10n\log10(d) + A) \quad (1)$$

Where *RSSI* is the RSSI value received (dBm), n is the path-loss exponent, d is the distance, and A is the RSSI value at a reference distance. [5] Each of these variables need to be modified for hardware configuration and environment. The path loss exponent, n, has to be determined experimentally for the environment in which the network is operating and ranges from values of 1.6 to 4.3, but for inside an office building or residential home is around 2.8. When proper values have been determined, this approximation should yield accurate values with a ~1m resolution, given line-of-sight to the transceiver.

B. Software and Network Configuration

To calculate approximated distances, store node to node distances, node locations, and display coordinates, a MATLAB program was created. By using MATLAB to compute distances and locations, the processing load on each node is substantially decreased, allowing for longer battery life and increased performance.

The software on each node simply transmits a packet to search for neighbors one hop away. When a node receives a packet, it requests the RSSI value of the packet via AT Command, then transmits the RSSI value, the source address, and its own address to the coordinator. The coordinator is

connected to a PC running the MATLAB program and only passes packet contents to MATLAB.

The MATLAB code cycles through 3 states: Establishing the origin and the Y axis, Establishing the X axis, and node localization. In the initial state, the code waits until the first packet arrives. Each packet contains the receiving address, the RSSI value, and the source address. The first receiving address is declared the origin of the virtual map, and is given (0,0) for its location. The source address is then given location (0,Y), where Y is its approximated distance from the first node from the RSSI value. A list is then created and maintained with each address, a distance value, and the address of the node corresponding to the distance value. The program then keeps populating the list with values as they arrive, and scans through the list until a third node is identified that has distance values to the first two established nodes. Using those distance, two circles are drawn around the first two nodes, resulting in two possible locations for the third node, as seen in Fig. 3 (A). For now, one of the two solutions is selected. To correct the values if the wrong node is selected, the user simply has to invert the sign of each X coordinate value. After three nodes have been assigned location values, the MATLAB code moves into its final state, in which it continues populating the list and updating location values.

C. *MATLAB GUI*

A MATLAB GUI was created to help visualize the data. The GUI includes a graph which populates node locations with a green dot. The GUI also includes node to node distances, and a button to print out node data. Fig. 5 shows an example of the GUI along with three localized nodes.

V. CONCLUSION AND FUTURE WORK

This article describes the development of an XBee C library and the framework of a localization algorithm which can be implemented using it. The library's modular structure allows it to be easily modified for other hardware platforms, and this localization application demonstrates how the library can be used.The MATLAB software currently does not account for motion within the network or updating each node's location. In future work, the program will adapt to localize any number of nodes and update locations as the topology changes and as the program becomes more confident in node locations.

The triangulation framework and library provides a ground for implementing the localization scheme on a robotic swarm equipped with wireless sensor nodes. Using the robot's accelerometer, magnetometer, and odometry data, a node displacement can be measured and applied to correct the inverted map problem.

REFERENCES

[1] "XBee-arduino", Available: https://code.google.com/p/xbee-arduino/

[2] "Bee Series", Available: http://www.seeedstudio.com/wiki/Bee_series

[3] "RSSI", Available: http://www.greatnote.com/2007_12_01_archive.html

[4] "Xbee Series 2 Documentation", Available: https://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-2.5-Manual.pdf

[5] Erin-Ee-Lin Lau; Wan-Young Chung, "Enhanced RSSI-Based Real-Time User Location Tracking System for Indoor and Outdoor Environments," Convergence Information Technology, 2007.

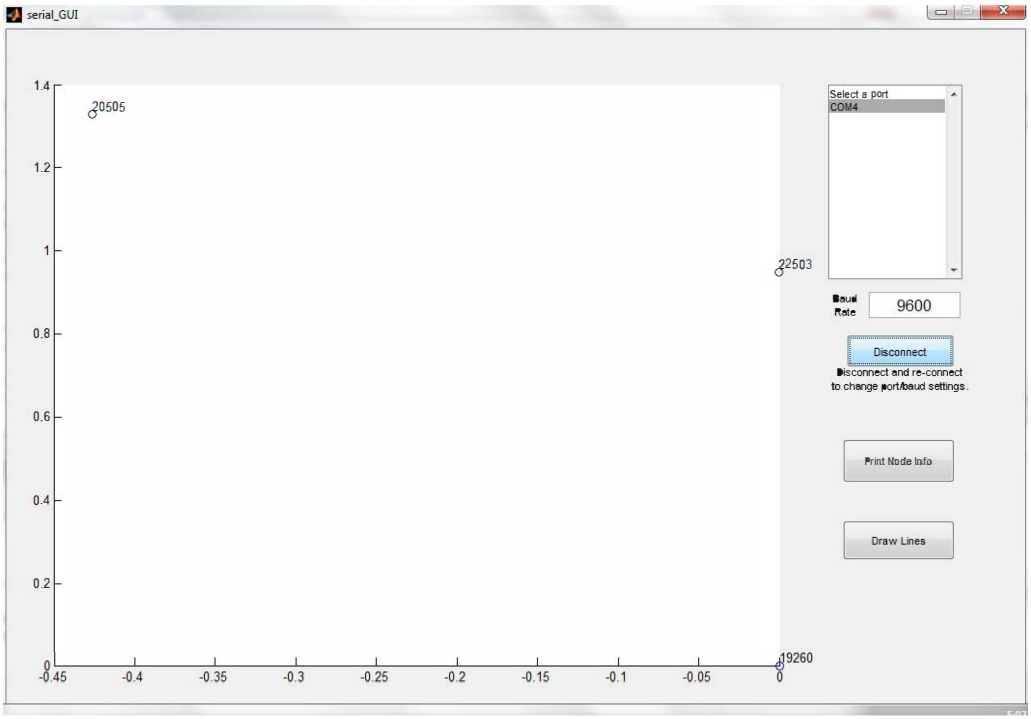

Fig. 5. The MATLAB GUI. The user may select the serial port which the Xbee coordinator is attached, specify the baud rate, connect and disconnect, print node information, and draw lines from node to node.