AC 2010-169: MESH-NETWORKED MOBILE ROBOTS: A FRAMEWORK OF LABORATORY EXPERIMENTS FOR COURSES IN WIRELESS COMMUNICATIONS

Wookwon Lee, Gannon University

Wookwon Lee, P.E. received the B.S. degree in electronic engineering from Inha University, Korea, in 1985, and the M.S. and D.Sc. degrees in electrical engineering from the George Washington University, Washington, DC, in 1992 and 1995, respectively. He is currently on the faculty of the Department of Electrical and Computer Engineering at Gannon University, Erie, PA. Prior to joining Gannon, he had been involved in various research and development projects in communications for more than 12 years in industry and academia.

Sreeramachandra K. Mutya , Gannon University

Sreeramachandra K. Mutya received his Bachelor's degree in electronics and communication engineering from Bharath University, Chennai, India in 2007. He is pursuing a master's degree in electrical engineering at Gannon University, Erie, PA, where he currently works as a graduate research assistant. His research interests include wireless communications, computer communications, and real-time systems.

Kirankumar Palthi , Gannon University

Kirankumar Palthi received his Bachelor's degree in electronics and communication engineering from Jawaharlal Nehru Technological University, Hyderabad, India in 2008. He is pursuing a master's degree in embedded software engineering at Gannon University, Erie, PA, where he currently works as a graduate research assistant. His research interests include wireless communications, signal processing, embedded systems, and digital Electronics.

Mesh-Networked Mobile Robots: A Framework of Laboratory Experiments for Courses in Wireless Communications

Abstract

In this paper, we present an exemplary framework suitable for laboratory experiments for undergraduate courses in communications. Initially designed to be a test-bed for a small wireless mesh-networked system, the framework consists of a graphical user interface (GUI) for a control center, a software-based interface referred to as the Synapse Portal, a mesh-networking node referred to as the Bridge, and multiple mesh-networking End Nodes each of which is integrated into a three-wheel mobile robot. The development of the test-bed requires an integration of two microcontrollers of different code-execution speeds in cascade. In this paper, along with design details and relevant specifics of all components of the test-bed, we discuss issues encountered during the development and how we addressed them to successfully realize a functioning mesh network of mobile robots. Based on our observations during the development, we believe that the test-bed can be useful for addressing ABET engineering criteria, and also developing a set of small-scale laboratory experiments for undergraduate courses in the areas of communications.

I. Introduction

The work presented in this paper initially began as a small research project involving master-level graduate students on indoor positioning. Research on indoor positioning has been intense over the past few years to facilitate a broader spectrum of location-based services and applications. It is well known that due to the inherent limitations of the satellite signals, the global positioning system (GPS)-based technologies do not work well in indoor environment. Several alternative approaches have been reported in the literature, and they can be classified largely into four categories: i) infrared signal-based, ii) ultrasound signal-based, iii) microwave (satellite) signal-based, and iv) radio-frequency (RF) signal-based. Partly due to the advantages in signal propagation under diverse indoor scenarios, the RF signal-based approach has become most popular for indoor positioning systems[1-4]. In these approaches, location estimation techniques are typically based on one or more of statistical parameters such as received signal strength (RSS), time of arrival (TOA), and angle of arrival (AOA). More recently, a new approach was proposed based on a new statistical parameter called the *angle of transmission* (AOT)[5]. The AOT is a spatial direction of the main lobe of a beam pattern generated by transmit-beamforming. The approach exploits the spatial information embedded in the signal transmitted from an antenna array. In this scheme, each mobile node with an omni-directional, single-element antenna estimates the AOT based on one (or more than one) signature signal(s) transmitted from the fixed node equipped with an antenna array. The mobile node then further estimates its distance from the fixed node based on the RSS of the signature signal to ultimately be able to pin-point its location in the polar coordinates where the fixed node is assumed to be at the origin.

Although our work on a prototype of an indoor positioning system exploiting AOT will need to be continued to the next stage, up to the current state as described below with more details, the

project has produced a complete test-bed system that can be used as a basis for developing laboratory experiments for undergraduate courses in wireless communications. The rest of this paper is organized as follows. Section II briefly describes our test-bed system, Section III provides details of implementation for mesh networking capability among the network nodes, and Section IV provides implementation details for mobility control of mobile robots. In Section V, relevant ABET criteria are discussed along with brief descriptions of suggested laboratory experiments. Finally, concluding remarks are provided in Section VI.

II. System Description

A. Test-bed of an Indoor Positioning System

Figure 1 shows a simplified diagram of our intended indoor positioning system consisting of a control center, a fixed node, and multiple mobile nodes. An antenna array is included at the fixed node for indication of a beamforming-based indoor positioning approach. The fixed and mobile nodes form a mesh network operating at the 2.4 GHz frequency band based on the IEEE 802.15.4 radio[7] and Synapse Network Appliance Protocol (SNAP)[9]. The control center is for control of mobile nodes' motions and also for processing of data such as location information from mobile nodes. Each mobile node has capability to roam around and is desired to estimate the angle-of-transmission θ and distance d for its location within the polar coordinates with the fixed node at the origin. Depending on the need from a particular application/service utilizing the location information, the mobile node may report its location to the control center through the fixed node/ gateway. If necessary, all or some nodes in the mesh network can share their location information as the mesh network protocol can easily accommodate a network node to communicate with the others in the same network.

In our test-bed, the control center is a host computer for a graphical user interface (GUI) developed in our lab with Microsoft Visual Basic. In the GUI, eleven commands are associated with buttons that can be clicked with a mouse to control mobile nodes' moving directions: Stop, Forward/Reverse, Forward/Reverse Right/Left by 30 degrees, and Forward/Reverse Right/Left by 60 degrees. The control center is also a host computer for an additional graphical user interface referred to as the Synapse Portal. For a mesh network to deliver control commands from the GUI to a mobile node (or multiple mobile nodes), Synapse's Bridge Node and Protoboards[9] (also referred to as End Nodes in this paper) are adopted for the fixed node and mobile nodes, respectively. Synapse's Bridge and End Nodes are a microcontroller board with a System-on-Chip (SoC) called the RF Engine[10]. Each RF Engine combines a Freescale S08GT family microcontroller (more specifically, MC9S08GT60A), an 802.15.4 radio, and an antenna. Synapse nodes are managed by the Synapse Portal for application-specific programming in python and loading the codes into the microcontroller. For the purpose of

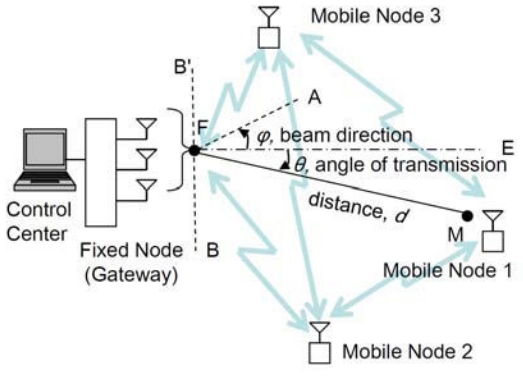

Figure 1. A simplified diagram of a test-bed for an indoor positioning system

communicating with other mesh-networking nodes, i.e., Bridge Node and End Nodes, the Synapse Portal has a virtual network node referred to as the Portal Node. The Synapse Bridge Node/fixed node is in one end directly connected by a serial cable to the control center and in the other end communicates with End Nodes/mobile nodes based on the SNAP/IEEE 802.15.4 radio. An individual End Node is mounted on a three-wheel mobile robot, referred to as ARobot[11], and maintains a physical connection by wire with a microcontroller for the robot, referred to as the BASIC Stamp[12]. The BASIC Stamp controls the motion of ARobot.

B. Location Estimation for Indoor Positioning

Implementation of a location estimation method on mobile nodes is based on our previous work on AOT estimation[5]. We briefly summarize the algorithm here for completeness of the description. When the fixed node broadcasts its signature signal $s(t)$ toward a predetermined angular direction, the instantaneous transmitted baseband signal can be expressed as $s(t) = \sqrt{P_t}\mathbf{w}s(t)$, where the $(\mathrm{M}\times 1)$ vector $\mathbf{w}$ represents the beam-forming coefficients of the antenna array and P_t is the transmitted signal power. The instantaneous received baseband signal $y(t)$ at the mobile node in a flat fading channel can then be written as[5]

$$y(t) = \sqrt{\xi}c\mathbf{a}^T(\theta)\mathbf{x}(t) + n(t) \tag{1}$$

where ξ is the power scaling factor for large-scale path loss, c is the channel gain for small-scale flat fading and $\mathbf{a}(\theta) = [1, e^{-j2\pi\beta\sin\theta/\lambda}, \ldots, e^{-j2\pi(M-1)\beta\sin\theta/\lambda}]^T$ is the (M × 1) vector. Here, we have used β for the spacing between two adjacent antenna elements and λ for the wavelength of the RF carrier. The superscript $(\cdot)^T$ denotes the transpose. The noise $n(t)$ is assumed to be stationary, complex-valued additive white Gaussian with its variance σ_n^2. These statistical variables are commonly accepted for and used in design of wireless communication systems.

Three steps are involved for location estimation: i) estimation of angle-of-transmission (AOT), θ, ii) estimation of distance, d, and iii) calculation of geocentric position C(L_h, L_v) of the mobile node. AOT estimation is based on the least-squares (LS) criterion to extract the spatial information, i.e., θ, embedded in the received signal. With details of the derivation omitted, we can write an LS estimator for unknown parameter θ as

$$\hat{\theta} = \arg\min_{\theta} \sum_{l=1}^{P} \left| y(t_l) - \sqrt{\xi P_t}\hat{c}\mathbf{a}^T(\theta)\mathbf{w}s(t_l) \right|^2 \tag{2}$$

where P is the number of RF signal samples used for estimation and $\hat{c}$ is for estimates of the channel gain. Distance estimation is based on the radio signal strength and a log-distance (LD) path loss model and can be done with

$$\hat{d} = d_0 10^{[PL(d) - PL(d_0)]/10n} \tag{3}$$

where *n* is the path loss exponent, $PL(d)$ is the measured average path loss in dB at the transmitter-receiver separation *d* and includes measurement error which can be modeled as a zero-mean Gaussian random variable with variance σ^2, d_0 is the close-in reference distance close to the transmitter, and $PL(d_0)$ is the measured average path loss in dB at the corresponding transmitter-receiver separation d_0. The transmit signal power P_t and $PL(d_0)$ are pre-determined

and easily known to the mobile node via upper-layer communication protocols, and $PL(d)$ is obtained based on measurements of the received radio signal power, P_r. Finally, with d and θ estimated, the geocentric coordinates C(L_h, L_v) of the mobile node can be ultimately obtained from

$$\hat{L}_h = L_{F,h} + \hat{d}\sin\hat{\theta}\cdot 180^\circ / \pi R \quad (4)$$

$$\hat{L}_v = L_{F,v} + \hat{d}\cos\hat{\theta}\cdot 180^\circ / \pi R \quad (5)$$

where the units of the latitude and longitude are [degrees, minutes, seconds] with negative numbers representing south latitudes and longitudes west of the Greenwich meridian, and R is the radius of the geocentric coordinate system.

III. Mesh Networking of ARobots

A. Mesh Network for Communication with Mobile Robots

A wireless mesh network typically consists of a gateway and several mobile nodes that operate as a communication entity engaged in communication with other communication entity outside of the network but also as a router to forward information packets to other nodes within the mesh network[8]. In our test-bed, a mesh network is employed for an intended utilization of the positioning system such that location information can be possibly relayed when a mobile robot may be far away from the gateway (and thus, the control center/GUI). Without reinventing the wheel, we have adopted a Synapse Wireless solution for mesh networking capability that offers a minimum level of trouble-shooting to make the network up and running. The solution offers three networking nodes, i.e., Synapse Bridge, End Node, and Protoboard. However, the integration of the networking nodes into the complete indoor positioning system utilizing a GUI and mobile robots still requires a considerable amount of laboratory work that is the primary scope of this paper.

B. Implementation for Control of Mobile Robots in a Mesh Network

Our GUI is developed to facilitate manual control of ARobot's moving directions with a set of buttons. Whenever a button in the GUI is clicked as a control command, the GUI writes the corresponding command in a form of binary string to a pre-specified text file. The Synapse Portal periodically reads the content of the text file and delivers the content to the Bridge Node, which serves as the gateway of the mesh network and eventually delivers the command to an ARobot. The Portal Node of the Synapse Portal facilitates communication between the Synapse Portal and the Bridge Node. Although the Portal Node can communicate with the other Synapse nodes (i.e., End Nodes and Bridge Nodes), as it sits outside of the gateway of the mesh network, it is not considered part of the mesh network. Figure 2 shows a block diagram for delivery of control commands from the GUI to an End Node in the network. The Synapse Portal displays the status of the nodes in the network on the screen of the host computer, i.e., the control center. While only active Bridge Node and End Nodes are displayed, the Portal Node is always displayed on the Portal even if it is not actually connected. The Synapse Portal communicates with the Bridge Node using a Packet Serial protocol over a RS-232 or USB 2.0 cable connection between the control center and the Bridge Node. The Bridge Node, as well as End Nodes, has

two Universal Asynchronous Receiver-Transmitter (UART) ports, UART0 and UART1, and UART0 is dedicated for the communication with the Synapse Portal.

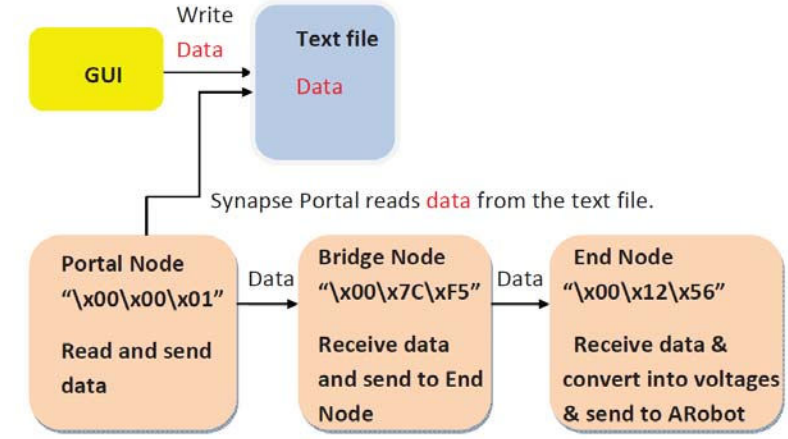

Figure 2. Delivery of control commands in mesh networking

With slightly different functionality of the Bridge Node and End Nodes for the test-bed, separate scripts were written in SNAPpy which is a subset of python for Synapse nodes [9, p. 9 & p. 36]. Key built-in functions, i.e., Application Programming Interface (API), for delivery of control commands are:

- `RPC()`
- `initUart()`
- `crossConnect()`

In contrast, the Portal Node is implemented based on a different set of built-in functions known as the Portal APIs[9, p. 89], and the key Portal API to deliver control commands is `sendData()`. Implementation for communication capability is done such that once powered on and initialized for mesh networking, the Bridge Node makes a remote procedure call (RPC)[13,14] to the Portal Node every second. The Portal Node then sends a short data packet (i.e., command) to the Bridge Node using `sendData()`. After this, the Bridge Node passes the control command to the End Node using another `RPC()`. The control command delivered to the End Node is finally converted to a binary signal (i.e., high and low) and is output to a designated pin in a serial fashion. The designated pin for the output from the End Node is directly connected to a pin on the microcontroller on the BASIC Stamp as the input to the board.

Every Synapse node contains an RF engine. Each RF engine has its own unique MAC address for differentiation from the other RF engines or other nodes in a wireless mesh network. Node addresses are the last three bytes of the MAC address that are read off the RF Engine sticker. For example, a node with MAC address `001C2C1E 86001B67` in hexadecimal format will have its node address `001B67`. In SNAPpy format, it is expressed as `\x00\x1B\x67` [9, p. 26], where 'x' indicates the hexadecimal format. The Portal Node carries a default node address `\x00\x00\x01`. Users can change this address anytime by using an appropriate Portal API.

B. Key Implementations and Lessons Learned

1) Pin functions and details: Table 1 shows the RF Engine pins used in our implementation[9]. Pins 5, 6, 8 and 9 for Data In/Out are the ones used for receiving/transmitting when UART is enabled; otherwise, they can be used as general unidirectional input and output pins. Pins for *Clear-To-Send* (CTS) and *Ready-To-Send* (RTS) are used for hardware handshaking. Hardware handshaking is also called the flow control. If the flow control is not enabled, they can be used as

bidirectional pins (i.e., input and output). The Synapse Portal uses its serial port(s) at a speed of 38.4 kbps. So users need to choose a baud rate of 38,400 for binary transmission.

Table 1. RF Engine Pin Assignments

Pin No.	Name	Direction	Description
5	GPIO3_RX_UART0	Input	UART0 Data In
6	GPIO4_TX_UART0	Output	UART0 Data Out
7	GPIO5_KBI4_CTS0	Bidirectional	GPIO, Keyboard In or UART0 CTS
8	GPIO6_KBI5_RTS0	Bidirectional	GPIO, Keyboard In or UART0 RTS
9	GPIO7_RX_UART1	Input	RS232UART1 Data In
10	GPIO8_TX_UART1	Output	RS232UART1 Data Out
11	GPIO9_KBI6_CTS1	Bidirectional	GPIO, Keyboard In or RS232UART1 CTS
12	GPIO10_KBI7_RTS1	Bidirectional	GPIO, Keyboard In or RS232UART1 RTS

2) Switchboard and connection matrix: The data flow through a SNAP device is configured via the Switchboard. It allows connections to be established between sources and sinks of data in the device. The following variables, also called the Data Sources/Sinks, are defined in the SNAP along with general-purpose input/output (GPIO) pins assigned on the RF Engine: `DS_NULL = 0, DS_UART0 = 1, DS_UART1 = 2, DS_TRANSPARENT = 3, DS_STDIO = 4, DS_CLI = 5,` and `DS_PACKET SERIAL = 6`. Note that these pin numbers for GPIO are not the original pin numbers of the RF Engine previously mentioned in Table 1. Table 2 shows a matrix of possible connection for the Switchboard[9]. The entries in the first column/row represent the data sources/sinks depending on the type of node devices. In our case, the Synapse Portal has data sources in the first column and each Synapse node has data sources listed in the first row. Each cell label describes the *mode* enabled by row-column cross-connection.

Table 2. Connection Matrix of the Switchboard

	UART0	UART1	Transparent
UART0	Loopback	n/a	n/a
UART1	Crossover	Loopback	n/a
Transparent	Wireless Serial	Wireless Serial	Loopback
Stdio	Local Terminal	Local Terminal	***Remote Terminal***
PacketSerial	**Local Gateway**	Local Gateway	Remote Gateway

There are two ways to set up data-forwarding paths in SNAP to connect data sources/sinks. They are `uniConnect(destination, source)` and `crossConnect(source1, source2)`. As it stands for, `uniConnect()` is for configuring one-way transmission from data source to destination. In this case, the destination cannot send data. In contrast, `crossConnect()` is for two-way communication in which each data source is able to send and receive data. In our implementation, `crossConnect()` is used.

Communication between data sources can be: a) Loopback, b) Crossover, c) Wireless Serial, d) Local Terminal, e) Local Gateway, f) Remote Terminal, and g) Remote Gateway. For instance, consider `crossConnect(DS_UART0, DS_UART0)`. This connection is a Loopback connection, and the transmitting node is sending data through its UART0 and the receiving node is getting the data through its UART0. The same Loopback can be also available when one connects nodes using `crossConnect(DS_TRANSPARENT, DS_TRANSPARENT)`.

3) Communication between Synapse Portal and Bridge Node: Figure 3 shows two communication links between the Synapse Portal and the Bridge Node 1) for control commands from the Portal to the Bridge Node and 2) for status data from the Bridge Node to the Portal to be displayed on the screen of the host computer. As mentioned earlier, for the purposes of wireless mesh networking, the Synapse Portal communicates with the Bridge Node using a Packet Serial protocol, i.e., `DS_PACKET_SERIAL`, and the Bridge Node uses its USB 2.0 (i.e., UART0) only for a connection to the Synapse Portal for the networking. This is a default configuration and its configuration command `crossConnect(DS_PACKET_SERIAL, DS_UART0)` would not appear in application-specific SNAPpy scripts for the Bridge Node. That means the UART0 of the Bridge Node cannot be used for other purposes such as exchanging non-networking data with the Portal; if UART0 is attempted for other purposes, the Portal will no longer be able to communicate with the Bridge Node and the SNAP `Communication Time Out` error will occur. This mode is shown in bold as *Local Gateway* in the last-row, second-column cell (PacketSerial-UART0) of Table 2.

For delivery of control commands from the Portal to the Bridge Node, an additional transparent mode needs to be set up at the Bridge Node using its UART1, which is a serial port for an RS-232 cable connection. Its configuration would be done by `initUart(1,38400)` and `crossConnect(DS_TRANSPARENT, DS_UART1)` in the SNAPpy scripts for the Bridge Node. This mode of communication is shown in Table 2 with underlined *Wireless Serial* in the fourth-row, third-column cell (Transparent-UART1). Once this is done, the Portal and Bridge Node are transparent and the Portal can insert data into the transparent-mode link to send control commands to the Bridge Node. Alternatively, to allow the Portal to get status data from the Bridge Node in transparent mode as well as sending control commands to the Bridge Node, and display the data on the screen, `crossConnect(DS_STDIO, DS_TRANSPARENT)` can be used along with `initUart(1,38400)` in the SNAPpy scripts for the Bridge Node. This mode of connection is shown in bold italic as Remote Terminal in the fifth-row, fourth-column cell (Stdio-Transparent) of Table 2. In Figure 3, the corresponding flows of data in transparent mode are shown in two dashed lines.

4) Implementation for End Node: Upon receiving the control commands from the Synapse Portal through the Bridge Node, the End Node converts them into digital signals that can be understood by the BASIC Stamp on an ARobot. The signaling for each command is done using a 4-bit binary signal with each bit lasting 75 ms. Logical bit 1 is represented by a high voltage, i.e., 2.8~3.3 [V], and Logical bit 0 by a low voltage, i.e., 0 [V]. The threshold between high and low voltages was 2.8 [V] at the BASIC Stamp.

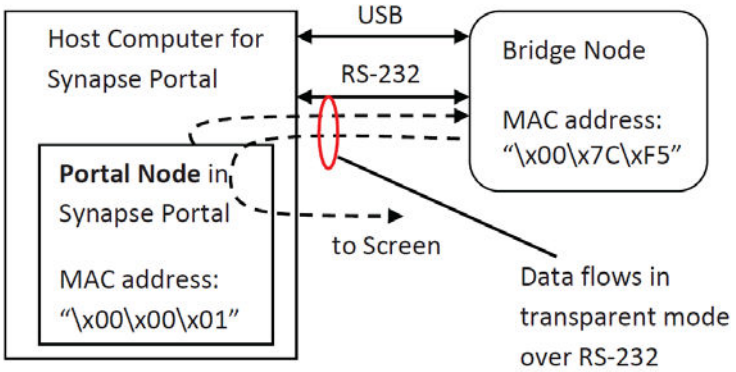

Figure 3. Communication between Portal and Bridge

5) SNAP Communication Timeout error: The SNAP `Communication Timeout` error occurs when the Synapse

Portal is unable to communicate with the Bridge Node. This can be due to the following reasons: a) some malicious script has been written into the Bridge Node; b) a wrong configuration of UART ports; and c) scripts are written such that a long delay occurs while making an RPC call. The SNAP `Communication Timeout` error may also be caused when changing the default Non-Volatile (NV) parameters. NV parameters are stored in a specific memory location and each parameter will have a specific ID. These are the Synapse configuration parameters and can be changed by two functions `loadNvParam(id)` and `saveNvParam(id, obj)`. After every change, one should reboot the node using `reboot()` function.

6) Restoring to default setting: When the Bridge Node loses communication with the Synapse Portal, a flag of SNAP `Communication Error` is raised, and there are three ways to recover from this: a) Erase SNAPpy image; b) set it back to the factory default NV parameters; and c) Upgrade Firmware. In general, one should use the Erase SNAPpy image option. The Default NV parameters option can be used when the SNAP `Communication Timeout` error is caused due to a change of the default NV parameter settings in a script. The Upgrade Firmware option can be used when the problem persists. During our implementation, it was found that the RF Engine may lose its firmware if a wrong script is downloaded onto it. It should be noted that while performing these options, one will have to use the serial port. In this case, the Synapse Portal will take care of configuration and there is no need for the user to manually set up UART for this purpose.

7) Unsupported Opcode error: One of the significant and possible reasons for this error `SNAPpy Image Manager Error: Unsupported Opcode: LOAD_ATTR at Line: line no.` is testing the Portal Node script with the Test SNAPpy script option in the Synapse Portal.

IV. Mobility Control of ARobots

The ARobot is a three-wheel mobile robot controlled by a microprocessor board called BASIC Stamp. It has two Whiskers as object sensors, as well as other conventional components such as LEDs, a buzzer, and control buttons. ARobot's motion is driven by Drive Motor, Encoder Wheel and Sensor, and Steering Motor. The drive motor moves the ARobot forward and backward using the H-Bridge driver circuit. The H-Bridge is controlled by the coprocessor receiving commands from the BASIC Stamp. The encoder wheel and encoder sensor are used to measure distance in relation to the rotation of the drive wheel. With 20 encoder slots per revolution, when the encoder wheel is rotated along with the drive wheel, the slots are detected by the encoder sensor and counted for distance calculation. As the drive wheel size is 3.25 inches in diameter, each slot represents approximately 1/2 inches of ARobot's traveling, i.e., $3.25\pi/20$ inches. The counts can be read from the coprocessor by the BASIC Stamp when necessary to determine travel distance. The steering motor is a remote control (RC)-style servo motor. The coprocessor can control four RC servo motors and #1 is used for steering. Using the servo motor, one can rotate the wheels in several directions. The values of different directions for our test-bed are shown in Table 3.

The BASIC Stamp includes a BASIC interpreter chip, internal memory, a 5-volt regulator, a number of general-purpose I/O pins, and a set of built-in commands for math and I/O pin operations. The pin voltages are in the range of 0-5 [V]. The BASIC Stamp is programmed with

Table 3. Code Values for Moving Directions

Direction (in degrees)	Value for Position Variable
Right by 60	'01'
Right by 30	'40'
Center	'80'
Left by 30	'C0'
Left by 60	'FF'

Parallax Beginner's All-purpose Symbolic Instruction Code (PBASIC), which is a microcontroller-based version of BASIC[12]. To load a PBASIC program, the BASIC Stamp needs to be connected to a computer hosting its graphical user interface through a serial port. When the BASIC Stamp needs to control a motor, it simply sends a command with necessary parameters such as speed, direction, and/or distance to the coprocessor for further handling of the necessary tasks. For this, two BASIC Stamp commands are used: SERIN and SEROUT[12]. Then, the microprocessor gets acknowledgement from the motor and continues for the rest of the main program. For the case of the servo motor, the motor aligns the wheel before sending acknowledgement. Further details and technical specifications of ARobot can be found in the Assembly and User Guide[11].

A. Communication between End Node and BASIC Stamp

All mesh-networking nodes employ a high-speed microcontroller with a code-execution speed of 11,400 instructions per second, and all mobile robots' mobility is controlled by a low-speed microcontroller with an execution speed of 4,000 instructions per second. In our implementation, data are coming from the Synapse End Node. Since the End Node runs on a microcontroller much faster than the BASIC Stamp (2/2e) microcontroller, it could become an issue if the signaling is not done properly when making them communicate with each other. To overcome this speed mismatch, communication between the two microcontrollers is implemented in a form of asynchronous serial communication (that is, these two microcontrollers are not time-synchronized prior to communication. Rather, each of them runs on its own timing clock.). To facilitate asynchronous serial communication, we have adopted a short frame format with one Sync bit and one Start bit preceding a four-bit binary sequence that represents a specific control command from the GUI. The four-bit representation for a control command is chosen as our design would have at most 16 different commands to control ARobot's motion. Figure 4 shows a bit pattern of the frame for the asynchronous serial communication.

The Sync bit is to achieve frame synchronization between the End Node and BASIC Stamp. When the Sync bit is recognized by the BASIC Stamp, the frame is considered synchronized. Then, the Start bit indicates that, after its predefined duration of 150 msec, data bits start. The combined pattern of the Sync and Start bits are different from any possible pattern of data bits for a control command or a sequence of consecutive control commands, the frame preamble can clearly identify the beginning of data bits. Illustrative PBASIC codes for the asynchronous serial communication are shown in Table 4. Note that 't' is declared as Nibble, and the input is directly assigned to output

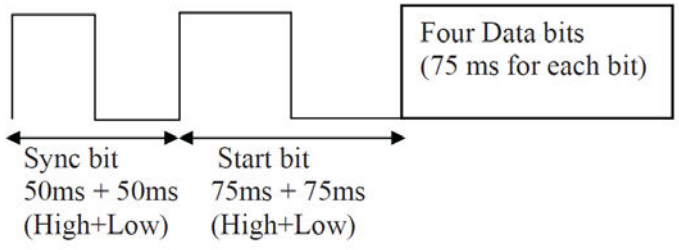

Figure 4. Frame structure for asynchronous serial communication

Table 4. PBASIC Codes for Asynchronous Serial Communication

```
' Async Communication between PROTOBOARD & BASIC Stamp
' {$STAMP BS2e}
' {$PBASIC 2.5}

t VAR Nib
i VAR Nib

INPUT 2              'PIN 2 of BASIC Stamp is configured as INPUT
OUTPUT 3             'PIN 3 of BASIC Stamp is configured as OUTPUT

tloop:

IF IN2=1 THEN        'Check for a SYNCH pulse
   PAUSE 100         'Time period of a SYNCH pulse (ON+OFF)

   IF IN2=1 THEN     'Check for a START Bit
      PAUSE 150      'Time period of the START bit (ON+OFF)

      FOR i= 1 TO 4  'Start of a Command (i.e., data bits)
           OUT3=IN2
           t=t << 1
           t=t+IN2
           PAUSE 75  'Time period of each data bit (75ms)
      NEXT
   ELSE              'If a START bit is not recognized,
      GOTO tloop     'go to 'tloop' to check again for a SYNCH pulse
   ENDIF
ELSE                 'If a SYNCH pulse is not recognized,
   GOTO tloop        'go to 'tloop' to check again for a SYNCH pulse
ENDIF
```

using the assignment operator. The codes in Table 4 will output the command bits at the output pin, i.e., Pin 3, and the bit pattern can be clearly observed on an Oscilloscope.

B. Flowchart for ARobot's Motion

The overall operation of an ARobot can be concisely described with a flowchart shown in Figure 5. With the power on, when user presses the START button, which is connected to Pin 14, the ARobot starts moving and then the program goes to the Event loop. In the Event loop, all possible 11 cases of commands, which are sent by user from the GUI, are implemented. Also, the Event loop contains two built-in events for left and right whiskers as events e1 and e2; this way, those whiskers are available and functioning while an ARobot follows user commands. After checking all events in sequence, the program goes back to the start of the Event 'e' which keeps the ARobot executing the current control commands. A subroutine for the Stop command "e3" is shown in Table 5 for illustration. Other subroutines e4 through e13 have a similar structure with a different subroutine(s).

C. Observations and Remarks

The BASIC Stamp 2/2e is a slow processor and works with a low level language such as PBASIC. As such, attention to timing issues of PBASIC Stamp is very critical to make the test-bed function properly. The microcontroller used in BASIC Stamp is Ubicom Sx28AC featuring a processor speed of 20 MHz and program execution speed of 4,000 instructions/second or 250 µseconds per instruction (minimum time). In Table 6, we present key measurement data relevant to the timing to execute PBASIC codes. Note that all numbers in the table are determined from several measurements in the lab. For instance, the execution time consumed by the processor for the command `HIGH` being 150 µsec means that whenever there is any command like `HIGH 10` which is for the processor to set Pin 10 to logic 1, 150 µsec is required. A similar amount of time is consumed for `LOW 10`, setting the pin specified to logic 0. For an `IF` condition, the processor takes 540 µsec to check for its condition. This time measurement was made with one simple condition for `IF`, i.e., `IF IN2=1, THEN 'null, ENDIF`. The time consumed by the microprocessor depends on the number of conditions for the `IF` statement and also the types of the conditions. For `GOTO`, the processor takes 300 µsec to jump to a specified routine.

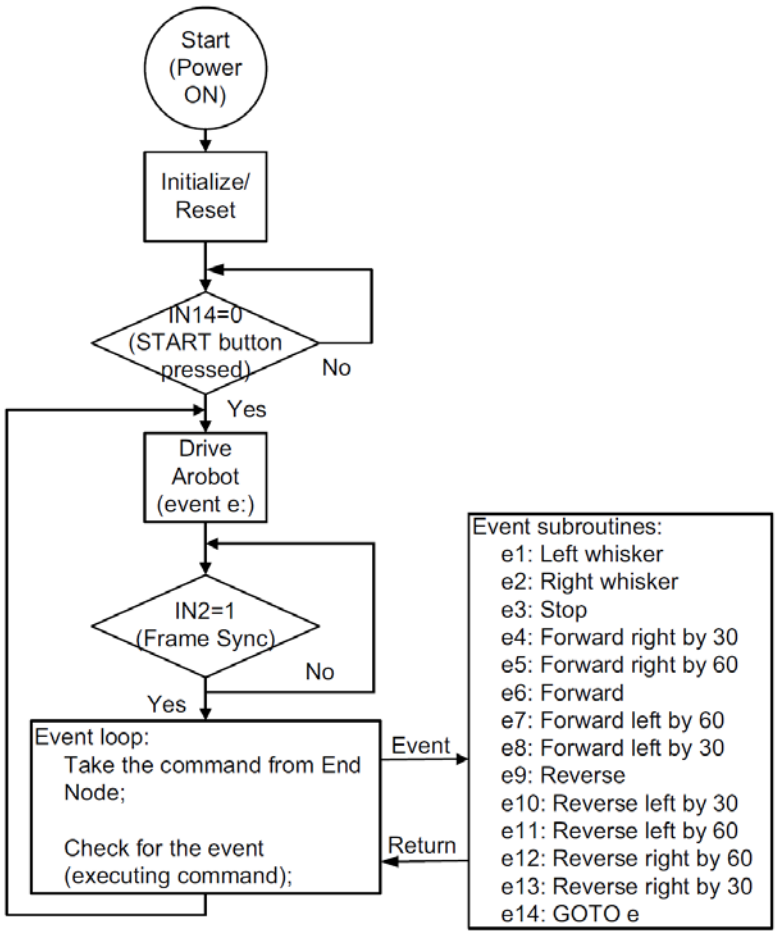

Figure 5. Flowchart for control of ARobot's motion

Figure 6 shows the completed test-bed components: two ARobots, Bridge Node, and the Control Center with the Synapse Portal. Although not explicitly shown, the GUI is also implemented on the host computer for the Synapse Portal. In our implementation, the minimum time required to execute the `Eventloop` (i.e., taking the command from the End Node to the final event e14) was 570 msec. This measurement was obtained from the case where all the conditions in all events were `FALSE`. If any condition is `TRUE`, then the time consumed to execute `Eventloop`

Table 5. PBASIC Code for Event 3: Stop

```
e3:
'Event for Stop command
IF t=%0000 THEN
   GOSUB stoprbt
   t=0
ELSE
   GOTO e4
ENDIF
GOTO tloop
```

Table 6. Execution Time of Key Instructions

Instruction or Command	Time consumed by the processor [μsec]
HIGH	150
LOW	150
IF ... ENDIF	540
GOTO	300

will vary according to the command received from the End Node. This minimum time was the basis of determining how quickly the Synapse Portal could go back to the text file to check for a new command. In our test-bed, as mentioned in Section III, the Synapse Portal checks for a new command and delivers it every one second as the Bridge Node issues an RPC call to the Portal every second.

V. Discussions on Relevant ABET Criteria and Laboratory Experiments

From the discussions above and along the course of our development activities for the test-bed, we noted that a successful implementation of our experimental indoor positioning system involving GUI, Synapse Portal, Synapse Bridge, Synapse End Nodes, and Basic Stamp for ARobot requires a great deal of attention to details. It effectively creates a framework of an end-to-end communication system with real-time operations and processing of logical and actual digital signals with visual observations to make as a measure for successful design. For an undergraduate course in wireless communications, we suggest that the following subject topics be considered for laboratory experiments:

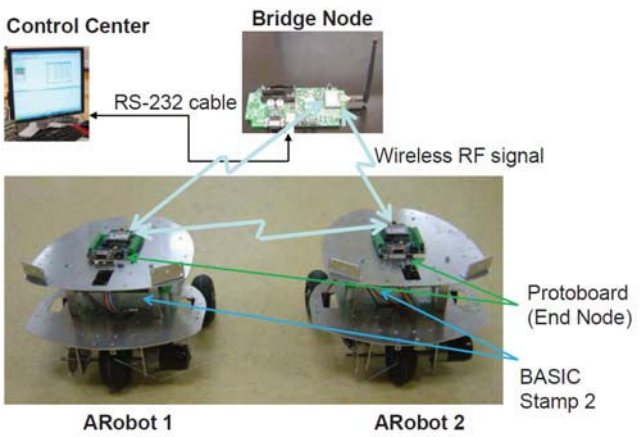

Figure 6. Photos of test-bed components

- Understanding of protocols required for data communication
- IEEE 802.15.4-based signal processing and measurement of its RF signals
- Understanding of upper layer protocols, e.g., SNAP, for data communication and its realization in a form of APIs.
- Understanding of the concepts of mesh networking, gateway, and network nodes
- Implementation of mesh-network nodes on microcontrollers (Synapse Bridge and End Nodes)
- Asynchronous communication and its application to communication between electronic devices built on different types of microcontrollers
- Conversion of logical bits to actual digital signals and its application to control mobile robots

These laboratory experiments along with a practical test-bed system are believed to greatly help students in an electrical and computer engineering program attain the following ABET outcomes:

- an ability to design and conduct experiments, as well as to analyze and interpret data

- an ability to design a system, component, or process to meet desired needs within realistic constraints
- an ability to identify, formulate, and solve engineering problems
- a knowledge of contemporary issues
- an ability to use the techniques, skills, and modern engineering tools necessary for engineering practice

VI. Concluding Remarks

We have presented our experience in designing networked mobile robots each of which involves two different types of microcontrollers for mesh networking and motion control. Creating a simple protocol for low-rate data communication and overcoming several issues arising from mismatched code-execution speeds of the two microprocessors, we have successfully integrated a pair of Synapse End Node and BASIC Stamp in cascade for each ARobot. With a Synapse End Node on the ARobot, we have established networking capability among ARobots and the Bridge Node based on the SNAP. With an in-house GUI for generation of control commands, we have successfully executed these commands on ARobots and controlled their motion as desired. The inevitable inter-command delay was reasonable for the purpose of controlling ARobots in indoor environment. Finally, we have suggested a set of laboratory experiments for undergraduate students in electrical and computer engineering to help them better prepared for their early-stage professional career measured by the ABET criteria.

Bibliography

1. Ali, L. A. Latiff, and N. Fisal, "GPS-free indoor location tracking in mobile ad hoc network using RSSI," in Proc. RF and Microwave Conf., Oct. 5-6, 2004, pp. 251-255.
2. C.-L. Chen and K.-T. Feng, "Enhanced location estimation with the virtual base stations in wireless location systems," in Proc. Vehicular Technol. Conf. - Spring, May 7-10, 2006, vol. 2, pp. 603-607.
3. W. Ni, G. Shen, X. Leng, and L. Gui, "An indoor location algorithm based on Taylor series expansion and maximum likelihood estimation," in Proc. Int'l Sympo. on Personal, Indoor and Mobile Radio Commun., Sept. 2006, pp. 1-4.
4. El Moutia and K. Makki, "Time and power based positioning scheme for indoor location aware services," in Proc. IEEE Consumer Communications and Networking Conf., Jan. 10-12, 2008, pp. 868-872.
5. W. Lee, "Indoor location estimation at mobile nodes based on angle-of-transmission," in Proc. Wireless and Microwave Conf., Apr. 20-21, 2009, Clearwater, FL, pp. 1-5.
6. D. Roddy, *Satellite Communications,* 4th Ed., McGraw-Hill, 2006, pp. 54-59.
7. IEEE Standard 802.15.4, Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs), Sept. 2006.
8. J. Jun and M. L. Sichitiu, "The nominal capacity of wireless mesh networks," IEEE Wireless Communications, pp. 8-14, October 2003.
9. Synapse, Synapse Network Appliance Protocol (SNAP) Reference Manual for Version 2.1, v1.1, Document Number 600-0007B, 2008.
10. Synapse, RF Engine & Evaluation Kit: Hardware Technical Manual, v2.0, Document Number 600-101.01A, 2007.
11. Arrick Robotics, ARobot - Mobile Robot: Assembly and User Guide, Rev. D, Nov. 2005, available at http://www.arrickrobotics.com/arobot/guide.pdf.

12. J. Martin, J. Williams, K. Gracey, A. Alvarez, and S. Lindsay, BASIC Stamp Syntax and Reference Manual, ver. 2.1, Parallax, Inc., 2005, available at http://www.parallax.com.
13. D. Marshall, “Remote procedure calls (RPC),” Jan. 1999, available at http://www.cs.cf.ac.uk/Dave/C/node33.html.
14. R. Thurlow, “RPC: Remote procedure call protocol specification - Version 2,” Internet Engineering Task Force (IETF), Draft Standard RFC 5531, May 2009.