Indoor Location Using SENSOR Networks

Timothy Porter
Jeremy Witmer

Dr. Chow, Advising
CS 522 – Network Communications
University of Colorado, Colorado Springs
Fall 2006

Abstract

New developments in wireless technologies has allowed for research into the possibility for real-time indoor location using wireless SENSOR networks. Indoor real-time location has drawn very strong interest from the emergency responder community, to improve incident management, personnel response to rapidly changing environmental conditions, and personnel safety. It has also drawn interest in the field of building security.

This project is interested in taking wireless SENSOR hardware and software and investigating the feasibility of tracking a moving sensor within a two dimensional network of fixed sensors. Ideally, the position of the moving mote will be determined based on the signal strength of the transmissions between the sensors, within a reasonable margin of error. After design and testing of the network, the location software should be able to determine the location of a moving sensor within a reasonable margin of error, under 5%. After testing the network and software, it was found that, in two dimensions, with no signal attenuation, the position of the moving sensor could be found with no more than 7% error.

Table of Contents

Table of Figures

Introduction

Emergency responders consistently look for ways to improve responder safety. To this end, many recent technologies have arisen, including vehicle tracking and traffic signal preemption, improved communications, and improved equipment. One area that has not seen concurrent development with other first responder technology is that of real-time personnel tracking within buildings and complexes during emergency events. This tracking would allow incident commanders the ability to closely direct personnel for improved response to rapidly changing environment conditions, and for improved safety. Real time personnel tracking would also be useful in the area of building security, for high security areas. Giving security personnel the ability to track visitors or other persons within the building or complex would improve the general security. Wireless technology and computer miniaturization have finally progressed to the point that this tracking is now feasible.

This project is meant to explore basic location tracking using wireless SENSOR network technology from Crossbow Systems. Ideally, the position of a single moving sensor will be tracked within a two dimensional plane formed by fixed sensors at the corners. Using a Trilateration algorithm, the position of the moving sensor will be determined from the known distances between the fixed motes, and the radio transmission signal strengths between the sensors in the network.

This project is meant to be a proof of concept. The signal strength exchange between sensors has been explored, as has sensor location based on radio signals in simulation, but both elements have not been combined to date within a network at UCCS. This project is intended to show that positioning with the hardware is actually possible, within a reasonable margin of error.

Wireless SENSOR Networks

The swift advancement of wireless technology has allowed for the development of new technologies utilizing wireless communication. A wireless SENSOR network is one that consists of numerous low-power microprocessor nodes, distributed across an area. Each node is equipped with a radio transmitter, and the nodes dynamically form mesh networks, allowing communication between the nodes, and back to a base node, allowing data to be fed through the base node, processed, and collected.

Each of the separate nodes (normally referred to as motes) can be configured with multiple different sensors, including temperature, sound, vibration, GPS, and many others. These sensors can be combined with the motes in various ways. The motes themselves can be specifically programmed to perform different operations.

Each mote runs on a battery, which means that the most critical element for mote longevity is power usage. Accordingly, the motes are optimized for low power usage, and can be programmed to spend most of their operational time in a sleep state, drawing only micro amps. The most power-hungry operation for the motes is data transmission using the radio.

The motes, as stated previously, form dynamic networks. Each mote has a unique ID, called the mote ID, and a group ID. The mote ID allows the motes to send messages to specific other motes in the network. The group ID allows for the use of multiple different networks of motes in the same area. When a mote receives a message, it checks the group ID against its own, and if they don't match, the message is ignored. For all of the motes with a given group ID, there is also a broadcast address, which allows a message to be sent to all the motes in a group.

Hardware

The motes used for this project were designed by Crossbow Systems. The motes are built on an Atmel ATmega 128L microprocessor, with a 433 MHz band radio. The radio has a maximum theoretical range of 1000 feet, but in practice, free space signal loss and attenuation due to buildings and equipment limit them to a shorter distance. Further information can be seen at http://www.xbow.com/Products/productsdetails.aspx?sid=3.

Two types of motes were used in this project, the Mica2 and the Mica2Dot. Both motes use the same processor. The Mica2dots simply have a smaller form factor, and run on a CR series lithium battery, providing three volts. The Mica2 series motes use 2 AA batteries to provide the necessary three volts.

For the purposes of this project, and the SENSOR network designs in general, the two types of motes are interchangeable. The different motes run the same software, so the behavior is consistent across the network.

Figure 1: Crossbow Systems Mica2 (left) and Mica2Dot (right) motes (From [1])

TinyOS

The motes from Crossbow Systems run the TinyOS operating system. TinyOS is an event based operating environment designed for use with embedded networked sensors. More specifically, it is designed to support the concurrency intensive operations required by networked sensors with minimal hardware requirements [2]. TinyOS uses a version of C, called nesC. nesC is a structured, component based language. It wraps hardware and software functionality into what are referred to as components. Each component

provides and uses a series of well-defined interfaces. Because component access is limited only to the interfaces provided, the modules are very loosely coupled, allowing for extensive flexibility. The component/interface system also means that the underlying component code can modified without changing the interface. Finally, because the various hardware on the mote itself are wrapped in components, the programmer can avoid having to do any low-level byte code programming, which allows the development process to go much more quickly.

Hardware Configuration

The location network for this project was designed to test a two dimensional version of the mote network (see Fig. 2). The motes are laid out in a roughly square configuration, with the base mote at the upper left. The base mote is connected to the laptop with an RS-232 serial cable. The laptop runs the location software and displays the location of the mobile mote as it moves within the network. The fixed motes are placed as shown in the diagram, at distances of up to 10 feet, for the test network. Dimensions within the network were measured in inches for testing, but the units are arbitrary.

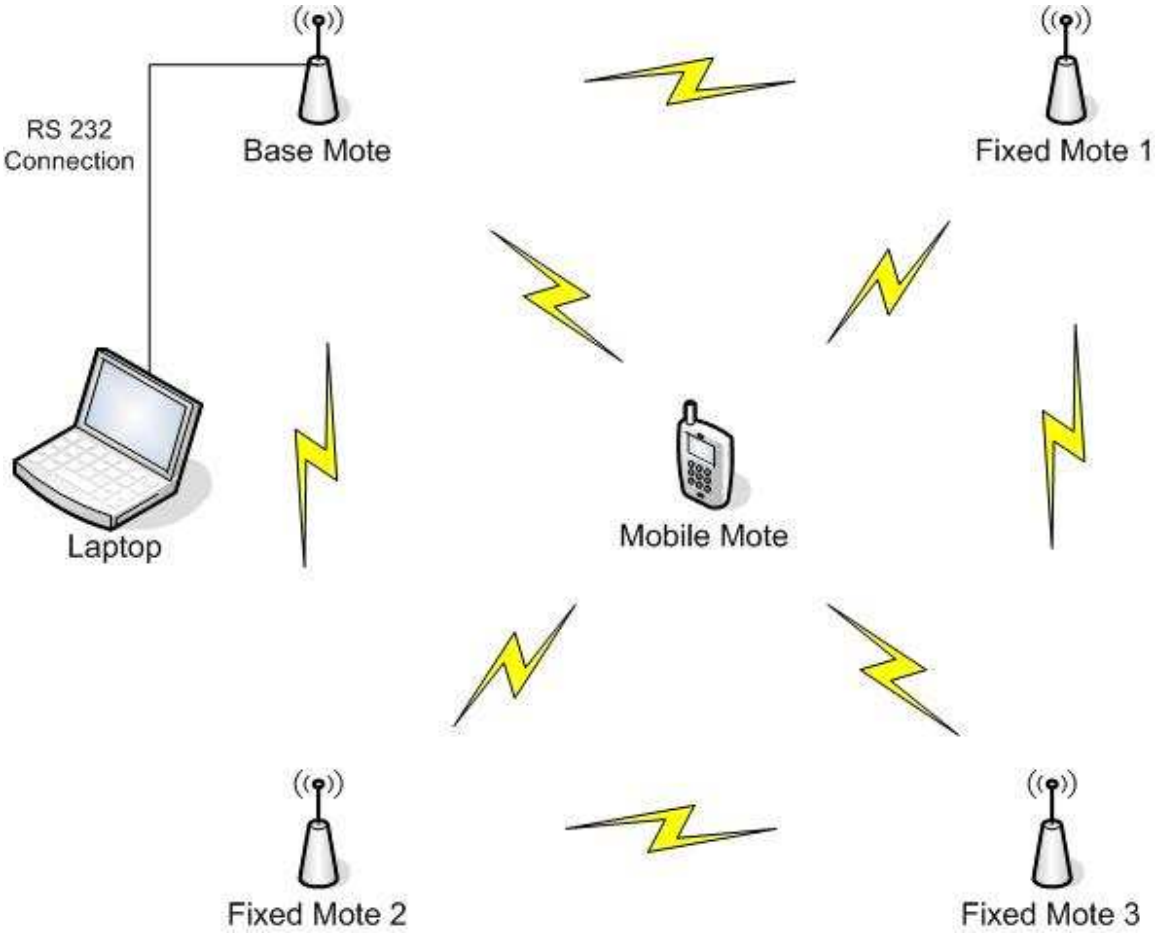

Figure 2: Test mote network configuration

Software

Mote Software

This project uses two different software versions for the motes. The remote (fixed and mobile) motes in the network use one version of the software, and the base mote, which passes data to the laptop, uses a second version of the software.

Both versions of the software rely heavily on the TinyOS ActiveMessage TOS_Msg structure, which is the OS defined structure that the motes use to send messages. The active message structure is defined as:

```
typedef struct TOS_Msg
{
  /* The following fields are transmitted/received on the
radio. */
  uint16_t addr;
  uint8_t type;
  uint8_t group;
  uint8_t length;
  int8_t data[TOSH_DATA_LENGTH];
  uint16_t crc;

  /* The following fields are not actually transmitted or
received
   * on the radio! They are used for internal accounting
only.
   * The reason they are in this structure is that the AM
interface
   * requires them to be part of the TOS_Msg that is passed
to
   * send/receive operations.
   */
  uint16_t strength;
  uint8_t ack;
  uint16_t time;
  uint8_t sendSecurityMode;
  uint8_t receiveSecurityMode;
} TOS_Msg;
```

The addr and group fields define the message receivers. The length, data, and crc fields are the actual message body. Of the non-transmission fields, the one most critical to the application is the strength field. When a message is received by a mote, this message structure is passed up the radio stack to the receive message handler. In that structure, the strength field indicates the transmission strength from the sender. This strength value is what the location software uses to calculate the location of the mobile mote. The strength field is a two byte field that ranges from 0 to 65535 for the signal strength, with 65535

being no connection, and 0 being the best possible signal strength. 65535 is the theoretical lower limit on two connected motes, but in operation, the motes will lose their connection well before a signal strength of 65535 is reached.

Each message data field can hold up to 29 bytes, which is the default TinyOS value. This can be changed, but the default value gives the best balance between size of message and throughput speed. Because of the way that the location software works, most of the message traffic is broadcast traffic, which means that it goes to all other motes with the same group ID, so throughput is important to reducing overall transmission lag in the network.

Both of the software versions use the following two message structures (BUFFER_SIZE is defined to be 13):

```
struct StrengthMsg
{
     uint16_t sourceMoteID;
};

struct SignalMsg
{
     uint16_t sourceMoteID;
     uint16_t sigdata[BUFFER_SIZE];
};
```

The StrengthMsg structure is used by both versions of the software as a way to send its own transmission strength values to all of the other motes in the area. The sourceMoteID field is set to the motes own ID, which allows the other motes to store the transmission signal value by mote ID. This message packet is sent approximately every half second.

The SignalMsg structure is used by the remote motes to send that mote's received values to all the other motes in the network. For instance, the network consists of motes 0-4, where 0 is the base mote, and 4 is the mobile mote. Take mote 2. Each time mote 2 receives a StrengthMsg packet from one of the other motes, it puts the strength value from the TOS_Msg structure in sigdata[transmittingMoteID]. This way, each mote's ID serves as the index into all the signal strength arrays that all of the motes send back to the base mote. The strength value for any mote is at the same position in every sigdata array. This message packet is sent every 3-5 seconds.

Remote Mote Software

The software on the remote motes has only two functions. The first is to, every 500 milliseconds or so, to send a StrengthMsg packet to all the other motes in the network. This gives all the other motes in the network a signal strength value for this mote.

The second function, every 3-5 seconds, each mote sends a SignalMsg packet to the base mote. This packet gives the laptop the signal strength data from every other mote in the

network to this mote. Getting all the SignalMsg packets from the motes in the network allows the location software on the laptop to calculate the position of the mobile mote as it moves within the network.

Base Mote Software

Unlike the remote motes, the base mote is connected to the display laptop over an RS-232 serial connection. It has the ability to transmit data to the laptop.

The software on the base mote transmits the same StrengthMsg packet as the remote motes, every 500 milliseconds, since the base mote participates as a fixed mote in the positioning network.

The second function of the base mote is to send its own SignalMsg packet down the serial line every 3 seconds, to provide its own signal strength data to the location software on the laptop.

The final function of the base mote software is to pass the SignalMsg packets from the remote motes to the location software on the laptop. Each time a packet is received, the base mote immediately passes it on to the laptop over the RS-232 line.

Location Algorithm

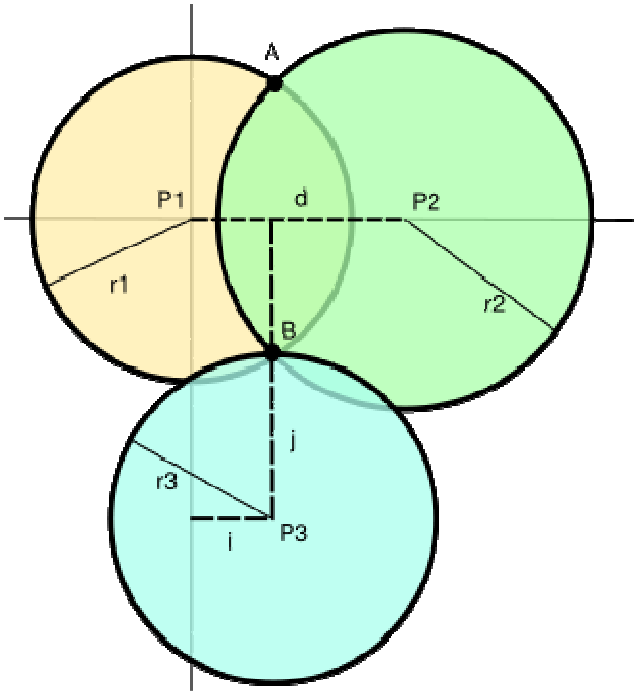

Figure 3: Trilateration in 2 dimensions (From [3])

The position of the mote within the sensor network is found using a Trilateration algorithm. Trilateration is a method of determining the relative positions of objects using the geometry of triangles in a similar fashion as triangulation [3]. Unlike triangulation,

which uses one known distance and the angles between points and the objective point, Trilateration uses two or more known points, and the distances between those known points and the objective point. To find a point in a two dimensional plane, three known positions are required. To find a point in a three dimensional plane, four known points are required. In three dimensions, replace “circle” with “sphere”, and all the geometry remains the same.

Taking Fig. 3 as a demonstration, P1 is the base mote, and P2 and P3 are fixed motes. The mobile mote is at point B. Measuring the distance to P1, r1, gives the location of the mobile mote as anywhere on the circle C1. With the addition of the measurement to P2, r2, the location of the mote is narrowed to the intersection points of circles C1 and C2, or points A and B. With the addition of the measurement r3, there is only one intersection point on all three circles, point B.

This diagram uses three circles to find the position of point B. The assumption here is that there is no error in the distance measurements. Because there actually will be error in the measurements, the software uses another fixed point, P4, and draws a fourth circle, and takes the average of the intersection points, to find the best-fit point for the location of the mobile mote.

The find the distances between the known points and the objective point (the mobile mote), the signal strength values are used. When the network is first set up, the distances between P1, P2, P3, and P4 are entered by the user. The motes at these points (the fixed motes and the base mote) then exchange messages to get the signal strength values. For instance, the base mote at P1 and the fixed mote at P2 are 36 inches apart. P1 receives a signal strength of 18944. This value is divided by 36 inches to find a ratio, 526. That ratio is then averaged with the ratios found using the signal/distance to P3, and P4, to get an overall signal/distance ratio. Using the P1-P3 ratio = 528 and the P1-P4 ratio = 521, the overall ratio is 525. Values are rounded to the nearest integer. At this point, the base mote at P1 has enough information calculate the distance to the objective point, the mobile mote, given a signal strength from that mote. If the signal strength from the mobile mote is received as 16877, the corresponding distance is 16877/525, or 32 inches, from P1 to the mobile mote.

This same operation is performed for the motes at P2, P3, and P4. Each of these motes calculates its own signal/distance ratio that it uses to find the distance to the mobile mote. After all these calculations are complete, the software then knows r1, r2, r3, and r4, corresponding to the above diagram, and can calculate

To account for errors, and to determine sequentially better ratio values for the four known points, the base mote and the fixed motes are constantly exchanging strength values, so the ratios are constantly being recalculated.

Location Software

The location software is written in Java, and displays the positions of the fixed mote, and the mobile mote, as it moves within the network. The positions of the base mote and

fixed motes are shown in blue, and the position of the mobile mote is shown in red. The circles indicate the areas being used for the Trilateration algorithm by the software. The location software accesses the data from the base mote over the network, served from the Forwarder software.

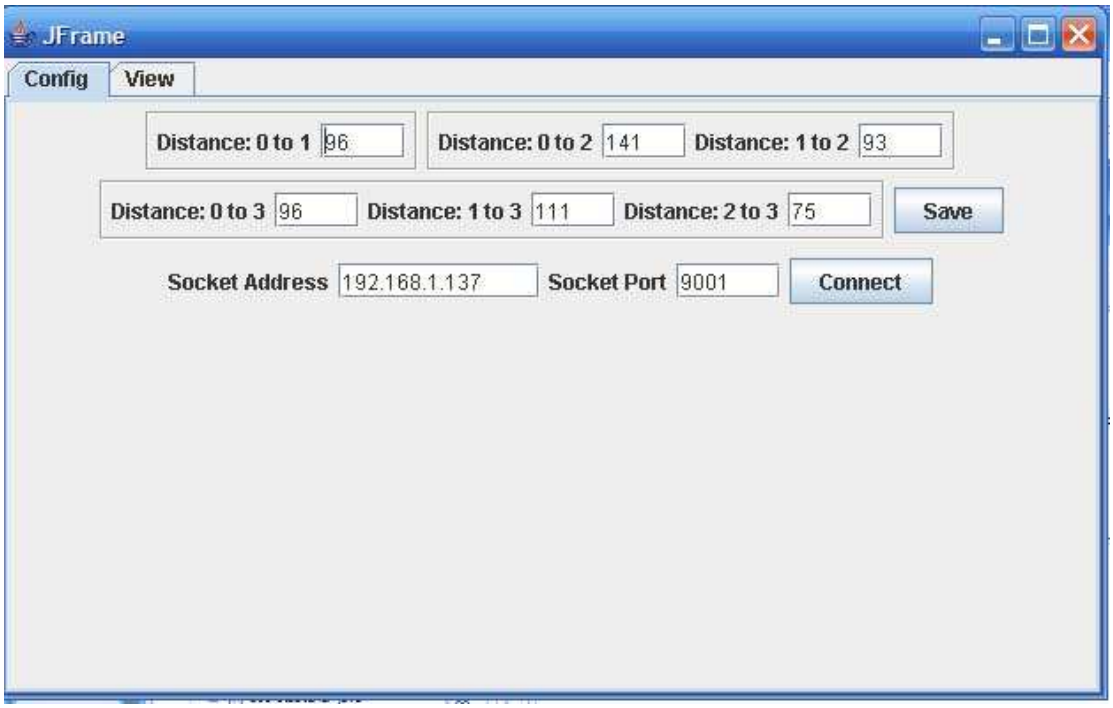

Figure 4: The configuration tab in the Location software

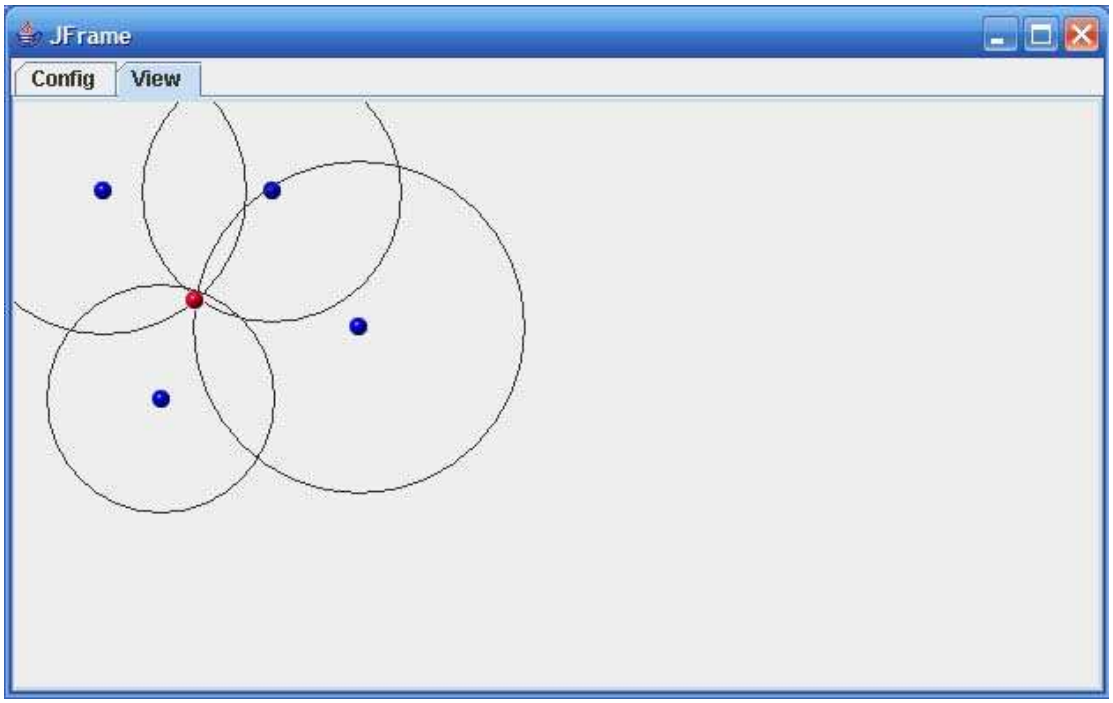

Figure 5: View tab in the Location software

A second piece of software, Forwarder.exe is used to receive the data packets from the base mote over the serial connection, check them for integrity, and make the packets available over the network. This software simply acts as a terminal server.

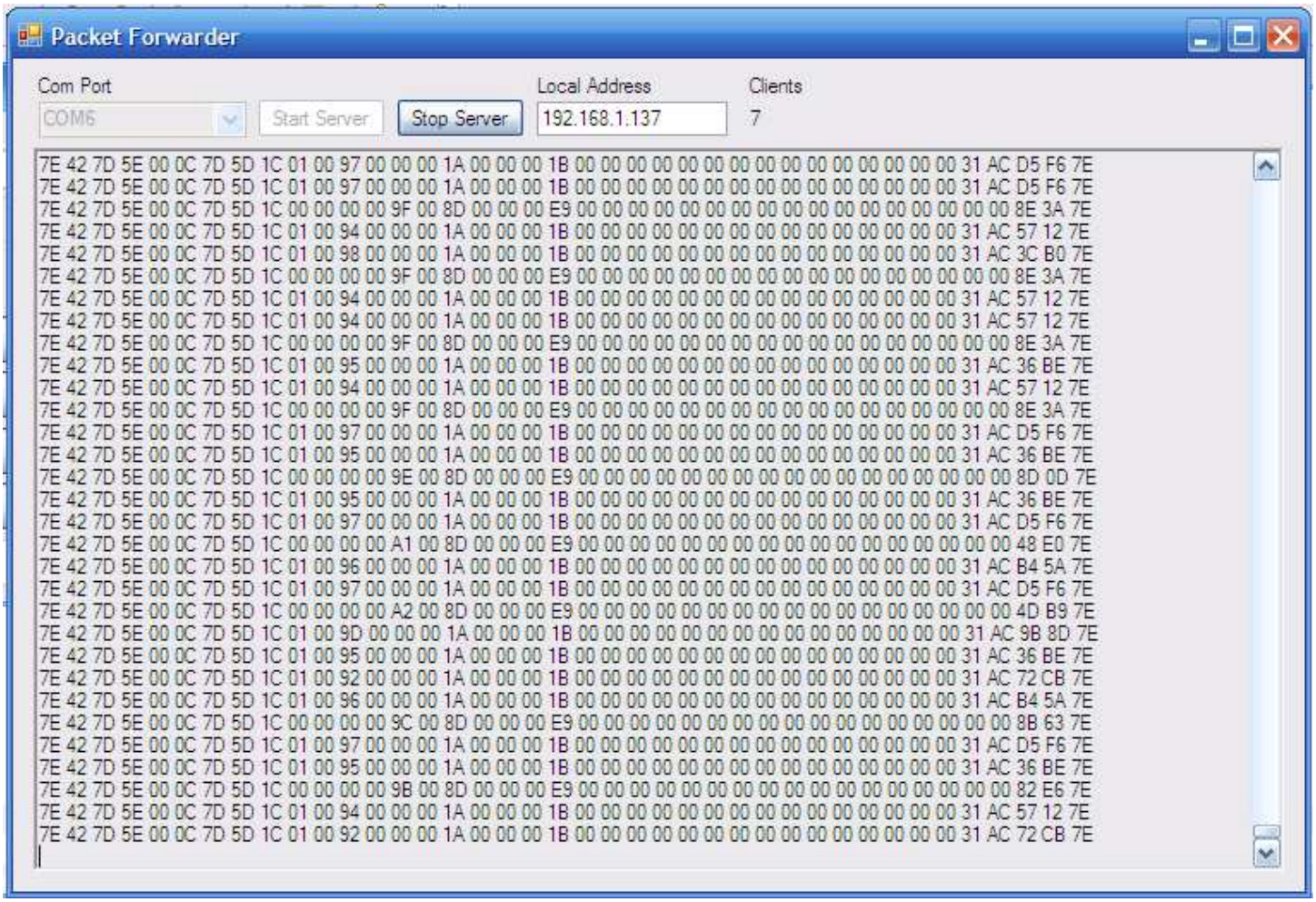

Figure 6: Forwarder.exe software in operation

Results

To test the software, a few different networks were set up, with edge lengths between 70 and 120 inches. In each network, the mobile mote was moved, the actual position measured, and that measurement was compared to the calculated measurement displayed by the software. In all cases, the actual position was within 5 inches of the calculated position, an error between 4% and 7%, which is quite acceptable. The error is calculated to be:

$$\frac{[(\text{actual } x - \text{calculated } x)^2 + (\text{actual } y - \text{calculated } y)^2]}{[(X \text{ side length} + Y \text{ side length})/2]}$$

This numerator is the Euclidean distance between the actual and calculated points. The Euclidean distance is divided by the average side length to get a percentage error. However, during the testing, a few problems arose with the network and the software.

The first problem was a concurrency issue with the mote data transmissions. Due to the design of the mote software, the SignalMsg transmissions from the remote motes are based on a timer. If two or more motes transmit their SignalMsg values to the base mote at the same time, the first transmission to reach the base mote is received, and the others are lost. This is due to the small amount of time that is required by the base mote to take the data from the radio message, and transfer it over the serial line to the laptop. This problem could be alleviated or eliminated by buffering the packets on the base mote, or by querying the remote motes for their SignalMsg data, a pull operational model, instead of the current "push" operational model, where the remote motes send their data based on a timer.

The second problem that arose was with the implementation of the Trilateration algorithm. Due to the fact that the radio signals are somewhat prone to interference, there was error in the ratio values used to calculate the circles, so the intersection points do not match up precisely. As a result, the location software must extrapolate the most correct point of intersection as the position of the mobile mote. This extrapolation needs more testing and refinement.

Finally, in the current implementation of the mote software, the motes are almost constantly transmitting, which is a significant drain on the batteries. The Mica2 motes will operate for approximately 24 hours on one set of batteries, and the Mica2dot motes not more than 12 hours.

Further Research

Attenuated Signal Testing

For indoor location to be a viable option in the real world, it must work with attenuated signals. In a normal indoor environment, there are walls, cubicles, equipment, and other sources of attenuation.

The test two dimensional mote network for this project was set up in open space, without any obstructions. To test the network in a more realistic environment, the network needs to be set up between rooms, between cubicles, between floors, and in other configurations to test whether or not the location software can handle the differences in signal strength and still calculate an accurate location for the mobile motes.

Standard triangulation requires two points to determine the distance to a point. Trilateration requires three fixed points to determine the location of a point. For redundancy, the location software expects four fixed points. Ideally, with this many fixed points, even in an attenuated environment, the average of the various calculations will give a correct position.

Updated Mote Software

Due to the basic design of the software on the motes, there are concurrency issues with the transmission of the SignalMsg packets to the base mote, and transfer to the laptop. Since the base mote software doesn't do any kind of packet buffering, it will refuse any packets sent to it while it is transferring data on the serial line. That transfer takes between 50 and 100 milliseconds. Any data packets sent to base mote are simply lost. Ideally, the base mote software should be updated so that it queues the packets and transfers continuously, as the serial line becomes available. In addition to this, the transmission times for packets on the radio need to be determined, which would allow each mote to send its SignalMsg updates to the base mote at optimal intervals. This would allow maximum throughput of the packets to the location software.

Furthermore, the software would need to be extended to allow for message passing between motes, to arrive at a base mote. For this network to be useful inside a larger building or complex of any sort, one network of motes with a single base station would not be able to cover the whole area. Therefore, a number of different base motes would have to all have connections back to the central server where the location software is run, probably over Ethernet, but possibly with another wireless link, such as 802.11b/g.

Currently, the sigdata[] array in the SignalMsg type will hold the signal values of up to 13 total motes, as each mote signal value is 2 bytes, and there are 26 bytes available for the data structure. To extend the network, a new data structure would have to be defined, to include the signal strength data and the transmitting mote ID, so more than 13 motes could be used in the network.

Updated Location Software

Due to tight time constraints, the location software is a somewhat imperfect implementation of the Trilateration algorithm. Due to the nature of the network environment, there are numerous error conditions and boundary/weak signal conditions that are not accounted for in the software. As the signal strengths vary, the program needs to be able to select the correct intersections from the various distance circles used to find the mobile mote, and must select between two likely positions.

Furthermore, the software currently will only handle a single mote, and only in two dimensions. However, the code can be updated to handle these two cases with relatively little effort.

Querying/Remote Control of Motes

Additionally, to save power in the whole network, a "pull" operational model could be implemented. The base mote itself runs on 12V DC from the wall, but all of the remote motes must run on battery, which requires that they conserve power. As the network is designed now, each remote mote pushes its data back to the base mote constantly, which is a significant power drain, relatively speaking. The remote motes are also always running once powered on, and there is no way to shut them down, short of removing the battery.

Since radio reception is a much lower-power operation than transmission, the remote motes could normally be in a waiting state, doing nothing but waiting for inbound messages. If the network needs to be used for location, the base mote can send a "power up" command to all of the remote motes, at which point they will come online and begin to broadcast the StrengthMsg packet, or the equivalent, to distribute the signal strength values. When the network is no longer needed, the base mote can send a "power down" signal, and the motes can go back into wait state. This method of operation would allow the battery life of the motes to be significantly extended.

Additionally, instead of timed sending of the SignalMsg packets, as the software does now, the base mote could, instead, query each of the fixed and remote motes, wait for a response, and move on to the next mote. With a timer, as the remote motes operate now, there is a significant amount of time, up to 50%, in some cases, where none of the motes are transmitting SignalMsg packets to the base. With the addition of querying, the base mote can ensure that one of the remote motes is always sending a SignalMsg packet, allowing for much more efficient use of the system.

Take for instance a large industrial structure, such as a chemical production plant. Each area of the structure is wired with a base mote, and a series of fixed remote motes, all of which are in a dormant state. In the event of an emergency, where the site coordinator needs to send personnel into hazardous areas, the base motes can send out the "power up" command, and the network can be put into operation for the duration of the event. Each responder who needs to go into the plant can be given one of the mobile motes to wear, allowing the incident commander to see the locations of all his personnel. While on, the system can report the positions of all the emergency personnel, allowing for faster response time, better incident management, and better safety for all the personnel involved in the fire. Once the incident is over, the base motes can send out the "power down" command, and put the remote motes back into a dormant state.

Ideally, the power-up/power-down would only be used for the fixed motes, as opposed to the mobile motes. The network should be designed with redundant fixed motes, to ensure coverage even in the case that one of the motes does not power up, but the mobile motes should still be in "always-on" mode. This would ensure that the mobile motes, once powered, would always transmit their own StrengthMsg and SignalMsg packets, eliminating a point of failure.

As a final step, for reliability, the base motes could send out a status query packet to all of the fixed motes once every hour, or every day, etc. This would check for fixed motes that are not operating due to damage, power failure, or some other circumstance.

Extension to Three Dimensions

This project served as a proof of concept in two dimensions. Using the location and mote software written for this project, the hardware network could probably be extended to three dimensions. Instead of the four motes used for these tests, forming a square, the network would have to be extended to eight motes, to form a cube. With a cube as the fixed network, the position of a mobile mote could be tracked in three dimensions.

Unfortunately, due to the transmission concurrency problem, with nine motes in the network, there would be significant lag problems. It would take some 10 or more seconds for the location software to get updates from all of the motes.

However, with updates to the software and concurrency model for the motes, the performance of the network could be significantly improved. For this project to have more than theoretical use, of course, positioning must happen in three dimensions.

Conclusion

This project has shown that indoor location using motes is certainly feasible, and warrants further research. Using only the signal strength values between the motes, the position of a moving mote can be determined within a reasonable margin of error. Although there are certainly issues with the network and software design, the basic operation has been demonstrated.

Though the network in this project was simple, the code and network can be extended to test location finding more thoroughly, including extension to three dimensions, and testing with signal attenuation, through walls, equipment, floors, etc.

Appendix A: Installation of Code and User Guide

All the software can be downloaded from the UCCS website at http://cs.uccs.edu/~cs522/studentproj/projF2006/jtwitmer/src/. Download all files and directories into a directory on your local computer.

Mote Software

Install Cygwin and TinyOS 1.1.0-1 from the Crossbow Systems guide at http://www.xbow.com/Support/Support_pdf_files/Getting_Started_Guide.pdf. Copy the BaseMote and RemoteMote folders, and the MakeXbowlocal file to your Cygwin home directory.

**Warning: when attaching and removing the Mica2Dot motes on the programming board, make sure that the power cord is disconnected from the board to avoid damage to the motes. Also, ensure that the battery switch on the Mica2 mote is off when attaching it to the programming board.

Using five motes Mica2 or Mica2Dot motes, choose and mark one of the Mica2 motes as the base mote. The Mica2dot will not work as a base mote. Place that mote on the MIB510 programming board, connect the board to your computer, and install the software with the following command from your Cygwin home directory.

```
cd BaseMote
make mica2 install,0 mib510,<comport #>
```

<comport #> is the COM port to which the MIB510 board is connected. Once the programming is complete, the base mote has been successfully programmed.

To install the RemoteMote software on the other 4 motes, place them on the MIB510 programming board one at a time, and execute the following commands from your Cygwin home directory.

```
cd RemoteMote
make <platform> install,<#> mib510,<comport #>
```

<platform> takes either mica2 or mica2dot, depending on the mote being programmed.
<#> is the number of the mote being programmed, 1-4.
<comport #> is the COM port to which the MIB510 board is connected.

Before each install, open the file RemoteMoteM.nc, and vary the SigTimer value between 2800 and 3300, and vary the StrTimer value between 600 and 1000, on lines 49 and 50. This skews the clocks on the remote motes so that they don't attempt to transmit over each other. The lines look like the following:

```
call SigTimer.start(TIMER_REPEAT, 3200);
call StrTimer.start(TIMER_REPEAT, 900);
```

Now that all five motes have been programmed, the network can be put into operation. Mote 0 is the base mote, and should be attached to the MIB510, and the MIB510 should be connected to the laptop using the serial cable.

Motes 1-3 are the fixed motes, and should be arranged relative to the base mote based on the directions in the Location Software section of this appendix. Mote 4 is the mobile mote, and can be moved within the confines of the plane defined by the fixed motes and base mote.

The base mote draws power from the MIB510 board, and does not require batteries. To turn the other motes on and off, replace or remove their respective batteries.

On the base mote, the red LED cycling indicates that it is sending StrengthMsg packets. The yellow LED cycles each time it receives a StrengthMsg packed from another mote in the network. The green LED cycles each time the base mote receives a SignalMsg packet from another mote in the network.

On the motes running the RemoteMote software, the red LED cycles each time the mote sends a StrengthMsg packet. The yellow LED cycles each time it receives a StrengthMsg packed from another mote in the network. The green LED cycles each time the mote sends a SignalMsg packet to the base mote.

Forwarder.exe Software

Once the motes have been programmed and powered up, the Forwarder.exe software should be started. This software requires that the computer has the Microsoft .NET 2.0 or higher framework installed. This framework software can be downloaded at http://www.microsoft.com/downloads/details.aspx?familyid=0856EACB-4362-4B0D-8EDD-AAB15C5E04F5&displaylang=en. Once the framework has been downloaded and installed, start the program by clicking on the .exe file named Forwarder.exe.

To start the server to the network, select the COM port to which the MIB510 is attached. Press the "Start Server" button. The local IP address will display in the box, and the server will listen for connections on port 9001.

Once the server has started, binary data should start appearing in the gray text box. If not, flip the small white switch on the edge of the MIB510 board. Once data is appearing in the text box, the server is forwarding packets.

The Forwarder server will accept up to 10 connections from clients.

Location Software

The location software requires that the Java 1.5.0 platform or better. This software can be downloaded from http://java.com/en/.

Download the moteview.jar file from src/MoteView/.

From the command line, use the command `java –jar moteview.jar` to run the program.

Click on the Config tab. Fill in the Socket address field with the IP address provided by the Forwarder software. The socket port is 9001.

To form the network for the motes, take the motes programmed 0-3 in the previous section. Mote 0 is the base mote; it is placed in the upper left corner of the square that will be formed. Mote 1 is the upper right corner, mote 2 is the lower left corner, and mote 3 is the lower right corner, looking at the square from above. The square need not be perfectly square, the location software will account for any discrepancies.

Now, using a tape measure, enter the distances between the various motes into the correct boxes on the Config tab. The units aren't important. Once completed, press the Save button. Now, press the Connect button, and switch to the View tab.

Move mote 4, the mobile mote, within the plane defined by the other four motes. In the View tab, the blue dots represent the positions of the base mote and the fixed motes, and the red dot represents the calculated position of the mobile mote. The four circles represent the circles used to find the position. The radius of each circle is found using the signal strength/distance average between the blue motes.

References

[1] Website: “Motes, Smart Dust Sensors, Wireless Sensor Networks”, http://www.xbow.com/Products/productsdetails.aspx?sid=3

[2] Website: “TinyOS FAQ”, http://www.tinyos.net/faq.html

[3] Website: “Trilateration”, http://en.wikipedia.org/wiki/Trilateration