CIRCUIT CELLAR®
THE MAGAZINE FOR COMPUTER APPLICATIONS

FEATURE ARTICLE

by Ethan Leland, Kipp Bradford, & Odest Chadwicke Jenkins

Freescale Wireless Design Challenge.
CONTEST WINNER

Robot Localization and Control

This team shows you how to use wireless nodes to simultaneously localize and control a WowWee Robosapien humanoid robot. The ZigBee nodes' outputs mimic the control signals. A simple GUI makes controlling the robot a cinch.

Imagine you're in a grocery store comparing the ingredients in different brands of applesauce. As you're reading one of the labels, the jar slips from your hand and shatters on the floor in a miniature explosion of glass shards and apple puree. Looking over the damage, you begin to contemplate whom you should inform about the mess, when suddenly, a small wheeled robot rounds the corner and makes a beeline toward the blast site

Intrigued, you watch as the robot seems to pause and assess the situation. Two more robots then arrive on the scene. One efficiently scoops up the mess, and the other wipes the floor clean. The first robot monitors their progress. After a few minutes, the mess is gone, along with the robots, and the store's management has been informed of its applesauce casualty.

Such a team of robots is the result of an area of research that deals with autonomous robot teams and swarm robotics. This interesting field covers problems that involve a diverse network of specialized robots sent out to accomplish a variety of specific tasks. In the grocery store example, one robot is a first-response unit that determines which kind of job is to be done. After surveying the situation, it calls on two other robots to remove the mess and wipe the floor clean. Additional robots in the grocery store network might specialize in different cleaning tasks, restocking shelves, and even leading customers around the store.

One of the hurdles to implementing this kind of robotic team is the problem of localization, which is a key part of any mobile robotic network. Localization is the process of determining the location of nodes within the network as accurately as possible. In this article, we'll present a ZigBee-based localization solution for estimating a robot's position.

SYSTEM OVERVIEW

The 802.15.4 ZigBee wireless standard is a compelling platform for implementing an elegant localization method. In addition to being inexpensive and low power, it allows for enough bandwidth for other types of communication, such as commands and tasks for robots within the network. We used a ZigBee evaluation kit from Freescale Semiconductor to implement a proof-of-concept network for a WowWee Robosapien 1.0, which is a humanoid robot. We implemented the prototype using the kit's three ZigBee nodes to simultaneously localize the robot with signal strength measurements while controlling it with a minimal command set.

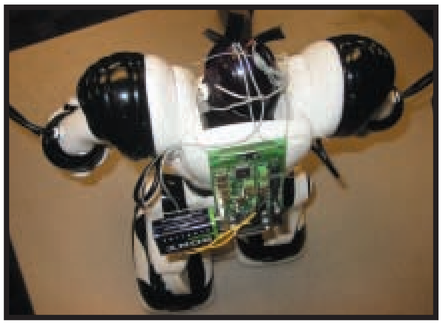


Photo 1—*Check out the Robosapien 1.0. We attached a Freescale ZigBee node to its back so we could control it.*

Our localization and control system features a Freescale MC13192 evaluation board with three accelerometers (MMA6261Q for the x- and y-axes and MMA126OD for the z-axis) mounted on the Robosapien robot (see Photo 1). The system also includes an MC13192 SARD board connected to a PC via an RS-232 serial connector and another MC13192 evaluation board in the environment.

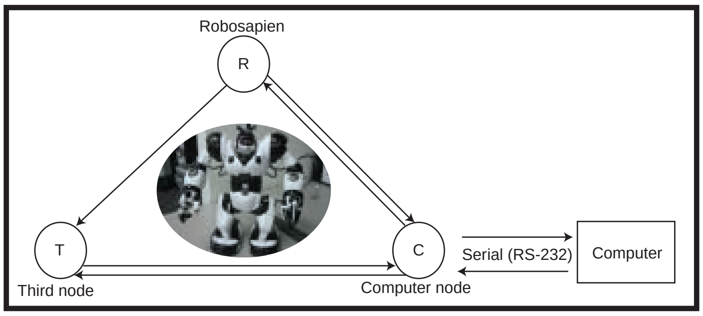


Figure 1—*The ZigBee nodes are labeled R, T, and C. The arrows represent sent packets.*

Figure 1 shows the nodes attached to the Robosapien. The communications model we used is based on a broadcast model in which the senders and receivers aren't addressed specifically, but the packets incorporate a basic header describing the kind of data they contain.

LOCALIZATION

The localization process is fairly straightforward. The Robosapien node (R) frequently sends out packets containing accelerometer data. Both the computer node (C) and the third node (T) receive the packets. (The accelerometer data isn't used in this implementation, but we'll explain it later when we describe some possible improvements to the system.)

After a packet is successfully received, a signal strength reading is taken by calling the `PLME_link_quality` method, which is found in the `simple_phy` class of the Freescale support code. After the third node has accomplished this, it constructs a packet containing the resulting signal strength measurement and sends it to the computer node. At that point, C, which has both the signal strength reading from R/T and R/C, composes a packet with these measurements to be sent via the serial port to the computer.

When the computer receives both signal strength measurements, it can begin hypothesizing about the robot's location. However, this can't be done with a straightforward geometric calculation because of the uncertainty associated with noisy data and the signal strength's nonlinearity.

This noise is such that if simple distance calculations were applied at each time step, the computer might find that the robot's location would vary on the order of meters when it is standing still. To deal with this large degree of uncertainty in our data, we used a probabilistic Monte Carlo localization technique to implement a particle filter to localize the robot. Let's take a look at how the particle filter reduces the uncertainty of the robot's location.

SOFTWARE

We wrote software for the PC in Java to receive signal strength measurements from the computer's serial port and incorporate the data into the current localization model. Our localization model is a probabilistic technique known as a particle filter, which is described in Sebastian Thrun et al.'s *Probabilistic Robotics*. Particle filters work by first distributing random samples called particles over the space being observed. Each particle represents a possible physical location in the environment. A probability value is assigned to each particle. This probability represents the likelihood that the robot is at the location specified by the particle.

At each time step, each particle is reevaluated and its probability value is updated according to the ZigBee signal strength measurements. Less likely particles are then redistributed around more likely particles. This is done by building a cumulative sum graph of the normalized probabilities of each particle (see Figure 2). This graph is then randomly sampled to create a histogram that dictates where the particles should be distributed at the next time step. The particles concentrate around locations that have a higher probability of being the robot's location.

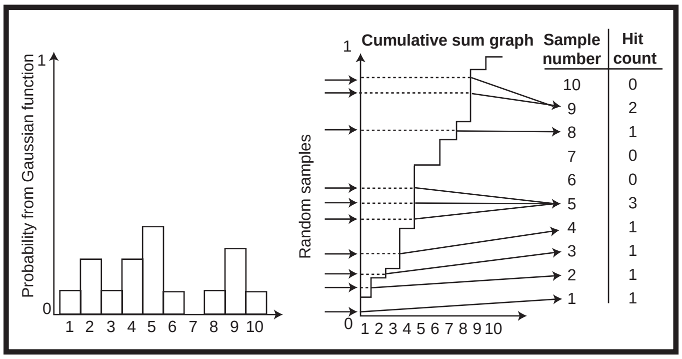
Finally, after the particles have their new coordinates, a small amount of random noise is added to each particle's location so that they're distributed around likely locations instead of concentrating at a single point. This helps maintain a small degree of uncertainty in our model and better reflects real-world conditions.

In our project, the localization software reevaluates each particle's position and probability when the PC receives the signal strength measurements from the computer node. It first performs a simple geometrical computation to convert the measurements into a coordinate pair (x_{DATA}, y_{DATA}) representing the approximate robot location. Because we only had the three ZigBee nodes that came with the Freescale kit to work with, we were not able to perform true triangulation, which requires at least three nodes to triangulate a forth node. Instead, we assumed that the T and C nodes were placed on a wall that the R node couldn't pass through.

If you were to assume that the signal strength measurements were 100% accurate, the location of the robot would reside at the intersection of two imaginary spheres centered at each of the sensing nodes, with radii proportional to the signal strength. The intersection of these two spheres is a circle that resides in a plane perpendicular to the line connecting the two sensors. If you use the plane of the floor as a constraint, you know that robot must be at one of two points on that circle. Finally, you can eliminate one of these points by putting the two sensors on the same wall and guaranteeing that the robot will always be on the same side of that wall.

On the computer, the coordinate pair (x_{DATA}, y_{DATA}) was averaged over 10 data-consecutive data points before being fed to the particle filter. Averaging helped reduce some of the noise in the data. The particle filter then used 1,000 particles to localize the robot. The probability value for a single particle was determined by plugging the particle's coordinates (x_{SAMPLE}, y_{SAMPLE}) into a standard Gaussian function based on the x_{DATA}, y_{DATA} point determined by the signal strength measurements. To guess at the robot's location, we simply took the weighted average of the 1,000 samples.

Photo 2 shows the particle filter running for three consecutive time steps. You can see the rapid convergence of the particles to the area of the robot marked in blue in the center of each picture. As the robot moves in the environment, the signal strength measurements are frequently updated, and the particle filter is updated in real time.



Sample number	Hit count
10	0
9	2
8	1
7	0
6	0
5	3
4	1
3	1
2	1
1	1

Figure 2—*The left graph shows the normalized probability versus robot locations 1 to 10, shown on the x-axis. By randomly sampling the cumulative sum graph (shown in the middle), we computed where the particles should be located given the latest signal strength measurement. The histogram shows that the particles will be relocated mostly in the proximity of locations 5 and 9. The particles are concentrated around the robot's most likely position.*

ROBOT CONTROL

A simple GUI runs on the PC during the localization process. The GUI gives you some simple options for commanding the Robosapien. For this project, the commands were limited to Stop, Go Forward, Turn Left, and Turn Right. When you select a command, the computer sends a command packet down the serial port to the C node, which broadcasts the command packet over the wireless network. When the R node receives this packet, it executes the command specified on the Robosapien and then broadcasts a packet to signify that it has performed the command. Meanwhile, the T node stops sending information after it receives the command packet so it won't bog down the network. It then begins its transmissions once it receives the command-executed packet from the R node.

To ensure that the correct command has been executed, both the command and command-executed packets also contain a command number indexed from zero at the beginning of operation. It wraps back to zero when the maximum command number has been reached. Table 1 includes all of the packets and their formats.

The final phase of our project involved executing a command at node R on the Robosapien. After dissecting the Robosapien, we decided that the easiest way to command the robot using a Freescale ZigBee node would be to replace the Robosapien's infrared receiver with the ZigBee node. The node was programmed to mimic the signals from the Robosapien's infrared receiver. These signals were discovered experimentally with an oscilloscope. We found that command signals sent to the Robosapien were made up of four distinct time slices: 7, 3.3, 1, and 0.7 ms. When data isn't transmitted, the logic level for the command signal is high (logic 1).

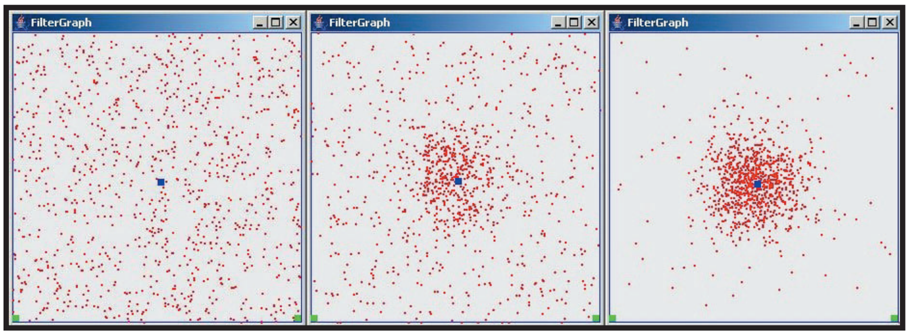


Photo 2—*The GUI displays three consecutive iterations of the particle filter while it's running. The two green squares in the lower corners represent the ZigBee T and C nodes. The blue square in the middle represents the best guess at the robot's location.*

Sent by node	Description	Received and acted upon by node(s)	Packet format
R	Accelerometer data (A)	C + T	[r, A_{XHIGH}, A_{XLOW}, A_{YHIGH}, A_{YLOW}, A_{ZHIGH}, A_{ZLOW}]
T	Signal strength reading (R/T)	C	[t, SS_{HIGH}, SS_{LOW}]
C	Command packet	R + T	[c, Command, Command_Number]
R	Command executed packet	C + T	[e, Command_Number]
C	Data to the computer over the serial (RS-232) port	Only the computer	[$SS1_{HIGH}$, $SS1_{LOW}$, 1, $SS0_{HIGH}$, $SS0_{LOW}$, 0, A_{XHIGH}, A_{XLOW}, A_{YHIGH}, A_{YLOW}, A_{ZHIGH}, A_{ZLOW}]

Table 1—*The various packets contain different information. Each packet broadcast over a wireless radio begins with data describing the sender. The HIGH and LOW bits of each number were used because larger 16-bit numbers had to be broken up into two smaller 8-bit numbers in order to be sent using the ZigBee radios.*

For example, take a look at the Go Forward command in Figure 3. These signals include only four time slices, so they are easily stored on a Freescale node as an array.

Take a look at the Go Forward command stored on a node:

```
int FORWARD_COMMAND[] = {1,2,3,4,3,4,3,
  4,3,4,3,2,3,2,3,4,3,-1};
```

Each number represents one time slice. The terminating `-1` exists because not all commands are the same number of time slices. This format allows for a simple `while` loop command implementation that uses a delay function to create each time slice. Port C5 on the MC13192 evaluation board was connected to the Robosapien's IR receive wire in place of the actual IR receive module. We were able to transmit commands from the ZigBee network to the Robosapien by toggling port C5 high and low.

POTENTIAL IMPROVEMENTS

There are many improvements that you can make to our project. You can use the accelerometer data sent by the Robosapien's Freescale node to predict which way the robot is moving relative to the accelerometers. This could be used to increase the accuracy of the probability function that evaluates samples in the particle filter. The result would be more accurate localization estimates. We successfully implemented this idea in the particle filter code, but it was impractical because of the Robosapien's motion. When the Robosapien moves, it shifts its weight from side to side violently, making the accelerometer readings useless. For this method to contribute to the localization estimates, you'll need a robot that moves more smoothly.

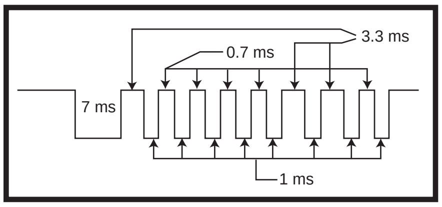


Figure 3—*We deciphered the Go Forward signal using an oscilloscope.*

The best possible improvement, however, would be to add more ZigBee nodes. We didn't have the resources to do so, but adding more nodes (to behave like the T node) would supply the localization routine with much more data. The result would be more accurate guesses as to robot's location and thus better control over the robot.

FORWARD THINKING

To take this project further, you could develop a client/server relationship in which the localization for all the robots would be performed on the server. The server could provide a host for other valuable functions to which client programs could subscribe in order to use the system. Robot control and message routing would be among these functions. The clients shouldn't have to handle any of them.

A low-power ZigBee radio is an inexpensive base for an autonomous robot team. By using inexpensive mobile robotics platforms with this kind of functionality, you can implement powerful higher-level applications such as the robot team we described at the beginning of this article.

There are other potential applications to consider as well. You can use robot localization for search-and-rescue missions, terrestrial exploration, and natural resource extraction. Using robots to move in potentially hazardous areas can save lives.

The world of gaming could be revolutionized too. Imagine physical embodiments of real-time strategy games that act out the commands that gamers send them.

This technology has the potential to change our lives. In the near future, autonomous robots may finally fulfill the promise of removing humans from dangerous situations. ZigBee technology can empower robot developers to create inexpensive, task-specific robots that may one day autonomously navigate through and perform work in highly dynamic, real-world situations.

Ethan Leland (eleland@cs.brown.edu) is working on a Master's degree in computer science at Brown University. Ethan is interested in artificial intelligence and robotics. He is currently working on a RoboCup project for his Master's degree.

Kipp Bradford (kb@kippworks.com) holds Bachelor's and Master's degrees in engineering from Brown University. For the past 10 years, he has been inventing and designing electro-mechanical medical devices, toys, and consumer products.

Odest Chadwicke Jenkins (cjenkins@cs.brown.edu) is an assistant professor of computer science at Brown University. He earned a B.S. in computer science and mathematics at Alma College, an M.S. in computer science at Georgia Tech, and a Ph.D. in computer science at the University of Southern California.

PROJECT FILES

To download the code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2006/188.

RESOURCE

S. Thrun et al., *Probabilistic Robotics*, The MIT Press, Cambridge, MA, 2005.

SOURCES

MC13192 Evaluation board
Freescale Semiconductor, Inc.
www.freescale.com

Robosapien
WowWee
www.wowwee.com