Parasitic Mobility for Sensate Media

by

Mathew Joel Laibowitz

B.S. Computer Engineering
Columbia University, 1997

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
In partial fulfillment of the requirements for the degree of

Masters of Science in Media Arts and Sciences

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2004

Author [illegible]

Program in Media Arts and Sciences
August 12th, 2004

Certified By [illegible]

Joseph A. Paradiso
Associate Professor
Sony Career Development Professor of Media Arts and Sciences
MIT Program in Media Arts and Sciences

Accepted By ______________________

Andrew Lippman
Chair, Department Committee on Graduate Students
MIT Program in Media Arts and Sciences

Parasitic Mobility for Sensor Networks

by

Mathew Joel Laibowitz

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
on August 12th, 2004, in partial fulfillment of the
requirements for the degree of
Master of Science in Media Arts and Sciences

ABSTRACT

Distributed sensor networks offer many new capabilities for monitoring environments with applicability to medical, industrial, military, anthropological, and experiential fields. By making such systems mobile, we increase the application-space for the distributed sensor network mainly by providing dynamic context-dependent deployment, continual relocatabililty, automatic node recovery, and a larger area of coverage. In existing models, the addition of actuation to sensor network nodes has exacerbated three of the main problems with these types of systems: power usage, node size, and node complexity. This work proposes a solution to these problems in the form of parasitically actuated nodes that gain their mobility and local navigational intelligence by selectively engaging and disengaging from mobile hosts in their environment. This body of work evaluates parasitically actuated sensor networks as a solution to these problems through extensive software simulation and by designing, implementing, and demonstrating a parasitically mobile sensor network.

Thesis Supervisor: Joseph A. Paradiso
Title: Associate Professor, Program in Media Arts and Sciences

Parasitic Mobility for Sensor Networks

by

Mathew Joel Laibowitz

The following people served as readers for this thesis:

Thesis Reader [illegible]

Joseph A. Paradiso
Associate Professor
Sony Career Development Professor of Media Arts and Sciences
MIT Program in Media Arts and Sciences

Thesis Reader [illegible]

David P. Reed
Adjunct Professor
MIT Program in Media Arts and Sciences

Thesis Reader [illegible]

William J. Kaiser
Professor of Electrical Engineering
University of California, Los Angeles

Acknowledgements

I would like to thank and acknowledge the continuing support, without which none of this would be possible, much less enjoyable, of advisor, readers, and colleagues Joseph Paradiso, William Kaiser, David Reed, Josh Lifton, Ari Benbasat, David Merrill, Hong Ma, Mark Feldmeier, Dan Lovell, and Michael Broxton.

I would like to thank my family, Robert Laibowitz, Laura Laibowitz, Ken Laibowitz, Danielle Laibowitz, Ben, Hannah, Julianna, Rebecca, Rachel, and my grandmother Giulianna (Nonna) Goetzl for all the support and love throughout my life leading to my ability to tackle challenges such as this.

I would like to thank my friends in New York, and apologize for always being at least 8 hours late to their events.

I would especially like to thank Talia Dorsey for her direct help with the soldering, cutting, taping, gluing, finding the plastic spheres, graphics design, writing, and photography; and for all her creative inspiration throughout my life.

And finally, I would like to thank everyone who made my test a success by being attracted to strange devices with blinking lights.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Basic Principle

We are at a point in time where advances in technology have enabled production of extremely small, inexpensive, and wirelessly networked sensor clusters. We can thus implant large quantities of sensors into an environment, creating a distributed sensor network. Each individual node in the network can monitor its local space and communicate with other nodes to collaboratively produce a high-level representation of the overall environment. By using distributed sensor networks, we can sculpt the sensor density to cluster around areas of interest, cover large areas, and work more efficiently by filtering local data at the node level before it is transmitted or relayed peer-to-peer. [1]

Furthermore, by adding autonomous mobility to the nodes, the system becomes more able to dynamically localize around areas of interest allowing it to cover larger total area with fewer nodes by moving nodes away from uninteresting areas. It is well suited to sampling dynamic or poorly modeled phenomena. The addition of locomotion further provides the ability to deploy the sensor network at a distance away from the area of interest, useful in hostile environments. Cooperative micro-robots can reach places and perform tasks that their larger cousins cannot. [2] Mobility also allows the design of a system where nodes can seek out power sources, request the dispatch of other nodes to perform tasks that require more sensing capability, seek out repair, and locate data portals from which to report data. [3]

But the creation of mobile nodes is not without a price. Locomotion is costly in terms of node size and power consumption. In dense sensor systems, due to the large quantity of nodes and distributed coverage, it is difficult to manually replace batteries or maintain all nodes. Some researchers [4] have explored using robots to maintain distributed networks, but this is difficult to implement over large, unrestricted environments. Additionally, the added intelligence and processing power required for a node to successfully navigate in an arbitrary environment further increases the power and size requirements of each node. Large nodes, in physical size, complexity, cost, and power consumption, prevent the sensor network from being implanted in most environments. [5] [6]

This research is concerned with exploring a novel type of mobile distributed sensor network that achieves the benefits of mobility without the usual costs of size, power, and complexity. The innovation that allows this to happen is the design of nodes that harvest their actuation and local navigational intelligence from the environment. The node will be equipped with the ability to selectively attach to or embed itself within an external mobile host. Examples of such hosts include people, animals, vehicles, fluids, forces (eg. selectively rolling down a hill), and cellular organisms. These hosts provide a source of translational energy, and in the animate cases, they know how to navigate within their environment, allowing the node to simply decide if the host will take it closer to a point of interest. If so, the node will remain attached; when the host begins to take the node farther away from a point of interest, the node will disengage and wait for a new host.

This area of research aims to develop and understand a potential method for the combination of mobile sensor agents, dense distributed sensor networks, and energy harvesting. This thesis presents the design and development of hardware and software systems to address the combination of these interests as parasitic mobility.

1.2 Related Work

Although this research has no direct precedent, it is inspired by systems in nature and human society (discussed in Chapter 2) and it builds upon current work in the encompassed fields of distributed sensor networks and mobile systems.

Wireless sensor networks have become a large area of research, with many universities and institutes contributing. Strategic seed programs begun in the 1990s such as DARPA's SENSIT initiative [7], have grown into an international research movement.

Early work on highly distributed computation and sensor networks at MIT that provides the lineage to this project can be traced back to the Laboratory of Computer Science's Amorphous Computing Group's research in emergent and self-organizing behaviors in computer systems [8]. This research conducted software simulations that provided a basis for designing distributed, cooperative systems, leading to the Paintable Computing [9] paradigm proposed by Bill Butera of the MIT Media Lab's Object Based Media Group. This platform has progressed from software simulation to the very recent development of hardware implementing a distributed sensor network comprised of about 1000 nodes. The Responsive Environments Group at the MIT Media Lab designed an earlier versatile sensor network test-bed inspired by the Paintable Computing concept called the Push-Pin computing platform [10], which can support over 100 nodes arbitrarily placed atop a 1x1 meter power substrate. Their subsequent interest in electronic skin as an ultra-dense sensor network [11] resulted in the creation of the "Tribble" project [12], a large sphere tiled by a hardwired multimodal sensor network. These systems consist of many nodes instrumented with environmental sensors that can communicate with each other to form a global picture of their situation. All the above projects illustrate many ideas in distributed sensor networks that motivate this research and provide a basis for the design of a system useful in experimenting with the concept of parasitic mobility.

The Smart Dust Project at UC Berkeley [13] has set a theoretical goal for extremely small nodes in dense embedded sensor networks. While the project itself did not put an actual hardware platform into production, it spun-off into the Mote [14] and more recently the Spec [15]. The Mote is currently the most popular platform for experimenting with compact wireless sensing. It has also served as a building block for many mobile sensor agent projects, all of which essentially involved putting a Mote onto some sort of robot [4]. The Spec is the current result of a project intended to shrink down the Mote to the theoretical goal of the Smart Dust project. While not yet that small, the Spec is around 4mm x 4mm (not including the battery or antenna) and will open the door for many dense sensor array experiments. Similar work is also proceeding at other institutions (e.g. The National Microelectronic Research Center in Cork, Ireland [16]); the research community is congealing around the goal of producing millimeter sized multimodal wireless sensor nodes. Parasitic Mobility is intended as a means to add mobility to systems built to meet the specifications of these projects with regards to size, power, and node complexity; as the nodes grow smaller, parasitic mobility becomes increasingly feasible and desirable. As the power source remains a problem and current research in energy scavenging [17] and adaptive sensing [18] is very relevant to this initiative. Adaptive sensing is the technique by which sensing capabilities (active sensors, sampling rate, power consumption, bit-depth, transmission, processing) are increased and decreased according to the sensor data itself, never decreasing below a level capable enough to determine when more sensing power is necessary. Such approaches are currently being implemented using the Stack Sensor Platform [19] at the MIT Media Lab. The Networked InfoMechanical Systems research area at the Center for Embedded Networked Sensing at UCLA conducts research and builds systems to investigate adaptive sensing [18] and mobility for distributed sensor networks [20].

The MIT Laboratory for Computer Science's Network and Mobile Systems group has conducted substantial research in wireless and sensor networks. Several of their projects are directly related to the work of this thesis. They include a protocol for networking Bluetooth nodes [21] and the LEACH protocol for sensor networking [22]. These are examples of self-configuring network protocols that support mobile nodes of any variety including parasitically actuated.

And finally, while not distributed sensor networks, there are several mobile sensor devices built by attaching large sensor packages to floating platforms that drift about in ambient flows while collecting data. Some examples include Sonobuoys [23] that acoustically hunt for submarines, drifting instrumentation packages to monitor ocean temperature [24], and balloon-borne modules for surveillance and proposed planetary exploration [25].

Chapter 2

Examples of Parasitic Mobility

2.1 Parasitic Mobility in Nature

The natural world provides us with many examples of parasitic mobility, including organisms that rely entirely on larger organisms to carry them to habitable locations. Parasitic relationships of this sort are called phoretic relationships from the word phoresis, which literally means transmission [26]. In the context of this thesis, these examples are separated into three categories: active parasitic mobility consisting of organisms that attach and detach at will from hosts with their own actuation, passive parasitic mobility consisting of passive nodes that are picked up and dropped off, knowingly or unknowingly, by hosts, and value-added parasitic mobility which consists of either passive or active parasitic organisms that provide additional value to the host in exchange for transportation.

2.1.1 Active Parasitic Mobility

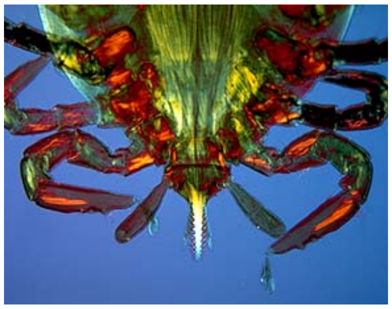


Figure 2-1: Close-Up of a Tick's Gripping Mechanism

The first example that comes to mind when discussing parasites in nature is the tick. The tick actively attaches to hosts by falling from trees or by crawling directly onto the host. It remains attached by using an actuated gripping mechanism which it can release whenever it decides to seek food elsewhere. Although the tick is transported to new locations by the host, its primary reason for attachment is to use the host as a source of food. It is therefore not normally considered a phoretic organism. It is still relevant to the topic as the main example of an active attachment mechanism.

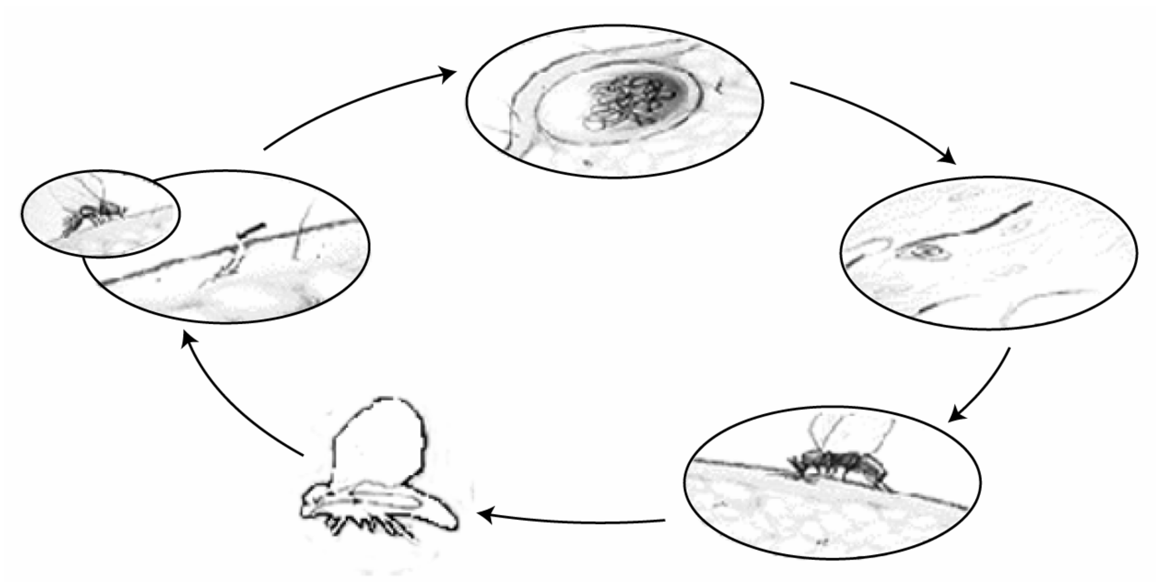


Figure 2-2: The Life Cycle of the Onchocerca Volvulus. Adult females release millions of microfilariae into the bloodstream of the host. There they are picked up by feeding blackflies and brought to a new host where they can start the life cycle again.

Several species of nematodes, a.k.a. round worms, exhibit phoretic behaviors. The Pelodera Coarctata is a nematode that is commonly found living in cow dung. When the conditions in the dung deteriorate and become inhospitable for the nematode, it attaches itself to a dung beetle which will carry it to a new fresh dung pat. [27] Another such nematode is the Onchocerca Volvulus which is infamous as the cause of “River Blindness.” This worm attaches itself to Blackflies that in turn bite humans allowing the worm to travel through the skin and infect the host human. These Blackflies themselves are also an example of parasitic mobility. Their larvae require an aquatic stage for growth, so they often attach themselves to freshwater crabs to bring them into the water and protect them. [27]

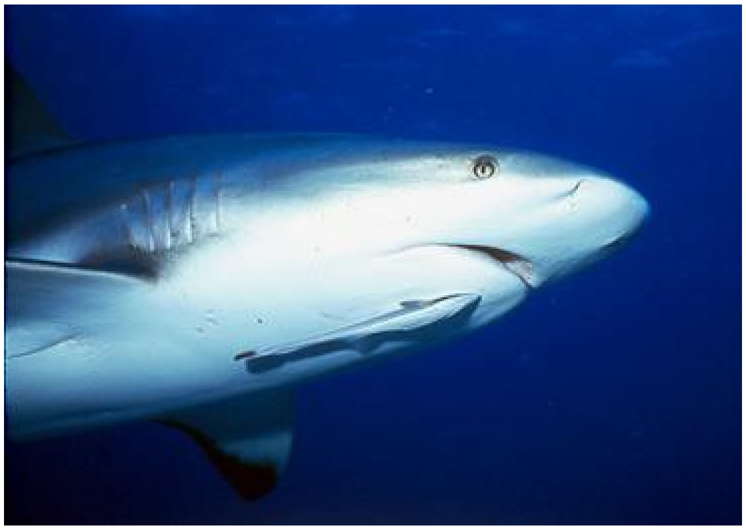


Figure 2-3: A Remora hitching a ride on a shark

Marine life is ripe with examples of active parasitic mobility. One example is that of the Remora or Suckerfish. These fish have developed a sucker-like organ that they use to attach to larger creatures such as sharks or manta rays. By attaching to these larger, faster animals the remora covers area faster giving it more access to food. [28]

2.1.2 Passive Parasitic Mobility

Plants often employ parasitic mobility as a means of distributing seeds. A common example of this is the dandelion. The dandelion seeds have a tiny parachute that carries the seed with the wind. This allows the seeds to travel some distance in hopes of landing in an area that provides the requirements of growth. It is completely passive and at the whim of the wind. It is not expected that all the seeds will land in arable areas. This is overcome by the sheer quantity of seeds released into the air. This is more opportunistic than parasitic, but still falls within the conceptual boundaries of this research.

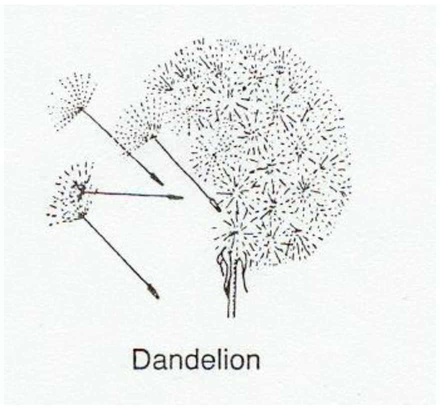


Figure 2-4: Dandelion seeds catching a ride from the wind

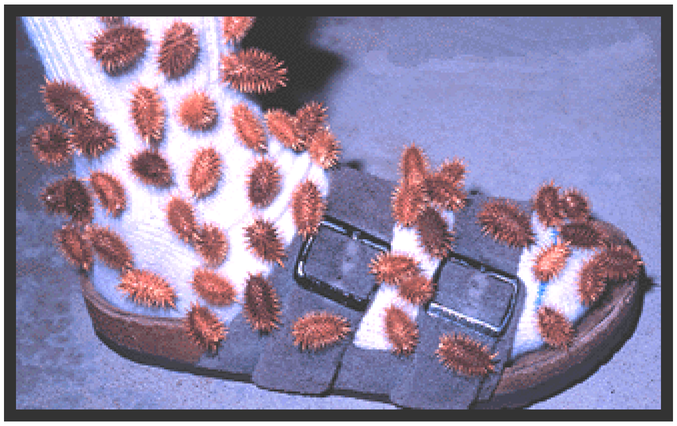


Figure 2-5: Burs stuck to a foot being brought to a new location

Other plants, with behaviors more aptly described as parasitic, distribute their seeds in bur casings. These prickly cases stick to animals that brush up against them or step on them. They are shaken lose or fall off as a result of shedding, usually at a new location.

2.1.3 Value-Added Parasitic Mobility

Fruit-bearing trees distribute their seeds in a value-added method. Animals gather the fruits as a food source and in turn spread the discarded seed-containing cores. This attraction and provision acts as an attachment mechanism for the seeds. The detachment mechanism is the inedibility of the seeds within the fruit, in other words, when the added value has been used up.

Figure 2-6: Bees are attracted to flowers by the petals and the nectar. Once inside the bees are covered with pollen which they carry to another flower to complete the pollination process.

Flowers use their scented petals to attract bees and other insects. The flowers also provide nectar. The bees use the nectar to make honey and carry the pollen from flower to flower. This is an extremely well evolved symbiotic system that has very little wasted energy or resources. [29]

The existence of many such well evolved systems in nature illustrates the validity of this type of mobility, and justifies researching further how to use this concept in our research.

2.2 Parasitic Mobility in Society

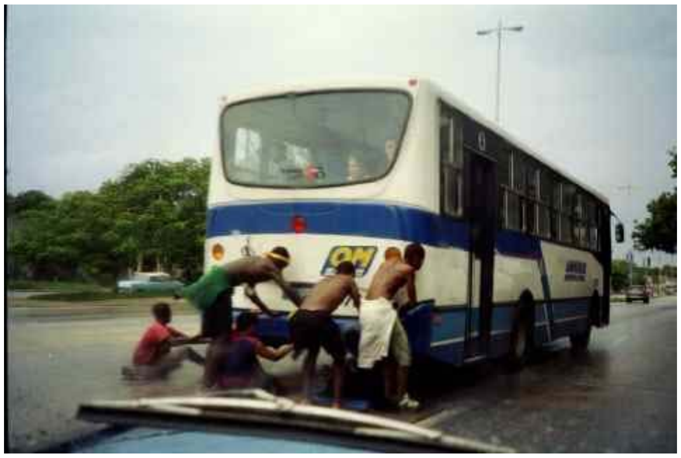


Figure 2-7: People hitching a ride on a bus

In human society, many of the systems surrounding us exhibit emergent behaviors that exemplify parasitic mobility. It is important to examine these systems, not only as conceptual examples, but also because it may be possible to embed sensor network technology directly into these existing systems and take advantage of their mobility.

Basic examples, such as people being pulled along by a bus as shown in Figure 2-7, exist throughout society. It is often beneficial to attach to something that can travel in ways that a person cannot. This example further illustrates the economies of parasitic mobility; the people are getting a free ride. This concept has been taken further by Neal Stevenson in his novel Snow Crash [30]. In Snow Crash, hitching rides on other vehicles is presented as a major method of transportation in the future setting of the story.

A simple example of parasitic mobility is when a lost object, such as a cellular phone, is returned to its owner. This method of actuation is a combination of the device identifying its destination and a desire for the host to bring it there. Keeping this in mind, it may be possible to design devices that could identify some sort of reward for bringing them to a point of interest to the device. Another example of this behavior is that of a consumer survey (a sensor of sorts) that is redeemable as a coupon when returned.

There are many everyday objects that are only useful for short bursts. One example of this is a writing utensil. A pen is needed to record information when it is presented or invented; afterwards the pen sits dormant awaiting the next burst of usefulness. During this period where the pen is not deemed useful it is free to be relocated. It is often relocated by a host requiring its use in another location. As a result, pens generally cover large areas over time, and due to their unlikelihood of being returned, people usually have redundant supplies of pens. Equipping pens with a sensor device is a good way to gain coverage of an environment, particularly an office or academic institutional building.

2.3 Fictional Examples of Parasitic Mobility

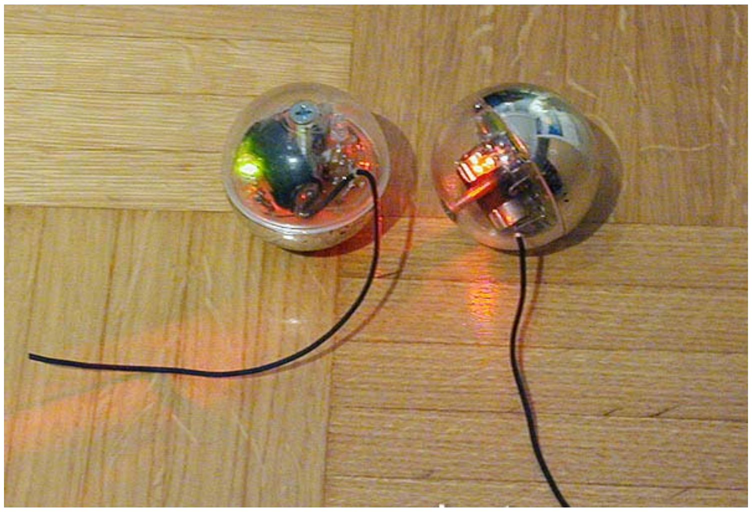


Figure 2-8: Dorothy Sensors from the movie Twister. Image copyright © Tim Ketzer.

In the movie Twister [31], a team of storm-chasers release a batch of sensors into a tornado. The sensors, collective called 'Dorothy', are sucked up into the vortex and collect data about the tornado from the inside. These sensor nodes are carried into the area of interest by winds themselves. In this case the sensor nodes are used to study the actuation force itself, and is mobile along with the force thereby always being at the area of interest. Although it seems possible that this system can be deployed, according to the National Severe Storms Laboratory [32], such devices have not been built. They have experimented with a large barrel-sized sensor device called TOTO (TOtable Tornado Observatory), but these tests have yielded only minimal success.

Finally, the most famous example of an object that travels without its own actuation is ‘The One Ring’ from the “Lord of the Rings Trilogy.” [33] This ring calls out to potential hosts to pick it up, and even renders the wearer invisible as a value-added service. And finally, the ring desires to be brought to a location which also happens to be the only place it can be destroyed; a promised reward for its successful journey.

Chapter 3

Software Simulation

In order to better delve into and examine the concept of parasitic mobility, extensive software simulation was performed. Through the process of designing the software simulator, the proposed systems were examined from the ground up, looking at all the factors that influence a potential sensor network of this type. This was a critical first step into research of this topic. Upon its completion, the simulator was an invaluable asset for testing and examining ideas and algorithms, collection of data identifying expected behaviors, further proving the validity of the overall concept, and providing insights directly used in the design of the hardware system described in chapter 4.

3.1 Design Overview

The design of the software simulator can be broken down into three sections: Environmental Simulation, Host Behavior, and Paramor Behavior. “Paramor” is the name given to a parasitically mobile node and is an apt anagram for PARAsitic MObility Research. On top of these areas, the simulator contains all the necessary hooks for interactively changing behaviors and trying out new algorithms, detailed logging of activity data, and unattended running of multiple simulations with a desired timescale.

The simulator is grid based, and has been tested with maps as large as one million cells. The hosts and paramors participating in the simulation move by transitioning from cell to cell.

3.1.1 Environment Setup

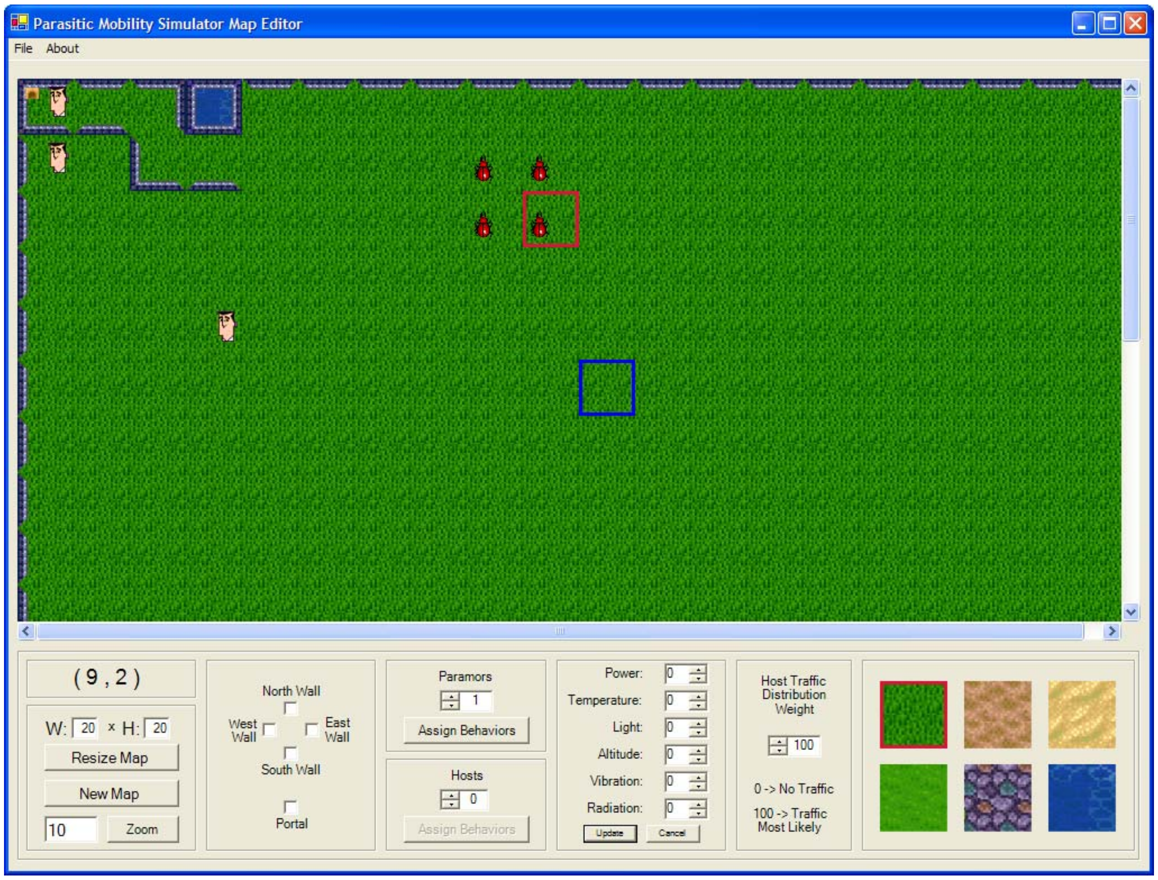


Figure 3-1: Screenshot of the Parasitic Mobility Simulator Map Editor

The first step in setting up a simulation is to layout the environment using the Parasitic Mobility Simulator Map Editor. This is the interactive graphical application shown above in Figure 3-1. The user interface for this tool consists of a window displaying a scrollable, tile-based map, and a control panel for editing the parameters of the selected tile. The red square around a tile indicates the currently selected tile editable with the control panel, and the blue square follows the mouse pointer as a cursor for selecting a new tile to edit.

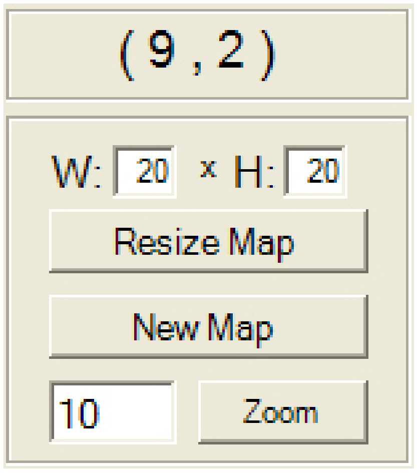


Figure 3-2: General Map Control section of control panel

The first section of the control panel located in the bottom left displays the coordinates of the currently selected tile. It also displays and allows modification of the current dimensions of the map. The zoom function scales the display, but has no effect on the map itself.

It is important to note how the dimensions of the map relate to distances in the real world. The resolution of the map should be directly related to the resolution of the sensors and location system on the sensor nodes. For example, if you want to simulate an environment of 100 meters by 100 meters populated by sensor nodes that can identify their location with a resolution of 2 meters by 2 meters, you would create a map that is 50 tiles by 50 tiles.

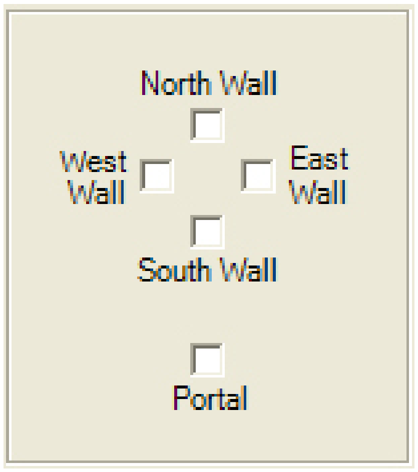


Figure 3-3: Wall and Portal section of the control panel

The next section of the control panel is where you can assign walls and portals to the currently selected tile. The walls allow you to design a map with particular paths for the hosts to follow and to design a map that is based on a real environment. The portals are tiles that act as entrances and exits for the host bodies to enter and leave the environment. When a host arrives at a portal, it can decide to keep moving or it can exit and take itself out of the area. The attached paramors can react to this and jump off. The host stays outside for a duration according to its behavioral parameters (see section 3.1.2 for details on host behavior), and then returns through any of the available portals. Portals are not required for simulation, but they allow multiple maps to be linked together. At the far right of the control panel, six options for textures are able to be applied to the current tile. This is for display only, and has no bearing on the simulation. If you want to make a square inaccessible for a host body, you need to surround it with walls or set the host frequency to 0 for that square. The host frequency, or host traffic distribution weight, is the likelihood that a host will travel past a given location. It can be set per tile using the section of the control panel shown in Figure 3-4. This is further explained in section 3.1.2.

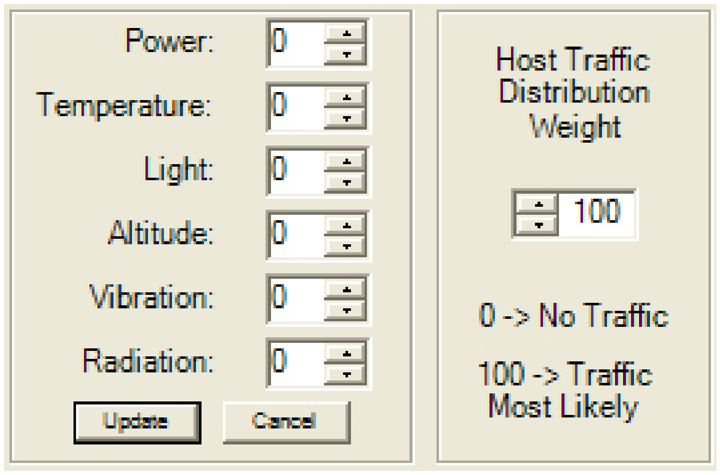

Power: 0
Temperature: 0
Light: 0
Altitude: 0
Vibration: 0
Radiation: 0
Host Traffic Distribution Weight: 100
0 -> No Traffic
100 -> Traffic Most Likely

Figure 3-4: Environmental conditions and host frequency section of the control panel

The section of the control panel shown in Figure 3-4 allows the user to create areas of interest for the sensor nodes and for the hosts. By setting the power parameter above zero, the tile becomes a source of power for the node to recharge its reserves. This is described in more detail in section 3.1.3 where the node behavior is described. The nodes can also be programmed to look for certain environmental conditions such as temperature. The Host Traffic Distribution Weight parameter sets the tile's attractiveness for a mobile host on a scale of 0 to 100, where 0 means there is no interest at the tile and the hosts will avoid it completely, and 100 means the tile is very attractive to hosts and has the highest likelihood of host traffic.

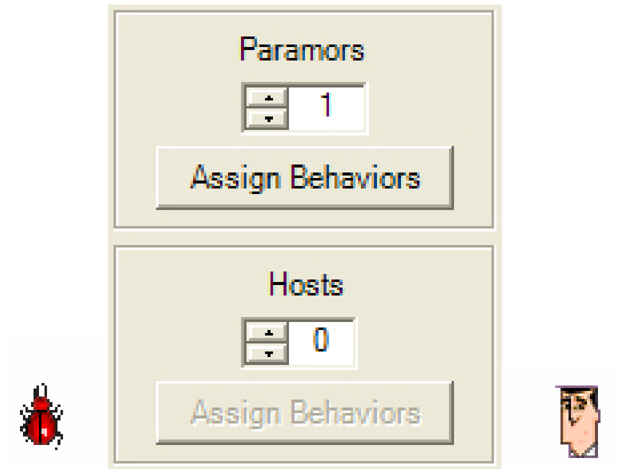


Figure 3-5: Host and Paramor Assignment Panel and map symbols for hosts and paramors

And finally, the most important section of the control panel allows you to populate the environment with host bodies, deploy paramor nodes, and assign behaviors to these entities. Up to 100 hosts and paramors can be deployed per tile. Tiles that have hosts or paramors assigned to them are indicated on the map by a head and a tick, respectively. Clicking on the Assign Behaviors button brings up a window where a name can be given to the behavior of each paramor or host. During the map creation process, only a name can be assigned; designing the actual behavior is done at a later stage of the simulation setup.

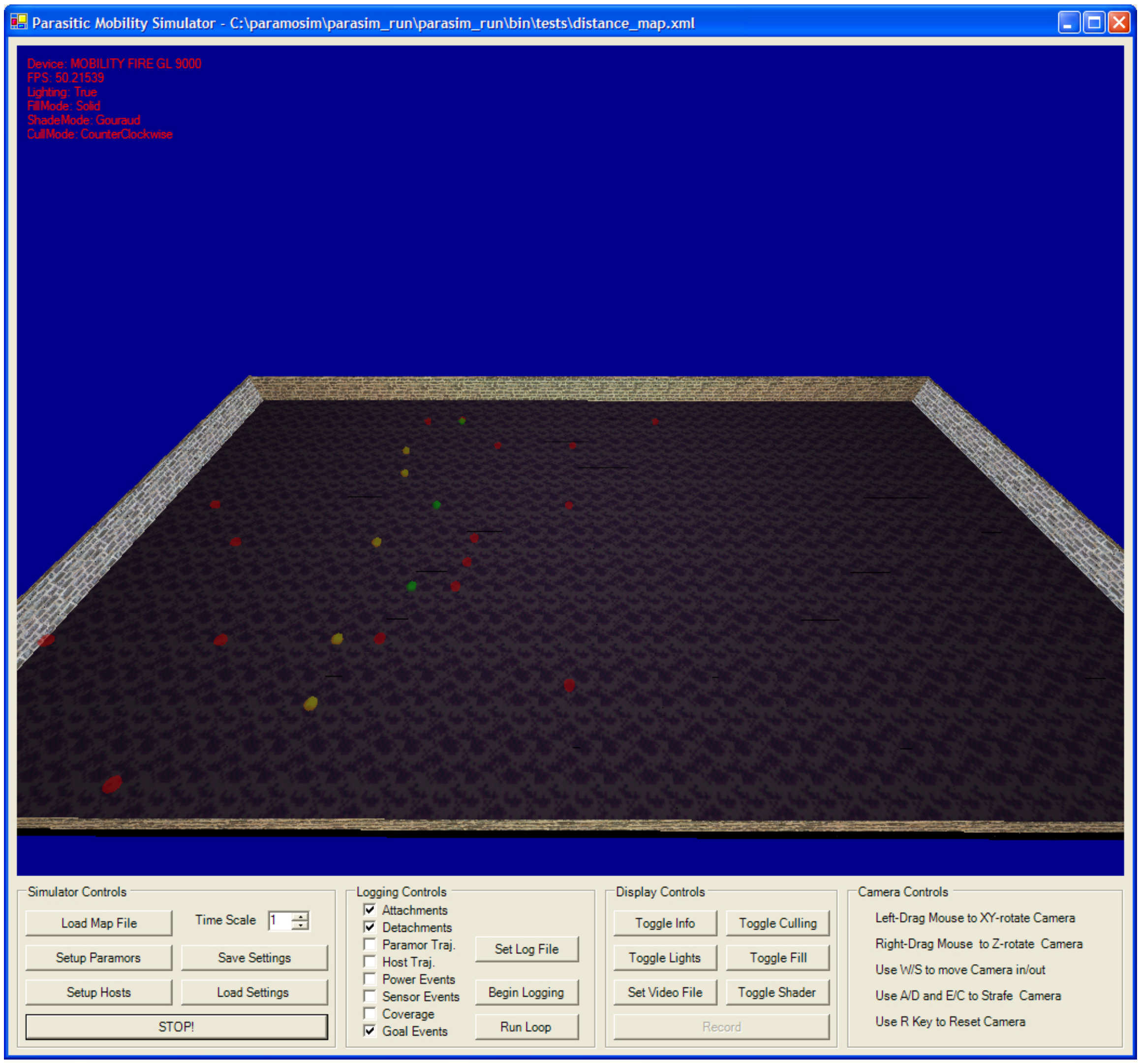
The created map is saved as an XML file. XML was chosen because it is easily readable by both humans and machines. As a result, maps can be created and modified without the use of the graphical map editor. It is very easy to set up a multi-pass simulation where the map is changed with each pass, using XML parsing tools now standard with every major operating system.

```
<?xml version="1.0"?>
<!--ParaSim Map File-->
<!--Filename:
C:\paramosim\parasim_run\parasim_run\bin\tests\distance_map_long2.xml-->
<!--Created: 5/17/2004 9:18:47 PM-->
<Map width="51" height="10">

                                    . . .

  <Tile X="3" Y="1">
    <WallN>False</WallN>
    <WallE>False</WallE>
    <WallS>False</WallS>
    <WallW>False</WallW>
    <Portal>False</Portal>
    <Paramors num="0" />
    <Hosts num="1">
      <B0>Behavior 1</B0>
    </Hosts>
    <Power>0</Power>
    <Temperature>0</Temperature>
    <Light>0</Light>
    <Altitude>0</Altitude>
    <Vibration>0</Vibration>
    <Radiation>0</Radiation>
    <Texture>0</Texture>
    <HostTraffic>100</HostTraffic>
  </Tile>

                                    . . .

  </Map>
```

Table 3-1: Example snippet of XML file created by the map editor

The complete code listing of the map editor is including in the appendices section.

Once a map file is created, it can then be imported into the Parasitic Mobility Simulator.


Parasitic Mobility Simulator - C:\paramosim\parasim_run\parasim_run\bin\tests\distance_map.xml

```
Device: MOBILITY FIRE GL 9000
FPS: 50.21539
Lighting: True
FillMode: Solid
ShadeMode: Gouraud
CullMode: CounterClockwise
```

Time Scale: 1

Logging Controls:
- [✓] Attachments
- [✓] Detachments
- [ ] Paramor Traj.
- [ ] Host Traj.
- [ ] Power Events
- [ ] Sensor Events
- [ ] Coverage
- [✓] Goal Events

Camera Controls:
Left-Drag Mouse to XY-rotate Camera
Right-Drag Mouse to Z-rotate Camera
Use W/S to move Camera in/out
Use A/D and E/C to Strafe Camera
Use R Key to Reset Camera

Figure 3-6: Screenshot of the Parasitic Mobility Simulator executing a simulation of a small, plain map containing only hosts and paramors. The red blobs are mobile hosts, the yellow blobs are stationary paramors waiting for a host to come by, the green blobs are hosts at a goal destination, and orange blobs are mobile hosts with a paramor hitching a ride. Other possible colors are black blobs indicating dead paramors that have run out of power, and white blobs indicating a node that is sensing or charging its battery.

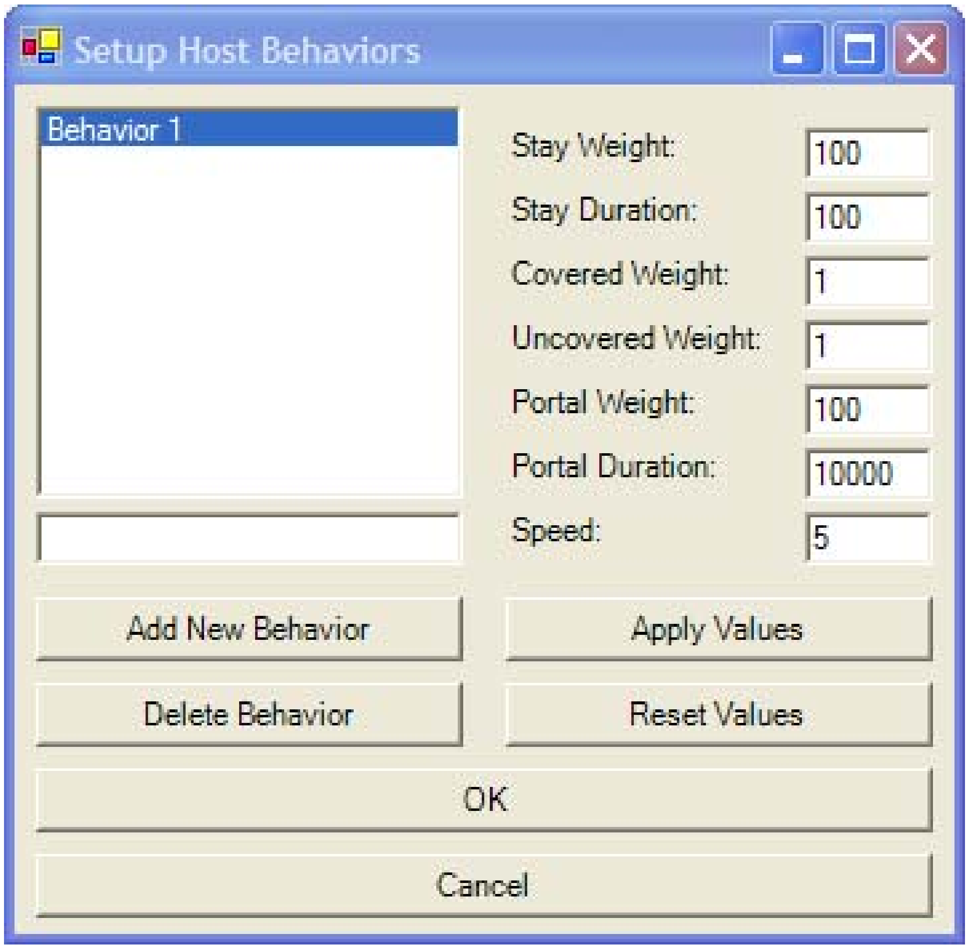
Once the map is loaded into the simulator, a user can set up behaviors for the hosts and paramors, set up logging, and begin execution of the simulation. The simulation is displayed with full animation in real or time-scaled real time. The decision to implement this in 3D although it is a 2D problem came from a desire to take advantage of the timing system available in the 3D coprocessor as well as to offload the graphical tasks to this coprocessor. It also allows control of the camera to view the simulation from any angle and from any distance. The camera can be made to follow a particular paramor or host, and entertaining videos can be captured.

3.1.2 Host Behavior Simulation

In order to accurately simulate parasitic nodes, credible hosts must be designed. It is also necessary to be able to adjust the hosts to simulate different types of real world entities.

At the lowest level, a host body needs to randomly wander through the environment. It needs to avoid obstacles and heed other attributes of the geography. The simplest host will stand at a location, list all the possibilities for a new location to travel to, and randomly select one with the randomness weighted by the environmental parameters. For example, if a host is at a location with walls on two sides (those directions have a weight of 0), a new location with a host frequency of 50 on one side, and one with a host frequency of 100 for the final side, it will randomly select between the two sides with a weighting of 2 to 1 likelihood in favor of the side with the 100.

Although this simulation can use just this simple model, a large number of other parameters are available to the Parasitic Mobility Simulator for more detailed host behavior simulation.


Setup Host Behaviors

Behavior 1

Stay Weight: 100
Stay Duration: 100
Covered Weight: 1
Uncovered Weight: 1
Portal Weight: 100
Portal Duration: 10000
Speed: 5

Figure 3-7: Popup window showing the parameters settable on a host-by-host basis

The window shown in Figure 3-7 appears after hitting the “Setup Hosts” button from the main control panel shown in Figure 3-6. This panel shows the rest of the parameters that can be assigned to a host. On the left side, the menu will list all the behavior names assigned to hosts from the map editor. By selecting a behavior, the user can edit the parameters for the hosts that have that behavior assigned to them. These parameters are:

- **Stay Weight** – This is the likelihood that the node will stay in its current location instead of heading towards one of the adjacent locations.

- **Stay Duration** – If a node has decided to stay in its current location this is the duration of time it will stay before repeating the selection process to identify its next move. By use of the Stay Weight and Stay Duration parameters, hosts can be designed that move often or hardly move at all.

- **Covered/Uncovered Weights –** The hosts keep a record of everywhere they have been. These weights modify the weighting from the environment based on how often a host has visited the locations it is deciding between. These parameters allow the design of hosts that have a tendency to always follow a known course or are more likely to explore uncharted areas.

- **Portal Weight –** This parameter denotes the likelihood that a host will choose to leave through a portal when it finds one at its current location.

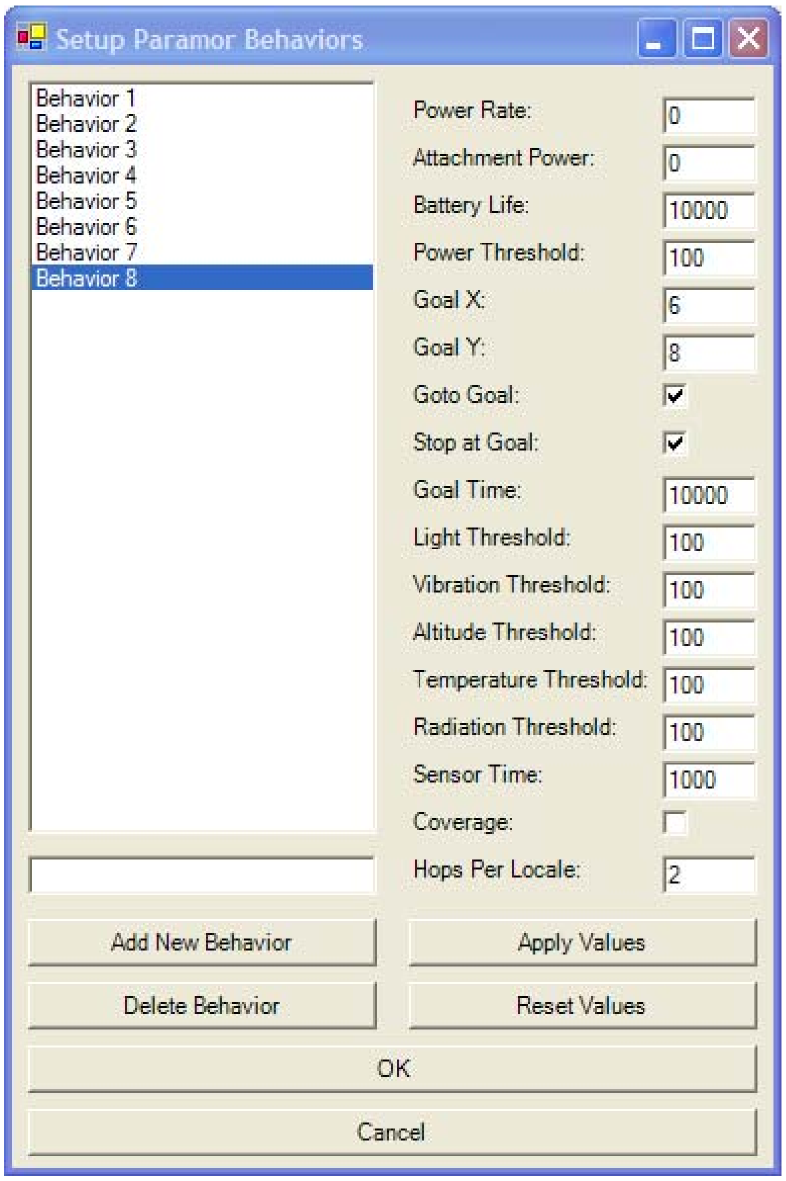
- **Portal Duration** -- This is the length of time a host that has left through a portal will wait before reemerging from the same or a different portal.

- **Speed –** This is a floating point number that sets the host's speed in simulator distance units per 1000 simulator time units. These units can be related to any value in the real world, provided the speed values that relate time to distance fit the scaling of the simulator units to real world units.

The execution of the host's behavior is fairly straightforward. When a host reaches a new destination, it identifies the possible choices for its next destination, and using the weightings for these choices, it randomly selects its next move. By setting these parameters, hosts can be created that have high levels of randomness or hosts can be created that have no randomness and follow a specific pattern. This implementation allows the simulation of most environments populated with mobile hosts, such as cars, people, and animals.

3.1.3 Parasitic Node Behavior Simulation

The basics behind a parasitically mobile node's behavior are a set of objectives for the node. When a node is idle and a host body comes in range of it, the node attaches to the host. While attached to the host, the node uses the information it can gain from the environment and host to determine if detaching will help it reach its objectives.



Power Rate:	0
Attachment Power:	0
Battery Life:	10000
Power Threshold:	100
Goal X:	6
Goal Y:	8
Goto Goal:	✓
Stop at Goal:	✓
Goal Time:	10000
Light Threshold:	100
Vibration Threshold:	100
Altitude Threshold:	100
Temperature Threshold:	100
Radiation Threshold:	100
Sensor Time:	1000
Coverage:	☐
Hops Per Locale:	2

Figure 3-8: Popup window showing the parameters for paramor nodes

Similar to the host behavior setup, the panel shown in Figure 3-8 allows the assignment of objective and behavior parameters to each of the named behaviors assigned to paramor nodes in the map editor. In more detail, the parameters are:

- **Power Rate –** This is the amount of power the node uses per 1000 units of simulator time. Currently, the simulator supports only a steady rate of power usage regardless of whether the node is sensing, traveling, or waiting. However, it can log the amount of time spent in each of these states allowing complete power calculations to be made after the simulation is complete. This is illustrated in Section 3.2.2.

- **Attachment Power –** This is the amount of power used for each attachment or detachment the node performs.

- **Battery Life –** This is amount of power the node has available. When this runs to zero, the node dies.

- **Power Threshold –** When the node's power level drops below this threshold, it enters a mode searching for power sources. In this mode, it will always detach if near a source of power.

- **Goal X/Y –** The coordinates that the node is told to head towards.

- **Goto Goal –** If this is checked, the node will try and reach the location stored in Goal X/Y. If it is unchecked, the value stored in Goal X/Y is ignored.

- **Stop at Goal –** If this is checked, when the node reaches the location stored in Goal X/Y, it will remain there indefinitely.

- **Goal Time –** This is the duration a node will stay at its goal if Stop at Goal is not checked. After the time is up, it will attach to the next host that comes by and start looking for a new area of interest.

- **Light/Vibration/Altitude/Temperature/Radiation Threshold –** These values tell the node what constitutes an area of interest. If the area contains quantities of these elements above the threshold value, it considers it interesting and will detach to start sensing. Setting these values to 100 disables checking for that element as values over 100 are not available in the map editor.

- **Sensor Time –** This is the duration that a node will stay at a sensor point of interest before trying to seek a new location.

- **Coverage –** If this is checked, the node will take into consideration where it has already been when deciding to remain attached or to detach. This is useful for applications that are looking for sensor points of interest, or trying to cover an entire area for reconnaissance.

- **Hops Per Locale –** When a node attaches to a new host and determines that the host is taking it in an undesirable direction, the node has the ability to hop off. However, it is possible that the node is at a spot where the only route that the host can take it is undesirable. The Hops per Locale setting assigns the maximum number of detachments the node should perform before it should stay on the host and ride to a new location. At this new location it can begin the process over again.

Once the above parameters are set up, the paramor behavior is easily implemented in the simulation environment. When a host comes within range, the paramor attaches. If it comes across an area of interest it will hop off and remain there for the specified duration. The power calculations are constantly being updated, and when it crosses the power threshold the node will not detach at a sensor point, only at a power location. It is possible to change these priorities. For example, if it is okay to risk running out of power in order to find something, the power can be lowered in priority below sensor events or destinations.

In general, if the node has a set destination, it will try to reach it. If it comes across an area of sensor interest, it will detach, stay for the sensor duration, and then try to continue towards its destination.

If a node is on a path to a destination or trying to go to where is has not been, it needs to hop off when it is on a host that is taking it farther away from its destination, or back into a covered area. This is a matter of simply calculating the distance to the destination at two points on the trajectory and testing the change in distance to the destination, or figuring out the direction and comparing it to the node's stored coverage map. It is important to allow the node to travel a distance long enough to sense the direction before hopping off. The "hops per locale" parameter is used to prevent situations with nodes being stuck as described above.

3.1.4 Final Simulator Design Notes

After all the parameters are set up, the simulation can be executed. Behind the scenes, the simulator takes 1ms for each simulation time unit. This unit can be scaled to any real world time unit provided that the time dependent values of host speed, power rate, goal time, sense time, portal duration, and stay duration are scaled similarly. To make the simulator execute faster, the amount of simulation time units that complete in 1ms can be increased. The distances can be dealt with similarly, scaling the distance dependent parameter of host speed and taking into account the distance metric when setting up behaviors.

Besides the ability to record the simulator into a video file, extensive logging capabilities are implemented. It is possible to log all the activities that happen to the hosts and paramors. The parameters are shown in Figure 3-9 and are pretty straightforward. Each

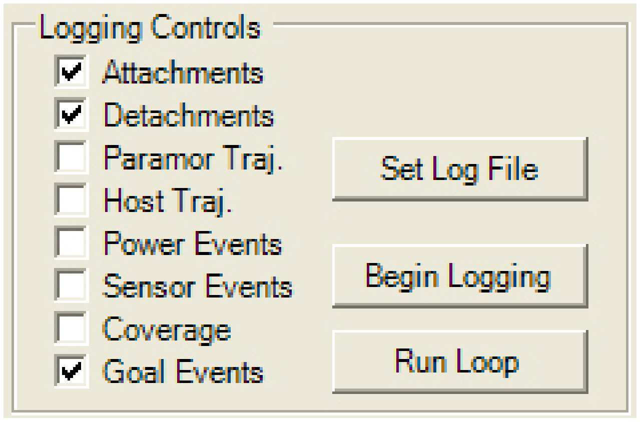

Logging Controls
[✓] Attachments
[✓] Detachments
[ ] Paramor Traj.
[ ] Host Traj.
[ ] Power Events
[ ] Sensor Events
[ ] Coverage
[✓] Goal Events

Figure 3-9: Logging controls and Run Loop Button

event is time-stamped in the log file. The host and paramor trajectory logging routine records every movement they make and the data can be reconstructed to map out all activity. The host trajectory section of the software places log results into large log files, and is mainly useful in debugging the host behaviors. The log files are recorded in XML format for easy parsing, as shown in Table 3-2.

```
<?xml version="1.0"?>
<!--ParaSim Log File-->
<!--Filename: C:\paramosim\parasim_run\parasim_run\bin\tests2\distance_log4.xml-->
<!--Created: 5/18/2004 3:14:51 AM-->
<Log>
  <LoggingStarted TimeScale="1">103613478</LoggingStarted>
  <ParamorAttachment Paramor="1" Host="6" X="5" Y="2">103616322</ParamorAttachment>
  <ParamorAttachment Paramor="5" Host="36" X="5" Y="6">103616632</ParamorAttachment>
  <ParamorAttachment Paramor="7" Host="43" X="5" Y="8">103616773</ParamorAttachment>
                                                     . . .
  <ParamorAttachment Paramor="0" Host="0" X="5" Y="1">103630242</ParamorAttachment>
  <ParamorDetachment Paramor="4" EventType="AwayFromGoal" X="6" Y="5">103630552</ParamorDetachment>
  <ParamorDetachment Paramor="2" EventType="Goal" X="9" Y="3">103630693</ParamorDetachment>
  <ParamorGoalEvent Paramor="2" X="9" Y="3">103630693</ParamorGoalEvent>
                                                     . . .
  <ParamorAttachment Paramor="0" Host="31" X="8" Y="1">103743415</ParamorAttachment>
  <ParamorDetachment Paramor="0" EventType="Goal" X="9" Y="1">103744146</ParamorDetachment>
  <ParamorGoalEvent Paramor="0" X="9" Y="1">103744146</ParamorGoalEvent>
  <LogStopped>103747641</LogStopped>
</Log>
```

Table 3-2: Example snippet of XML log file

The “Run Loop” button shown in Figure 3-9 will execute a block of code that sets up exit conditions for the simulator, such as when all the paramor nodes have reached their destinations or the entire map has been covered by the nodes collectively. It then runs the simulation multiple times, making defined parametric changes with each pass.

3.2 Simulator Data and Results

With the appropriate parameters, the package can simulate many environments and help test out algorithms and ascertain the requirements for a particular parasitically mobile sensor network. It can also be used to generate numerical data for predicting behavior, such as how long it will take a node to reach a location according to the hosts in the environment, the power usage compared to standard robotic devices, and how many nodes should be deployed to cover an area in a particular amount of time. This section presents these types of data collected from thousands of hours of simulation time.

3.2.1 Velocity Data

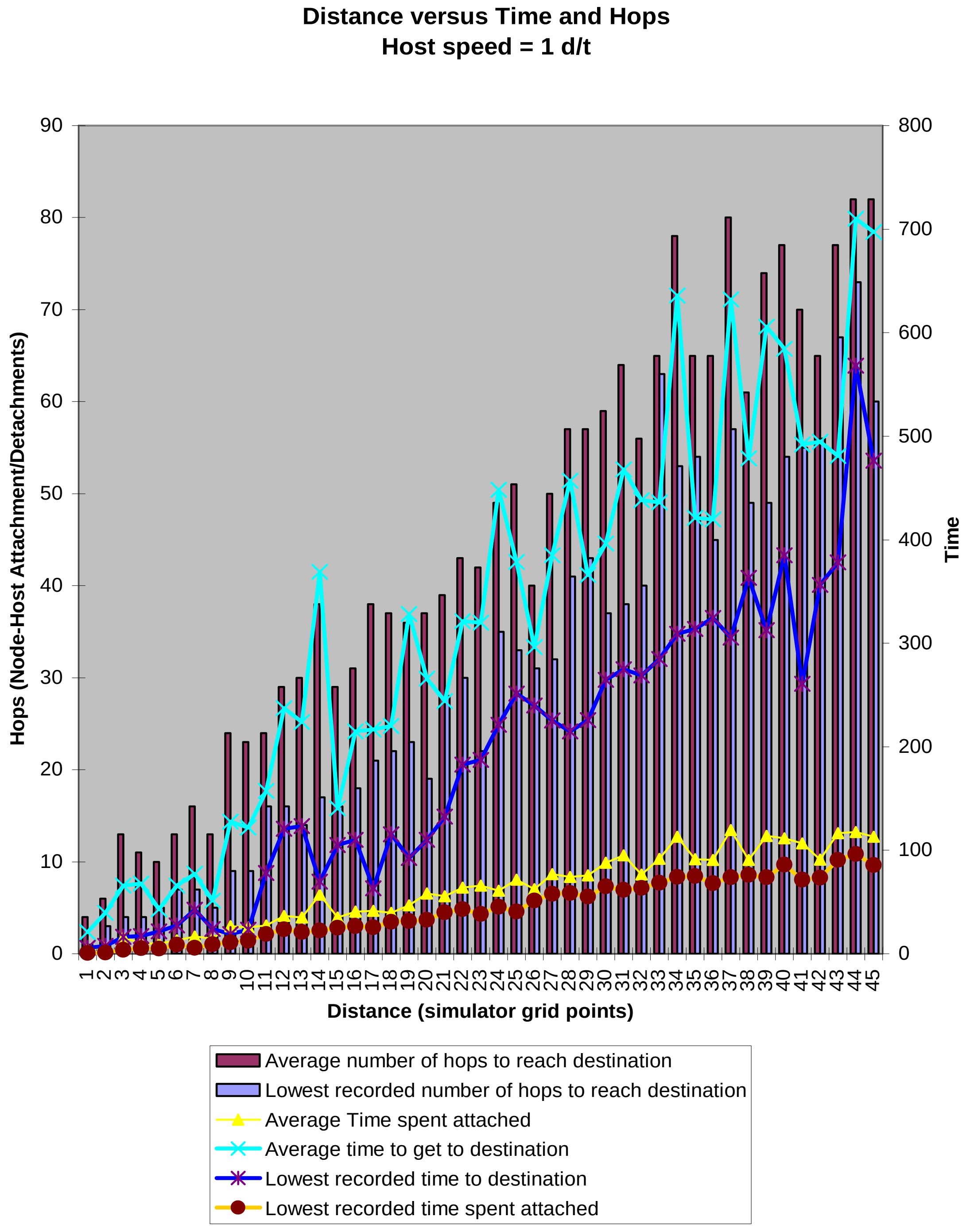


Figure 3-10: Graph showing distance versus time data collected and averaged from repeated simulation passes.

The data in the graph shown in Figure 3-10 results from simulating an environment with a constant size and host population. The speed of the hosts is also a constant 1 unit of distance per unit of time. The simulation parameters can be mapped to any units provided everything is scaled appropriately. In the next section, this data is used to calculate real-world energy usage values, and the simulator distance units are related to meters and the timing units are related to seconds. This would mean the hosts travel with a speed of 1 m/s, similar to the walking pace of a human.

The paramor behavior chosen for this test is simply to go to a particular location at a known distance away. The algorithm for attachment and detachment is to attach to every host that comes by, decide whether it is bringing the node closer to or farther away from the destination. For a real device to be able to ascertain this information, it must ride the host for long enough to get a fix on the motion or the new location. Based on the GPS and Bluetooth localization systems described in the next chapter, this was set in the simulator as 1/2 of a time unit before the node knows the new location with a resolution of 1/2 of a distance unit.

This simulation was executed 25 times for each distance and the results were averaged. The linearity of this graph is due to the host's random behavior with respect to the node's destination. In other words, at each point (a point being at the end of the node's minimum cycle required to ascertain the direction that the host is traveling) the host is just as likely to turn away from the node's destination as it is to continue moving towards it. Therefore, the average time it takes for a host to take you from one location to the next location that is closer to the destination is the same regardless of how you arrived at the current location. Since the host reassesses its path at each point, the average time it takes a node to find a host going one step closer to the destination and to ride it to the next point, is constant over the entire travel. Hence, the average time it takes to go N number of steps should be N times the average time it takes to go one step, leading to a linear relationship. A simulation that uses hosts that behave less randomly, as they might in a real world situation where the hosts are governed by pathways and destinations, is discussed in Section 3.2.3.

Besides the total time it took to reach a destination, the graph also shows the time spent attached to a host, in other words, the time spent non-idle and actually traveling. This line is quite smooth; especially in comparison to the total time including time spent idly waiting for a host to pick it up. This smoothness shows that the algorithm is working properly as most of the time is spent waiting for a beneficial host, which varies according to the random flows of the hosts. This randomness is eliminated by the paramor's decision process. The reason that the attached time and distance don't exactly scale with the host velocity is because the nodes still need to attach to the host to ascertain its direction, and even hosts that head towards the destination are not guaranteed to go exactly straight, especially considering the coarse location system of the node.

The graph also contains the number of hops, a complete cycle of attach and detach, which, along with the total time and attached time will be used to calculate the energy consumption in Section 3.2.2.

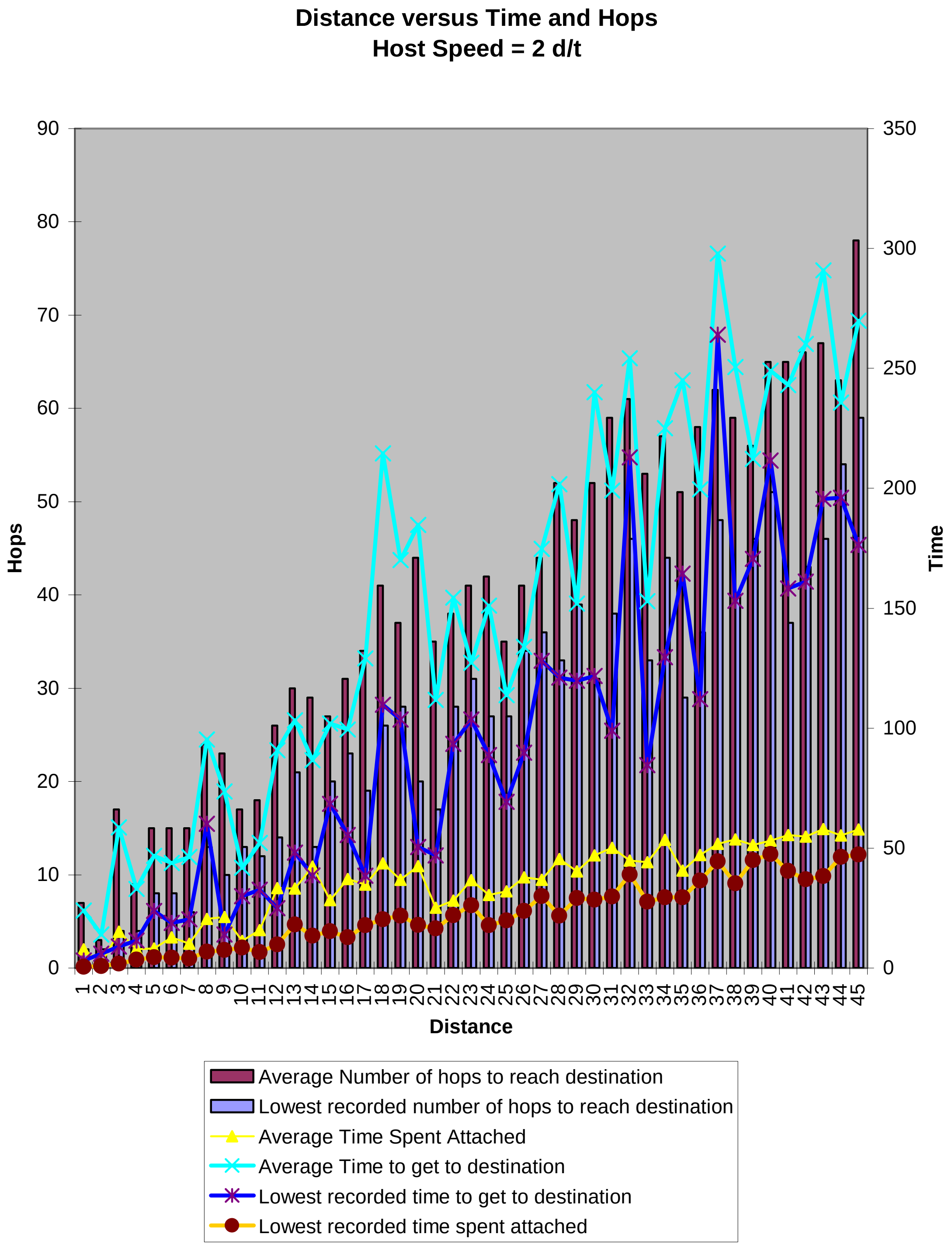


Figure 3-11: Graph showing distance versus time data collected and averaged from repeated simulation passes. This run uses hosts that move at a speed of 2 d/t.

The graph shown in Figure 3-11, repeats the same test as the graph shown in Figure 3-10, but with an environment consisting of hosts that move at a speed of 2 d/t, twice the speed of the first test. It shows that the overall node velocity is almost exactly proportional to the speed of the hosts, in an environment where the hosts move fairly randomly but with uniform distribution.

The line illustrating the time spent traveling in this second test is slightly less smooth than the same line in the first test. This is due to the increased speed causing the node to travel further off course before it can sense the direction it is traveling and detach from an inhospitable node. This is discussed further with the data in Figure 3-12.

Also shown on the graphs in Figure 3-10 and Figure 3-11 are the lowest recorded total times, lowest recorded attached time, and lowest recorded number of hops. Due to the proposed inexpensiveness of parasitically mobile sensor nodes, redundancy may be utilized. One hundred nodes can be deployed in situations where only one needs to reach a destination. In this case, it is more than likely that the first node will be there in a shorter amount of time than the average time of all the nodes.

Several additional runs, each with different host speeds, were executed. From these runs, average node velocity versus host velocity data was collected and is shown in Figure 3-12.

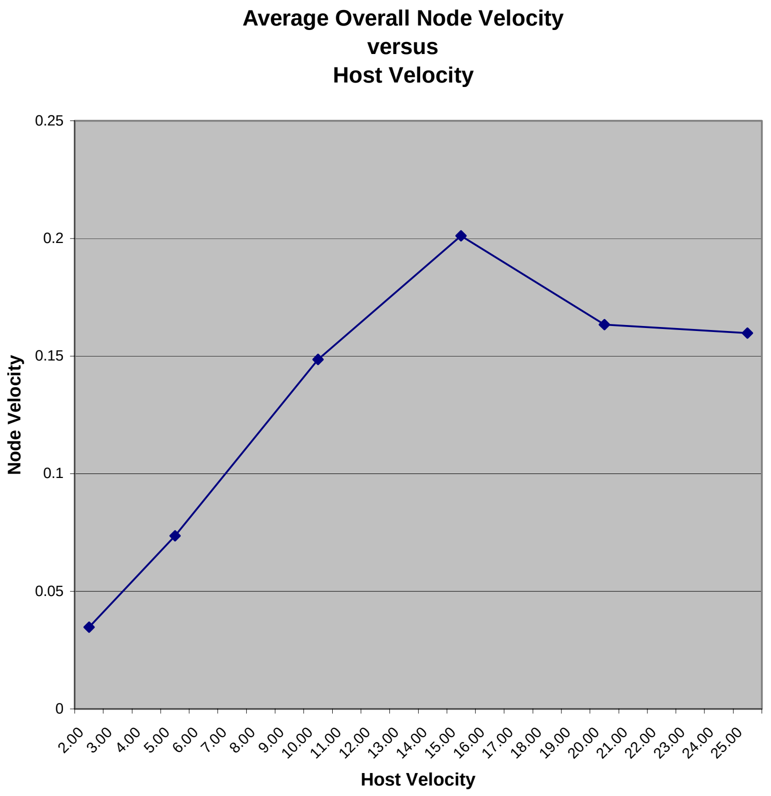


Figure 3-12: Host Velocity versus Node Velocity

The beginning of the curve looks as expected, a linear relationship between host velocity and node velocity. But the second half of the curve looks quite strange at first glance. The node velocity peaks at around a host velocity of 15 d/t and then drops. This is where the resolution of the location sensing system becomes a more serious factor. When the host speed increases, it takes the attached node farther off course in the time that the node needs to sense its trajectory. At some point, the algorithm is rendered completely useless and the nodes attachment and detachment becomes completely random. In the simulator, the time needed to sense the trajectory can be turned very small, or even eliminated, by allowing the nodes to sense the host's direction before attaching. But this will definitely be an issue when designing real mobile systems where in all but very special cases (such as scheduled vehicular systems like trains) the node will have to attach to the host to find out where it is going.

The graph shown in Figure 3-13 displays the results of a similar simulation, except that in this run the distance to the destination is kept constant and the host frequency surrounding the destination is altered. The host frequency was described in the prior section under environmental simulation.

The attached time in this test is fairly constant, whereas the total time varies greatly in proportion to the host frequency. This shows that the node is spending an increasing amount of time waiting for a proper host as the frequency of such hosts goes down. The large discrepancy between the quickest node and the average is further justification for the redundancy allotted from cheap sensor nodes. Low host frequencies can be combated with the deployment of enough nodes to guarantee that one will find a host headed for the destination regardless of how rare it is.

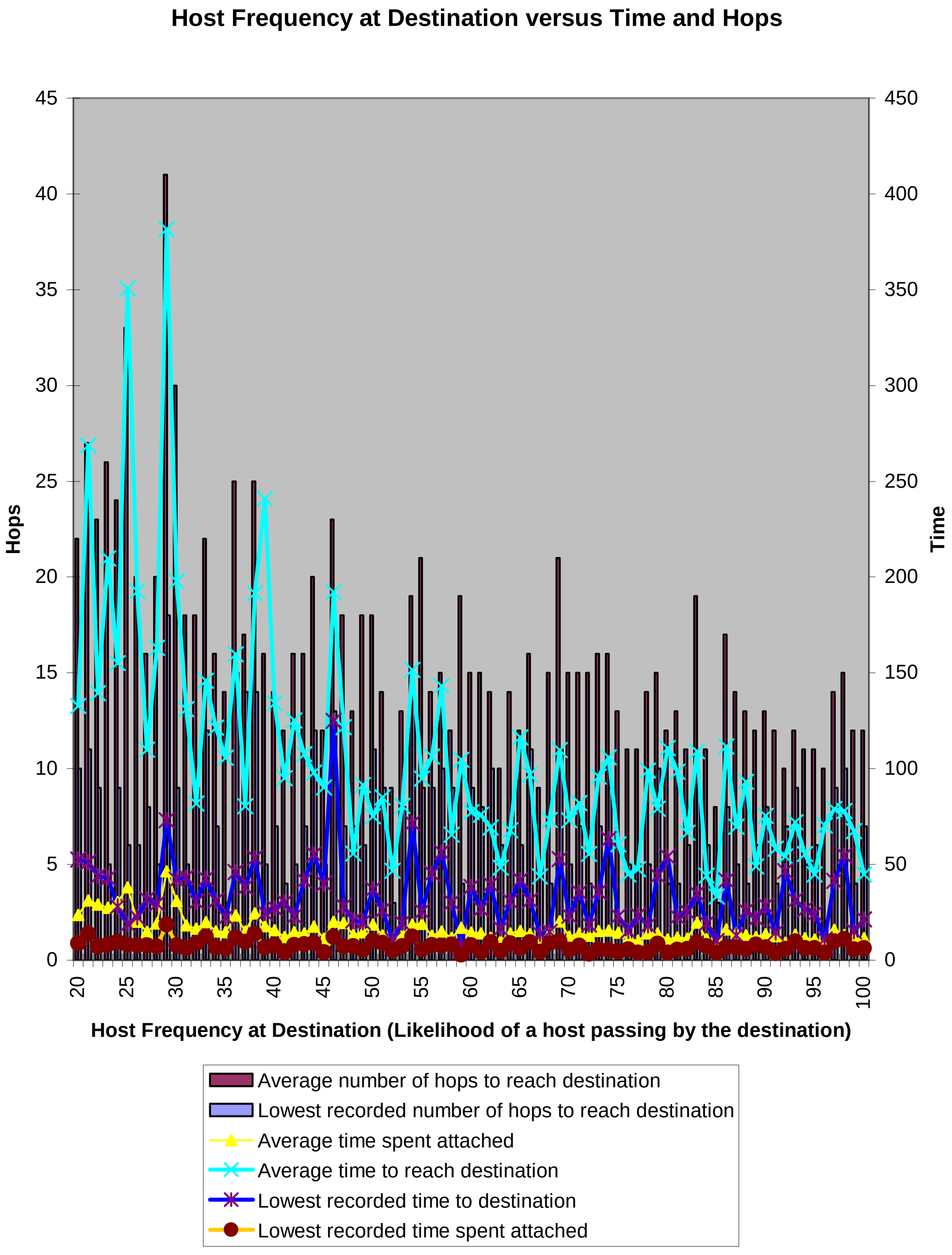


Figure 3-13: This graph shows the results of the simulation where the distance is kept constant and the host frequency around the destination is varied.

Of particular interest to mobile sensor networks, is the ability to release nodes without a specific destination and have them attempt to scan over the entire area. In order to simulate this behavior, a new algorithm for attachment and detachment had to be implemented. The simulator allows quick trial and error of such algorithmic ideas.

The first component of the algorithm is to equip the nodes with the ability to record where they have been. When a host takes them back to a location they have already covered, they detach. This proved to be inadequate as the nodes quickly found themselves surrounded by places they had already covered up to the radius of their ability to sense where the host was taking them. Depending on the processing ability of the node, it may be possible to analyze the entire map of coverage and determine general desired directions even if it the node first must travel through an already covered area.

After experimenting with several behaviors, it appears that the key to coverage is to keep moving even if you might be heading in a direction that has a area in the immediate vicinity that the node has already visited. The chosen algorithm for this simulation is to limit discarded hosts (hosts that are heading to a location already visited) to one per location. In other words, when a host comes by an idle node, the node will attach, and decides if the host will take it to an unvisited location. If not, the node detaches and waits for the next host. This time it will take the new host without question and ride it until it finds an uncovered location, or at least arrives at a location that has nearby unvisited locations so it can detach and have a high likelihood of a host coming by that will go in that direction. It is also important not to attach to a host that already has a paramor attached to it. If two nodes are on the same path, both looking to cover the environment, they will most likely remain together by making the same decisions. Simple broadcast commands sent from individual nodes can aid with the dispersal of mobile sensor nodes. These commands can tell other nodes which areas have been covered and areas of high host traffic.

The graph in Figure 3-14 shows the timing results of the coverage simulations. The environment was set up as a 200 square unit area with a coverage map resolved in 1 unit squares. The test was run with sets of 5, 10, 15, and 20 node deployments.

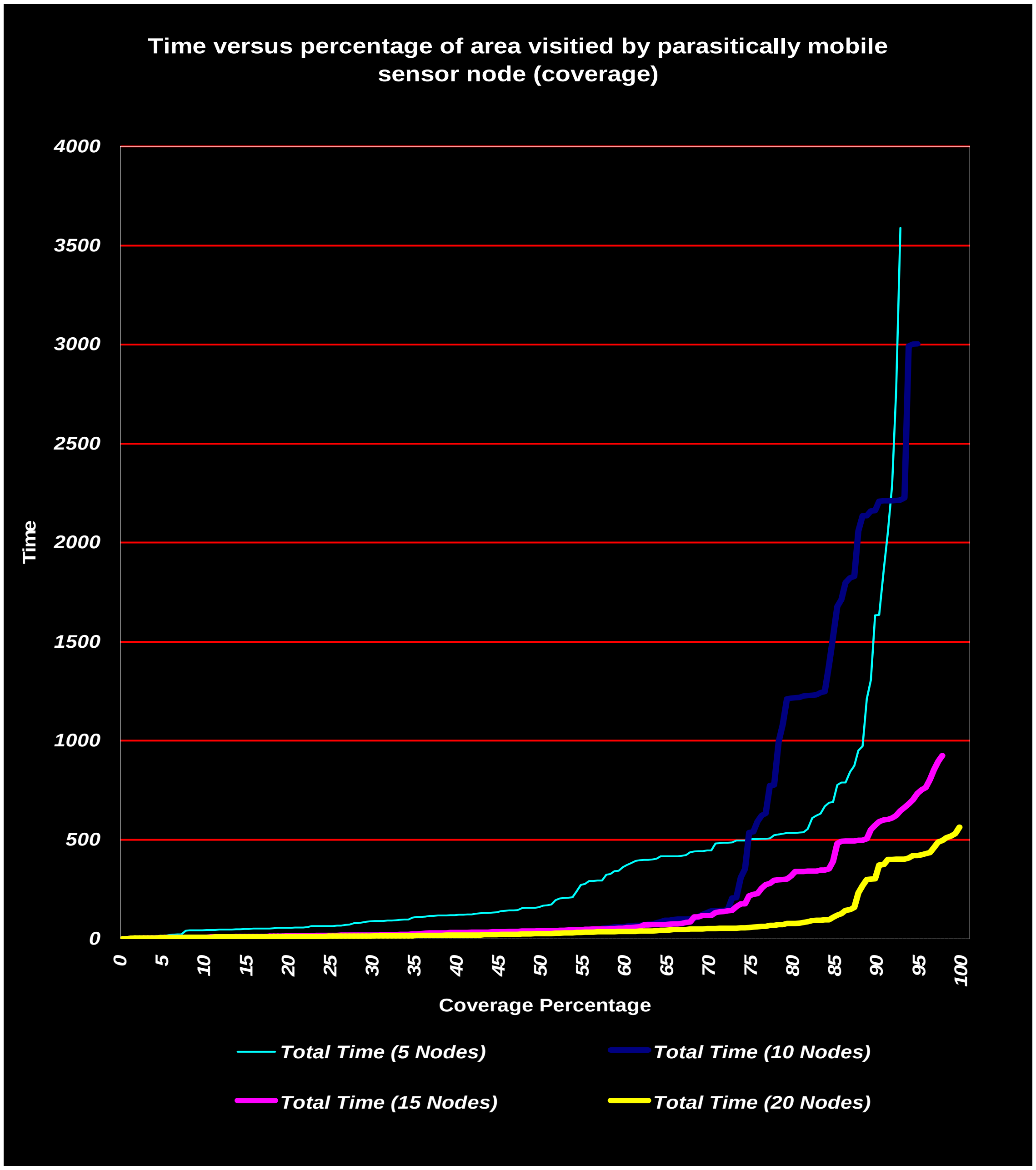


Figure 3-14: Graph showing the time to cover an area for different quantities of deployed nodes

The results are fairly straightforward. A large percentage of the area is covered fairly quickly. The last ten percent, usually comprised of unvisited locations surround by covered areas, takes most of the time and is completely proportional to the number of nodes deployed. This behavior is quite similar to most mobile robotic systems seeking to cover an area, such as the system designed by Maxim A. Batalin and Gaurav A. Sukhatme [34] from the University of Southern California’s Robotic Embedded Systems Laboratory. Their system evaluates algorithms for mobile robots deploying sensors intended to maximize sensor coverage area. The data of the coverage area versus number of deployed robots for several of their algorithms is very similar to that for the parasitic mobility simulation, further identifying parasitic mobility as a potential replacement for standard means of mobility.

3.2.2 Energy Usage Calculations and Comparison

By examining the hops, attached/traveling times, and wait times from the simulator as described in section 3.2.1, we can calculate predicted values for the energy consumption rates of a parasitically mobile sensor node. In this section, we introduce the two kinds of nodes designed as the hardware components of this research and compare their predicted power usage statistics to that of two standard mobile robotic sensor devices.

3.2.2.1 Semi-passive Parasitic Node Power Calculations

The first device designed for this experiment is a 1 cubic inch device mimicking the passive attachment mechanism of a bur with the ability to actively detach by shaking itself loose. This device is further detailed in Chapter 4.

Since the attachment is passive and it sticks to every nearby host (bur-like attachment), it requires no additional power, actuation, or sensing during the host discovery and attachment process. When it is idle and waiting for a host, it can remain in a low-power mode and wake up on motion caused by being picked up by a host as defined in Chapter

4. The low power mode runs using a 32 KHz clock and keeps alive a comparator on accelerometer data. This low power mode draws around 35 uA. Additionally, the node will wake up once per second and check for a wireless message. This check lasts around 10 ms and uses 16 mA for that duration. However, the power for the communication system will not be included in these calculations because, at this point, we are just looking at the power needed for mobility to compare to standard techniques of moving sensors.

While attached, the semi-passive node can enter a different low-power mode and periodically wake up to check its location, progress, and sense the new surroundings. Assuming that the location system has a resolution of one meter, in the simulation of the environment with the hosts that move at 1m/s, the device will have to wake up and sense once per second of attached time. Depending on the type of location system and sensors, the node could use up to 60 mA for up to 60 ms for gathering data about the location and conditions surrounding it. Two location systems are described in Chapter 4, both systems require less power than this estimate.

And finally, when the node decides that it is time to detach, it needs to activate a detachment mechanism, which in the case of these sticky nodes is a pager motor with a draw of 15mA activated for 500ms to shake loose.

So the power usage of the proposed semi-passive node can be defined in three parts as follows:

- $PowerUsedPerHop = 15 \times 500\left(\frac{1}{1000}\right)\left(\frac{1}{1000}\right) \times 3.3 = 0.0245 \text{ Joules/hop}$

- $AttachedPowerRate = 60 \times 60\left(\frac{1}{1000}\right)\left(\frac{1}{1000}\right) \times 3.3 = 0.0119 \text{ Joules/s}$

- $DetachedPowerRate = 35\left(\frac{1}{1000}\right) \times \left(\frac{1}{1000}\right) \times 3.3 = 0.00012 \text{ Joules/s}$

Using these formulas and the data shown in Figure 3-10, time and hops versus distance, we can graph power used versus distance. Figure 3-15 at the end of this section shows this calculation.

3.2.2.2 Active Node Power Calculations

In addition to the semi-passive nodes, an active node was design that adds an actuated method for attachment as well as detachment. Detailed design information on this “tick” modeled node is provided in Chapter 4.

The active node power calculations are fairly similar to the semi-passive node. The active node requires 30 seconds drawing 10mA to wind the spring and execute an attachment or detachment. This node hops at a height of around 6 cm and weighs around 40 grams. If the device were 100% efficient the power usage per hop could be calculated as:

$$Energy = m \times g \times h = 0.040 \times 9.81 \times 0.060 = 0.024 \;\; Joules$$

The above calculation is for a single jump; two jumps are required for a complete attachment/detachment cycle. Hence, the calculated value of 0.048 Joules per hop is substantially less than the observed value of 1.98 Joules per hop calculated below. This makes sense because the hopping mechanism is far from 100% efficient. Therefore, we will use the observed values in the overall power calculations.

Additionally, when the active node is not attached it requires more power than the semi-passive node because it needs to identify and locate a potential host to attach to. This requires 30mA of constant power draw to run an IR proximity detection circuit with a fairly high sampling rate. This sampling rate can be reduced, but worst-case ratings are being used for these calculations.

The power equations for the active node are:

- $PowerUsedPerHop = 2 \times 10\left(\frac{1}{1000}\right) \times 30 \times 3.3 = 1.98$ Joules/hop
- $AttachedPowerRate = 60 \times 60\left(\frac{1}{1000}\right)\left(\frac{1}{1000}\right) \times 3.3 = 0.0119$ Joules/s
- $DetachedPowerRate = 30\left(\frac{1}{1000}\right) \times 3.3 = 0.099$ Joules/s

3.2.2.3 Power Comparison 1 – NASA Urban Reconnaissance Robot

The first robot chosen for power comparison with parasitic mobility is NASA's Urban Reconnaissance Robot. [35] This robot is equipped with an enormous array of sensors, actuators, and processing power. It is designed to navigate through very tricky environments. Parasitically mobile nodes gain this ability from the hosts they attach to, and these hosts have evolved to navigate their environment in the best way possible. The NASA robot is an ideal comparison as it is a prime example of the power needed to build a device that navigates in a way that parasitically mobile nodes potentially get for free.

This robot draws 145 Watts while moving, sensing, and navigating on flat ground at a rate of 80cm/second. It can also climb stairs using 245 Watts of power. For this comparison, we will assume it is on flat ground. Using this energy consumption rate and speed of travel we can create power versus distance data and compare it to that of the parasitic node. This comparison is shown in Figure 3-15.

3.2.2.4 Power Comparison 2 -- The RoboMOTE

On the other end of the spectrum from the NASA robot is University of Southern California’s RoboMOTE [36]. It is a small wheeled robot measuring less than 6 cubic centimeters in volume. While it does not have the navigation or actuation abilities of the NASA robot, it makes up for it in size, cost, and power consumption. The RoboMOTE exhibits many of the desirable attributes in mobile sensor networks, such as small cheap nodes that can work together. When all the features required for navigation are active, the RoboMOTE uses 1.5 Watts and can travel at a speed on 0.27 km/h.

Parasitic Mobility is an attempt to bring the navigational power of the NASA robot into a device the size and cost of the RoboMOTE. That is why these two projects were chosen as points of comparison for the power consumption of these new types of networks. The RoboMOTE power consumption values are added to the data graph in Figure 3-15.

3.2.2.5 Power Comparison Results

Figure 3-15 illustrates the results of the above formulas, simulation tests, and comparison with robot information, in the form of power with respect to distance. The power scale is shown logarithmically since the power consumption of the NASA robot is drastically much more than that of the parasitic nodes.

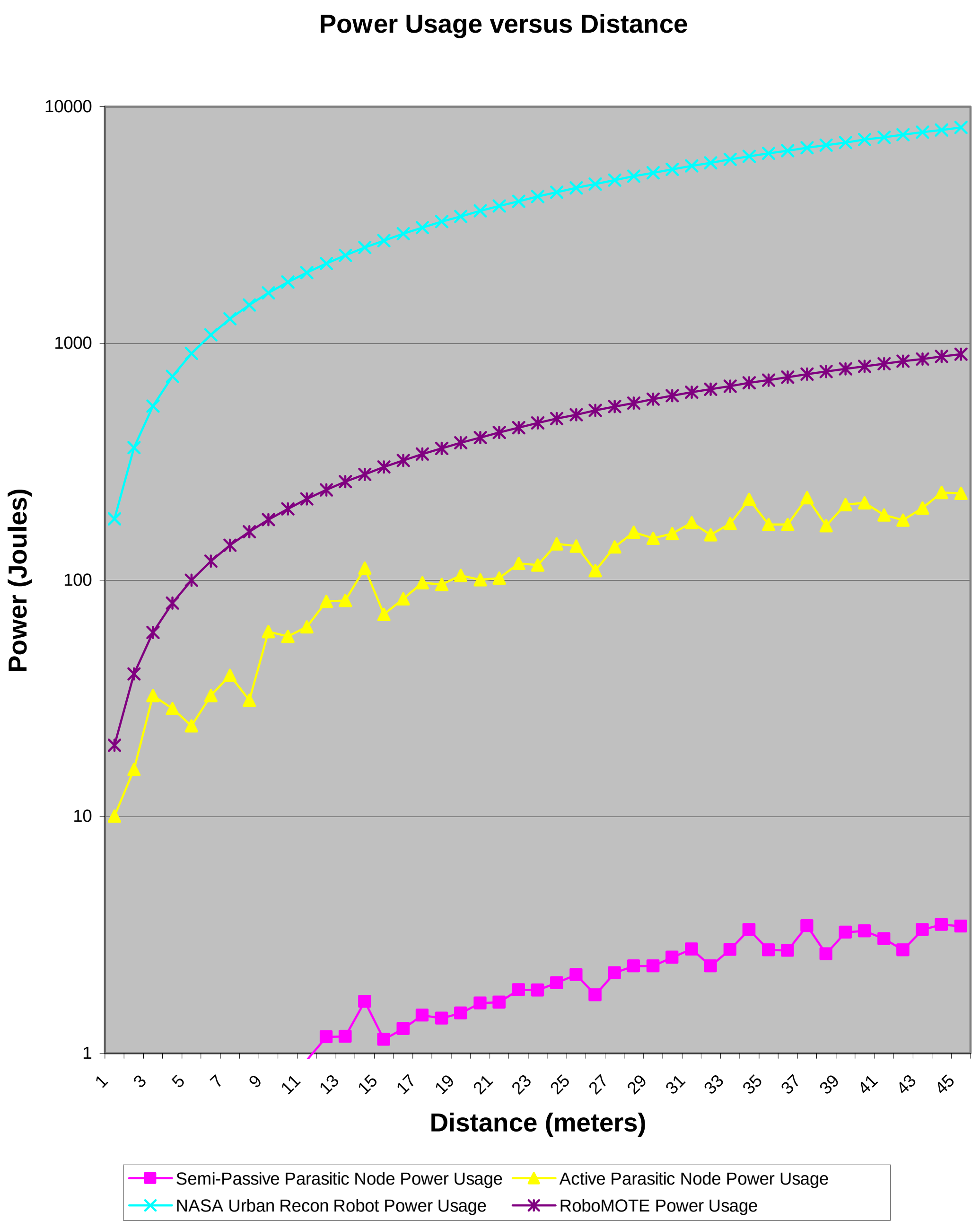
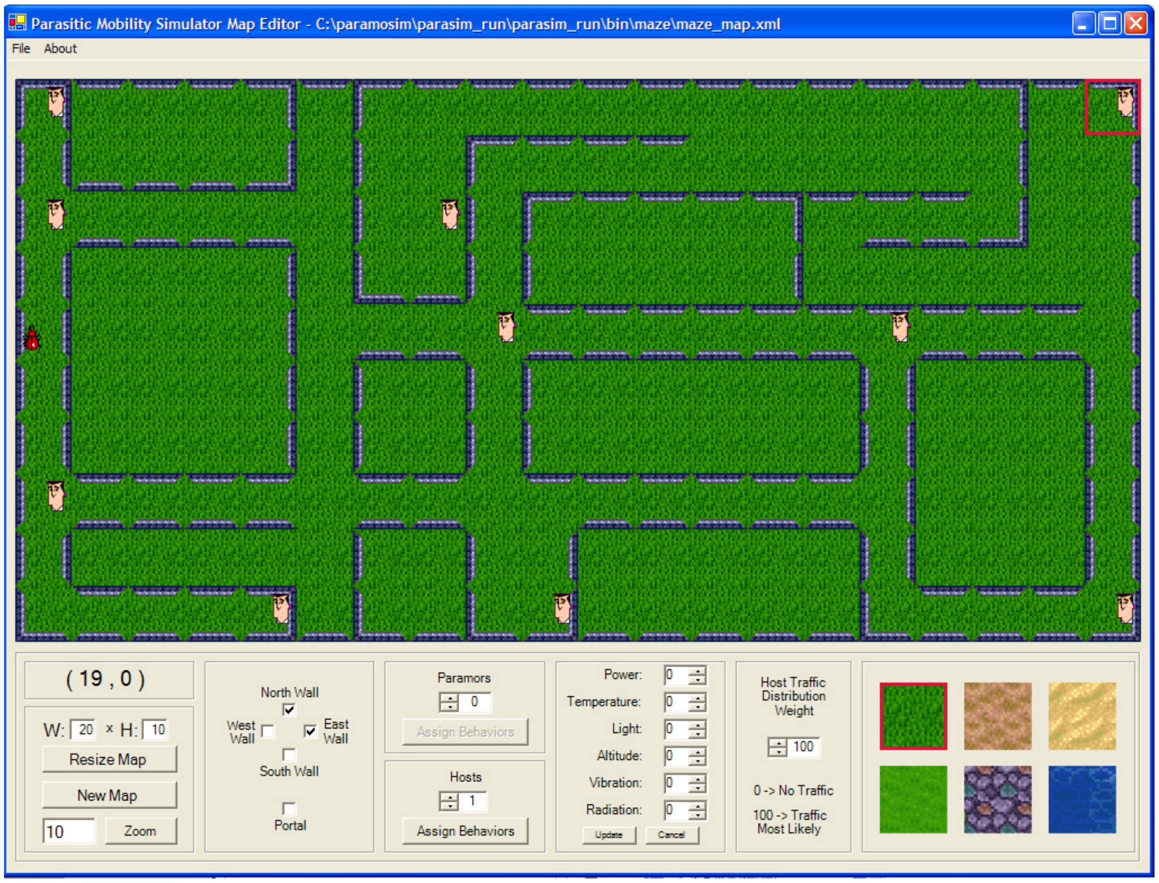


Figure 3-15: Power versus Distance of parasitic and non-parasitic mobile devices.

Figure 3-15 shows that power usage of even the small RoboMOTE with its low power actuation is close to an order of magnitude greater than that of the active node, and over 2 orders of magnitude greater than that of the semi-passive node. The NASA urban robot is another order of magnitude greater than that of the RoboMOTE. This confirms the hypothesis that parasitic mobility leads to large power savings when compared to standard mobile devices.

3.2.3 Maze Simulation

The final simulation was an attempt to simulate a real-world host behavior scenario.


Parasitic Mobility Simulator Map Editor - C:\paramosim\parasim_run\parasim_run\bin\maze\maze_map.xml
File
About
Resize Map
New Map
Zoom
Assign Behaviors
Assign Behaviors
Update
Cancel
0 -> No Traffic
100 -> Traffic Most Likely

(19 , 0)

W: 20 × H: 10

10

North Wall [✓]
West Wall [ ]
East Wall [✓]
South Wall [ ]
Portal [ ]

Paramors: 0

Hosts: 1

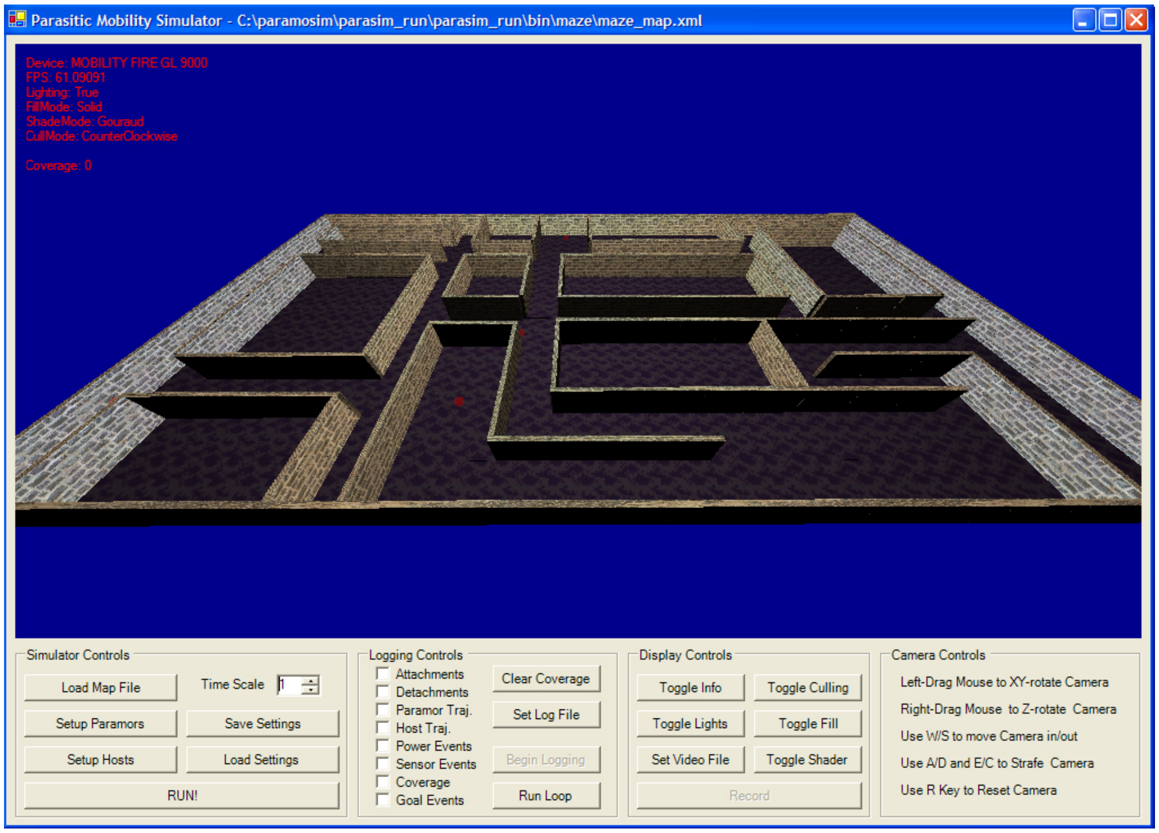
Power: 0
Temperature: 0
Light: 0
Altitude: 0
Vibration: 0
Radiation: 0

Host Traffic Distribution Weight: 100

Figure 3-16: Maze layout for simulation of a real scenario

For this test, a maze was laid out, populated with hosts, and a single paramor node. The paramor was given orders to try and reach the cell in the top-rightmost corner indicated in Figure 3-16 by a red square. The hosts were programmed to move forward along their current path until reaching a point of decision. At this point they decide which way to go not including the direction that they came from, unless it is a dead end. Of the possible directions at an intersection, the host chooses randomly, but gives a slight preference to places that it has already been. This is intended to simulate people in an office building or other environment familiar to the host; the assumption being that people in a familiar environment will tend to take the same paths to get to where they want to go, which is normally a small subset of possible locations in the environment that differs from person to person. When a host reaches a dead end, before turning around, it remains stationary for some time to simulate arrival at a destination such as an office.

Three different paramor behaviors were implemented and compared. The first paramor behavior simulated was the as-the-crow-flies distance calculation that is the same as the previous simulations described in this chapter. The second is a basic maze-solving algorithm. For this behavior, it is assumed that the node can sense or has enough knowledge of its immediate surroundings to be able to tell which directions that the host can travel. If the host takes any path but the rightmost path, the node will detach and wait for the next host. The final behavior assumes that the node has knowledge of the complete map, not an unreasonable assumption considering the diminishing size of GPS mapping devices. In this behavior, the node decides upon pickup as soon as it can predict the direction of travel, whether the presumed destination maze point that this host is heading towards, is better than the current one. Essentially, it simplifies down to the node choosing the shortest route from its current destination and only remaining attached to hosts traveling on this chosen path.



Parasitic Mobility Simulator - C:\paramosim\parasim_run\parasim_run\bin\maze\maze_map.xml

Device: MOBILITY FIRE GL 9000
FPS: 61.09091
Lighting: True
FillMode: Solid
ShadeMode: Gouraud
CullMode: CounterClockwise

Coverage: 0

Time Scale 1

Left-Drag Mouse to XY-rotate Camera
Right-Drag Mouse to Z-rotate Camera
Use W/S to move Camera in/out
Use A/D and E/C to Strafe Camera
Use R Key to Reset Camera

Figure 3-17: Maze simulation executing

The simulation was run ten times with each of the three behaviors with a single node with its start and goal positions remaining the same for all runs. The environment has ten hosts in it, each starting at a different location and behaving as described above.

Since the hosts were programmed to favor places they have already been, and they were deployed in unique locations, the trend in host behavior throughout these simulations was to make a few random decisions in the beginning to establish its route, then remain generally in that area, looping, and converging with the other hosts at some of the intersections towards the center of the maze. This turned out to be quite a favorable situation for the nodes which could get relayed from host to host along the central path. The simulation was executed and the averaged data collected is shown below in Table 3-3.

	Node Behavior 1 (distance algorithm)	**Node Behavior 2 (right-hand rule)**	**Node Behavior 3 (path omniscient)**
Total Time	230	600	210
Attached Time	100	300	80
Number of Hops	22	30	12

Table 3-3: Maze simulation results

As expected, the omniscient behavior performed the best. This makes sense since these nodes will always take the same path, and the number of decision points they will pass is fixed. So it becomes a probabilistic function of the number of decision points and possible directions that a host can turn at each of these points. It is further helped by the fact that hosts in the test do not turn around, cutting down the number of wrongful directions.

The distance algorithm behavior did not perform that much worse than the omniscient behavior. This is probably due to the fact that the maze is relatively simple, and the distancing algorithm can easily resolve the situation and more or less find the same path as the omniscient node.

Of the three behaviors, the right-hand-rule behavior is the only one that has the potential to get way off track. By nature of this type of pattern it can take a long time to reach the destination, but it is guaranteed and the coverage of the area is procedural and predictable, which may be desirable for some applications.

3.3 Software Simulation Conclusion

The software simulator has proven an invaluable tool with which to experiment with ideas and specific algorithms for implementing parasitic mobility. The experiences with the software simulation and the presence of these types of systems in nature have presented a favorable proof of concept for this type of mobility. The following chapter describes the design and implementation of an actual parasitically mobile sensor network, which takes into account all the quantitative and qualitative results from the software simulator.

Chapter 4

Hardware System

4.1 Electronics Design

To test the concept of parasitic mobility in a real-world setting, a hardware system comprised of electronic nodes equipped with all the necessary elements to implement the specific ideas introduced through the software simulation was designed and built. The nodes required processing, communication, data storage, a location system, a suite of sensors, and an onboard rechargeable power source. The electronics should also facilitate experimentation with different types of attachment and detachment mechanisms.

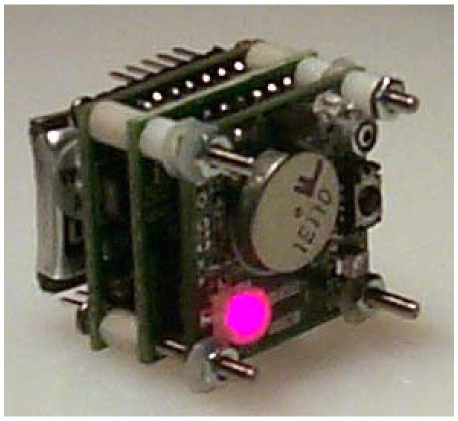


Figure 4-1: Node Hardware with 3 layers

The electronics were designed as small as could be easily built by hand using easy to obtain components. The design is based on stackable layers each around 1 square inch in size. When four layers are stacked, they are less than 1 inch high. More details on the mechanical specifications of the node hardware are given following the breakdown of the individual layers. The complete schematics, circuit board layouts, and bill of materials are listed in the appendices.

4.1.1 Power Module

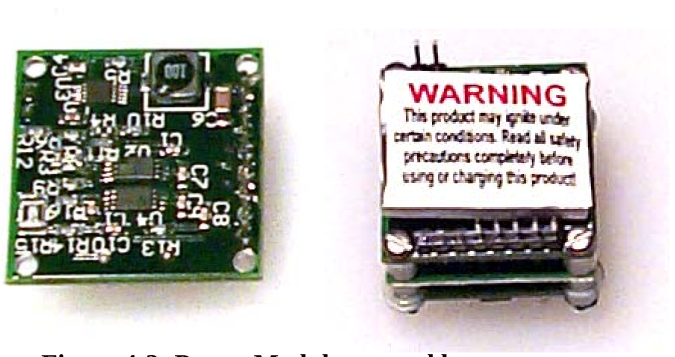

WARNING
This product may ignite under certain conditions. Read all safety precautions completely before using or charging this product!

Figure 4-2: Power Module top and bottom

The power module is based around a Lithium Polymer rechargeable battery. Lithium Polymer, LiPo for short, was the chosen battery chemistry because it is the current leader in charge density (capacity with respect to volume) amongst easily obtainable rechargeable battery cells. Charge density is important when trying to design a node as small as possible.

The specific battery chosen is a flat package measuring 25mm by 20mm by 4.5mm and weighs 3.5 grams. This is within the size requirements of the node. The battery has a capacity of 145 mAh at a voltage of 3.7 V. The battery can discharge at rates up to approximately 1 amp.

The power module also needs to be able to switch off the battery when an external power source is present and power the node from this source as well as recharge the battery. This requirement is to facilitate harvesting power from the environment whenever it power is available and make the switch transparent to the node systems.

The power module also contains a highly efficient step-down converter that provides a regulated 3.3V to the rest of the node. The battery at full charge provides a voltage of 4.2V. When the battery drains and the voltage drops below 3.3V, the step-down converter allows the battery voltage to pass through directly. When the battery drops below 3V the step-converter turns off and stops draining the battery. Draining a lithium polymer battery below 3V can destroy the battery, so this feature acts as a battery protection circuit. In addition, a resistor network is used on the feedback circuit that senses the battery voltage to provide some hysteresis preventing the battery from turning off and on due to the battery voltage rising when the load is removed.

The last feature of the power module is a gas gauge chip. This chip uses a 0.02 ohm current sense resistor to monitor battery usage and calculate remaining battery life in seconds. It provides a HDQ digital interface to allow a microcontroller to query the battery life information. HDQ is a bi-directional serial interface over one wire; it is Texas Instruments’ version of the 1-Wire protocol from Dallas Semiconductor [37].

4.1.2 Processing

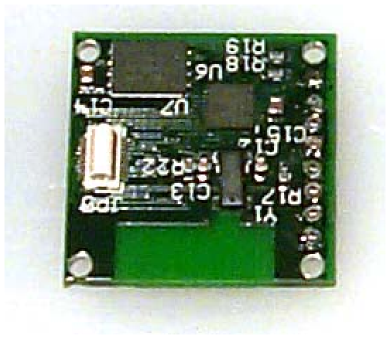


Figure 4-3: Top side of Processing and Communication Module showing the Microcontroller and Memory

A Silicon Labs C8051F311 microcontroller was chosen as the processor for the node. This processor can run up to 25 MHz using its internal digitally controlled oscillator and will use 300 uA per MHz. Also included in the processing module is an external 32 kHz crystal which is used when the processor goes into a low power mode. At 32 kHz, the analog peripherals and internal timers are still sufficiently alive to wake the processor. By alternating between these two oscillators, the battery usage can be minimized.

This processor is a fully-featured mixed signal processor with a hardware SPI controller, hardware UART, hardware I^2C controller, 4 PWM/Frequency generator outputs, and a 17 input 10-bit analog-to-digital converter. It has 1.2K of internal RAM and 16KB of flash for program storage. The C8051F311 comes in a 5mm-square leadless MLP package, making it the smallest processor available at the time of design with the peripherals needed for the Paramor node.

Lastly, the processing layer of the paramor node also contains an Atmel Dataflash memory chip with a capacity of 16MB. This storage is required to store the firmware for the GPS module (discussed in section 4.1.4), which requires 1MB of storage, and to store collected sensor data for later retrieval or transmission.

4.1.3 Communication

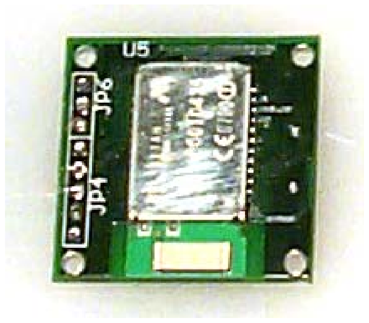


Figure 4-4: Bottom side of Processing and Communication Module showing the Bluetooth Radio

Each node is equipped with a wireless communication system to allow nodes to communicate with each other for distributed sensing applications, passing of navigational information, and cable-less retrieval of data from nodes.

Currently, the mobile telephone industry has driven down the size and power consumption of Bluetooth modules, and Bluetooth is easily interfaced to (if not already connected) by PCs and other devices. For these reasons Bluetooth was chosen as the wireless protocol and hardware. Bluetooth modules are available with embedded antennas and communication ranges up to 100 meters in a 13mm by 24mm package.

The specific Bluetooth module used is the BR-C11A Class 1 Bluetooth module from BlueRadios, Inc [38] which includes an antenna and has a Bluetooth protocol stack programmed directly into the module itself. This allows complete control of the Bluetooth radio via simple UART commands.

The embedded Bluetooth stack supports Bluetooth Inquiry to find other devices in range and can connect to and communicate with 7 nodes simultaneously. It is easy to discover, register, connect, and disconnect nodes allowing large-scale peer-to-peer networks to be generated.

4.1.4 Sensor Suite

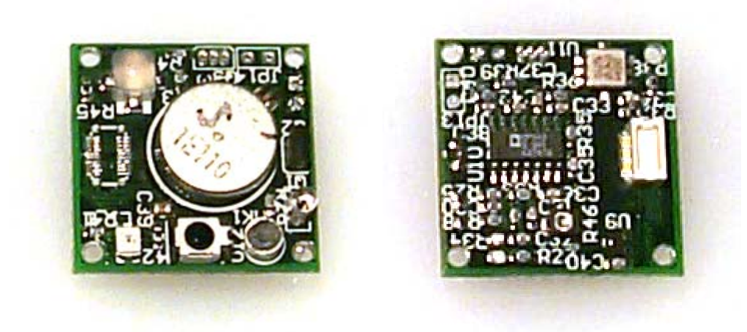


Figure 4-5: Sensor/Actuation Module top and bottom

The sensor and actuation module contains the following input and output mechanisms:

- **2 Axis Accelerometer –** Used to determine if the node has been picked up or dropped off, as well as for collecting vibration and inertial data
- **Microphone –** Used to collect audio data from the environment
- **Active Infrared Proximity Sensor –** Used to test node distance from an external object or used to test presence of a potential host
- **Temperature Sensor –** Used to collect environmental data
- **Light Sensor –** Used to collect environmental data
- **RGB LED –** Used to display status and sensor information, as well as to act as a signal or attractive device to potential hosts
- **Pager Motor –** Used as a detachment device in the semi-passive node design or to signal host to release node

- **Motor Controller –** Used to control an external motor used for attachment and detachment in the active node design

This module also contains the analog circuitry necessary to interface the sensors and outputs to the microcontroller

4.1.5 Location System and Monitor System

Each node needs to be equipped with a system for placing itself in the environment. The first location system designed for this system is a GPS module.

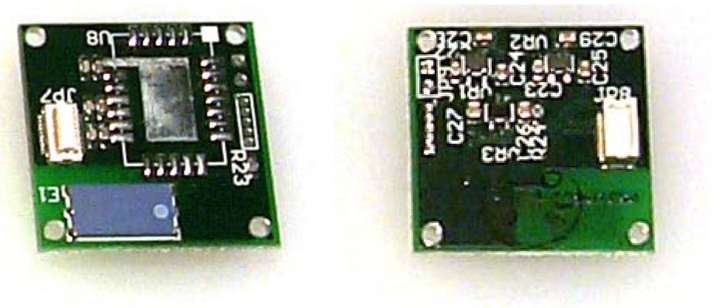


Figure 4-6: Top and bottom of the GPS Module layer shown without GPS chipset populated

The design of the GPS layer is based around the Motorola FS OnCore single chip GPS module. This module can work in Assisted Mode (which requires a GPS beacon) and Autonomous Mode. The design of the GPS layer includes the Motorola Module, a Yageo Embedded GPS Antenna, and the required power supply components. The GPS system also requires enough storage to store the GPS firmware, and a processor capable of writing the firmware into the GPS module over its SPI interface at boot time.

A prototype GPS module was built using the sample FS OnCore chip included in the development kit. Using the Assisted GPS mode, a position fix, accurate within a few meters, could be achieved in 1 second using 75 mW of power. While not working on a position fix, the module could be put into a sleep mode where it will draw only a few micro-amps. The GPS module worked decently indoors (with the Assisted-GPS beacon placed in front of a window) and outdoors and proved itself as a usable location system for this application.

Unfortunately, at the time of writing this, Motorola could only provide a few samples of their GPS chipset, and could not yet ship the quantity needed for this research. So a second type of location system was designed to be used until the GPS chipsets become available.

The second location system is a series of Bluetooth beacons, each with a 10 meter range, placed in an overlapping grid around the area of interest. The nodes can inquire to find out which beacons are in range and figure out their location from this information.

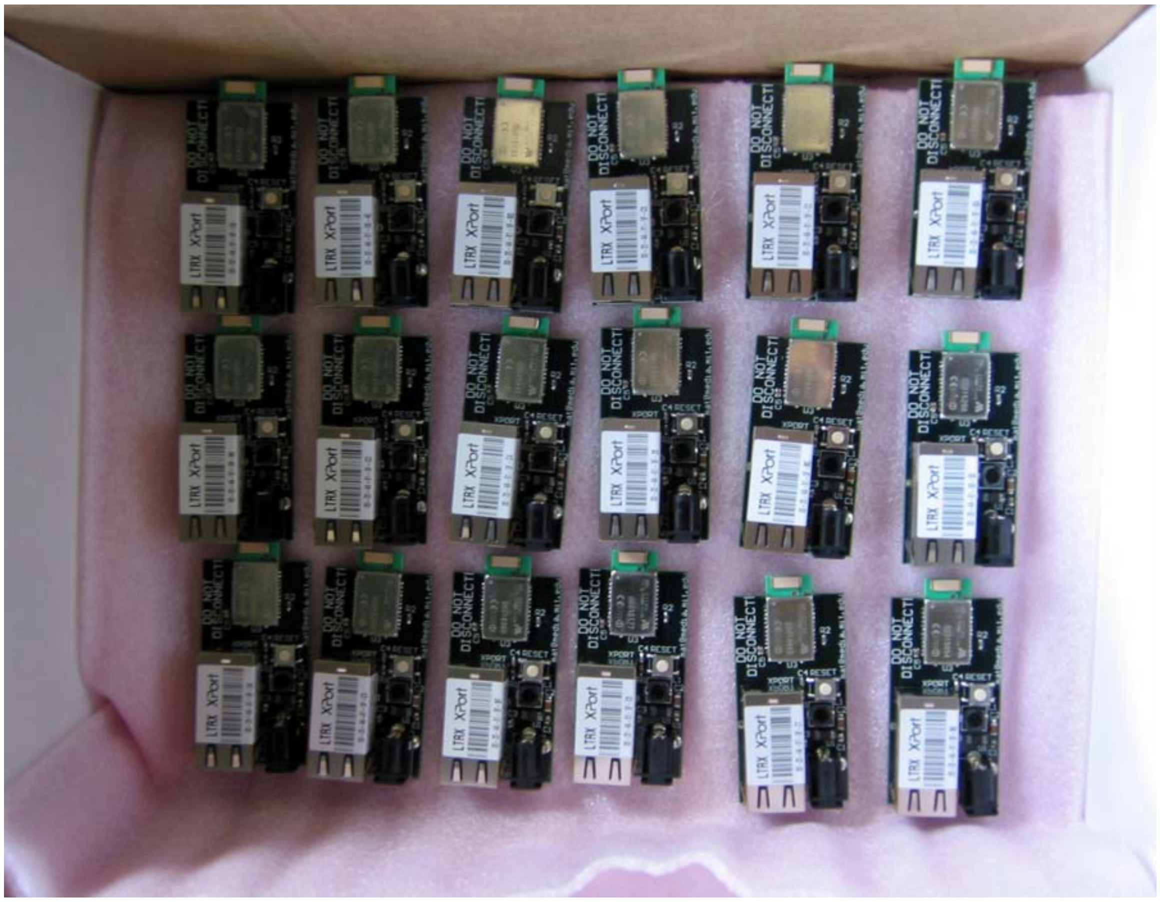


Figure 4-7: Bluetooth Location beacons ready to be deployed

Each Bluetooth beacon is comprised of a Bluetooth radio module, a power supply, and a Lantronix XPort. The Xport is an Ethernet controller and a processor built into an Ethernet connector form-factor. The XPort allows quick and easy development of a sockets interface to the functions of the Bluetooth module.

With this network capability, the beacons can also be used to connect from a central location on the network to any of the nodes that are in range of any of the beacons. This facility can be used for test purposes to track the nodes, retrieve any collected data, or manually control the functions of the node.

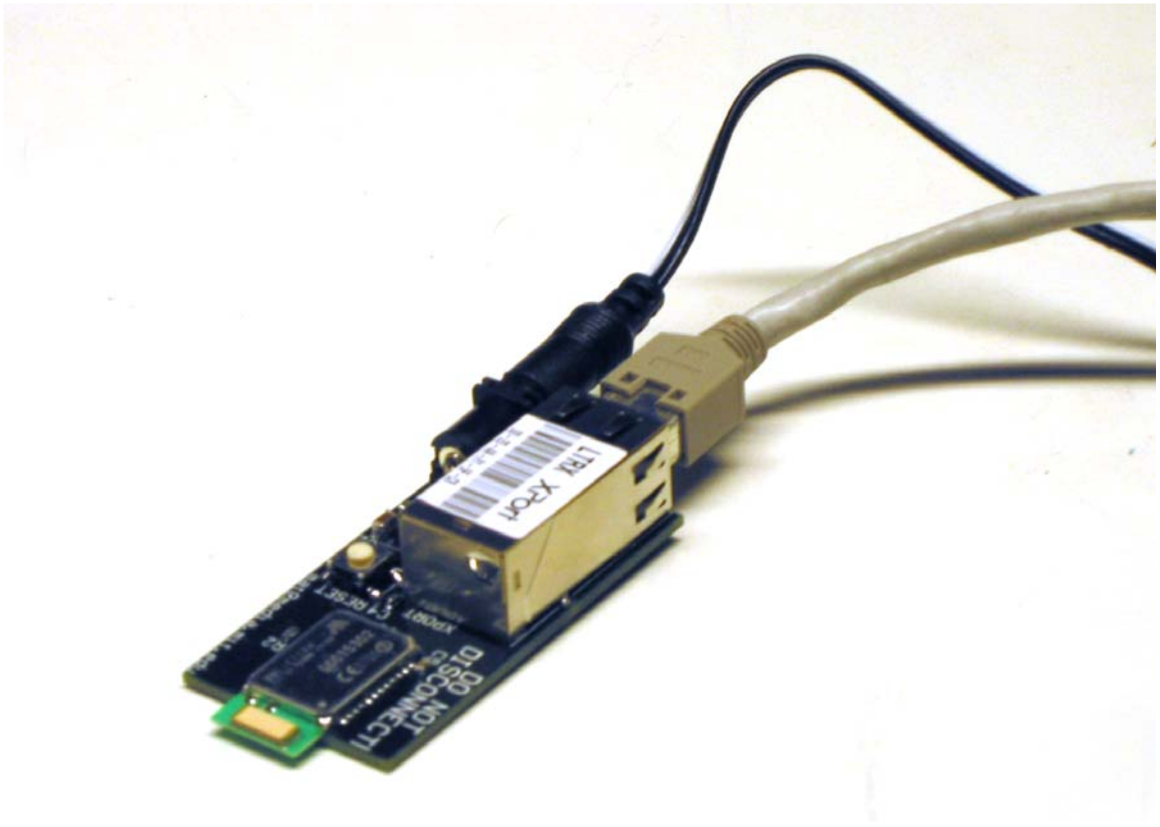
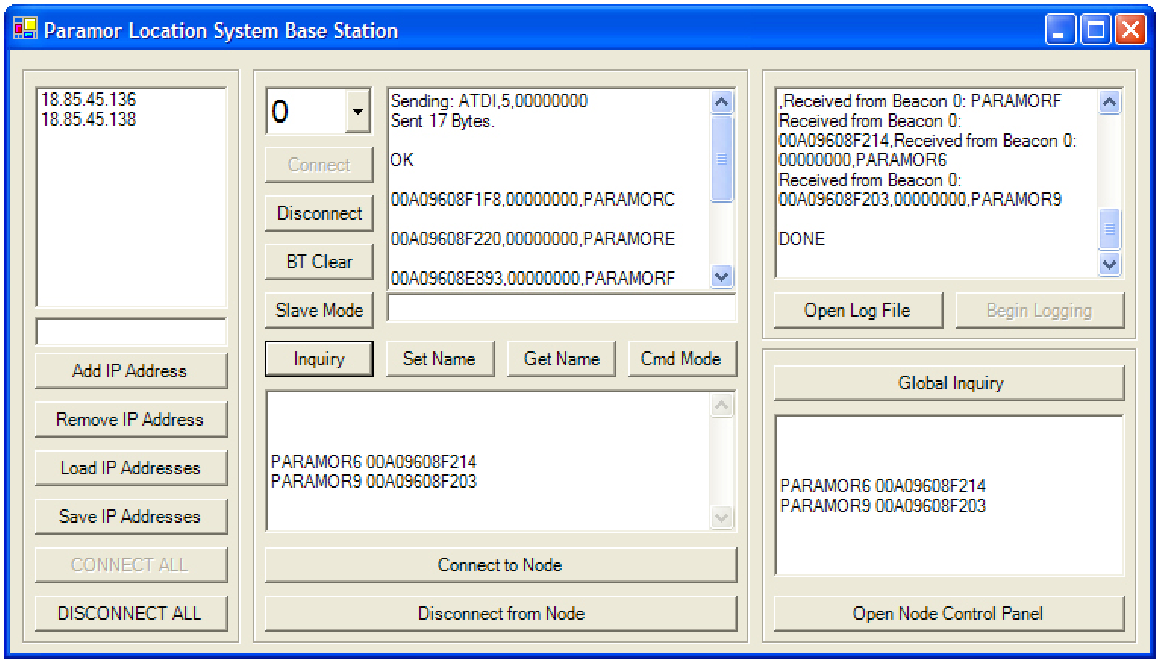


Figure 4-8: Bluetooth beacon with LAN cable attached and power supply plugged in

The central monitoring software maintains a list of IP addresses of the beacons in the systems, and can inquire to find out what nodes are in range of each beacon. It can then put together a node-centric view which will list all the nodes and allow connection to an individual node.



```
18.85.45.136
18.85.45.138
```

```
0
```

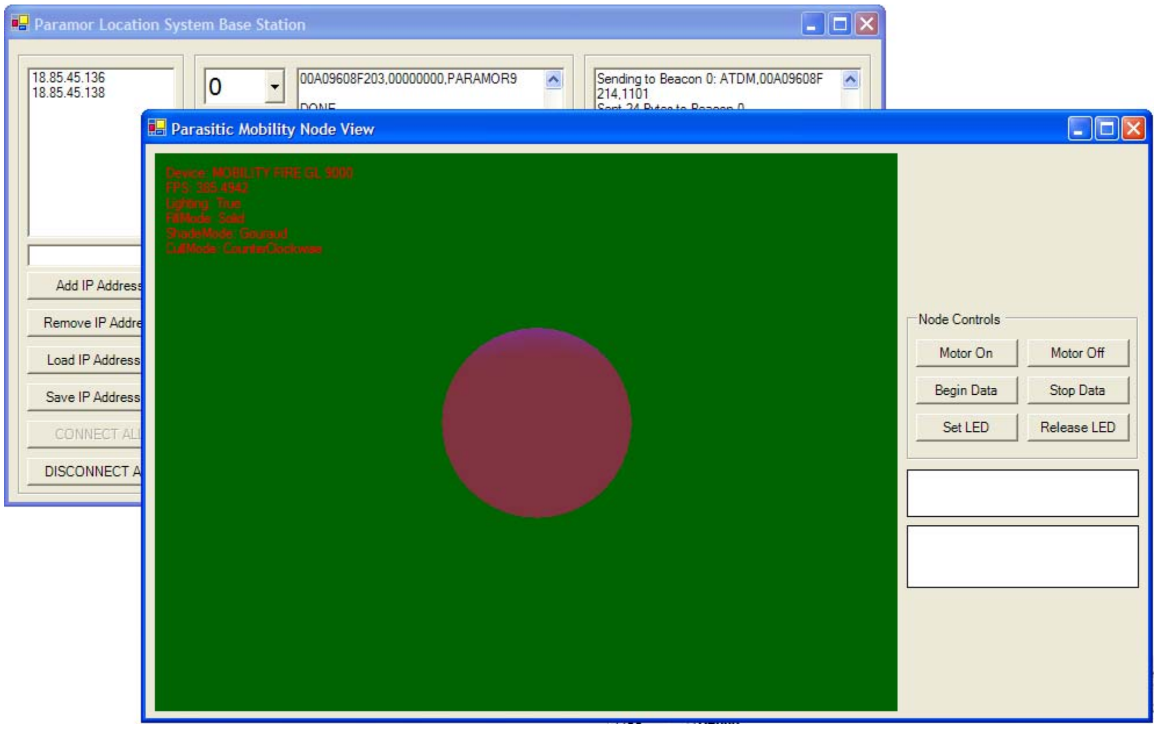
```
Sending: ATDI,5,00000000
Sent 17 Bytes.

OK

00A09608F1F8,00000000,PARAMORC

00A09608F220,00000000,PARAMORE

00A09608E893,00000000,PARAMORF
```

```
,Received from Beacon 0: PARAMORF
Received from Beacon 0:
00A09608F214,Received from Beacon 0:
00000000,PARAMOR6
Received from Beacon 0:
00A09608F203,00000000,PARAMOR9

DONE
```

```
PARAMOR6 00A09608F214
PARAMOR9 00A09608F203
```

```
PARAMOR6 00A09608F214
PARAMOR9 00A09608F203
```

Figure 4-9: Screenshot of the central control software for monitor node behavior from the network. The left panel contains IP addresses for the beacons, the center panel allows commands to be sent and received from a particular beacon, and the panel in the bottom right allows inquiry commands to be sent to all the connected beacons and a global node list to be built. A node can be selected from this list and a control panel for that node can be opened. Manual commands can be sent to a node once connected using the beacon controls in the center panel.



```
18.85.45.136
18.85.45.138
```

```
0
```

```
00A09608F203,00000000,PARAMOR9
DONE
```

```
Sending to Beacon 0: ATDM,00A09608F
214,1101
Sent 24 Bytes to Beacon 0
```

```
Device: MOBILITY FIRE GL 9000
FPS: 385.4942
Lighting: True
FillMode: Solid
ShadeMode: Gouraud
CullMode: CounterClockwise
```

Figure 4-10: Screenshot of the node control panel allowing control and visualization of a specific node. The red sphere moves according to acceleration data, and glows according to light sensor data. Using this control panel, a user can also turn the motor on and off, change the LED color, listen to the audio from the microphone, view temperature sensor data, and view the remaining battery life.

4.2 Mechanical Design

The electronics described in the preceding section need to be encased and equipped with the mechanisms to support parasitic mobility. In chapter 2, we have identified four types of parasitic mobility attachment/detachment mechanisms: active, passive, semi-passive, and attraction/value-added. In this section, hardware designs to support experiments for these four types of mechanisms are described.

4.2.1 Active Node Design

The first node was built to test the concept of an active node based on the natural parasitic behaviors of fleas and ticks. The basic idea is to build a hopping robot that can sense a nearby object, hop at it or onto it, and attach. The mechanism needs also to be able to cause the node to detach on command and fall off the host.

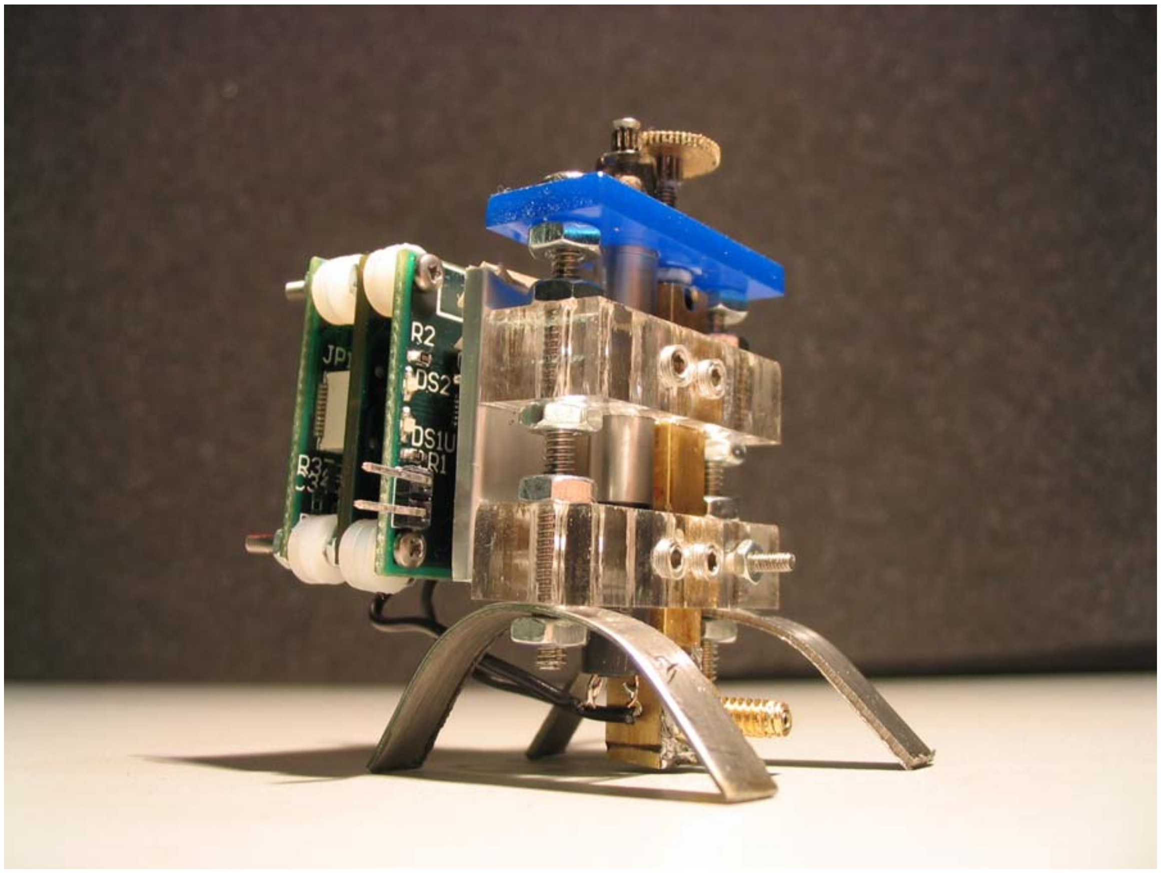


Figure 4-11: The active node, nicknamed the ParaHop.

This active node, shown in Figure 4-11 and Figure 4-12, is 40mm tall by 30mm wide by 30 mm deep, including a mechanical launching mechanism and all electronics, comprised of the power module with battery, processing and communication layer, and sensor/actuator module as described in the preceding section.

The device consists of the electronics mounted to a frame consisting of 3 horizontal bars and two vertical bolts using nuts to position the plastic crossbars. The frame also contains two aluminum feet to hold the node upright, ready to jump. A future enhancement will be to encase the node in a self-righting egg-shaped plastic case.

Down the center of the frame are the hopping piston and a planetary-geared, 8mm in diameter motor used to reset and release the piston. The design of the piston is shown in Figure 4-12.

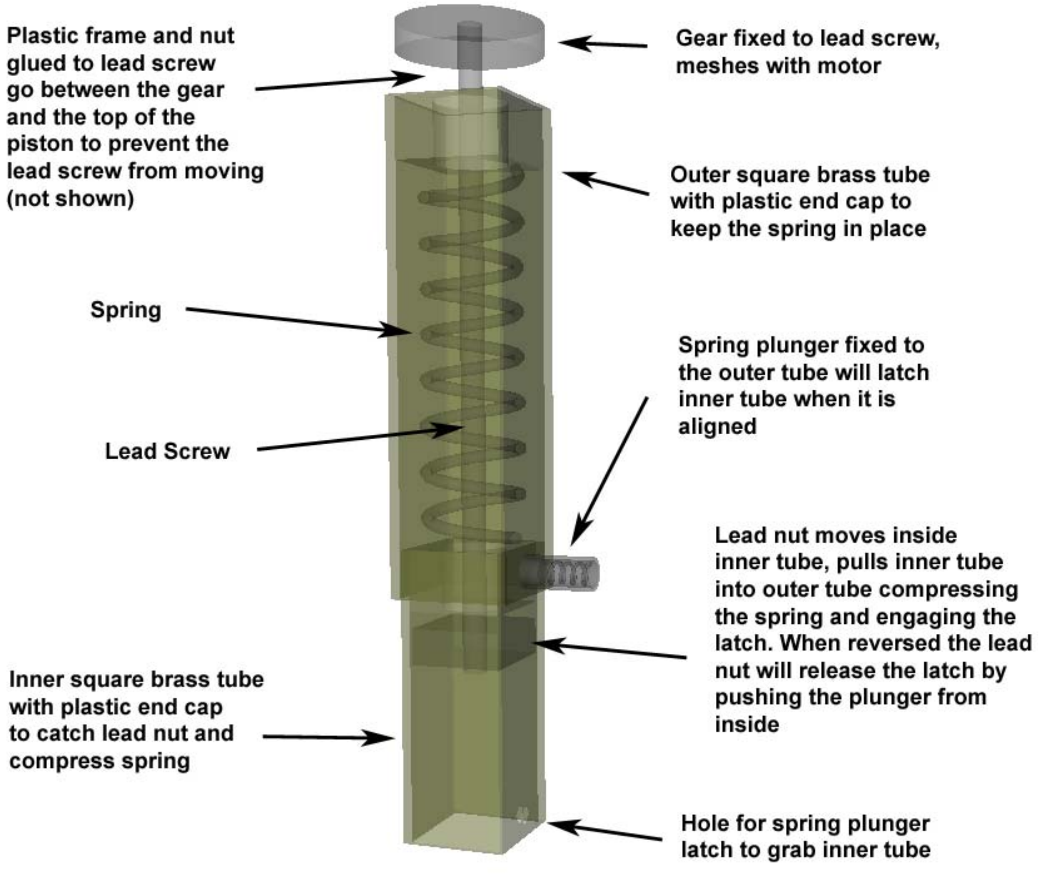

Plastic frame and nut glued to lead screw go between the gear and the top of the piston to prevent the lead screw from moving (not shown)
Gear fixed to lead screw, meshes with motor
Outer square brass tube with plastic end cap to keep the spring in place
Spring
Spring plunger fixed to the outer tube will latch inner tube when it is aligned
Lead Screw
Lead nut moves inside inner tube, pulls inner tube into outer tube compressing the spring and engaging the latch. When reversed the lead nut will release the latch by pushing the plunger from inside
Inner square brass tube with plastic end cap to catch lead nut and compress spring
Hole for spring plunger latch to grab inner tube

Figure 4-12: CAD Drawing of the hopping actuator.

The hopping actuator is designed around two telescoping square tubes. Both of the tubes have plastic end caps with a hole for the threaded rod (lead screw) to pass through the middle of the piston. Around the lead screw in the outer tube is a spring. By turning the lead screw, the lead nut located inside the inner tube is pulled upwards, compressing the spring, and pulling the inner tube up inside the outer tube. When the inner tube is pulled completely inside the outer tube, a spring plunger latches the inner tube in place through a hole near the bottom end of the inner tube.

Once the inner tube is latched, the motor is no longer under any strain and can be disengaged until it is time to hop. When it is time to hop, the motor reverses and sends the lead nut downwards until it hits the plunger on the inside of the inner tube. When the lead nut keeps going, it will push the plunger and release the latch. This will free the spring to expand, pushing the inner tube outwards and causing the node to hop. For more precision, the motor can reverse after the latch engages, and stop with the lead nut just above the latch. In this state, it is ready to trigger a hop on a moments notice. A minor change in the design would allow the node to be deployed with a spring pre-wound for a certain number of hops.

The node was then equipped with 5 hooks protruding in all directions, each with a curvature of 1 inch in diameter. The robot would hop to heights around 8 cm from the ground at an angle of around 70 degrees. Due to the placement of the motor and the battery it always hopped in the same direction relative to itself. This height proved enough to hook into a person’s pant leg or shoe. Other attachment devices were tried including Velcro and silicon adhesive, but only a large hook could grab clothing, given the irregularities in the approach vector. The power usage statistics were given in Chapter 3. Other attachment mechanisms such as shuttered magnets for vehicles and electrically activated adhesives were briefly examined. Most of these methods were deemed unsuitable for testing on humans.

4.2.2 Semi-Passive and Passive Node Design

The next experiment was to design nodes with semi-passive and fully passive attachment and detachment mechanisms. The power calculations shown in Chapter 3 denote huge power savings for these types of nodes.

To build nodes of this type, the electronics were enclosed in a plastic sphere with an outer diameter of two inches. The electronics were attached strongly to the spherical case, allowing the pager motor's vibrations to affect the external surfaces.

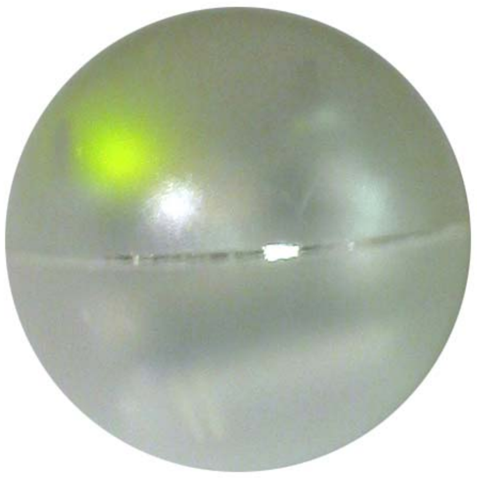


Figure 4-12: The node electronics in their spherical casing

To make these spherical devices into semi-passive nodes, the surface was coated with a polyester double-sided adhesive tape from 3M. Polyester tape was chosen because it can stretch and form a tight, smooth layer around the sphere. It is necessary that the surface is smooth to prevent too much of the surface from sticking to a host, making it difficult for the ball to shake loose. The polyester tape is very thin and works perfectly in this regard. It is available in a wide range of sticking strengths. Through trial and error, the appropriate stickiness was found that allowed the ball to easily stick to anything it touched and remain stuck until it is shaken loose from inside with the pager motor.

In order to create passive nodes, a different adhesive needed to be used. The bond, when attached, would have to degrade over time, allowing a new bond formed on the opposite side of the sphere to be the stronger bond allowing the node to be pulled from its original host. If no new host came about, the bond would degrade and the node would eventually fall free without the use of the pager motor or any form of actuation. Various consistencies of silicon were mixed and tested. Through this trial and error, a working silicon adhesive was created that performed as desired. When a bond was made, the silicon would get weakened at the spot of the bond by getting soiled by the host's surface. This worked quite well; however, the ball would have a limited number of attachments before it became too dirty to stick at all. The polyester tape did not have this problem because it was much stickier and less greasy, but required the pager motor to dislodge it. Other attachment mechanisms, such as devices based on hooked microstructures, could be potential candidates for parasitic nodes, but would require more resources to develop.

4.2.3 Value-Added/Attraction Node Design

The basic spherical node without any sticky surface falls into this category by way of its full spectrum LED. This LED can be programmed to display attractive patterns that catch the eye of a passerby, especially in an academic research institution where everyone is attracted to blinking lights and novel objects of technology. Once attracted, the host can receive instructions from the node as to what the node does and what the host can do with it. Figure 4-13 shows a label that can be applied to the node to give the host specific instructions.

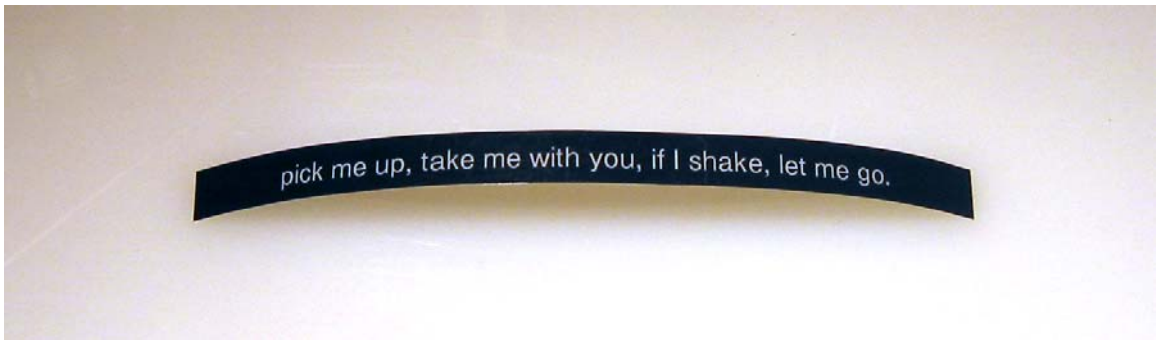


Figure 4-13: A label that can give the host instructions as to what to do with the node. This can be combined with a reward from the node for the host if the instructions are followed

The nodes can then reward their attracted host for following the instructions. Example rewards can be discounts provided on purchases while carrying the node, or providing useful information to the host. When the node wants to be dropped off, it can stop providing these rewards, vibrate or make a sound signaling a desire to be put down, and/or turn off the LED until it is ready to be picked up again.

While extensive testing was done on all the mechanisms described in this section, the complete multi-node test described in section 4.4 was done using the simplest of the nodes, the labeled value-added sphere. As shown in section 4.4, this attachment mechanism worked quite well given the test environment, a building at the Massachusetts Institute of Technology.

4.3 Firmware Design

The firmware design is based around the software simulator’s node behavior design. The node firmware can be seen as being formed from three entities: data structures that hold state information, behavior information, and map information; background processes that handle actions implemented in hardware or through the use of hardware peripherals such as the wireless communication and the sensor readings; and the main firmware code that executes the node’s activity. The firmware is written for the Silicon Laboratories series of 8051-style processors using the Keil C compiler.

4.3.1 Data Structures

The basic data structures used by the firmware to store its internal information are shown below in Table 4-1.

```
//---------------------------------------------------------------------------
// Data Structures
//---------------------------------------------------------------------------

typedef struct {
        unsigned int btX;
        unsigned int btY;
        unsigned char latDir;
        unsigned int latDeg;
        unsigned int latMin;
        float latSec;
        unsigned char lonDir;
        unsigned int lonDeg;
        unsigned int lonMin;
        float lonSec;
} location;

typedef struct {
        unsigned int powerThreshold;
        location goal;
        unsigned char gotoGoal;
        unsigned char stopAtGoal;
        unsigned char coverage;
        unsigned int LightThreshold;
        unsigned int VibrationThreshold;
        unsigned int TemperatureThreshold;
        unsigned int AltitudeThreshold;
        unsigned int AudioThreshold;
        unsigned int senseTime;
        unsigned int stopTime;
        unsigned int hopsPerLocale;
} behavior;

typedef struct {
        location *good;
        location *bad;
        location *visited;
        location *unvisited;
} map;

typedef struct {
        unsigned int sensors[];
        unsigned int powerLeft;
        unsigned int hopsRemaining;
        unsigned int senseTicks;
        unsigned char state;
        location current;
        map node_map;
        behavior node_behavior;
} node;

enum { IDLE, ATTACHED, SENSING };
```

Table 4-1: Data Structures used in node firmware

There are four basic data structures used in the firmware. The first structure, location, contains information to identify a location in a coordinate system. The structure uses both the Bluetooth location system, which gives an XY coordinate, and the GPS system, which returns the coordinates in longitude and latitude. The GPS location system is not currently implemented in the firmware, but the data hooks for it are included for future use.

The map structure is used to store geographic information. This structure is created from lists of locations. The first list includes locations considered as good. These locations are ones that the node considers as attractive to visit. These can be locations known to have a high host frequency or hosts that are most likely to bring a node to a point of interest. The structure also contains a list of bad locations, which are locations that are known to have a low host frequency or to be a dead end. Good and bad locations can be entered manually, either prior to deployment or over a communication channel if available. They can further be discovered and identified by the nodes themselves. Once discovered, a node can broadcast these locations to any other nodes in range. The map structure also contains a list of the locations visited and a list of the locations that are known to be unvisited. These lists are used by a node trying to maximize coverage. Like the good and bad locations, the unvisited locations can be entered in manually or discovered and transmitted from node to node.

The behavior structure holds data that is very similar to the parameters identified in Section 3.1.3's discussion of the node behavior in the software simulator. Please refer to this section for details about the behavioral parameters.

The last data structure is the main structure for the node, including instances of the map and the behavior structure. This structure contains the current state, which uses the enumerated values of idle, attached, or sensing. The senseTicks variable is an internal counter that tracks the time left. The hopsRemaining variable is also an internal variable used to track how many hops have happened at the current location. The sensors array contains the current value of the six sensors described in Section 4.1.4 and the powerLeft variable stores the remaining time left on the current charge of the battery. Finally, the node structure contains the current location returned from the location system.

4.3.2 Background Processes

The chosen microcontroller and the hardware design combine to allow several activities to happen without taxing the main program loop.

The first of these systems is the Bluetooth radio. The radio module contains an embedded processor with a Bluetooth communication stack. This stack defaults to a mode where other nodes can find it, connect to it, and communicate to it. The data is then passed to the processor through a serial connection. On the processor side, the serial communication is handled by a hardware UART and the data reception is interrupt-driven. The serial interface is set to a speed of 9600 bits per second to allow successful operation in the low power mode running at a processor speed of 32 kHz.

Most of the possible wireless communication packets contain information that gets stored in the node's data structures for the main loop to use in its state machine. This happens completely in the interrupt service routine and the data is immediately available to the main loop.

The environmental sensors are read using the processor's analog to digital conversion hardware. This is also an interrupt-driven processor. After the conversion is finished, the interrupt service routine stores the values in the sensor array. The sampling is much slower when the node is in the low-powered idle state. In this state, the sensors are mainly used to detect when the node has been picked up and should transition to the attached state. The lower sampling rate is still adequate to identify this occurrence.

The battery life is monitored by a specialized gas gauge chip. However, the interface to read the battery life value from this chip is not connected to a hardware peripheral in the microcontroller hence must be controlled in the firmware. Furthermore, this communication is too fast to be performed in the low power mode. This proves to be adequate since the power drain in the low power mode is so minimal. Accordingly the battery life is checked routinely during the attached and sensing states.

4.3.3 Main Firmware Execution Code

The main firmware execution can be illustrated by the three diagrams shown in Table 4-3, Table 4-4, and Table 4-5. Each flowchart illustrates one of the three states that the node can be in: idle, attached, and sensing, respectively.

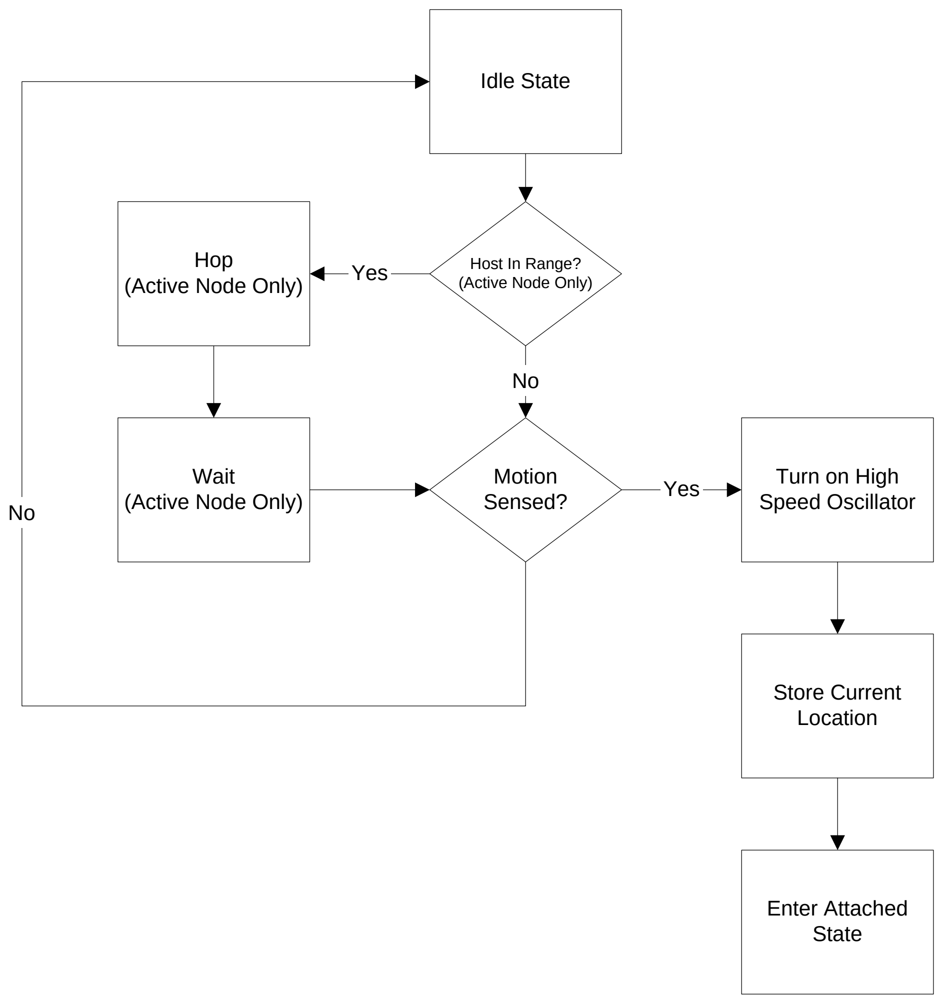


Table 4-3: Flow chart showing the basic operation in the idle state for both the active node and the semi-passive node.

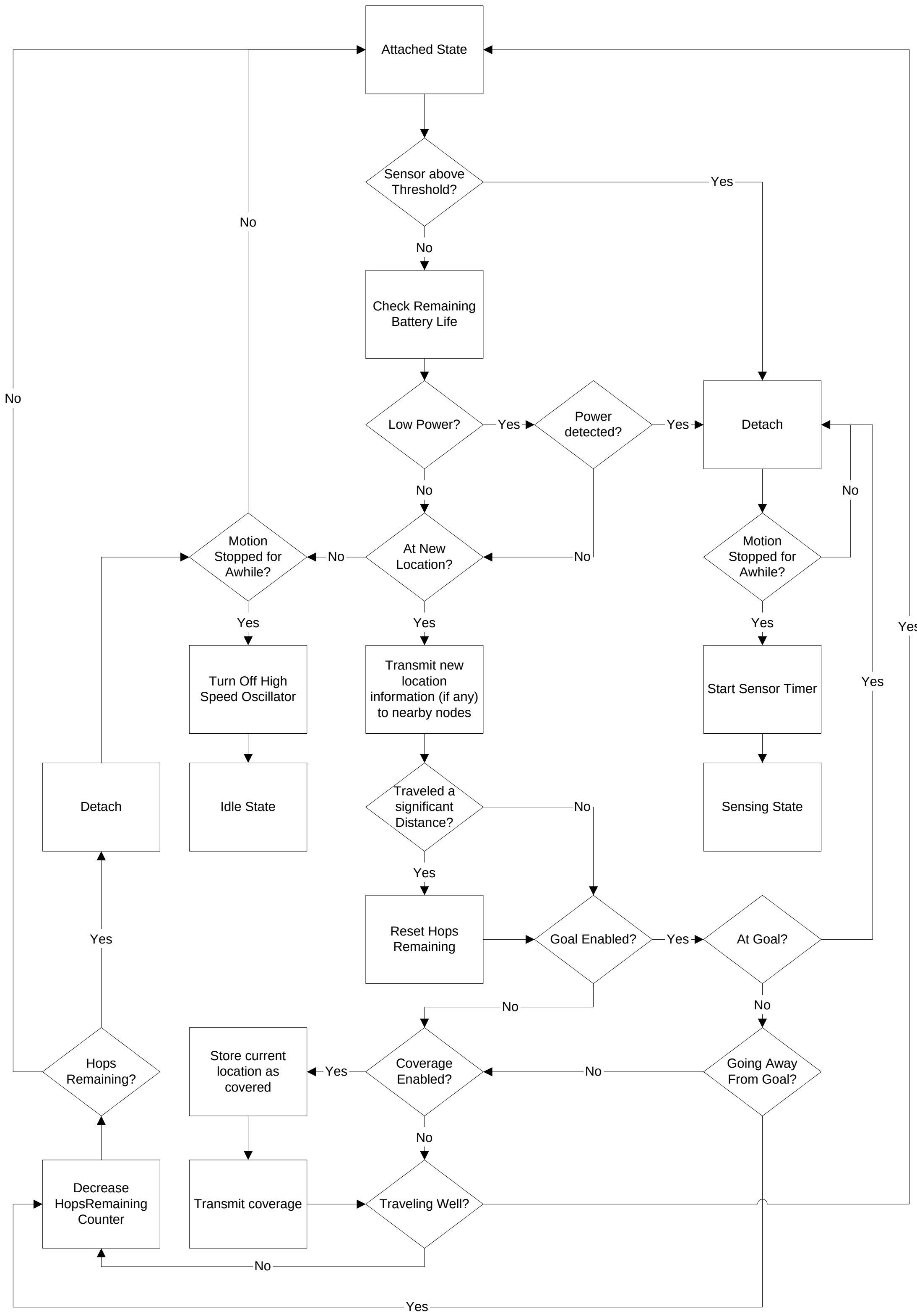


Table 4-4: Flow chart showing the firmware execution when the node is in the attached state

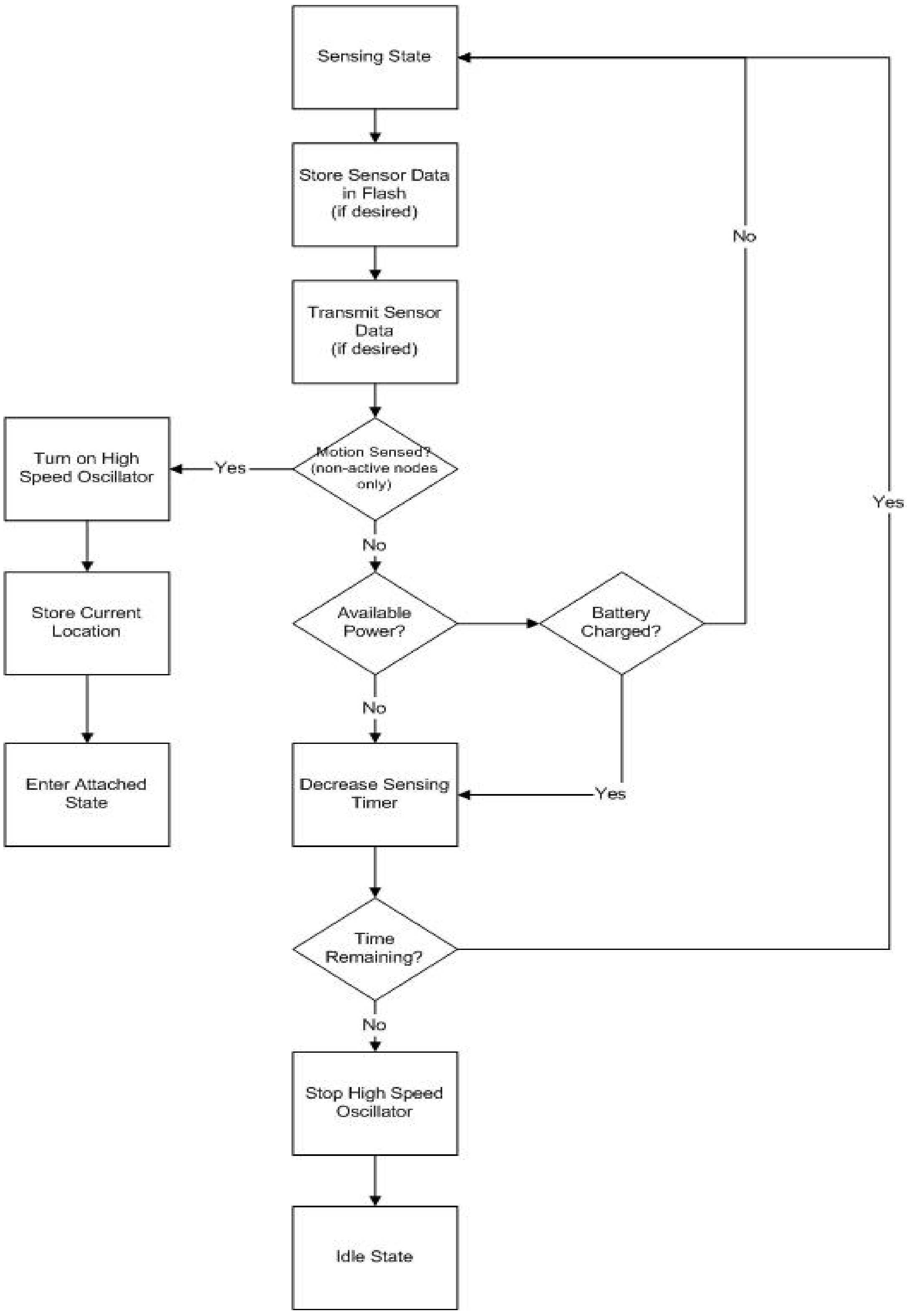

Sensing State
Store Sensor Data in Flash (if desired)
Transmit Sensor Data (if desired)
Motion Sensed? (non-active nodes only)
Yes
Turn on High Speed Oscillator
Store Current Location
Enter Attached State
No
Available Power?
Battery Charged?
No
Yes
No
Decrease Sensing Timer
Yes
Time Remaining?
No
Stop High Speed Oscillator
Idle State

Table 4-5: Flow chart showing firmware execution while in the sensing state

Tables 4-3, 4-4, and 4-5 illustrate the general firmware flow for a generic parasitic node based on the ideas developed from the software simulator. It is quite easy to modify this basic code to support many specific sensor network applications.

It is important to note that the sensing state is the state where the node has detached and is collecting data, but it is not the only time the sensor data can be collected. The sensors are fully active in the attached state and the data can be stored or collected in this state as well. It is also possible to collect data in the idle state, but with a sacrifice of power depending on the sampling rate and how many sensors are active. Furthermore, it is possible to combine the idle state and the sensing state into one state that is always collecting data and looking for a host.

In general, the three states exist only for power management reasons. The states allow the node to enable and disable peripherals and functions according to what the node needs in a particular state. For applications with less of a power restriction, the node can always be sensing, checking its location and power, and calculating whether it should detach or attach.

For semi-passive and passive nodes, the node can transition to the attached state from either the idle or the sensing state when it senses that it is moving. This is shown in Table 4-5. Active nodes, on the other hand, can control their state change into and out of the attached states.

The complete code listing of the firmware used in the test is given in Appendix B.

4.4 Test Application and Results

The final stage in this body of work was to execute a test of the parasitic nodes described above in this chapter by releasing them into a real-world situation. More specifically, ten semi-passive, value-added, spherical nodes were given orders and released into an environment populated by human hosts.

4.4.1 Test Application Description

The area selected for this test application was the third floor of the Media Lab at the Massachusetts Institute of Technology. This floor is usually inhabited by about 40 students and faculty and has light but steady traffic through its pathways. Furthermore, its inhabitants are known to be attracted to strange devices with blinking LEDs and are more than willing to pick up and carry around the sensor nodes.

In order to run this test, the floor first needed to be covered with the Bluetooth location beacons. The entire floor was able to be covered with only 6 beacons. These beacons were set at the 100-meter-range power output class of Bluetooth. However, the real range of the beacons indoors is closer to 25 meters. With these six beacons and the areas where two or three beacons overlap, we were able to divide the floor into a grid of 16 distinct zones. This resolution is more than adequate for this test. The beacons are given base-two numbers allowing unique numbers to be formed for the overlapping areas.

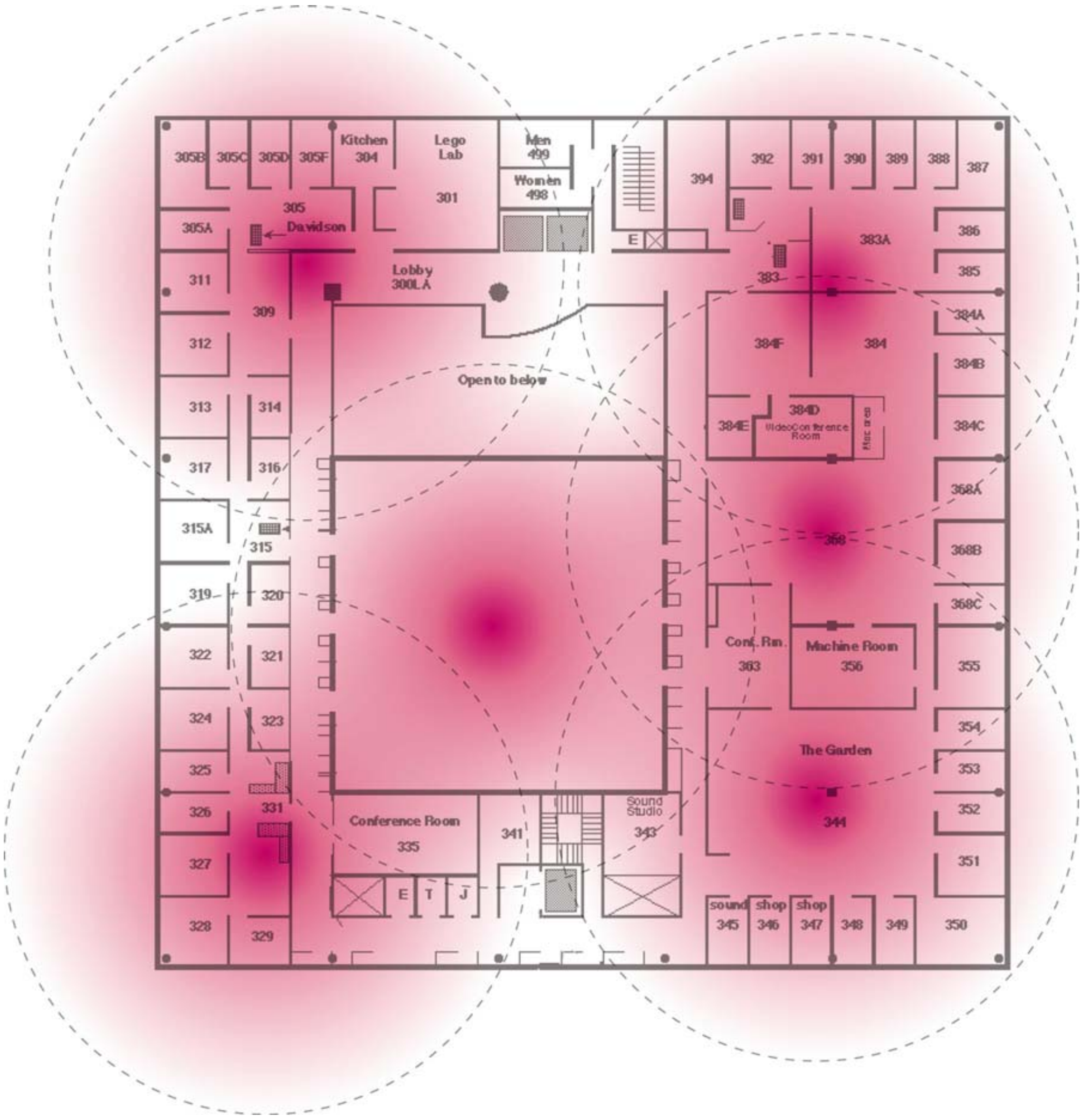


Figure 4-14: Bluetooth Location system coverage of test area

The location system's Bluetooth-based locations were mapped to the test area by performing a walkthrough calibration carrying a test node. These locations were given X/Y coordinates that the sensor nodes can use in calculations concerning direction of travel related to their goals.

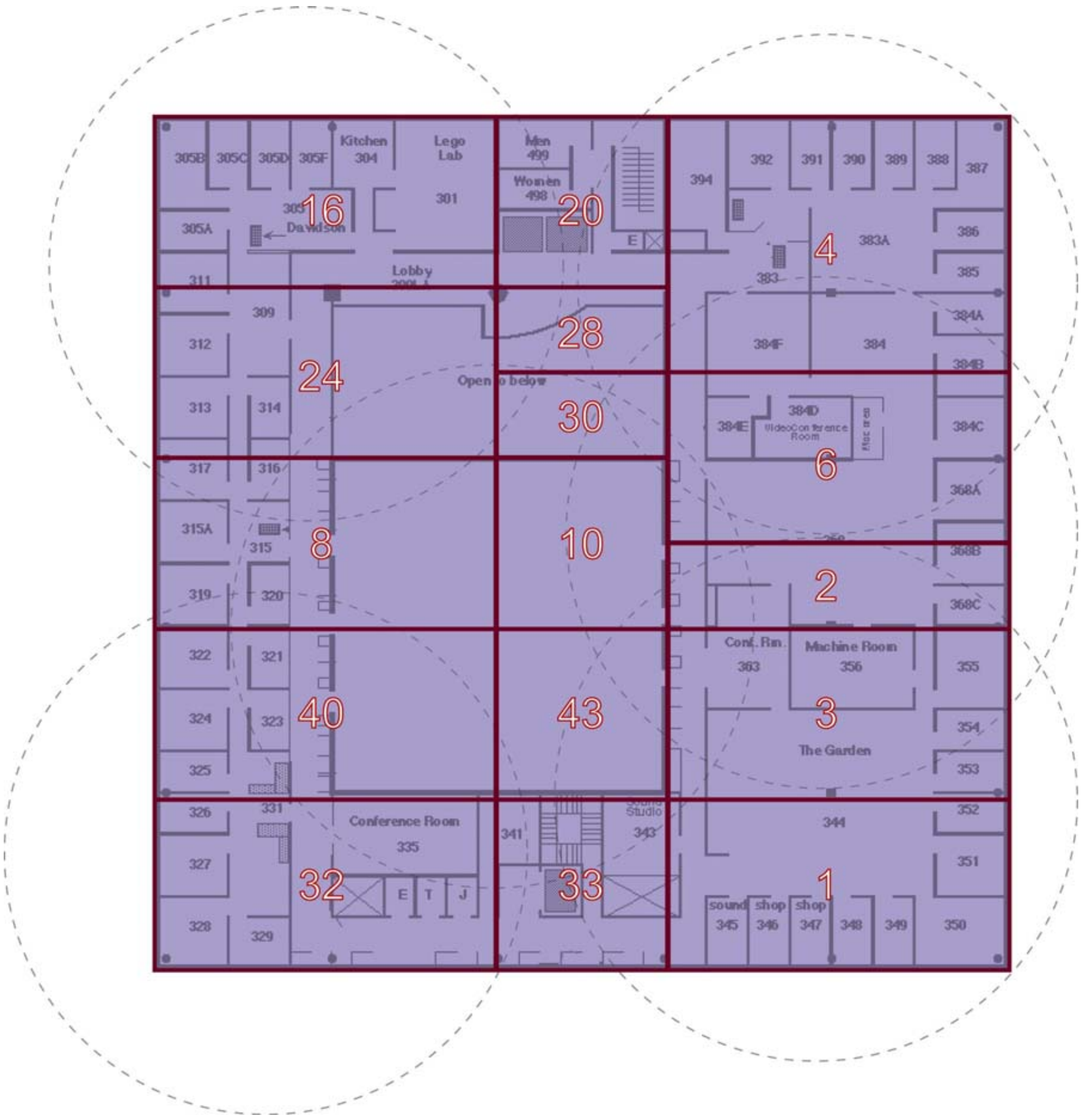


Figure 4-15: Distinct zones formed from overlapping beacons in the location system.

The nodes were then prepared, first by programming the application-specific firmware into each node. The firmware for this test closely follows the flowcharts in Section 4.3. This test application firmware uses all three identified states (idle, attached, sensing) and specifically enables and disables peripherals and alters the sampling rates accordingly. This allowed the application to be optimized for power usage.

The firmware for this test run logged the sensor data, state changes, and location information to the flash memory for the entire test run, regardless of state. Special modes were added for retrieving the data from the nodes via the Bluetooth location system and for general health monitoring.

The nodes were then given their specific behaviors. Of the ten nodes released, six were told to try and get to specific geographic locations using the distance checking algorithm from the software simulator. The version of this algorithm that runs on the embedded platform in the node is quite scaled down from the algorithm used in the software simulation due to processing power restrictions. The algorithm used in the nodes just simply uses the last two locations it recognized to guess the next location that it will be brought to based on a linear extrapolation. If this guessed location is closer (with an adjustable threshold) to its goal than the current position, it will stay in the attached state; if it is farther away from the goal than the current position, it will try to detach. Detachment for this test involves vibrating the pager motor and flashing a red LED in an attempt to be put down. It also maintains a counter of how many detachments it has attempted and will ignore its distance-checking algorithm if it has hopped multiple times and not gotten any closer to the destination. This prevents getting stuck in a situation such as being at a dead end, where all hosts will take the node away from the goal, at least temporarily.

The remaining four nodes were given specific sensor conditions to look for. All ten nodes will constantly collect sensor data throughout the test, but these nodes are also programmed with desired sensor conditions that will cause them to detach. The node will then stay in this location to observe the phenomenon it has found interesting for a pre-determined amount of time, or until the sensing condition disappears. If the node is picked up from this state or from a state where it has reached its geographic goal, it will immediately attempt to detach until its, time spent at the goal or sensor point of interest has elapsed.

The batteries were then completely charged and the nodes were sealed into their plastic spheres as shown in Figure 4-12 and labeled with the instruction label shown in Figure 4-13. To additionally aid the attachment and detachment process, an email was sent to the building's inhabitants telling them to keep an eye out for the spheres, and to feel free to pick them up and carry them around, being sure to put them down when they shake. No information was given about the devices or the test, and the nodes all looked identical, regardless of their goal. There were about 40 people in the building when the test started.

The nodes were then deployed in a high-traffic hallway, where they would wait to be picked up. The positions of the nodes can be monitored from any PC on the network. Within fifteen minutes all of the nodes had found their way to a new location. Some were knocked around and rolled, and some were picked up a brought around to new locations. The people carrying them mostly obeyed the device when it shook and wanted to be put down, sometimes even tossing it away, startled by the vibration.

The batteries lasted for close to four hours. This is discussed further in Section 4.4.2. While the test was running, the nodes collected sensor data at a rate of 30 Hz. The sensors were sampled faster when the power modes allowed it, but logged at a rate that would let the flash storage last for the duration of this test. The nodes collected location data, attachment data, state changes, sensor data, and the time when they reached their goals. The networked location system also recorded the trajectory of each node on its central computer.

The test generally ran without a hitch, other than the disappearance of one of the ten nodes. This is an expected loss considering the unknowns of the host's behaviors. This node was thought to have been carried outside the range of the location system, purposefully stolen, or had a power issue. Trajectory data for this node was still able to be recovered from the location system up until it disappeared. Fundamental to the concept of parasitic mobility is that the nodes are cheap and the potential to lose many nodes is made up for with redundancy of inexpensive nodes. The node was found one week later and its sensor data was recovered.

Another node got locked inside an office soon after deployment and remained there for the duration of the battery life. And one other node was discovered to have manufacturing defects preventing it from recording data to the flash, so it was removed from the test. The remaining eight nodes easily covered the test area.

4.4.2 Power Usage Discussion

As mentioned above in Section 4.4.1, the test lasted for 4 hours on a single charge of the battery. The battery that was used has a capacity of 145 mAh, however, the protection circuitry on the node’s power layer disconnects the battery when it has drained to 3.0V. At this point the battery still has 20% of its capacity. Accordingly, the nodes had an average current draw of just less than 30 mA. This test run incroporated many power optimizations, but also had many features enabled for the logging and observation of the test operation such as the constant flash writes, data dumps over the wireless network, and health monitoring communications with the observer’s PC. Further power savings could be achieved by cutting the output power of the radio transceiver from Bluetooth’s specified high power mode to its normal output power mode.

The following two sections, Section 4.4.3 and Section 4.4.4 present the actual trajectories the nodes traveled and an environmental mapping observed by the sensors, respectively.

4.4.3 Trajectory Data

The following Figures show the floorplan of the area used for this test and the trajectory that the nodes followed, including any major stops. These trajectories are re-created from the log files of the location system and the data collected and recovered from each node. The position data can have a refresh rate as slow as 3 minutes, so some interpolation is done based on the layout of the floor and regions where people can walk free from obstacles. Furthermore, since the resolution of the location system is fairly sparse, some activities that happened completely within a single zone of the location system are omitted from these maps. For example, if a node is picked up that has reached its goal or is in an area that it does not want to leave, it shakes to be put back down, hence (ideally) is released into the same general area. This scenario does not show up in these maps because they are generally not important to the trajectory discussion. However, all significant attachments and detachments leading to changes in trajectory or to the completion of the node's goal are shown with timing information. Section 4.4.4 presents the sensor data which gives a more detailed narrative of the individual node's encounters.

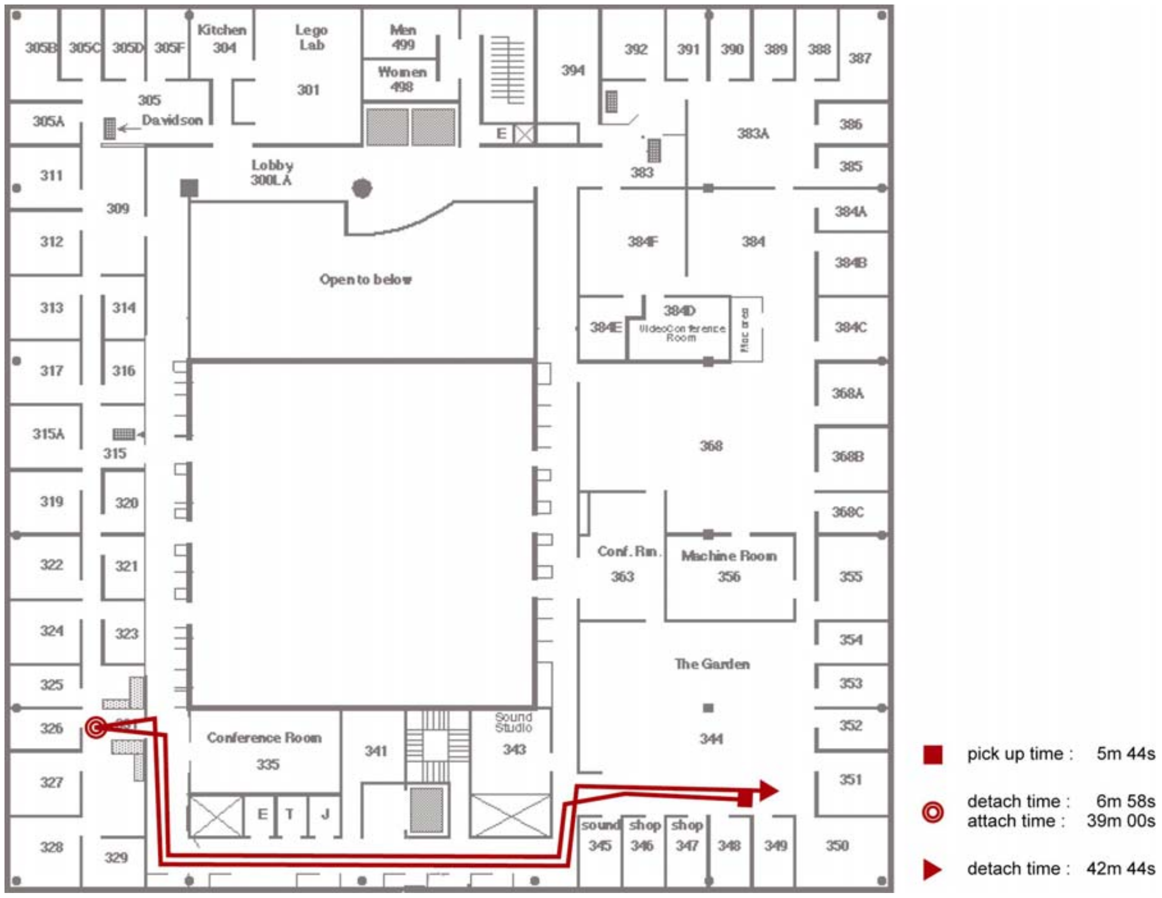


Figure 4-16: Trajectory for node #1

Node 1 was given the orders to try a find a specific geographic location. This location was zone 1 as shown in Figure 4-15. Since this is the deployment location, this node was also told to wait until it has crossed through a few other zones to try and find its goal. It returned to its goal after 42 minutes, where it remained and was promptly released every time it was picked up. It had one significant detachment, after it filled the requirement of leaving the original zone; it eventually realized it was going further away from its goal and detached outside room 326. It waited there for 33 minutes until a suitable host came by and picked it up; this host was walking towards its goal, it hung on until its destination was reached. According to the data collected, it took close to 4 minutes to walk the length of this hallway. This data is taken from the location system which has a refresh rate dependent on many factors, including the number of other Bluetooth devices in the area; it can refresh as quick as a few seconds or as slow as every 4 minutes.

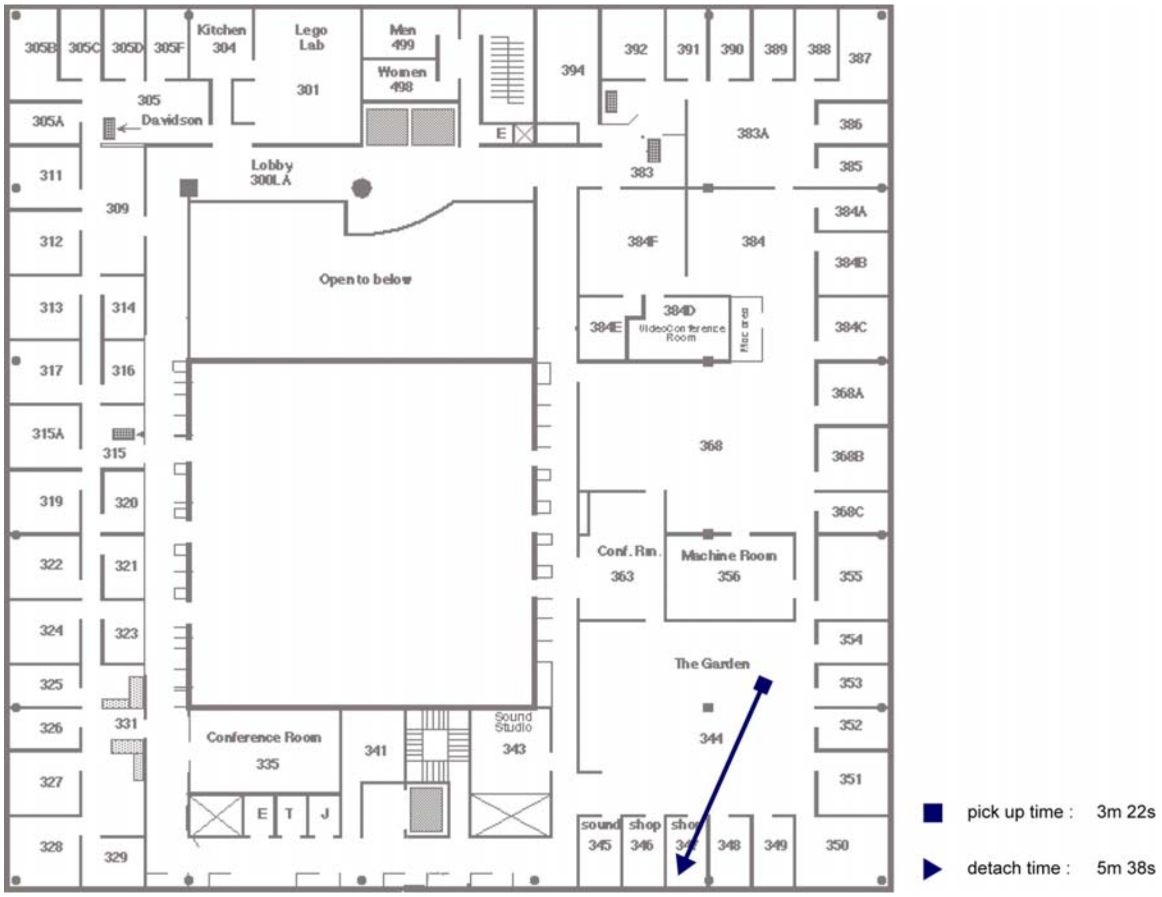


Figure 4-17: Trajectory for node #2

Node 2 was given orders to try a find location zone 2. Location zone 2 is one of the more difficult locations to find, because it is a narrow zone existing where beacon 2 is not overlapped by any other beacon. When the map was built, it turned out that beacon 2 was mostly overlapped by beacons 1 and 4, leaving only a sliver of beacon 2 left to build zone 2. Unfortunately, node 2 wound up being locked in an office within the first 5 minutes of the test and could not be recovered until the next morning, long after the batteries had died, leaving its goal unfinished. With better power management techniques, such as those discussed in Ari Benbasat’s paper on adaptive power management [39], this situation does not have to drain the batteries, even though it is technically considered to be in a sensing state. These techniques allow low-powered sensing based on using exactly as much sensing power is needed to accurately depict the environment.

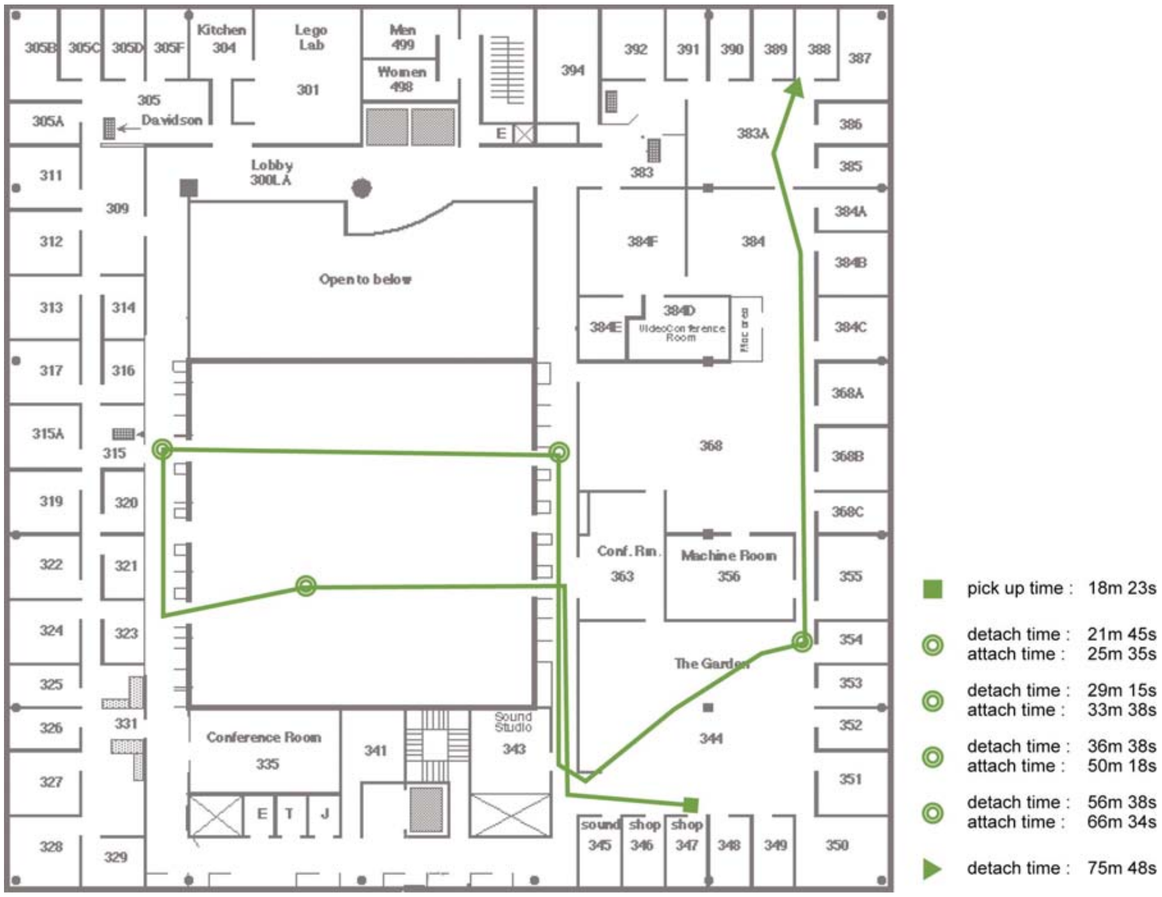


Figure 4-18: Trajectory for node #3

Node 3 was given the orders to find its way to location zone 4, which is in the top right corner of the map. It found its way there in 75 minutes. The batteries lasted for around 4 hours; however, since the test was started in the evening, the number of people in the building was decreasing and fell to almost none at around 2 hours into the test. So it can be assumed that very little or no mobility happened after this time. The nodes did continue to sense and collect data.

Node 3 made a fairly large loop going away from its goal. This area of the map is the sparsest as far as the location system goes, so the detachment algorithm for finding the specific goal has trouble here. However, it did detach 3 times in the span of 15 minutes, showing that it was trying to get back on track, which it eventually it did.

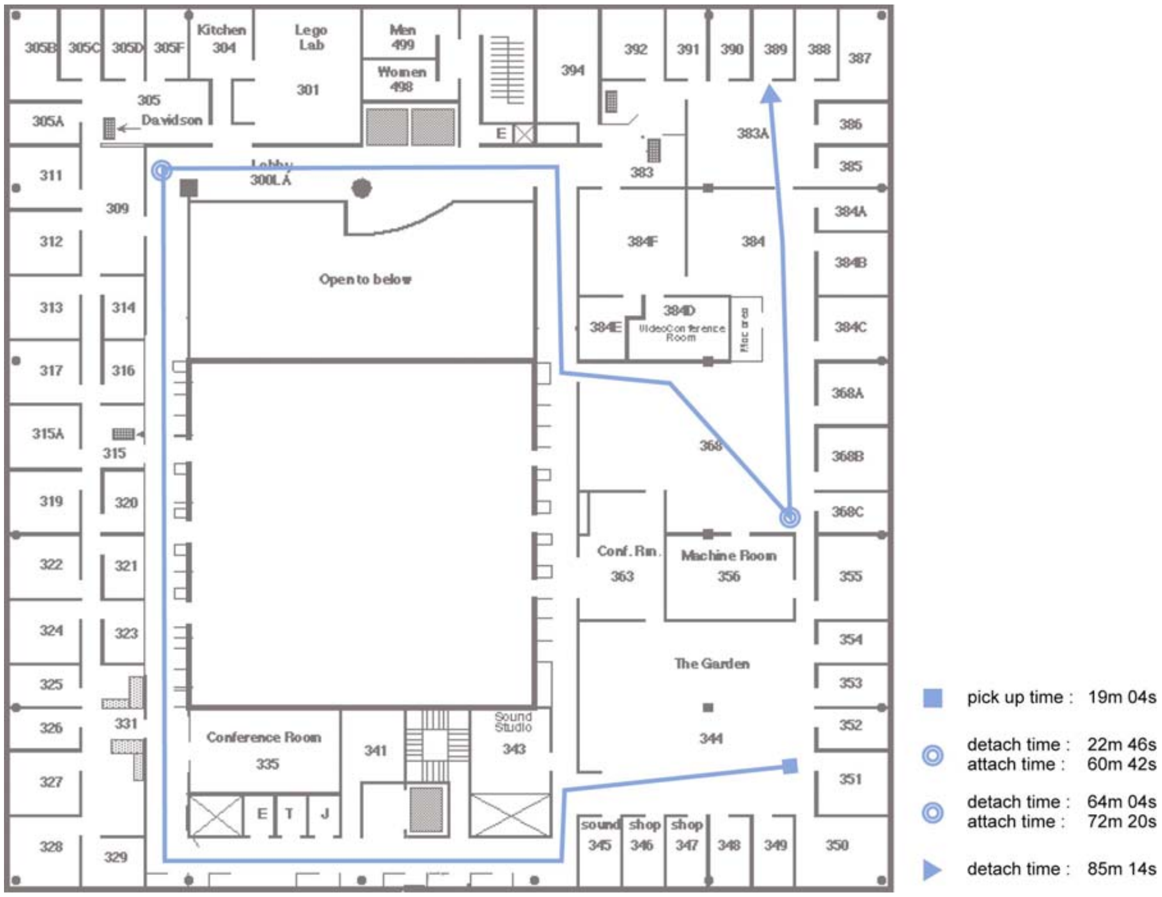


Figure 4-19: Trajectory for node #4

Node 4 was told to try and find location zone 8, which is located around the left side of the open area in the center. It appears that the node passed right through this zone. This means that either the person carrying the node ignored its wishes to be dropped off or it was carried too quickly across the zone to react. This was the node that disappeared and was recovered one week later. The sensor data recovered shows a lot of accelerometer activity during the time that it was in-between zone one and zone 16. The logs show one attempted detachment at the goal location, but no stopping of motion. It seems that the host ignored its call for detachment.

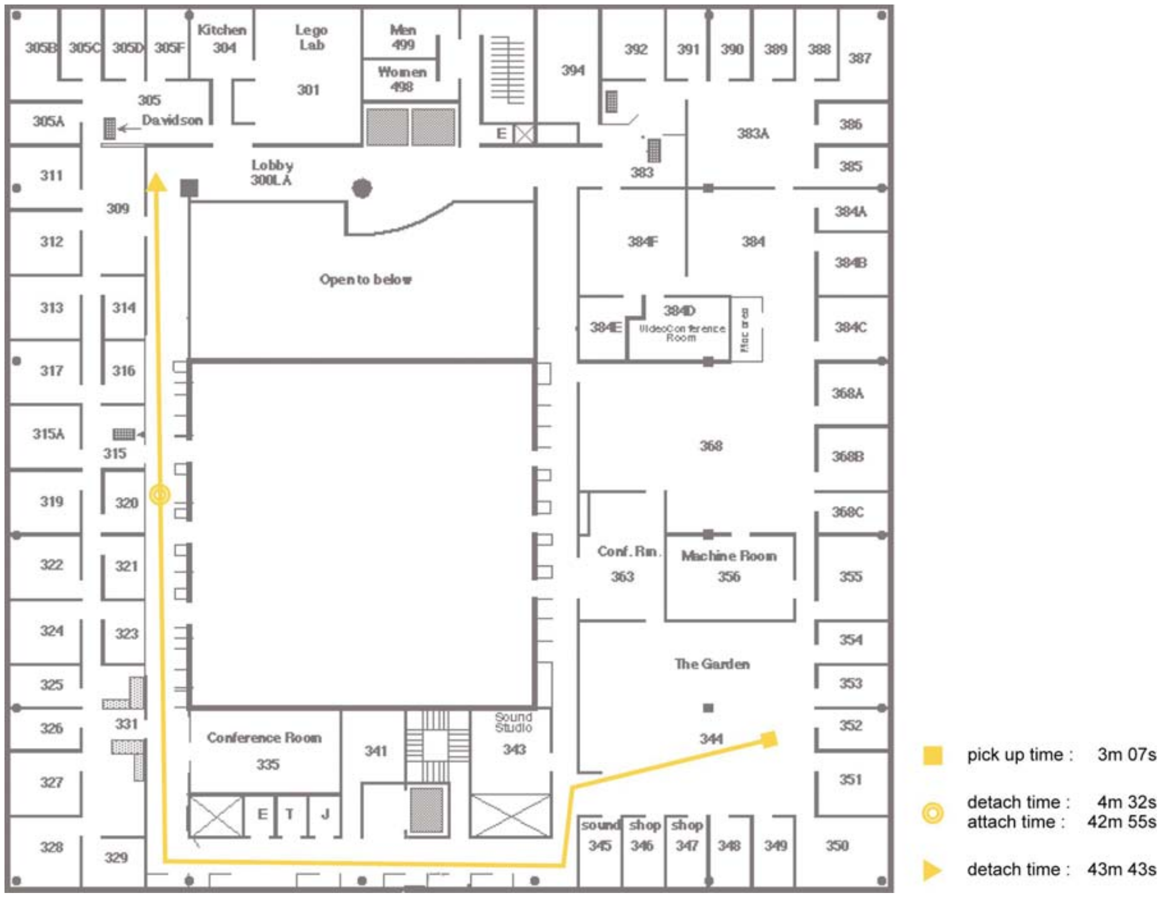


Figure 4-20: Trajectory for node #5

Node 5’s trajectory was pretty straightforward. It was told to try and get to zone 16 which is in the top left corner of the map. It made it there in 43 minutes with one significant detachment due to the host changing course and leaving it in the hallway when it shook. It waited 38 minutes for another host to come by and took it to its destination.

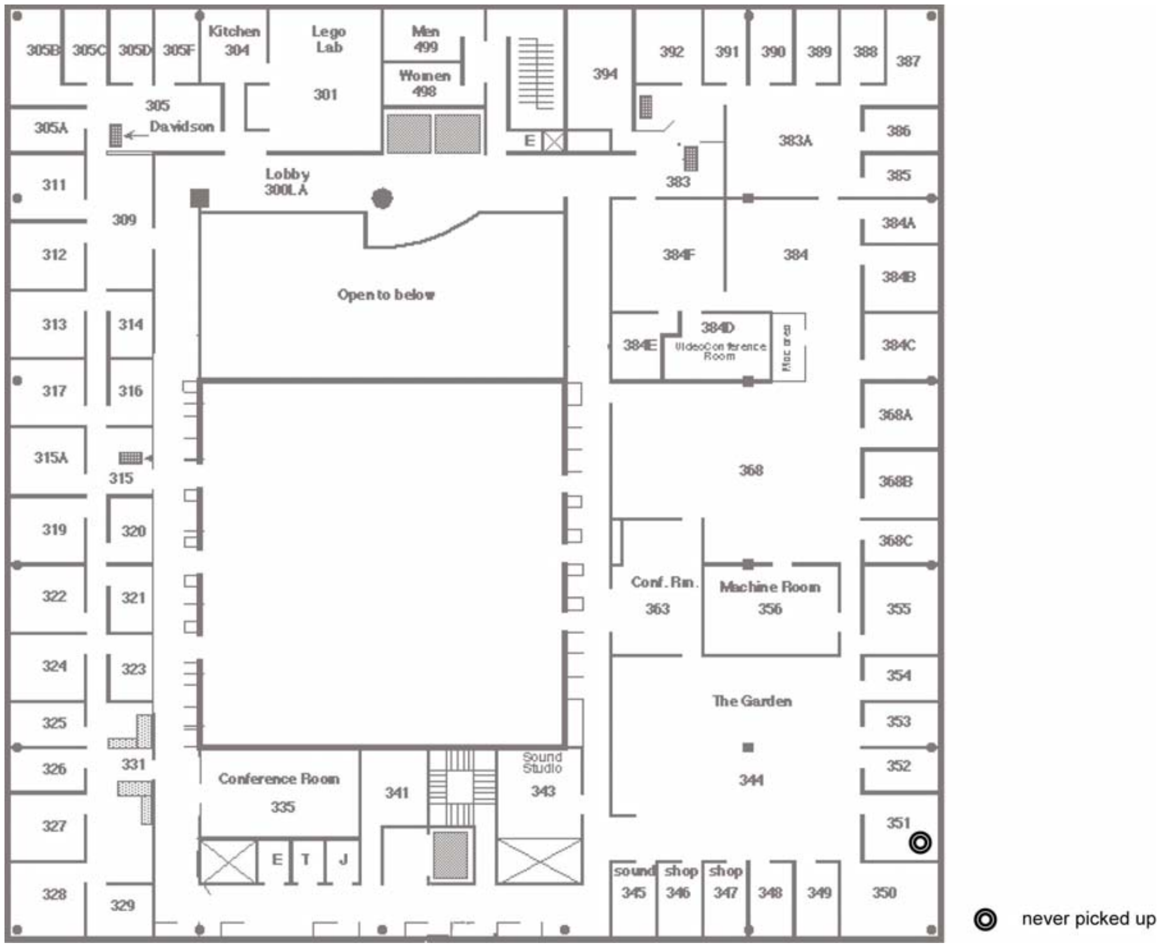


Figure 4-21: Trajectory for node #6

Node 6 was the last of the nodes told to go to a specific location. It had some hardware problems and was removed from the test.

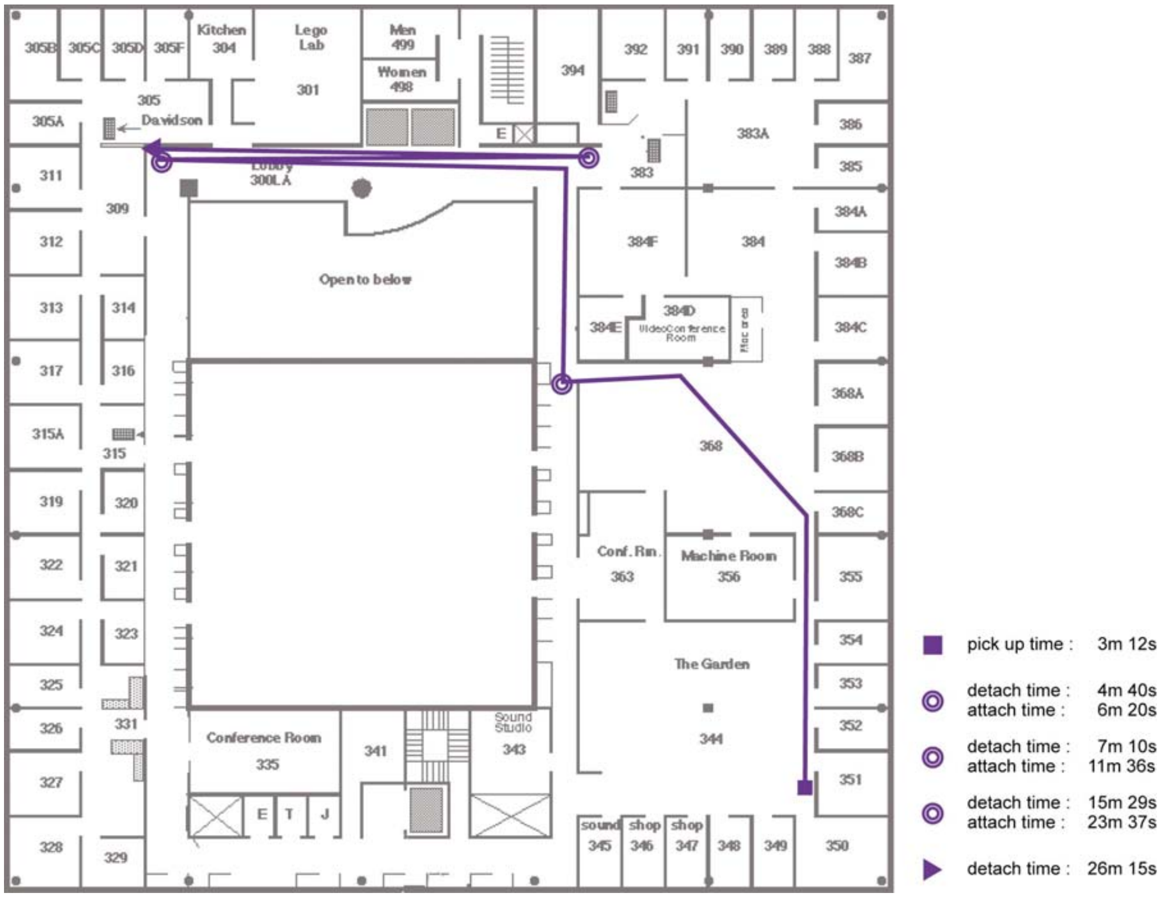


Figure 4-22: Trajectory for node #7

The remaining four nodes whose trajectories are shown in Figures 4-22, 4-23, 4-24, and 4-25, were given environmental conditions to try and find. The first two of these nodes were told to look for bright lighting conditions. When an acceptable condition is found, the node will detach and start a sensing timer. If it is picked up before this timer runs out and the sensing condition persists, it will immediately ask to be put down. If the sensing condition becomes uninteresting and it gets picked up again, it will take the ride. If the sensing timer runs out and the sensing condition still exists, it will still catch the ride. If it returns to the same location or any other location where the sought-after condition is present, then it will detach and start the timer again. The duration of the sensing timer is set as part of the node’s behavioral orders.

Node 7 found a location in the top left corner that it considered to be of interest, in this case contained a bright light source, as the node was instructed to seek bright illumination. The data recovered from the node showed that this was indeed the case, but that the lighting condition dipped and became uninteresting. Since the nodes are spherical, how the node winds up being oriented becomes important for the light sensor especially. According to the data from the node, this node was picked up but then noted a lighting condition that was below the threshold for it to consider it interesting. In fact, it was far below the level that it had seen before it was picked up. This indicated that the person who picked it up must have covered the light sensor. The node therefore thought that the location was undesirable and did not ask to be put back down. When it returned the second time, it was better oriented and noticed the lighting condition, detached, and remained there for the rest of the test. This took only 26 minutes to happen, meaning that the node was able to fend off hosts picking it up for the remaining hour and a half of high traffic time. The data collected shows 4 attempted pick ups, all successfully ending in the node being placed back in the same location. This can be seen in the sensor data graphs of Section 4.4.4 as areas where the location curve is flat and there are level shifts (re-orientation) in the accelerometer data.

The issue with the sensor readings being influenced by the way in which they are carried and how they are oriented when at rest can possibly be improved by the addition of redundant and symmetric sensors to each node.

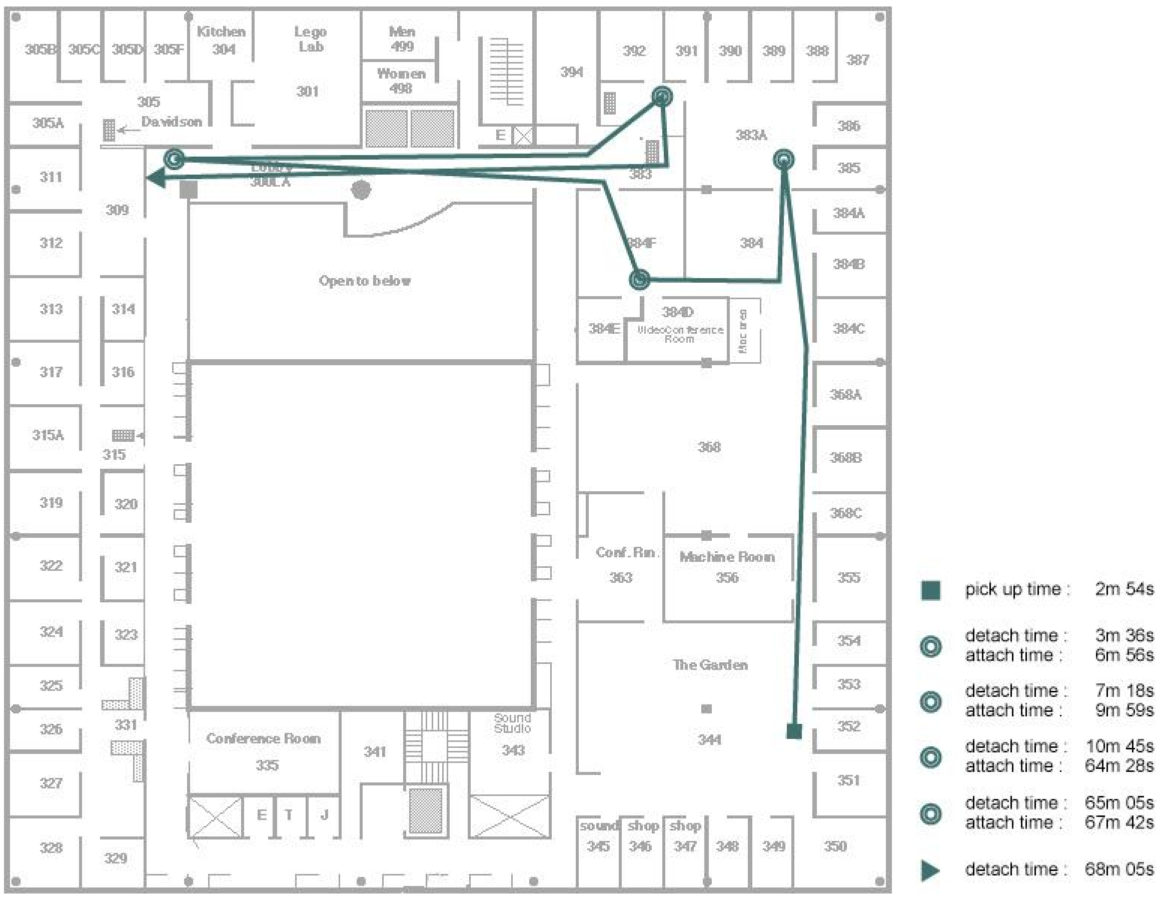


Figure 4-23: Trajectory for node #8

Node 8 was given the same orders as Node 7. It also wound up in the same spot as Node 7, verifying that it is a spot of brightness. In actuality, it is a spot that receives light from several sources and is clearly the brightest spot on the floor on the average.

Node 8 did a similar zig-zag as Node 7. It found the location, but then was taken away briefly and found its way back. However, unlike Node 7, this node allowed itself to be picked up because it had exhausted its sensor timer. The end result was the same, and it returned to the area of interest. Both nodes remained there until the end of the test.

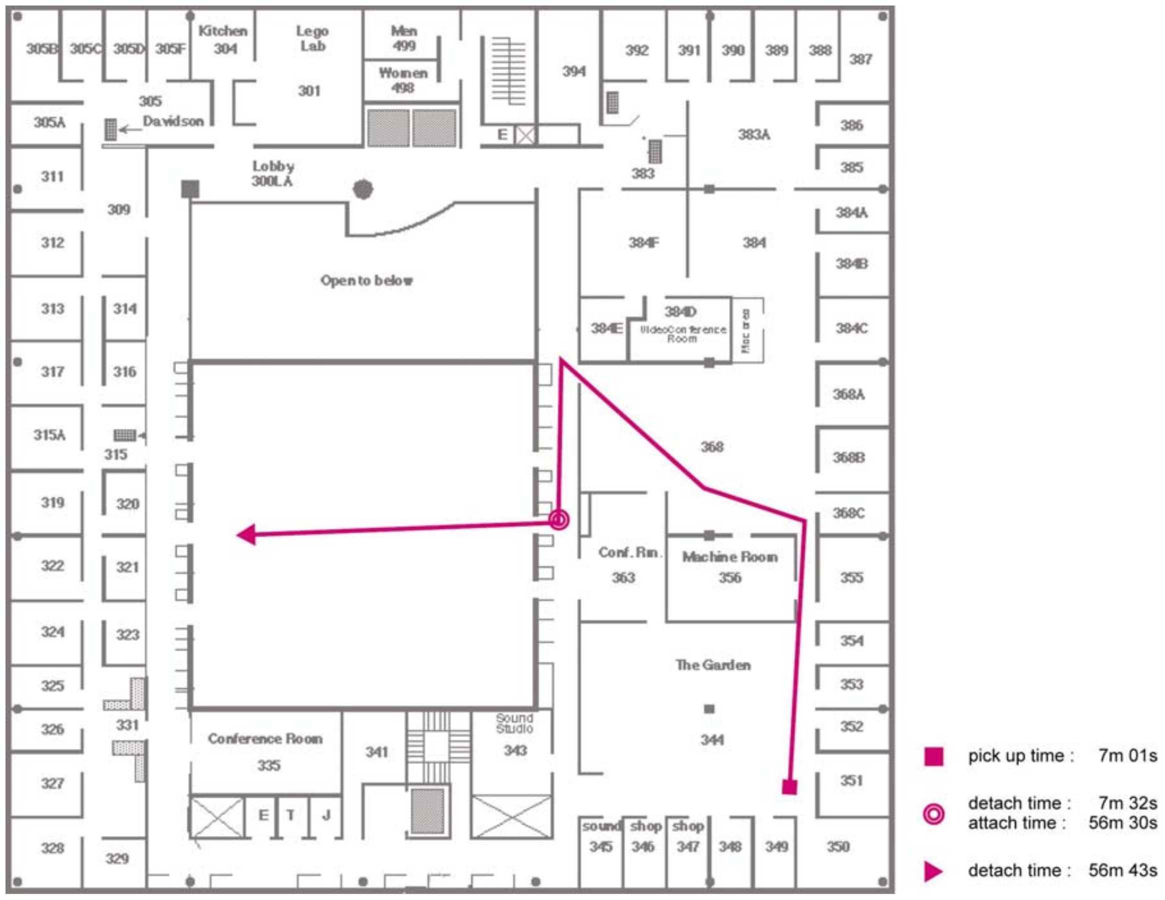


Figure 4-24: Trajectory for node #9

Node 9 was told to look for an area of high temperature. It found one right away at its first detachment location. It stayed there until it exhausted its sensor timer. When it was picked up again, it fairly quickly found another location that met its criteria. In fact, it was encountered as soon as the node crossed into the next zone. This indicates that this area was most likely a few degrees warmer on the whole.

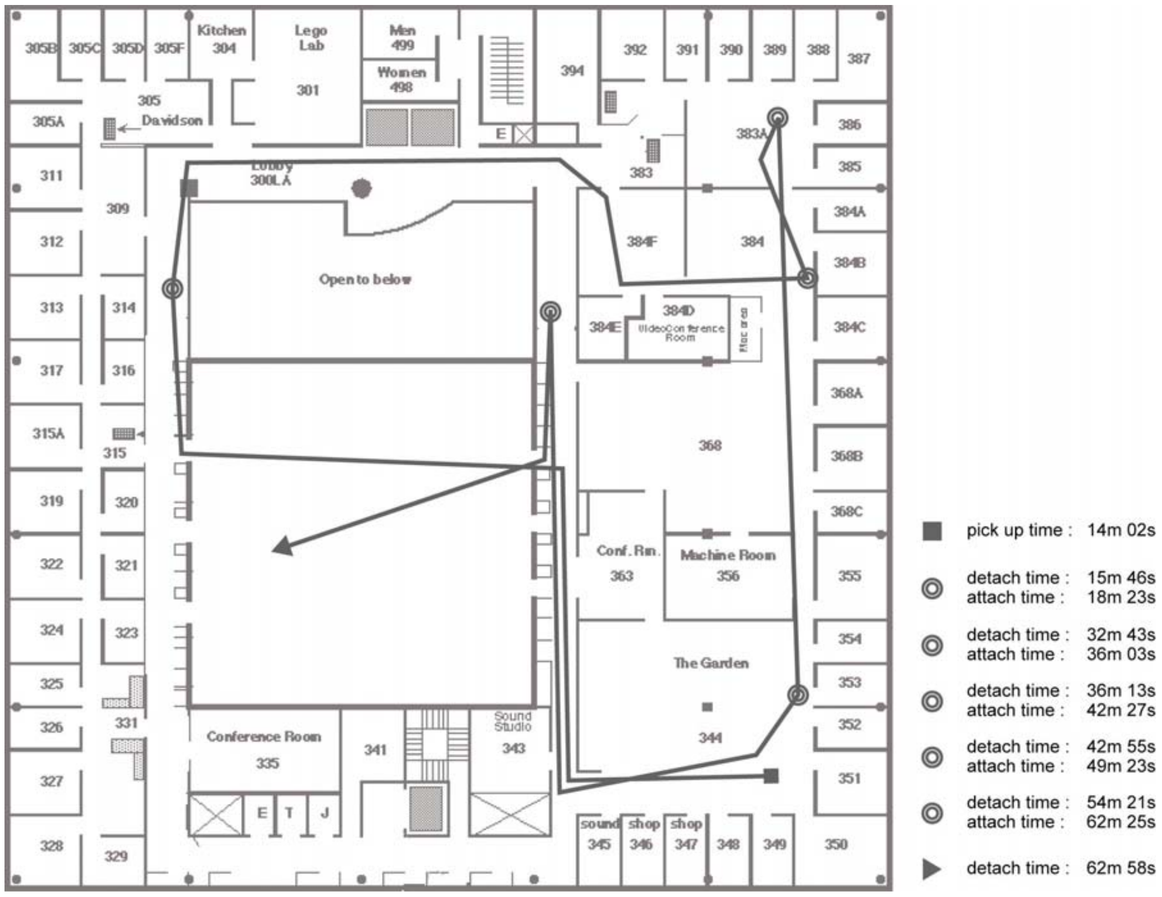


Figure 4-25: Trajectory for node #10

Node 10 was given similar orders as Node 9 but with a higher degree of temperature to look for. It passed right by the area that Node 9 had detected as interesting, but since it was in between the thresholds on the two nodes, Node 10 did not detach. It came back close to an hour later and the temperature was now high enough to now find it acceptable to detach and start sensing.

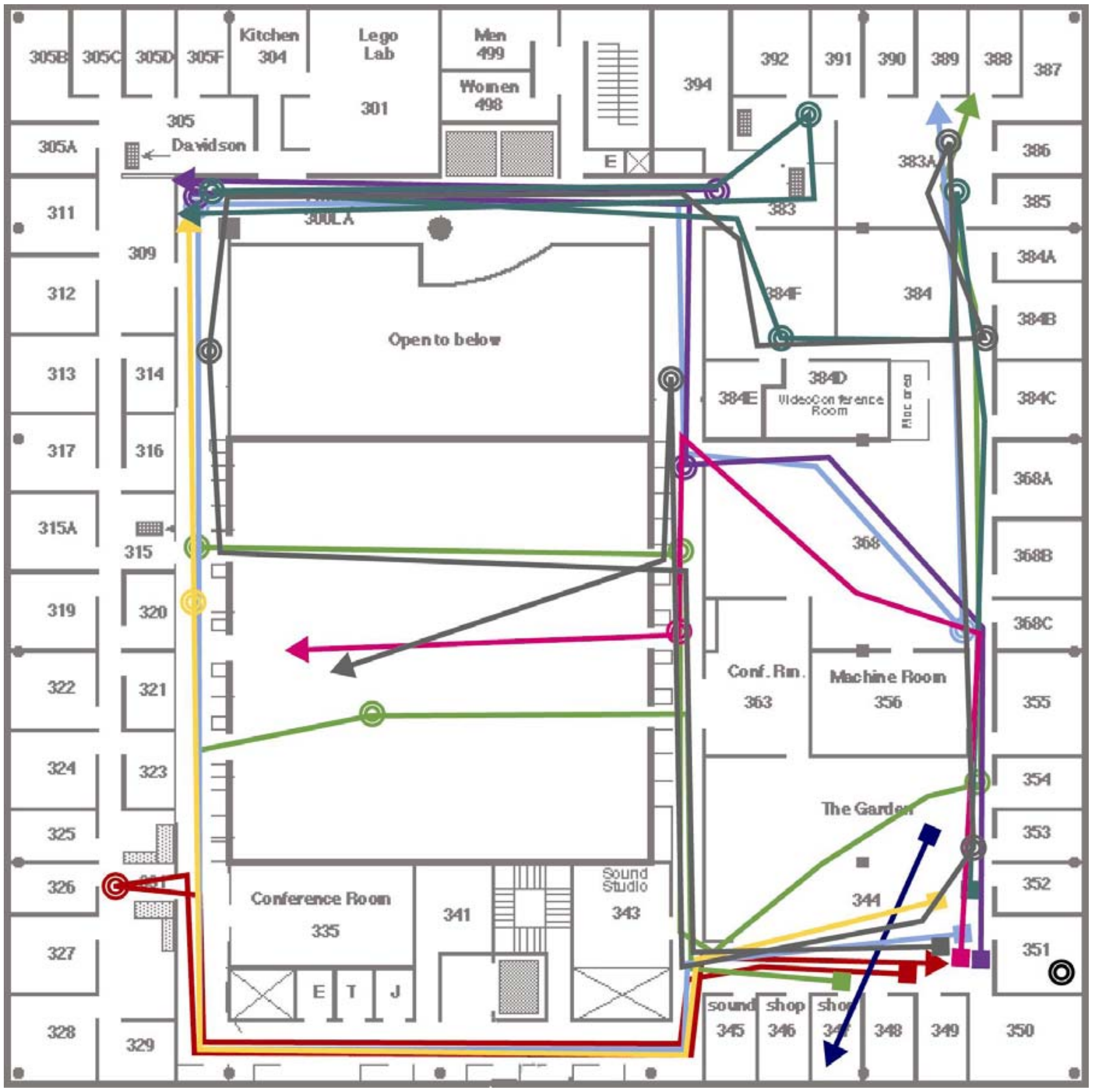


Figure 4-26: Trajectories for all nodes

Figure 4-26 shows the travels of all the nodes. It shows that the nodes pretty covered many times over all the publicly accessible areas of the floor. Figure 4-27 shows which node was in which zone at what time. This information is presented in greater detail in Section 4.4.4.

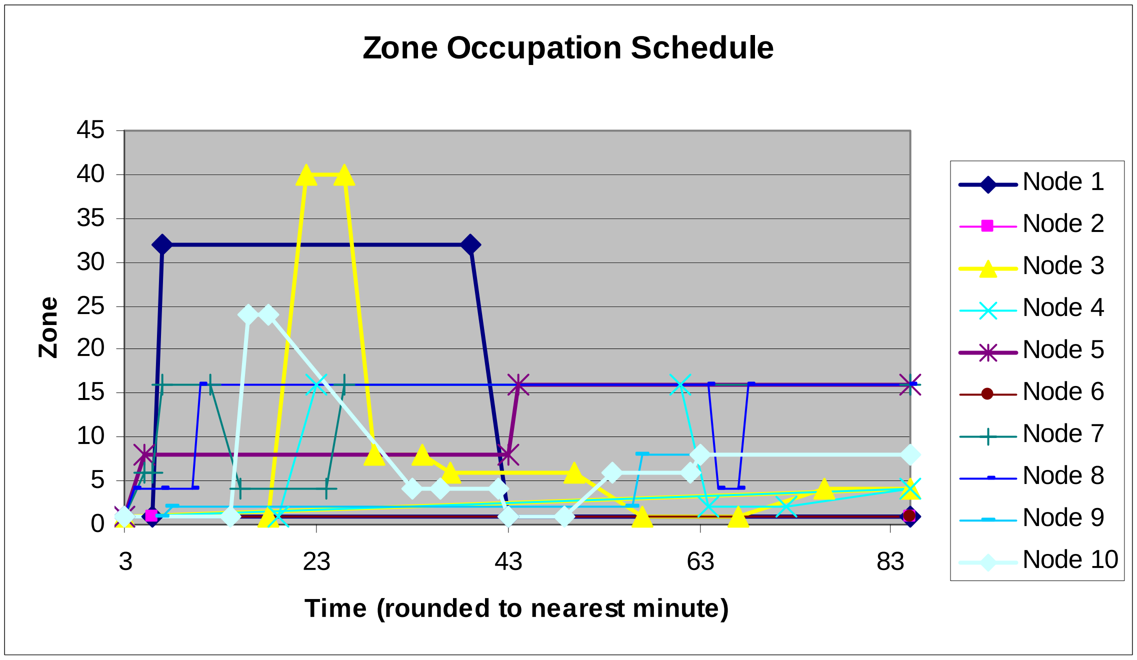


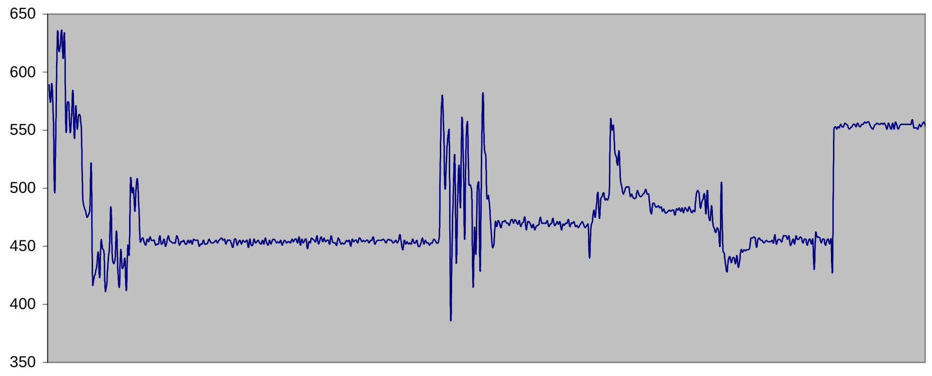
Figure 4-27: Occupation schedule for zones (which node was in which zone at what time)

4.4.4 Sensor Data

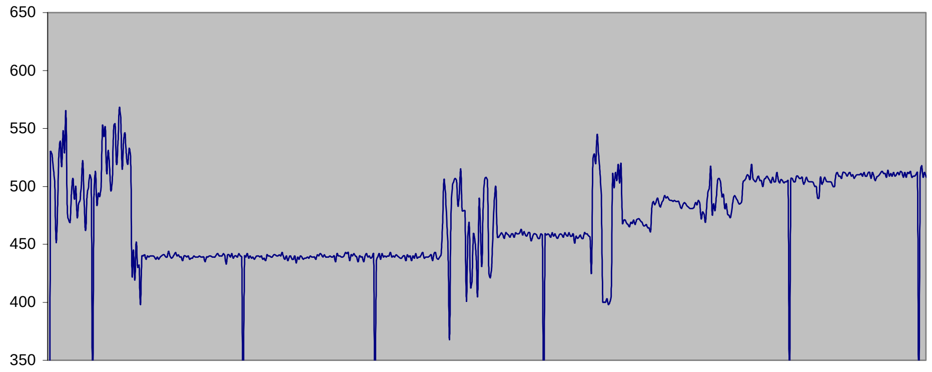
This section presents the sensor data retrieved from the nodes after the test was complete. The nodes had enough storage to store just less than 2 hours of sensor data at around 30 samples per second. This proved enough as little or no activity happened after 85 minutes and the trajectory data discussed in Section 4.4.3 does not pass this time point. The sampling rate varied slightly from node to node and some of the nodes used up their memory faster than others. The data is presented normalized across all the nodes for comparison. For each node, six subplots are shown all with a common X axis of time in seconds as shown on the location plot. The position vector is from the location system which is less resolute than the sensor data; therefore there is usually some slight framing error correlating the sensor data with the location data. This is minor and it is quite easy to observe the relationship between the trajectory activity and the sensor activity.

Node #1 Sensor Data

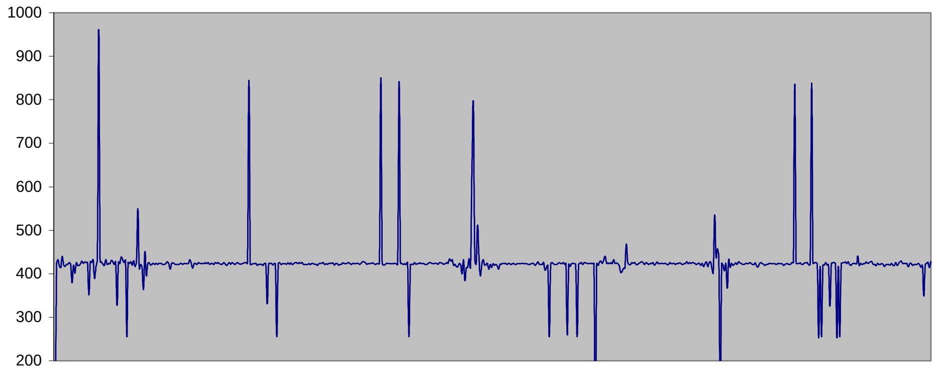
Accelerometer X



Accelerometer Y



Audio Magnitude



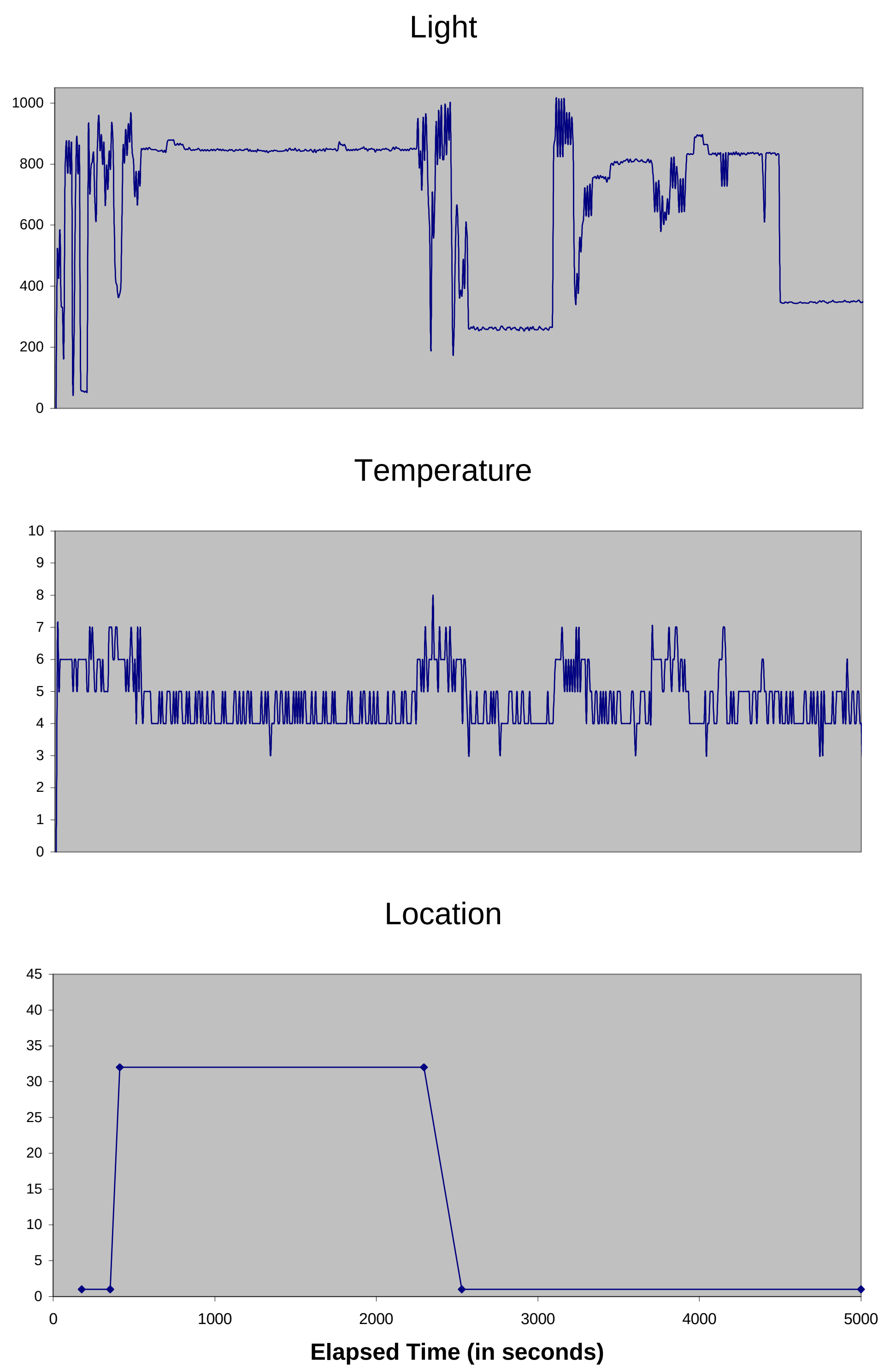
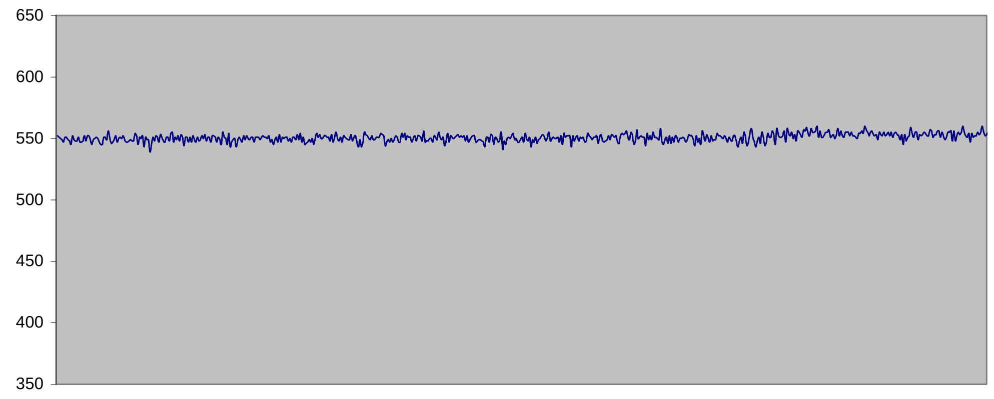


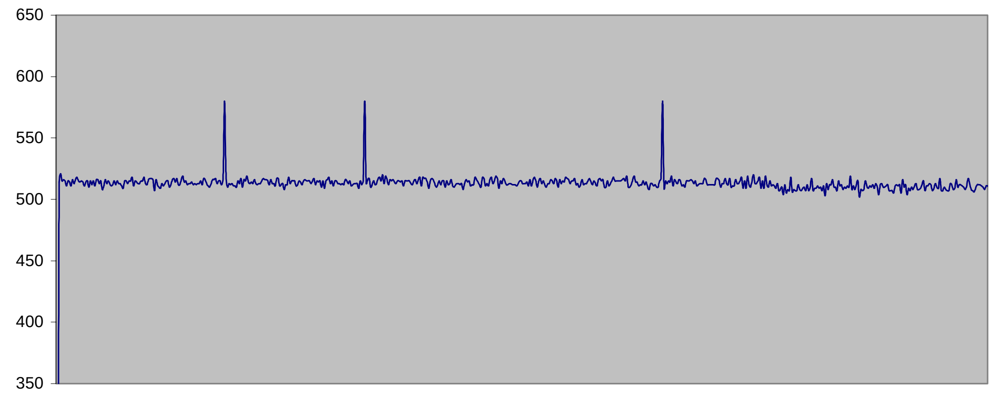
Figure 4-28: Sensor Data for Node #1

Node #2 Sensor Data

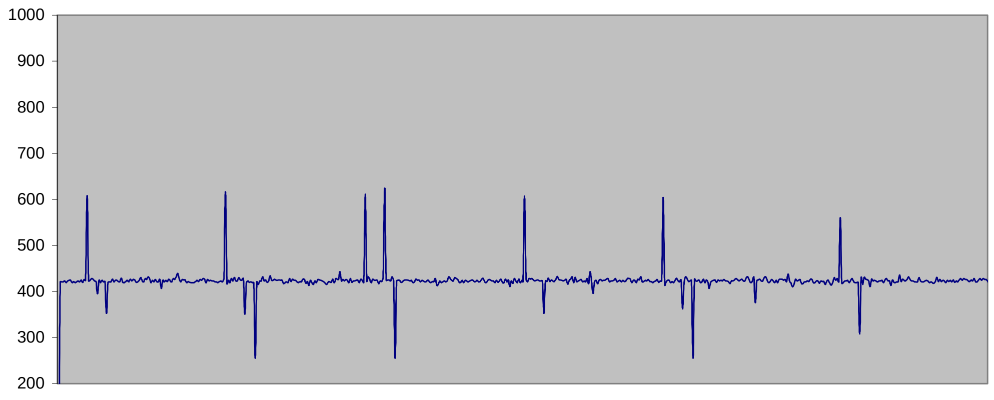
Accelerometer X



Accelerometer Y



Audio Magnitude



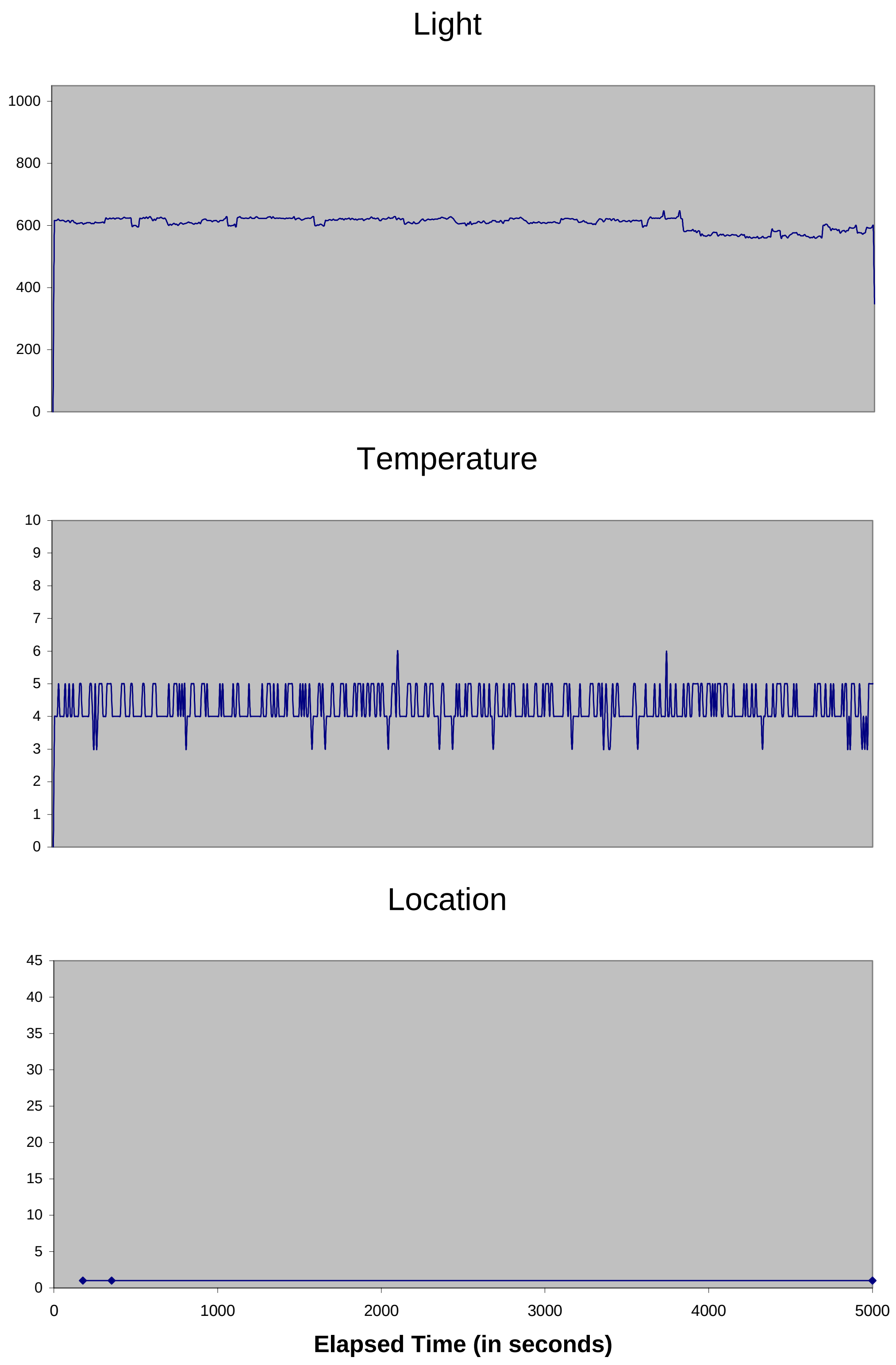
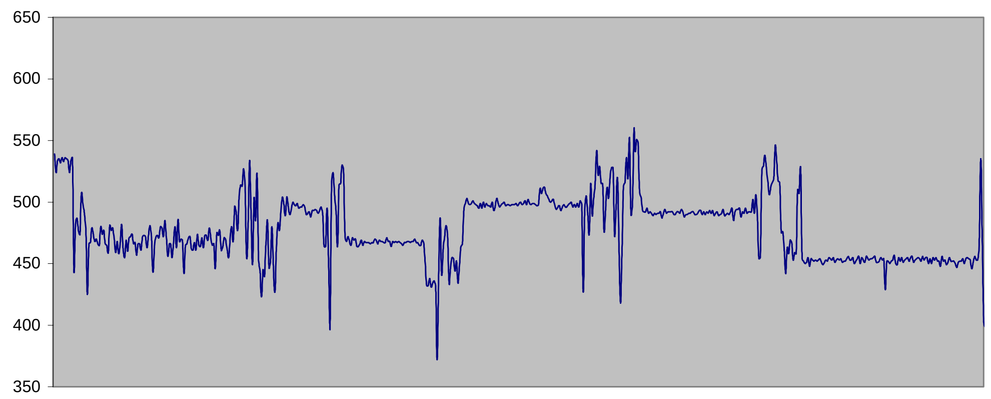


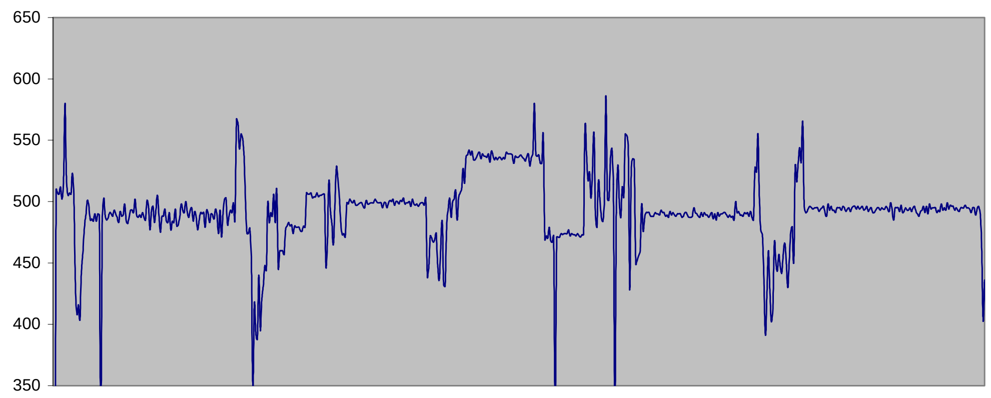
Figure 4-29: Sensor Data for Node # 2

Node #3 Sensor Data

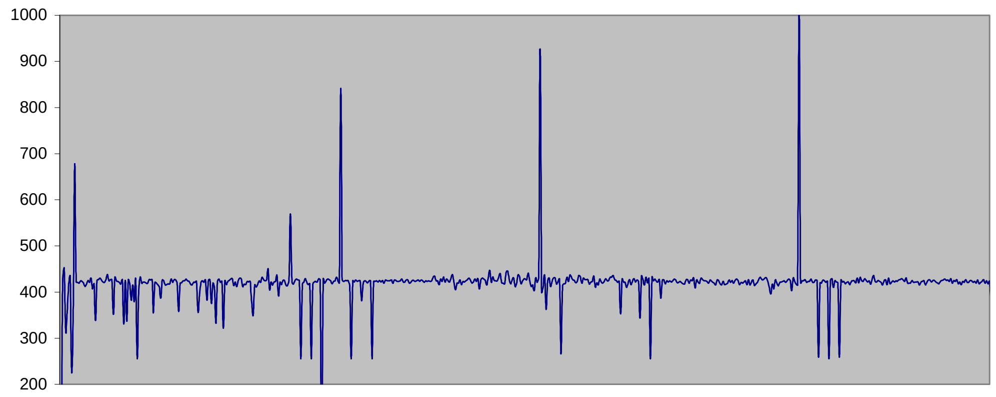
Accelerometer X



Accelerometer Y



Audio Magnitude



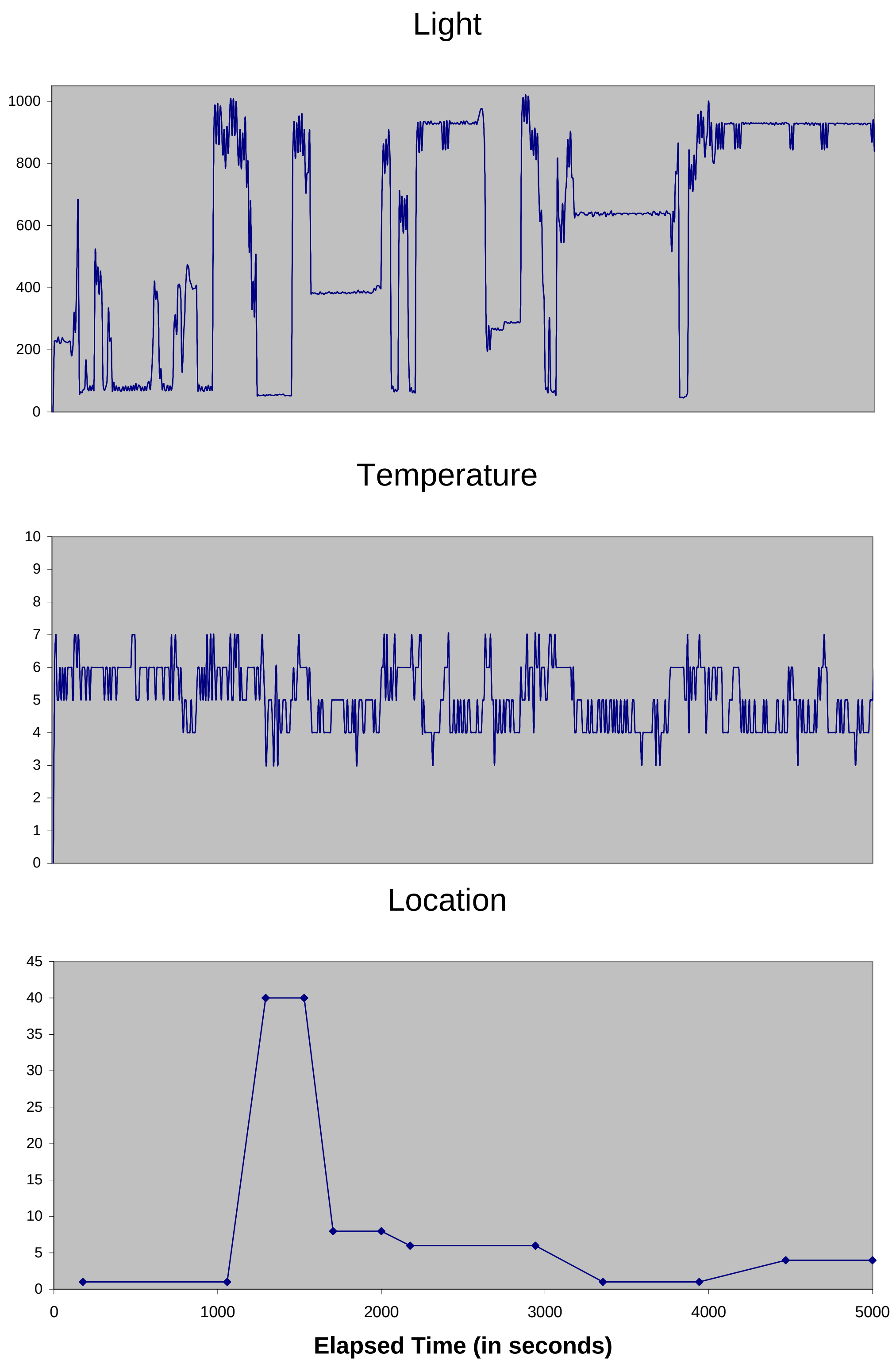
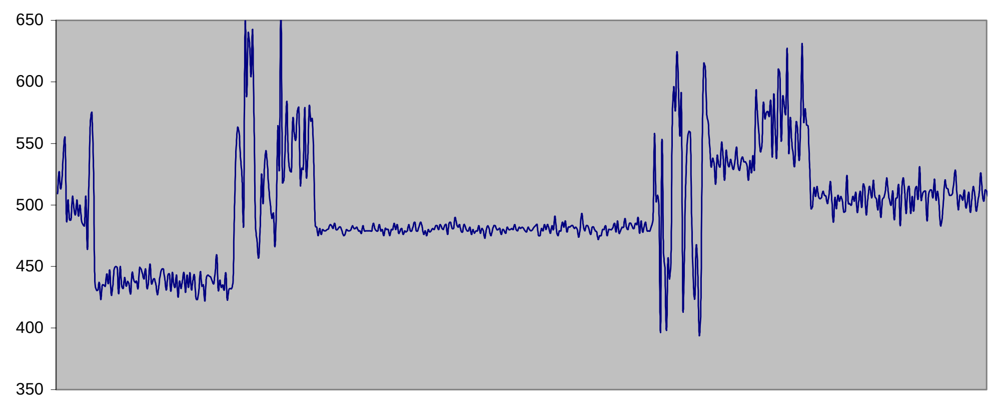


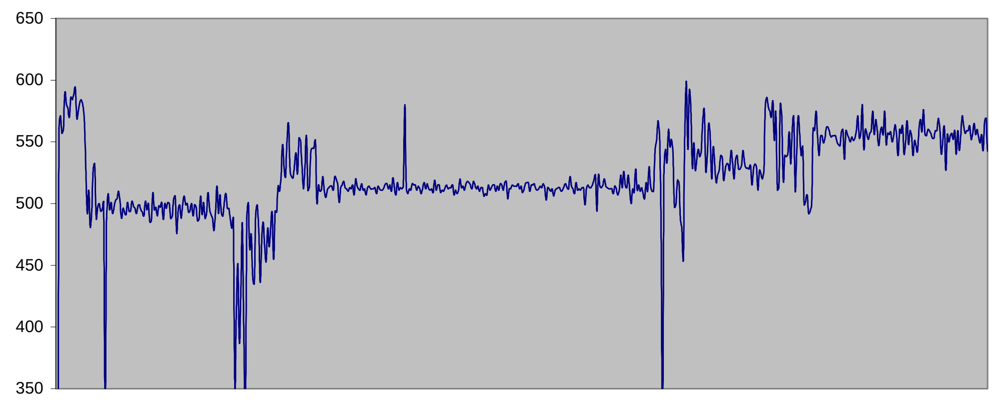
Figure 4-30: Sensor Data for Node #3

Node #4 Sensor Data

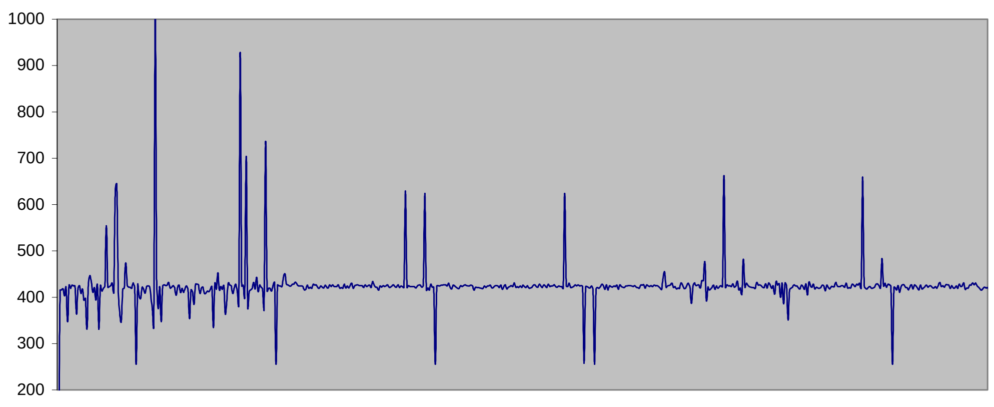
Accelerometer X



Accelerometer Y



Audio Magnitude



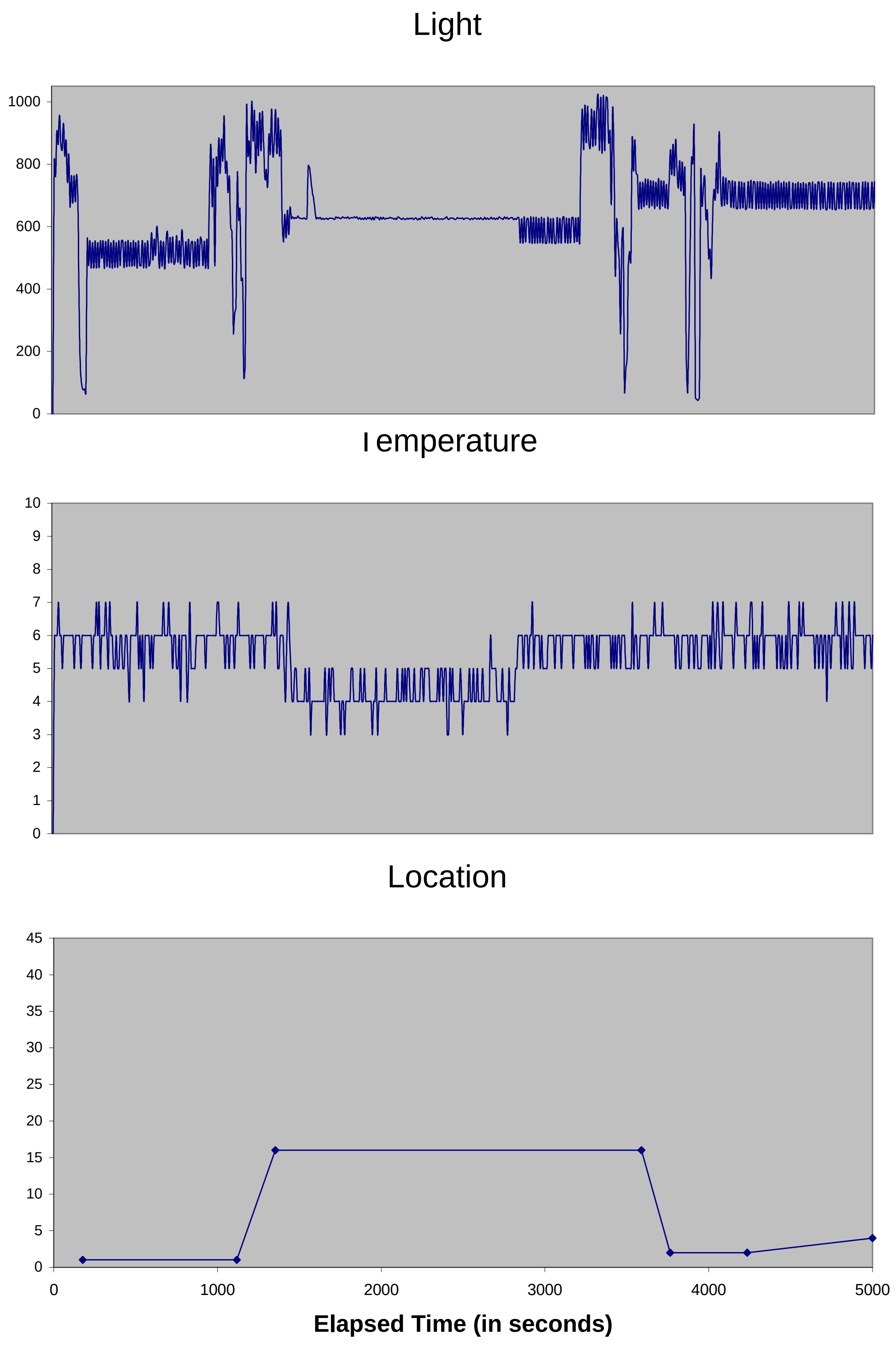
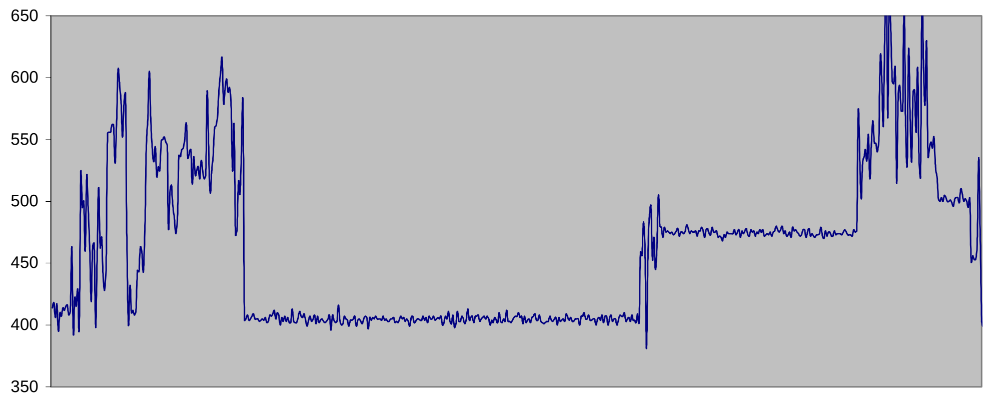


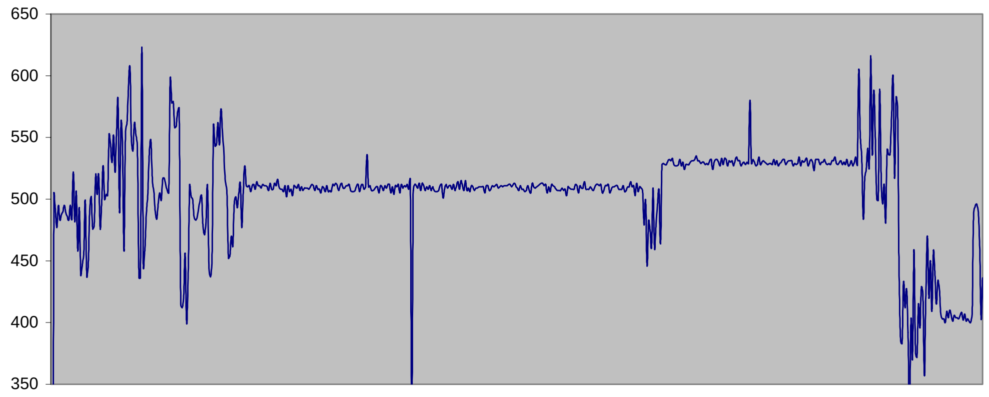
Figure 4-31: Sensor Data for Node #4

Node #5 Sensor Data

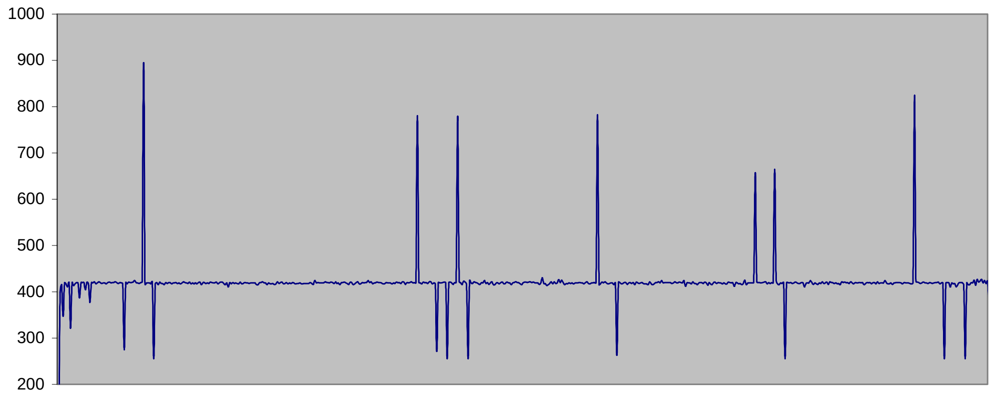
Accelerometer X



Accelerometer Y



Audio Magnitude



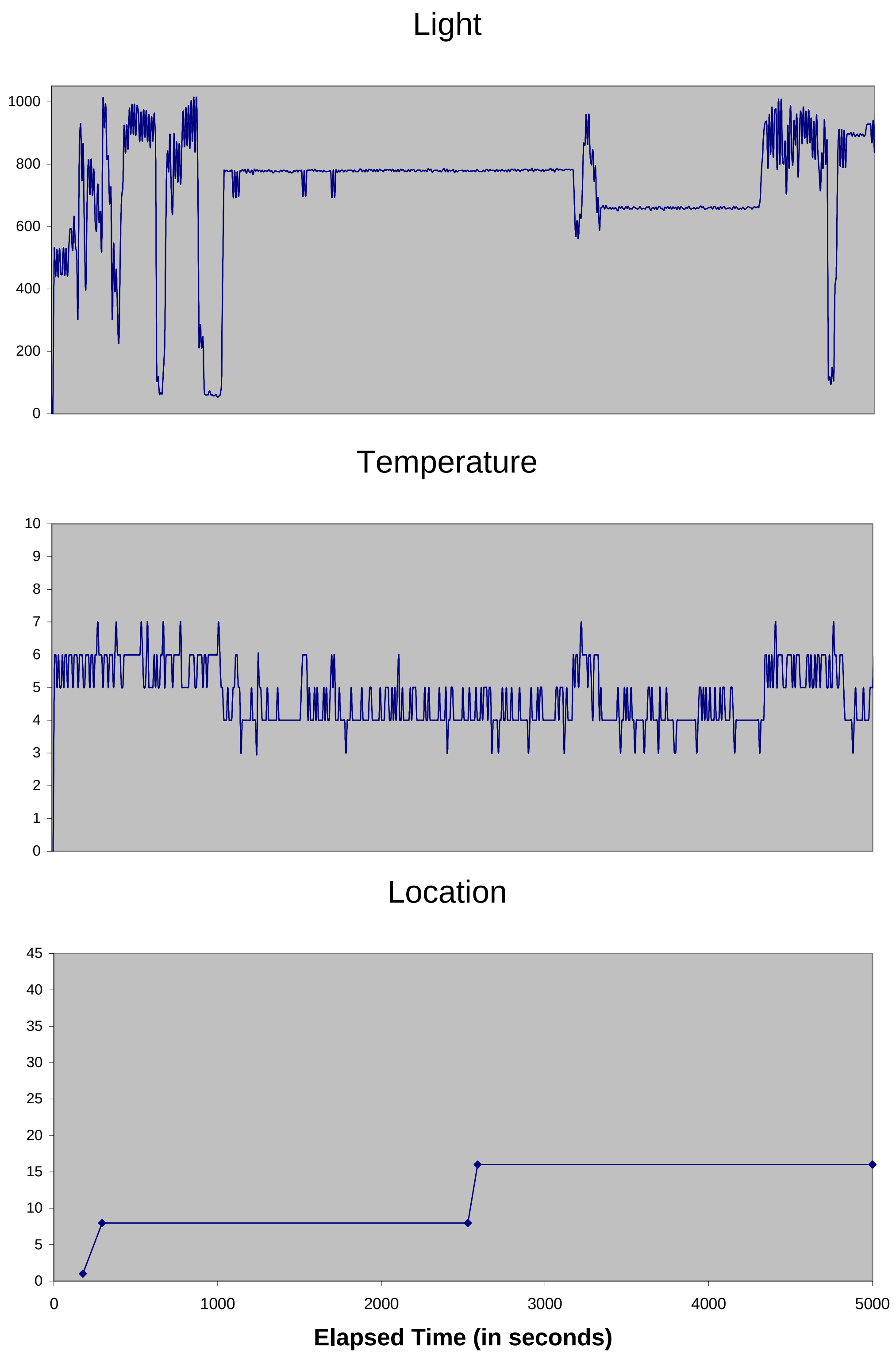
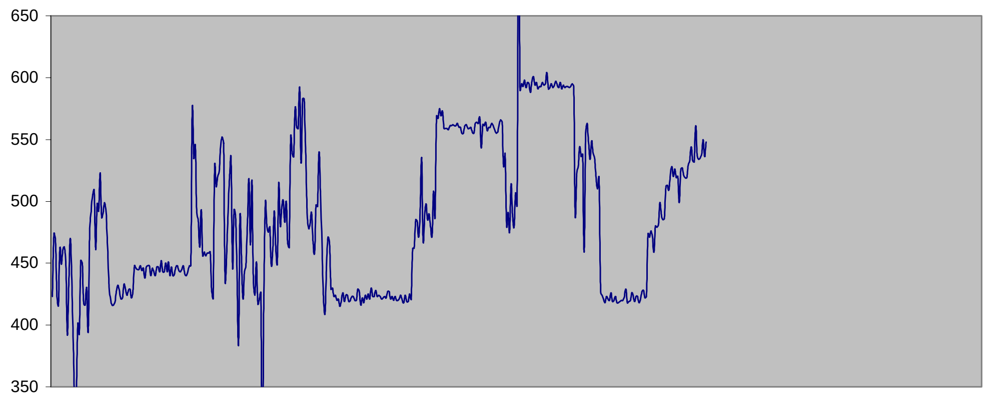


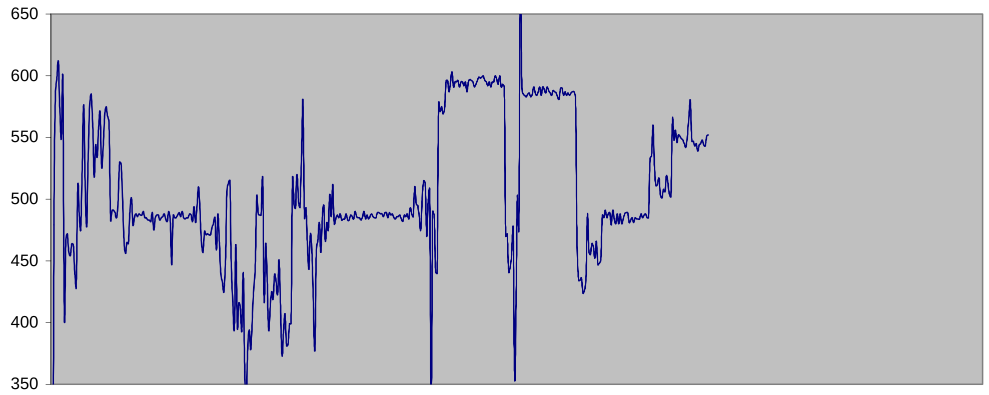
Figure 4-32: Sensor Data fro Node #5

Node #7 Sensor Data (Node #6 was D.O.A.)

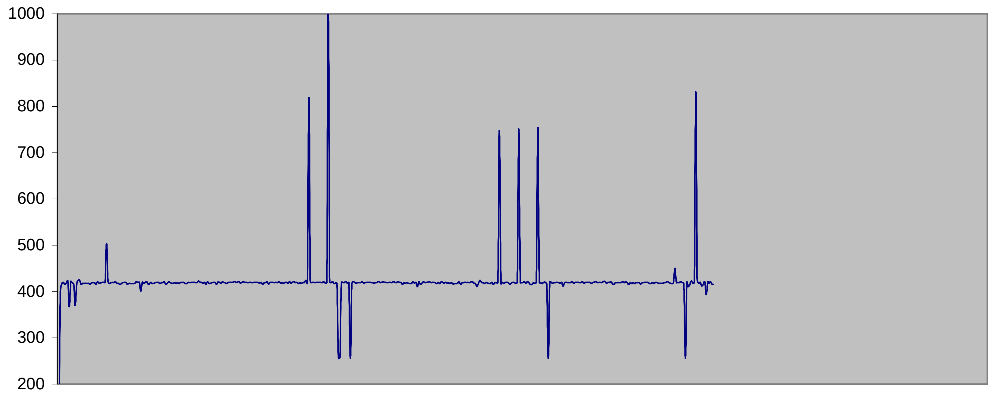
Accelerometer X



Accelerometer Y



Audio Magnitude



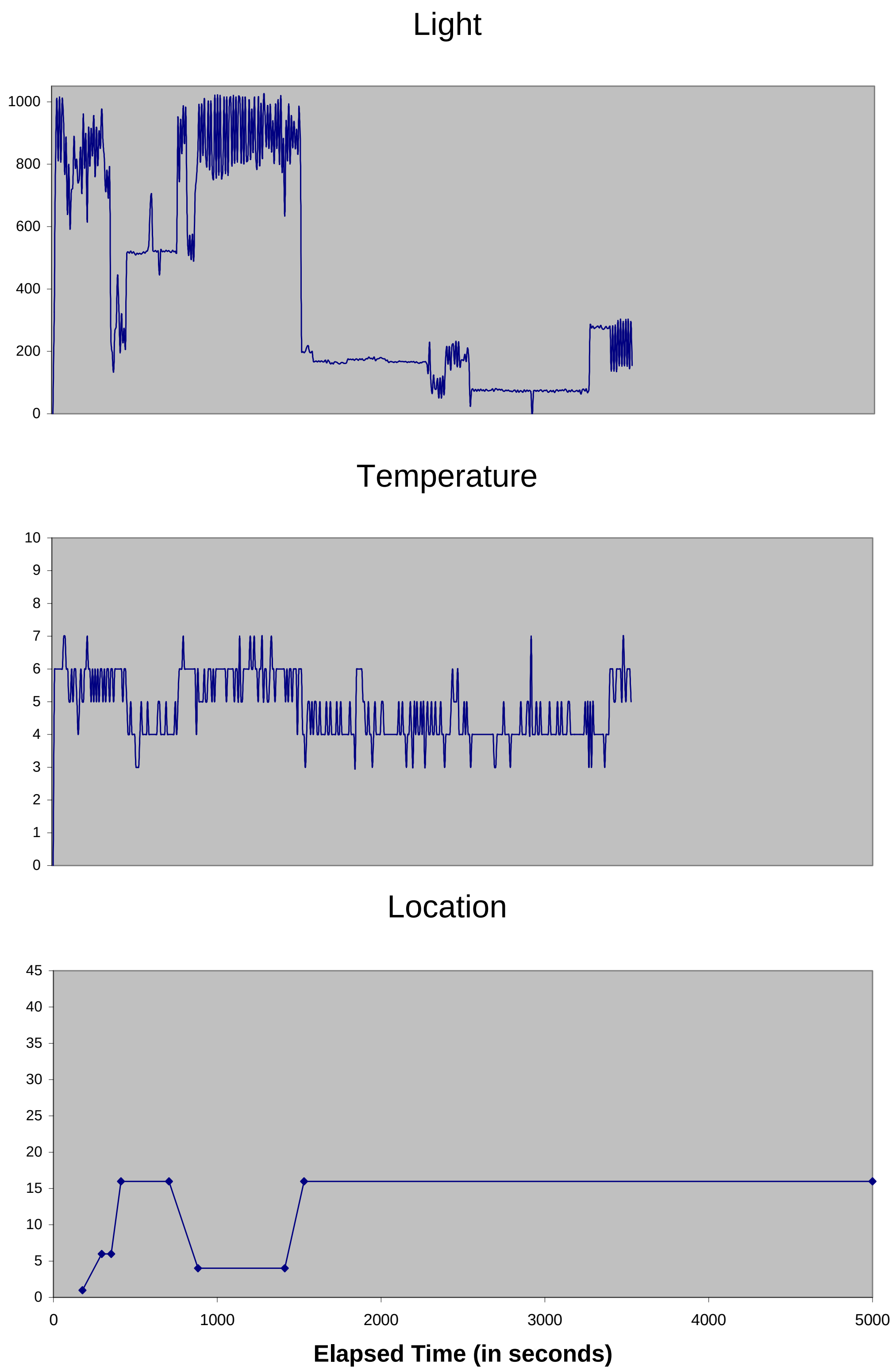
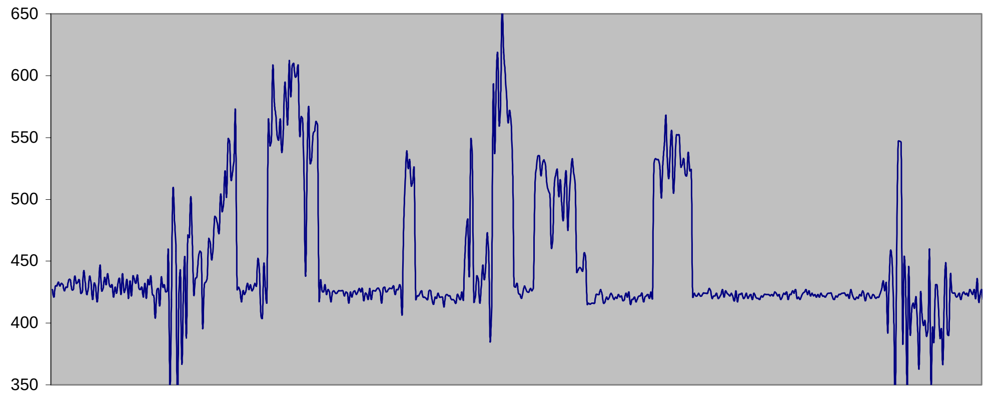


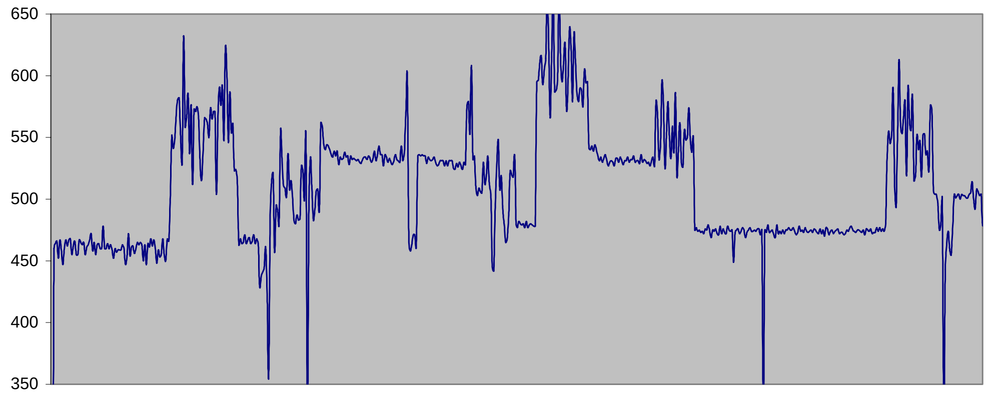
Figure 4-33: Sensor Data for Node #7

Node #8 Sensor Data

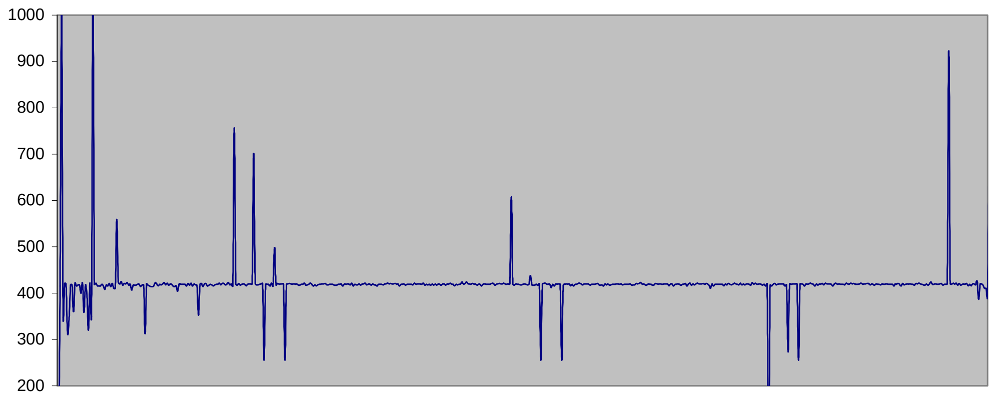
Accelerometer X



Accelerometer Y



Audio Magnitude



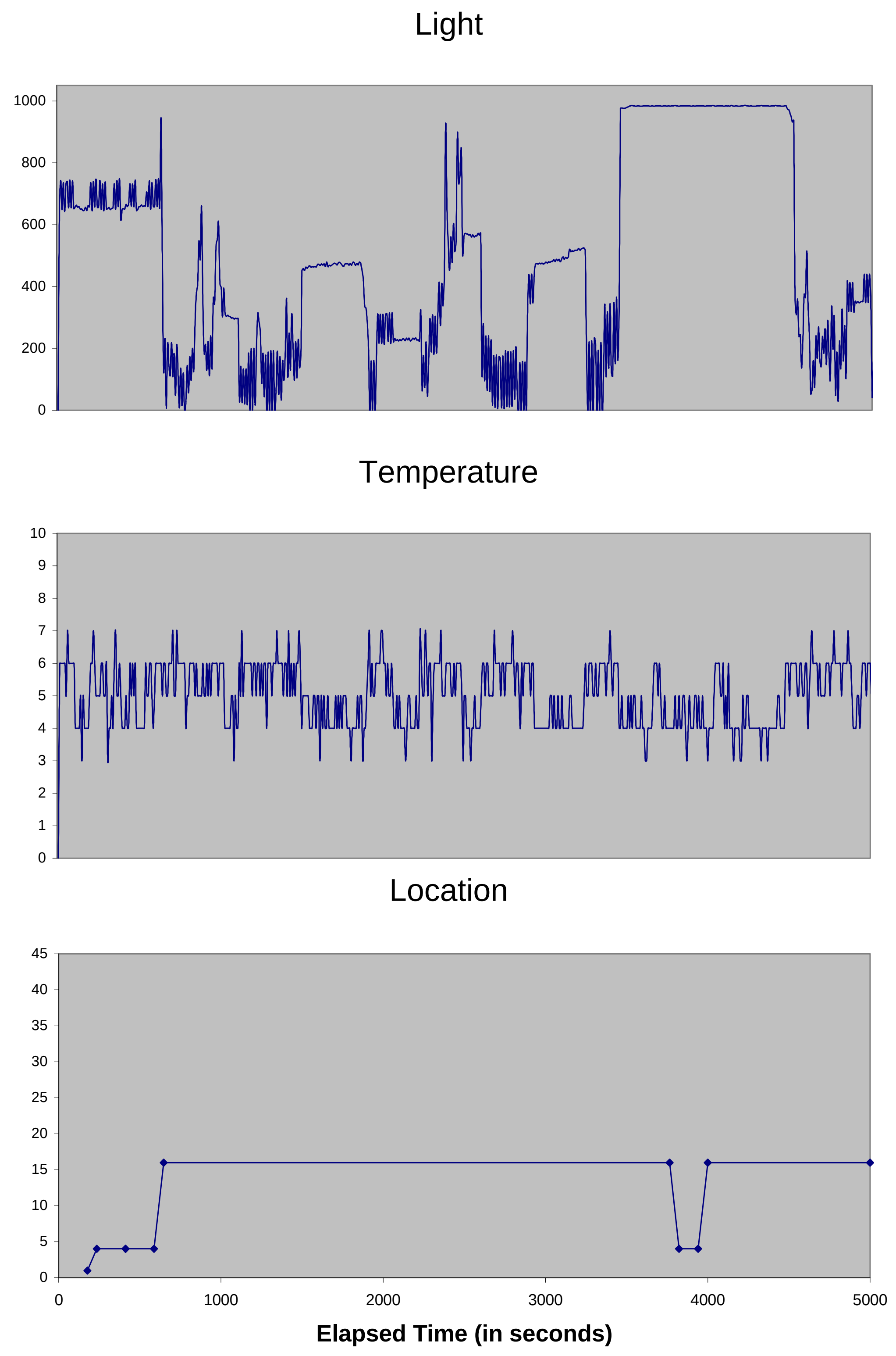
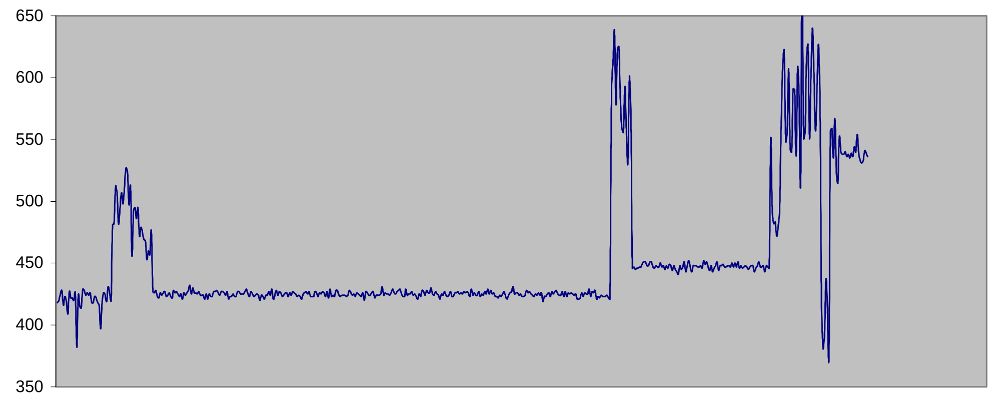


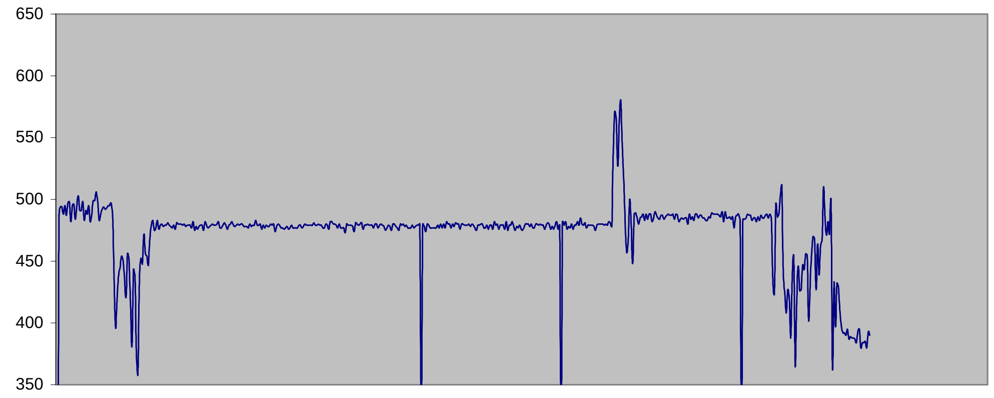
Figure 4-34: Sensor Data for Node #8

Node #9 Sensor Data

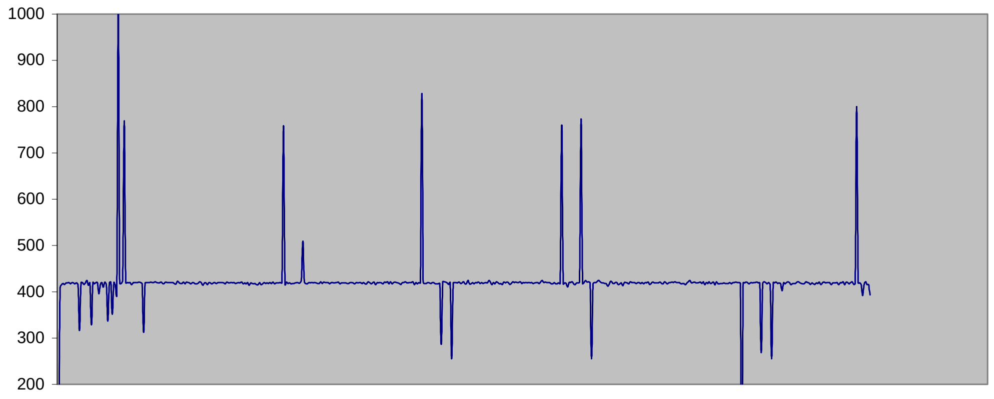
Accelerometer X



Accelerometer Y



Audio Magnitude



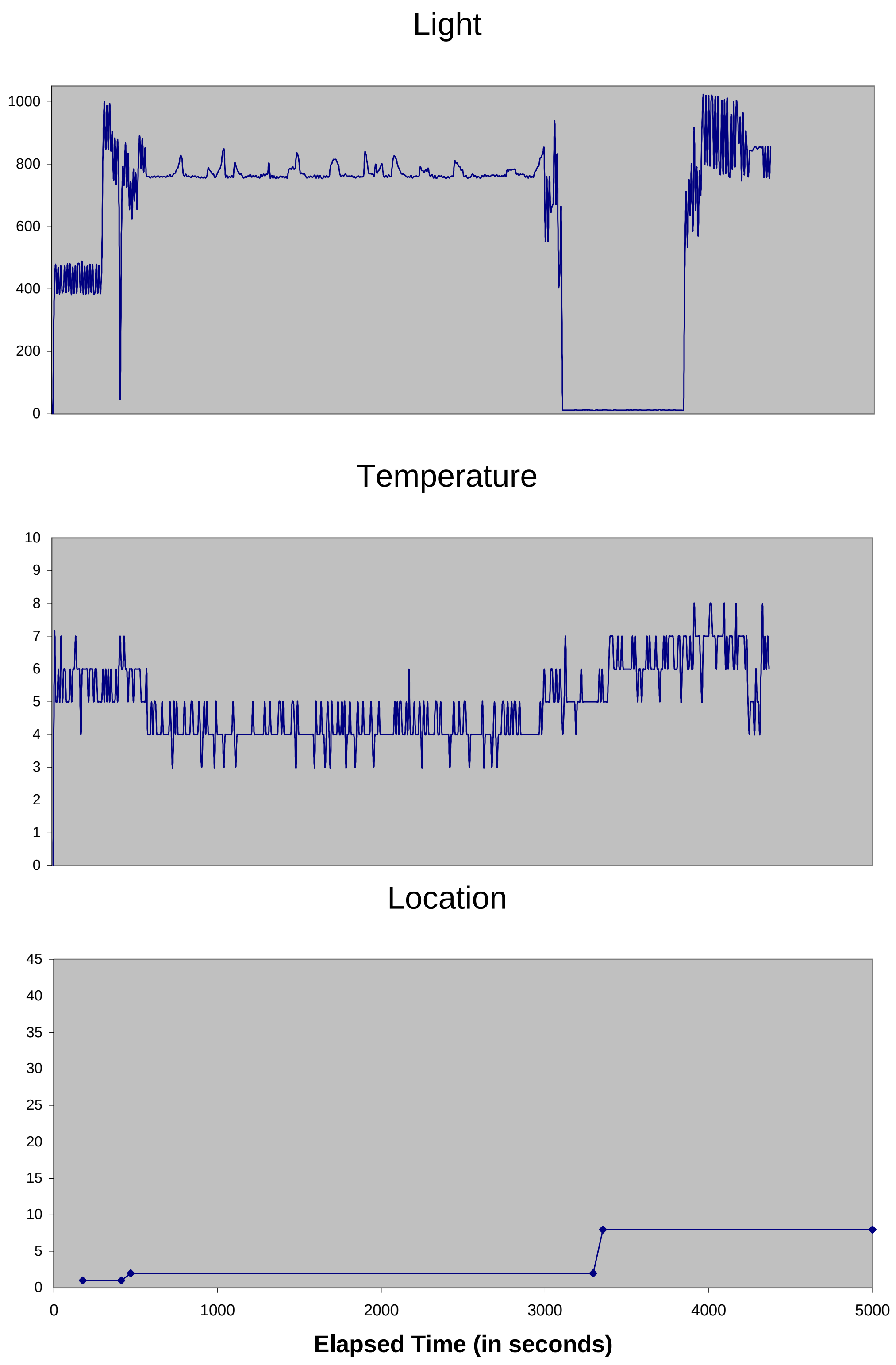
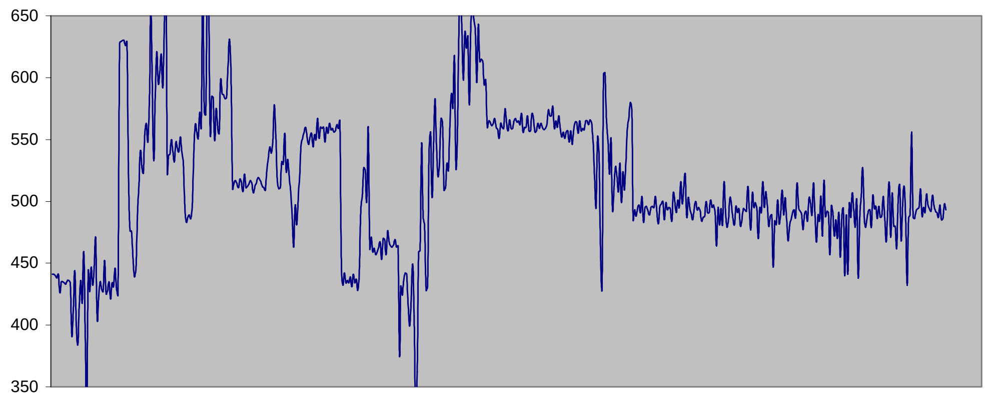


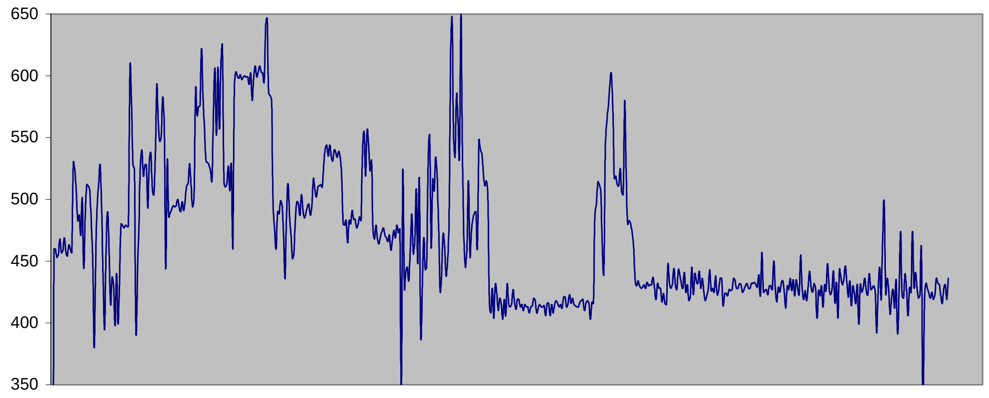
Figure 4-35: Sensor Data for Node #9

Node #10 Sensor Data

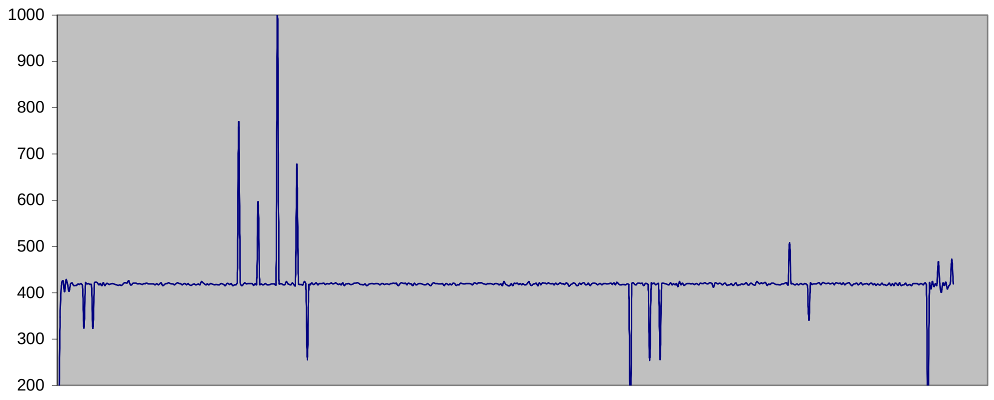
Accelerometer X



Accelerometer Y



Audio Magnitude



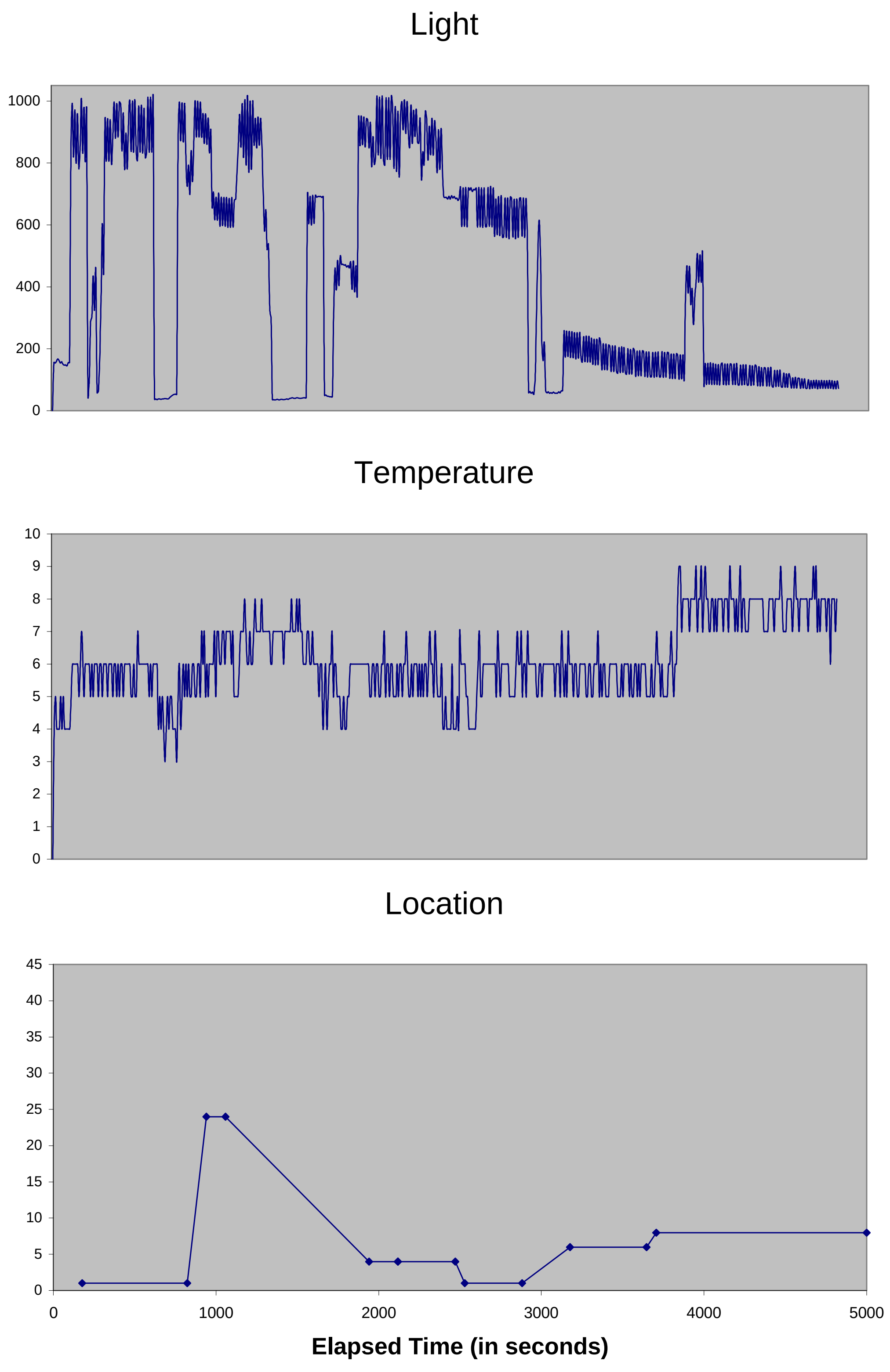


Figure 4-36: Sensor Data for Node #10

The first thing that can be seen from the graphs in Figures 4-28 through 4-36 is the relationship between location change and sensor activity. The nodes themselves use the accelerometers to assess their current state (attached/moving, idle, sensing). Combining all this data presents a very detailed picture of the behavior of the node.

For example, in Section 4.4.3 we mentioned that Node #7 found a light source and remained there for 4 attempted attachments. This can easily be seen by looking at the graphs in Figure 4-33. The light sensor shows two areas where the light value dropped very low, indicating a bright area. These two areas roughly match the times that the node was in zone 16. During the second stay in zone 16, there were several level shifts in the accelerometer data. These indicate a change in orientation, and the activity in between these levels is due to the node being picked and vibrating to be put back down.

The un-calibrated data shown in Table 4-6 was gathered by taking the average of all sensor data collected by all the nodes in each location zone. Please refer to Figure 4-15 to see where these zones map to on the floor plan.

Zone	Light (lower is brighter)	Temperature	Audio Level
1	633.9	5.0	423.0
2	624.8	5.1	423.6
3	624.9	5.0	424.9
4	603.9	4.8	424.8
6	613.3	4.8	424.8
8	608.4	5.7	423.9
10	609.2	5.5	424.6
16	594.1	4.7	423.0
20	603.2	4.7	425.1
24	603.4	5.1	423.3
28	602.8	5.0	424.8

30	605.4	5.0	423.8
32	617.3	4.8	425.4
33	626.3	4.9	426.2
40	612.2	5.2	424.1
43	613.2	5.3	424.2

Table 4-6: Average Data per Zone (raw un-calibrated sensor data)

The data in Table 4-6 gives an overall view of the area. It is important to note that some of the sensor data can be influenced by how the device is attached and carried by the host. For example, a rolling node can have alternating light and dark views presented to the light sensor. These factors average themselves out somewhat and general information about the environment can be gleamed. This is verified by the trajectory data from section 4.4.3, which constantly identified certain areas as higher in temperature and brighter in light. Generally, the sensor data stayed around a known baseline, as the environment is fairly controlled and regulated.

4.5 Hardware Evaluation

The hardware design proved quite adequate for experimentation with the concept of parasitic mobility. The specific version used for this test was mechanically robust and none of them stopped working due to mechanical stress.

Of the ten nodes deployed, only one suffered from complete failure due to the hardware, and it was most likely due to a manufacturing problem. Another node may have had some hardware issues, but it is not conclusive. Further testing on the node after its recovery was successful and show that it is working fine.

The firmware also proved to be quite robust, and there were no unexpected firmware crashes or missed or mangled data.

One issue was that the sampling rate of the sensors was too slow. This is an easy fix for the next version. It was kept low while testing to simplify development, testing, and debugging, as well as to allow the flash to last for the entire test. As the development progresses, the sampling rate will most likely increase and become adaptive. The sampling rate at the time of this test was too slow to pull features out of the audio other than the net amplitude.

One of the major successes of the hardware design was the power supply and power management systems that each node incorporates. The battery lasted as long, if not longer, than expected considering the additional data logging and communication requirements of running such a test. Clever wakeup and adaptive sensing can improve this considerably.

The size of the nodes is currently too big to attach to a human or an animal without its knowing. The current size is more than adequate, however, for applications where the hosts are vehicles.

The location system, while it worked adequately for this test, was not resolute enough for fine-grained indoor tracking. Using this type of system properly on such a scale would require 3 times as many location beacons. It would also need to refresh faster and more consistently. Many indoor location systems are under active development [40], hence this situation is rapidly evolving.

Chapter 5

Conclusion

5.1 Summary

Through the work described in this thesis, a new field of research has emerged. This field sits at the crossroads of distributed sensor networks, mobile systems, and power harvesting.

Through simulation and test we have illustrated that it is possible to develop a parasitically mobile sensor network. Our results indicate that they can, in some ways, perform as well as standard robotic mobile sensor networks, but with huge potential savings with regards to power consumption, node complexity, and general robustness due to their relative simplicity.

5.2 Possible Applications of Parasitic Mobility

In certain environments, parasitic mobility can be used as a replacement for standard mobility for dense, distributed sensor systems. Systems of this sort include applications to sense toxic areas requiring sensor deployment at a safe distance, dynamically reconfigurable systems such as weather monitoring sensors that need to follow the relative phenomena, and systems where the accuracy of node deployment is minimal such as for nodes being released in water or from a vehicle.

Going further, parasitic mobility can possibly lead to applications that can only be done (or are better done) with parasitic mobility than standard mobility. Any example where the host behavior is part of what is desired to be monitored would fit this category. In these systems, parasitically mobile nodes would attach to their subjects and would always be at the points of interest.

One application that would be interesting to explore is the idea of a rating system based on breadcrumb trails. Essentially, the parasitic nodes would attach to hosts and pool up in spots of high traffic. These points can propagate through the system and provide information on the popularity of certain pathways and locations.

5.3 Future Work

Our systems can be further perfected, e.g., a first step would be to increase the performance of the system by increasing the sample rate of the sensors, the onboard processing power, and the resolution and refresh rate of the location system. Deploying the GPS system would also be advantageous. By increasing the node’s capabilities, it will be possible to give the nodes more information about the environment such as onboard databases of map information.

Hooking this research up with actual power harvesting could be a natural fit, allowing self-maintaining, perpetual systems to be developed. These systems can harvest the power from their environment (taking inspiration from the tick, which harvests chemical energy from its host) or from forces acted upon the nodes, such as when they are in the attached state. Smarter power management can be developed, as well as power adaptive sensing, to improve battery conservation.

Also, adding more distributed, node-node communication to the test system would open up some new venues for research. By collaboration, the sensor nodes could optimize their mobility and detachment and attachment algorithms.

More experimentation with new types of attachment and detachment mechanisms could lead to new applications of parasitic mobility, e.g. attaching to vehicles. Also, embedding sensor nodes into everyday objects is an exciting prospect. These directions can benefit from smaller nodes. Adding sensors (e.g. camera, motion sensor, and magnetic sensor) can also allow detection and attachment to a larger variety of hosts, as well as a wider range of sensing applications.

Finally, the major outstanding piece of work would be to develop and deploy the system for a real application. Some possible applications were mentioned in Section 5.2 and can arise from inspiration that comes about from further technical enhancements and conceptual experiments.

Appendix A

Schematics and PCB Layouts

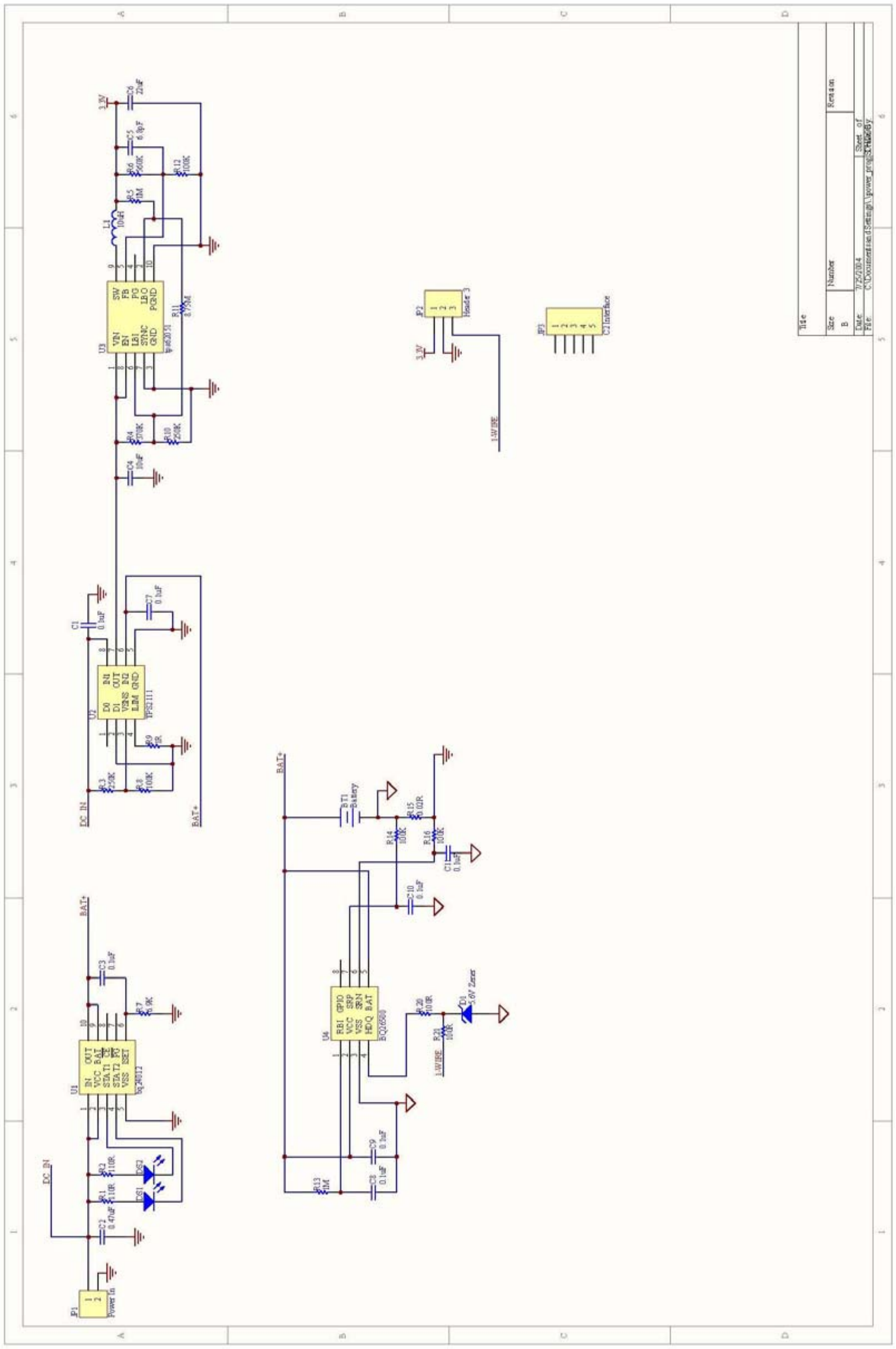


Figure A-1: Node Power Module Schematic

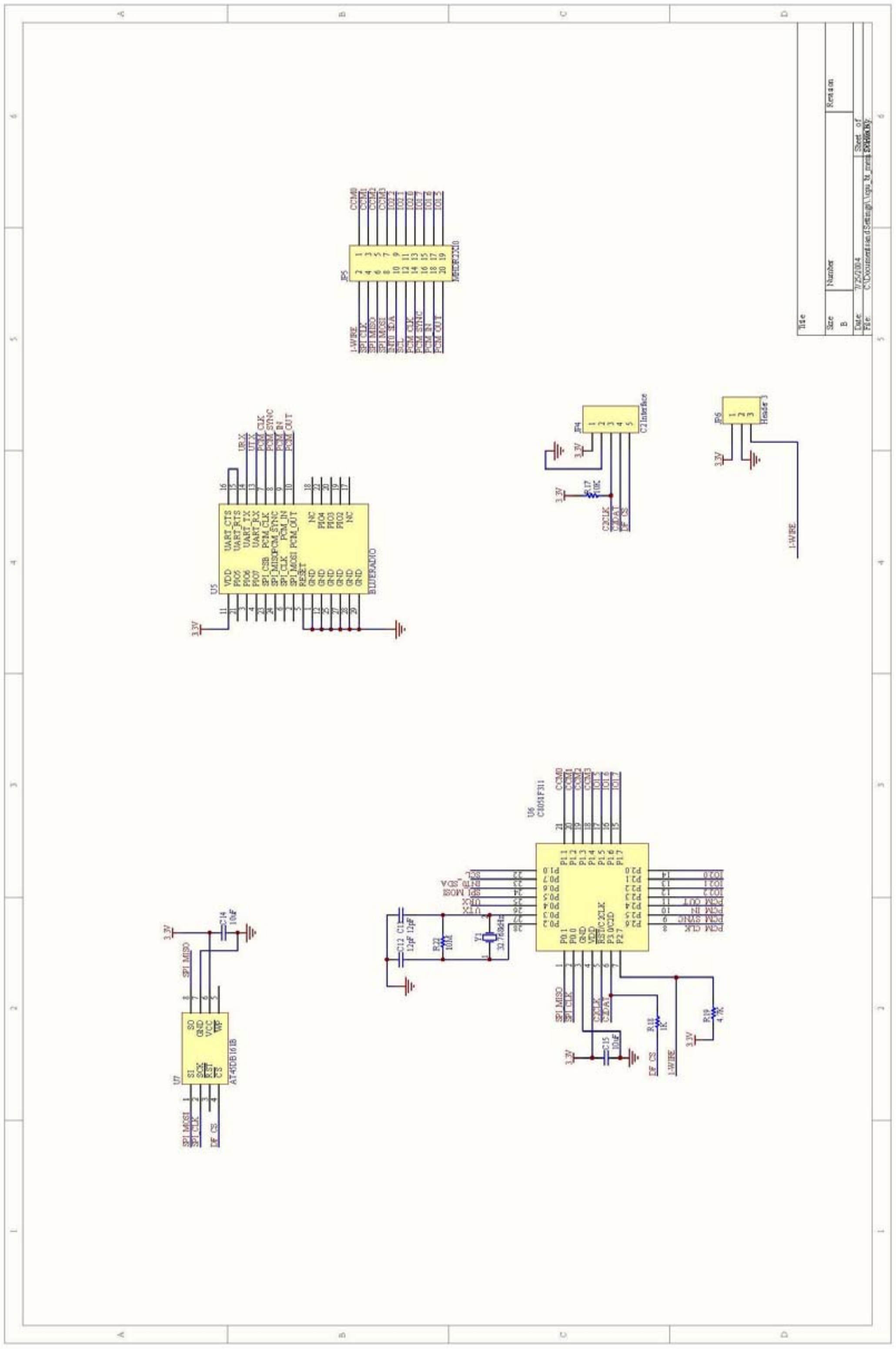


Figure A-2: Node Processing and Communication Module Schematic

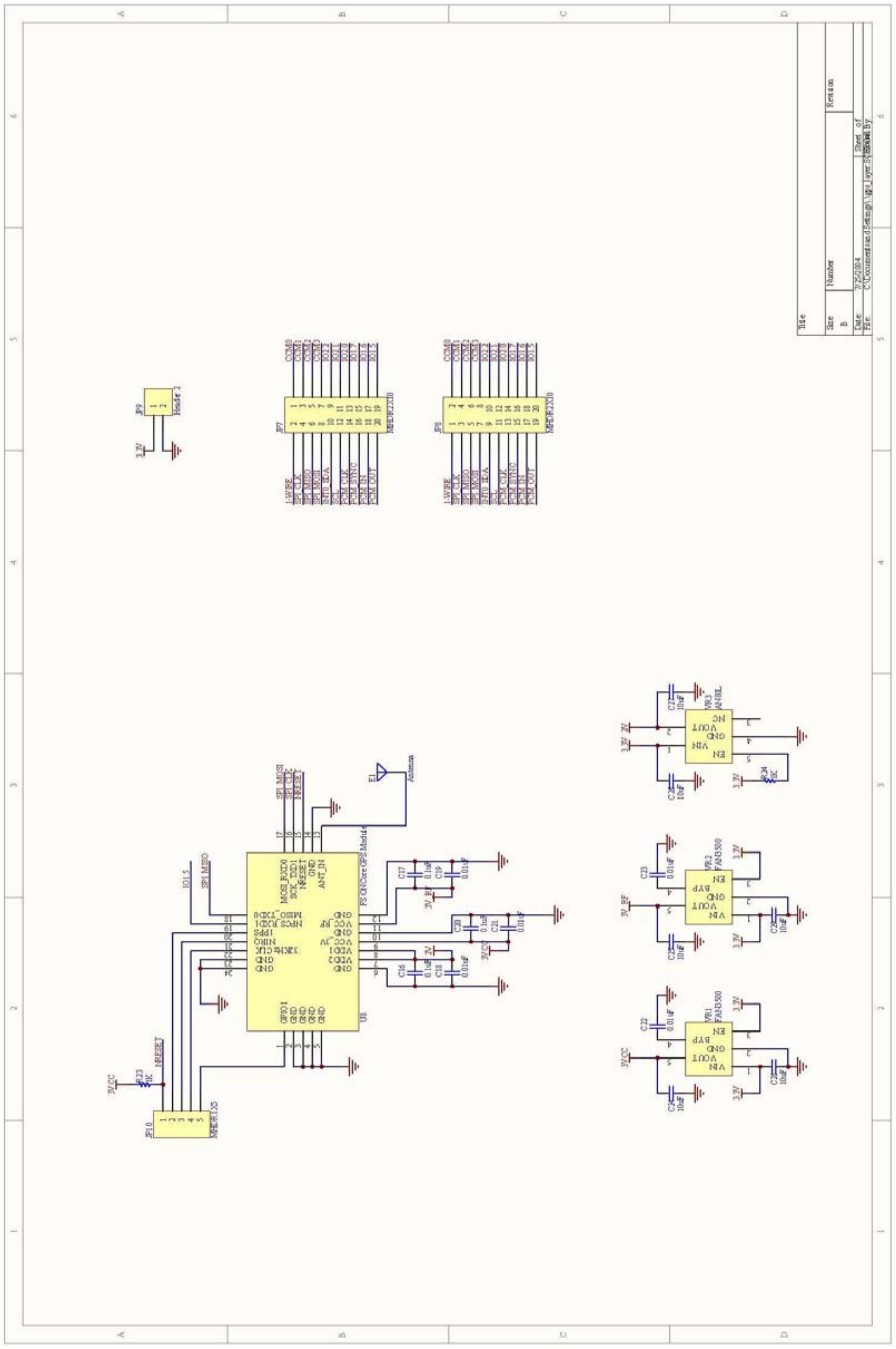


Figure A-3: Node GPS Module Schematic

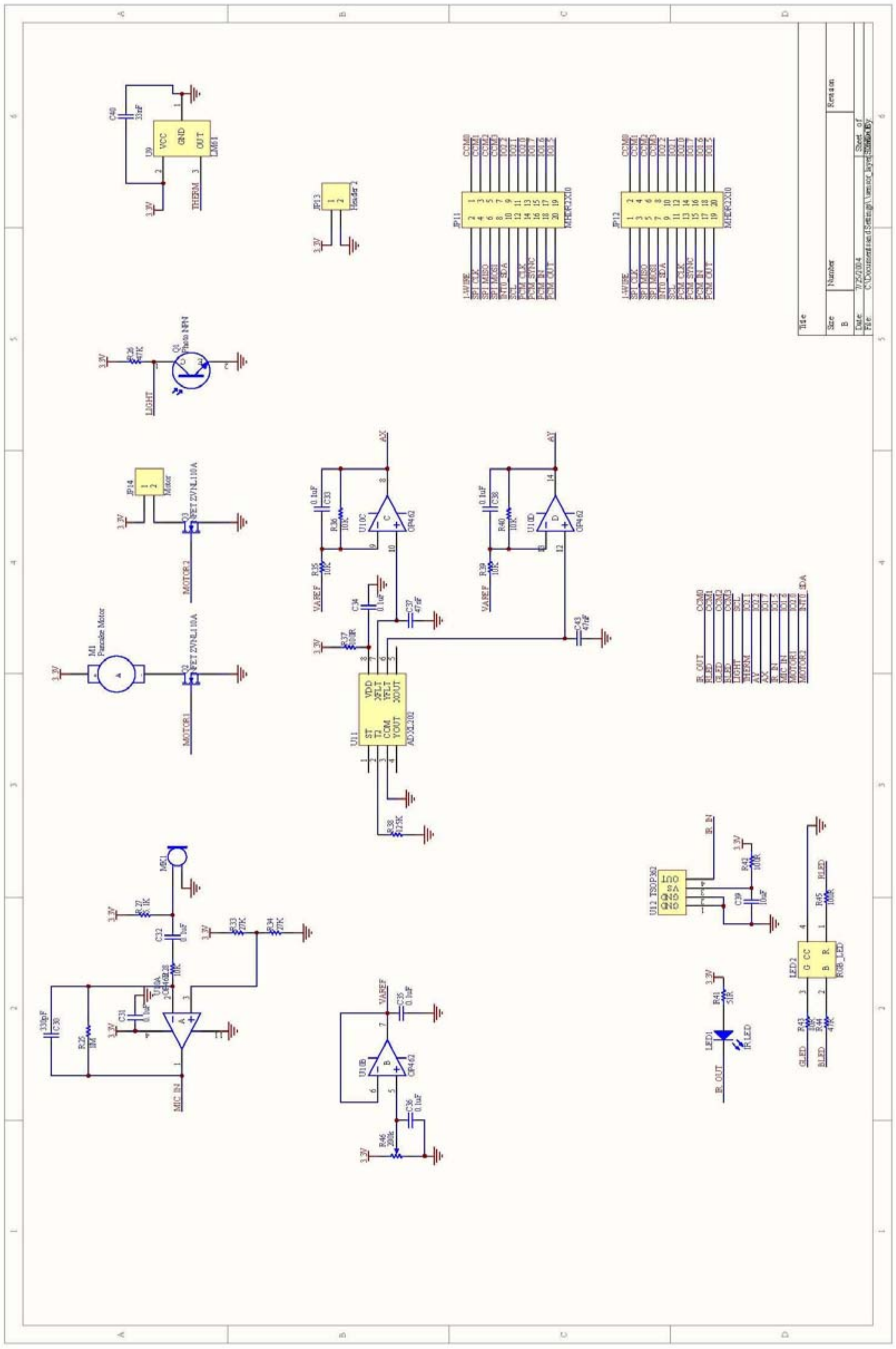


Figure A-4: Node Sensor Module Schematic

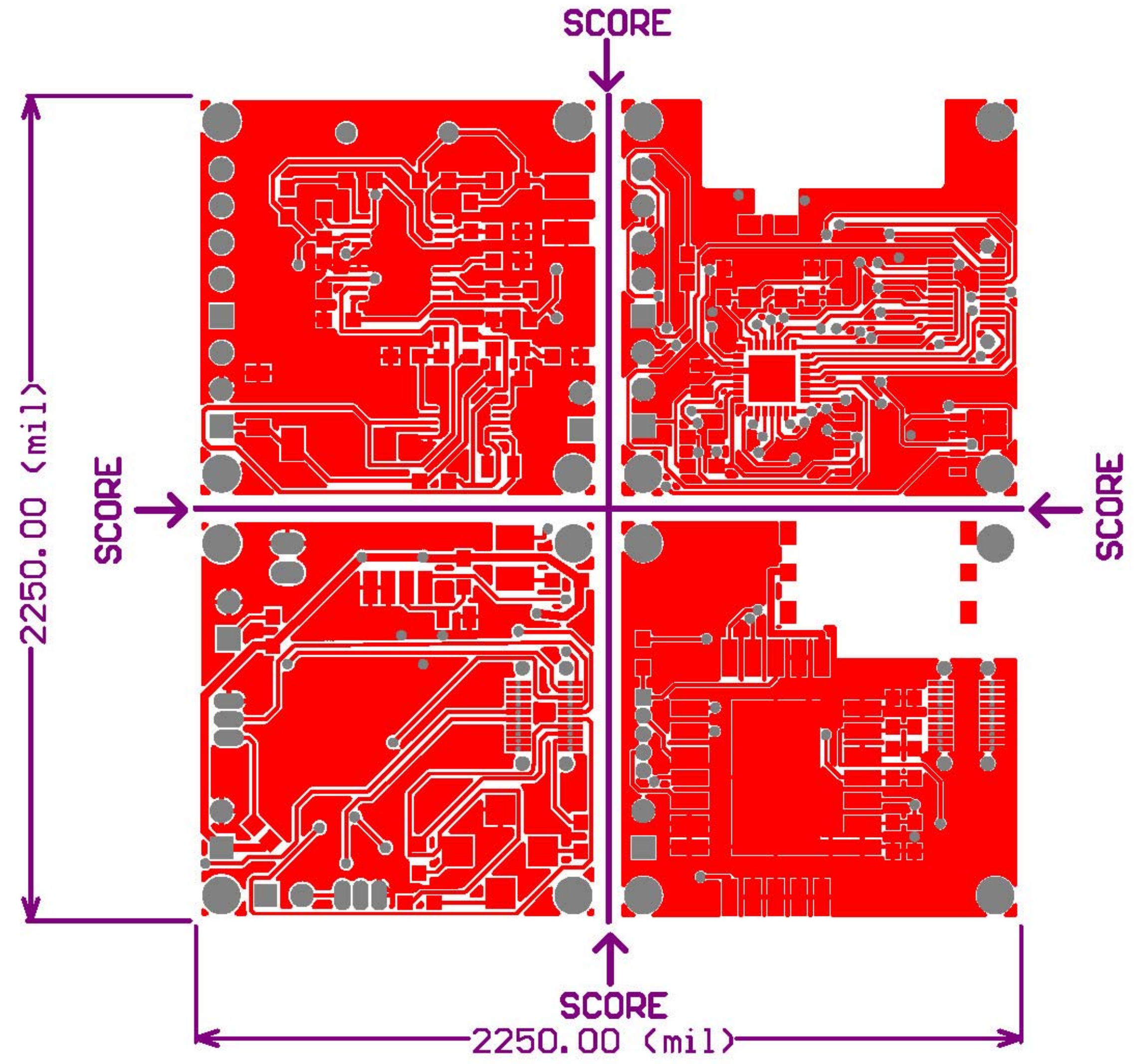


Figure A-5: Node PCB Layout Top Layer

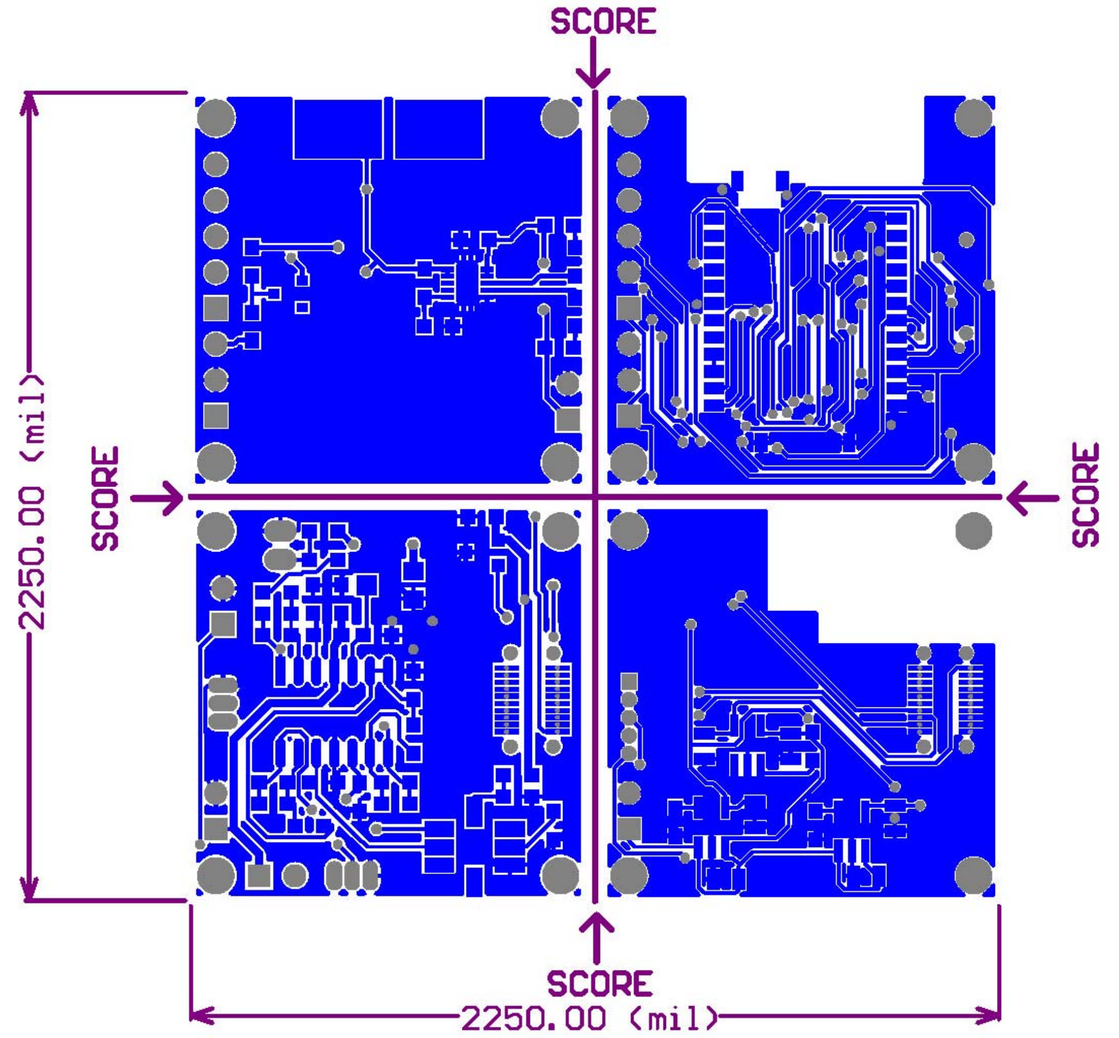


Figure A-6: Node PCB Layout Bottom Layer

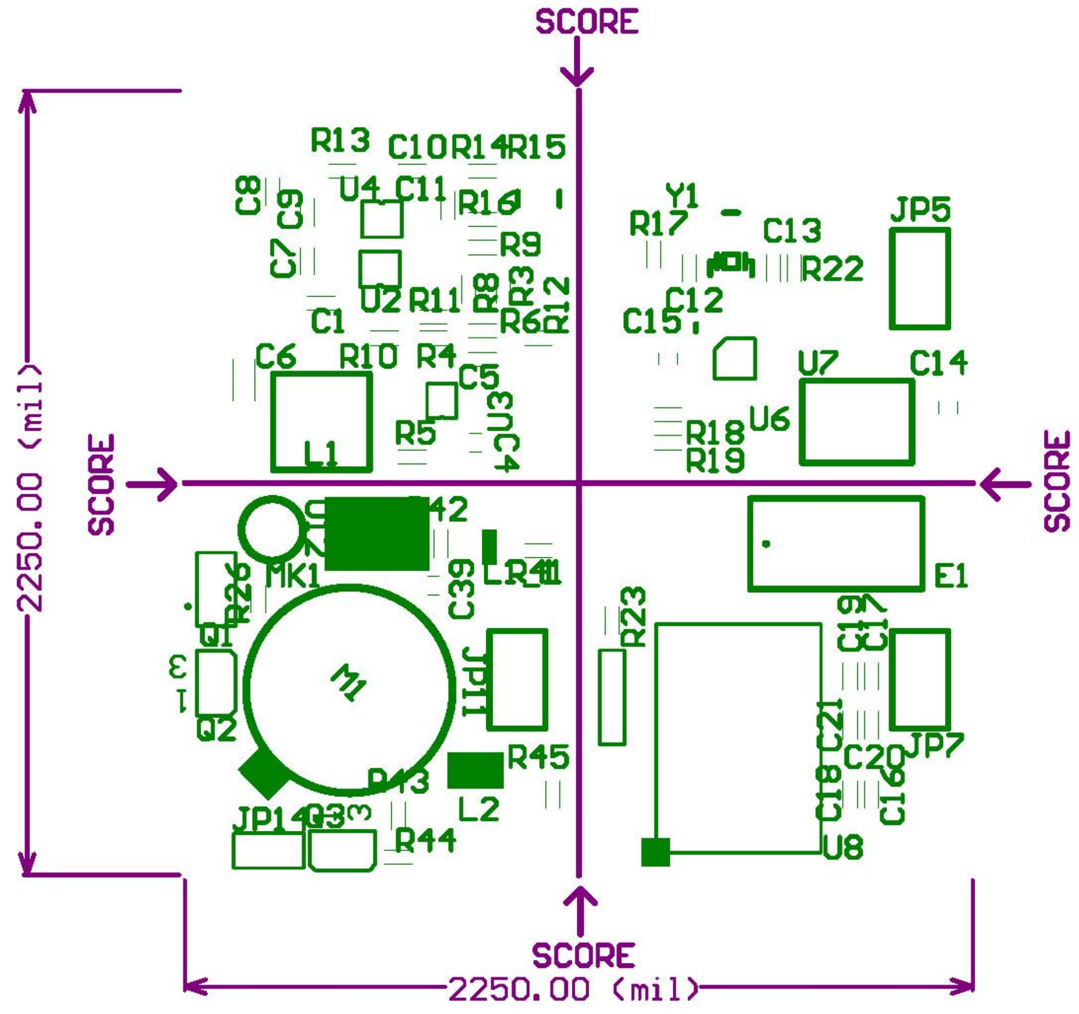


Figure A-7: Node PCB Layout Top SilkScreen

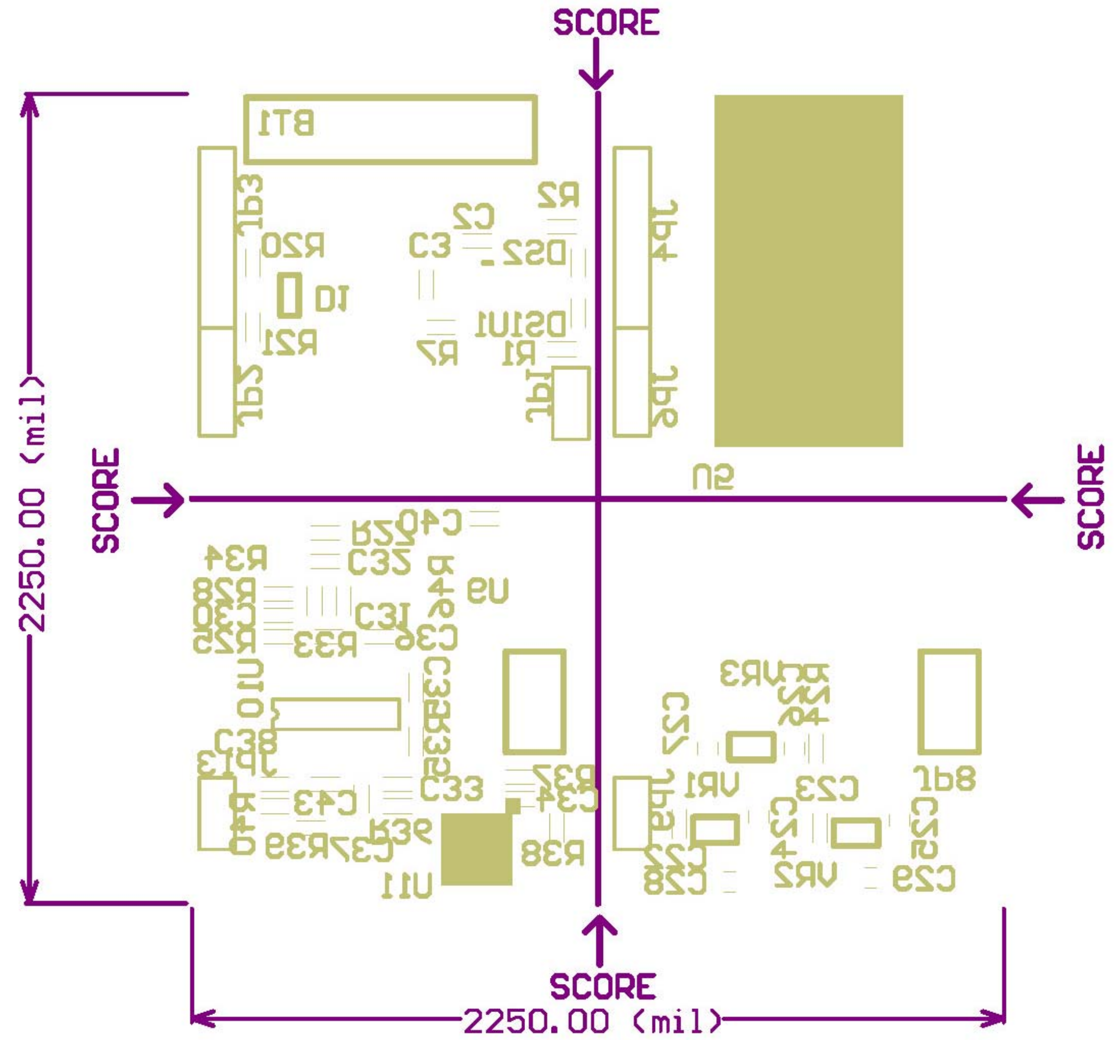


Figure A-8: Node PCB Layout Bottom SilksScreen

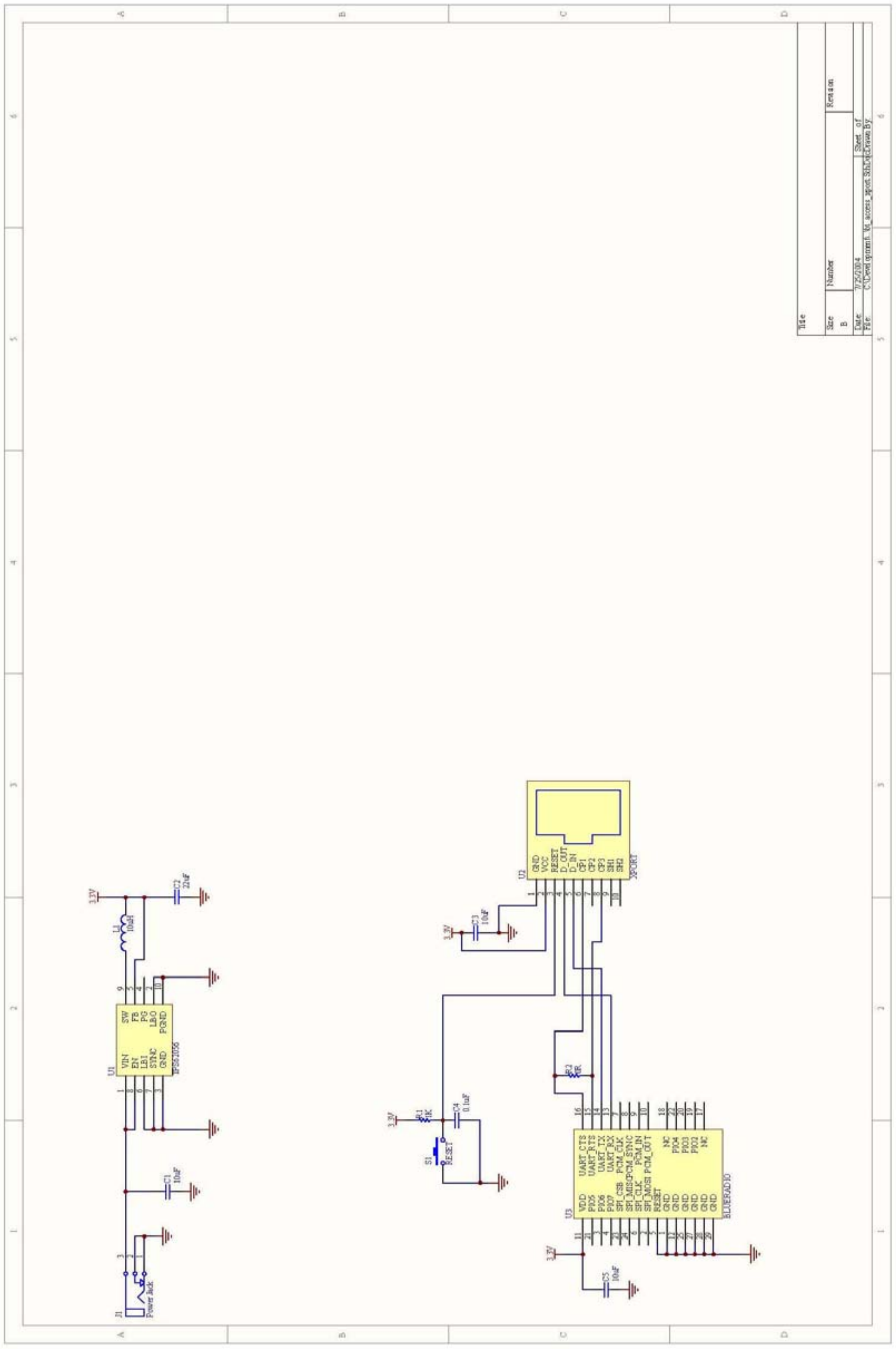


Figure A-9: Schematic of Bluetooth Location System Beacon

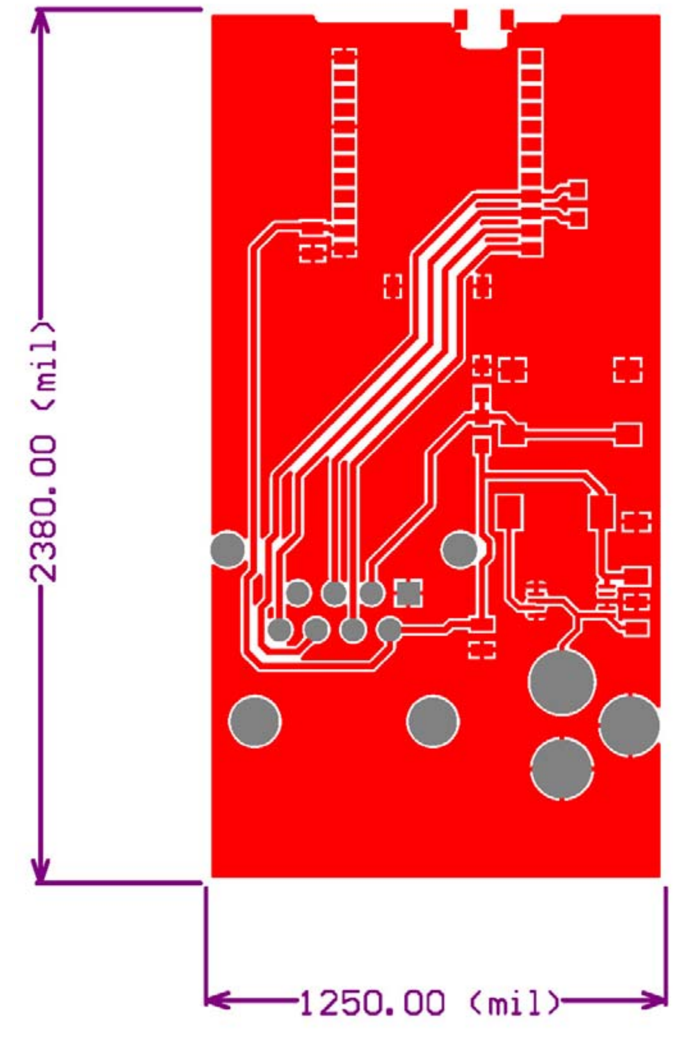


Figure A-10: Bluetooth Location System Beacon PCB Layout Top Layer

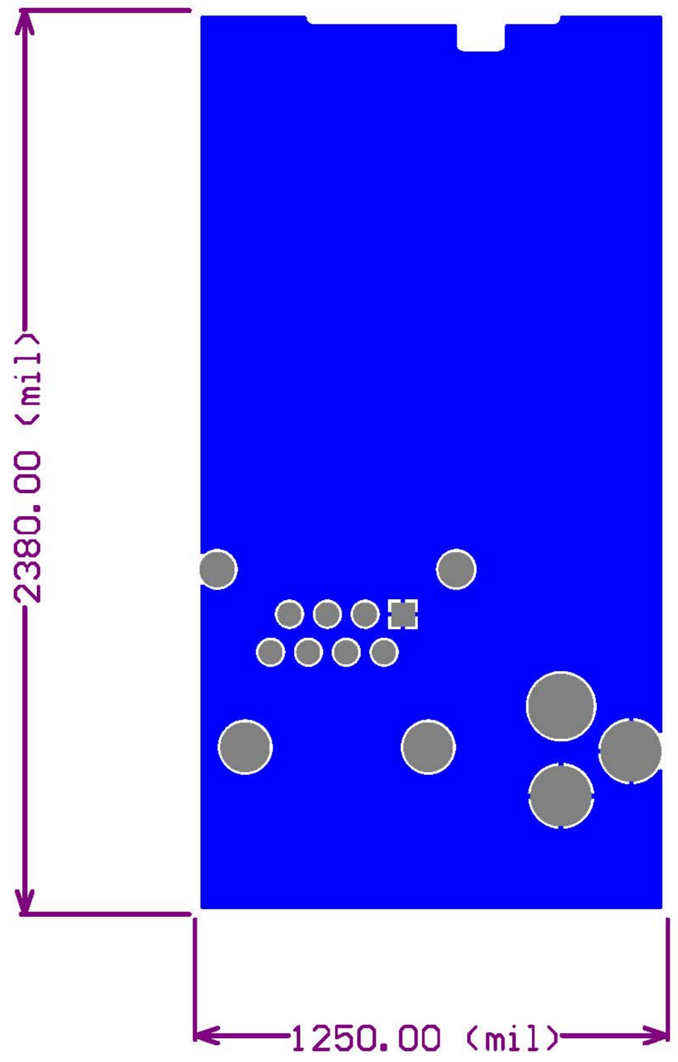


Figure A-11: Bluetooth Location System Beacon PCB Layout Bottom Layer

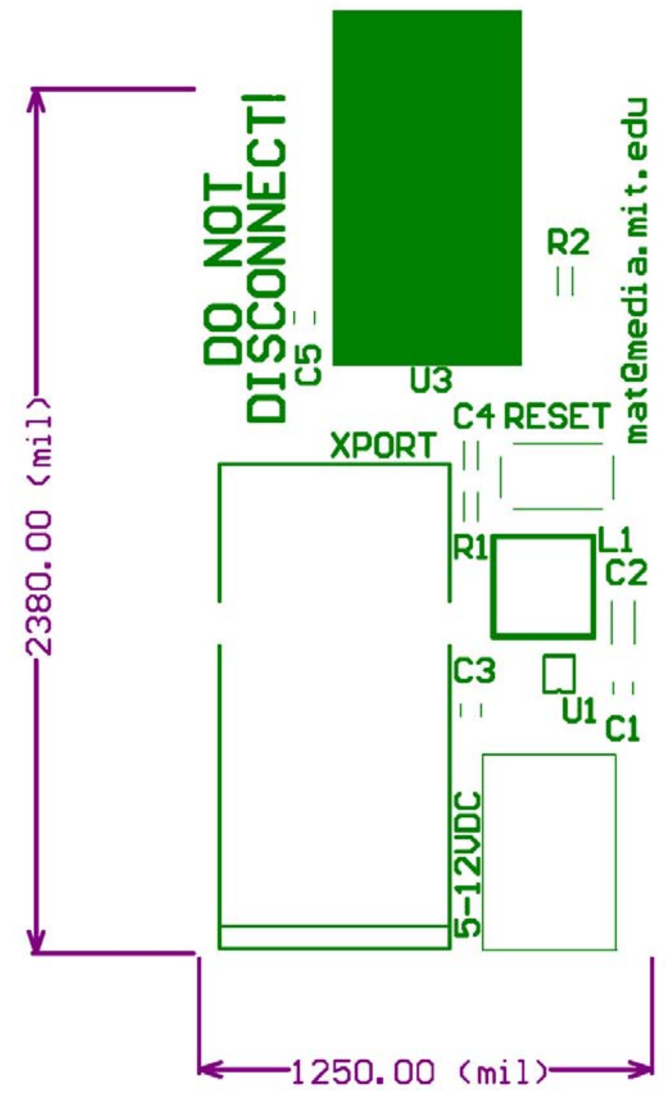
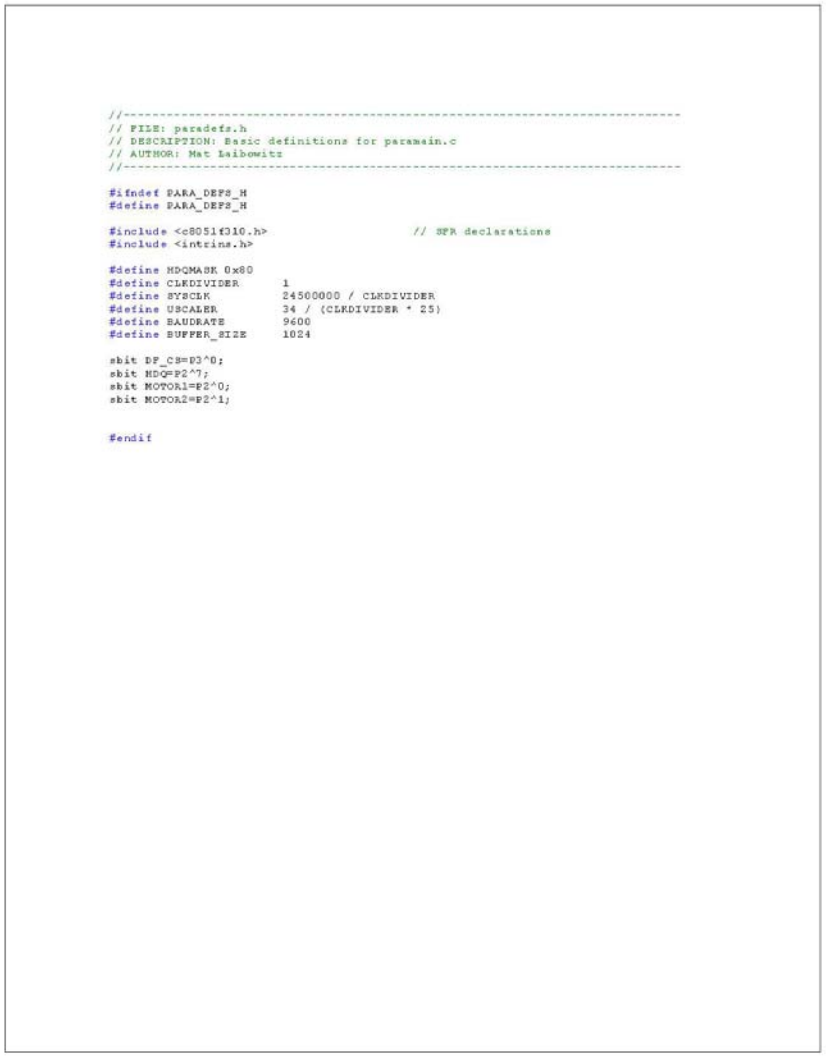


Figure A-12: Bluetooth Location System Beacon Top SilkScreen

Appendix B

Microprocessor Code



```
//---------------------------------------------------------------------------
// FILE: paradefs.h
// DESCRIPTION: Basic definitions for paramain.c
// AUTHOR: Mat Laibowitz
//---------------------------------------------------------------------------

#ifndef PARA_DEFS_H
#define PARA_DEFS_H

#include <c8051f310.h>                    // SFR declarations
#include <intrins.h>

#define MDQMASK 0x80
#define CLKDIVIDER      1
#define SYSCLK          24500000 / CLKDIVIDER
#define USCALER         34 / (CLKDIVIDER * 25)
#define BAUDRATE        9600
#define BUFFER_SIZE     1024

sbit DF_CS=P3^0;
sbit MDQ=P2^7;
sbit MOTOR1=P2^0;
sbit MOTOR2=P2^1;


#endif
```

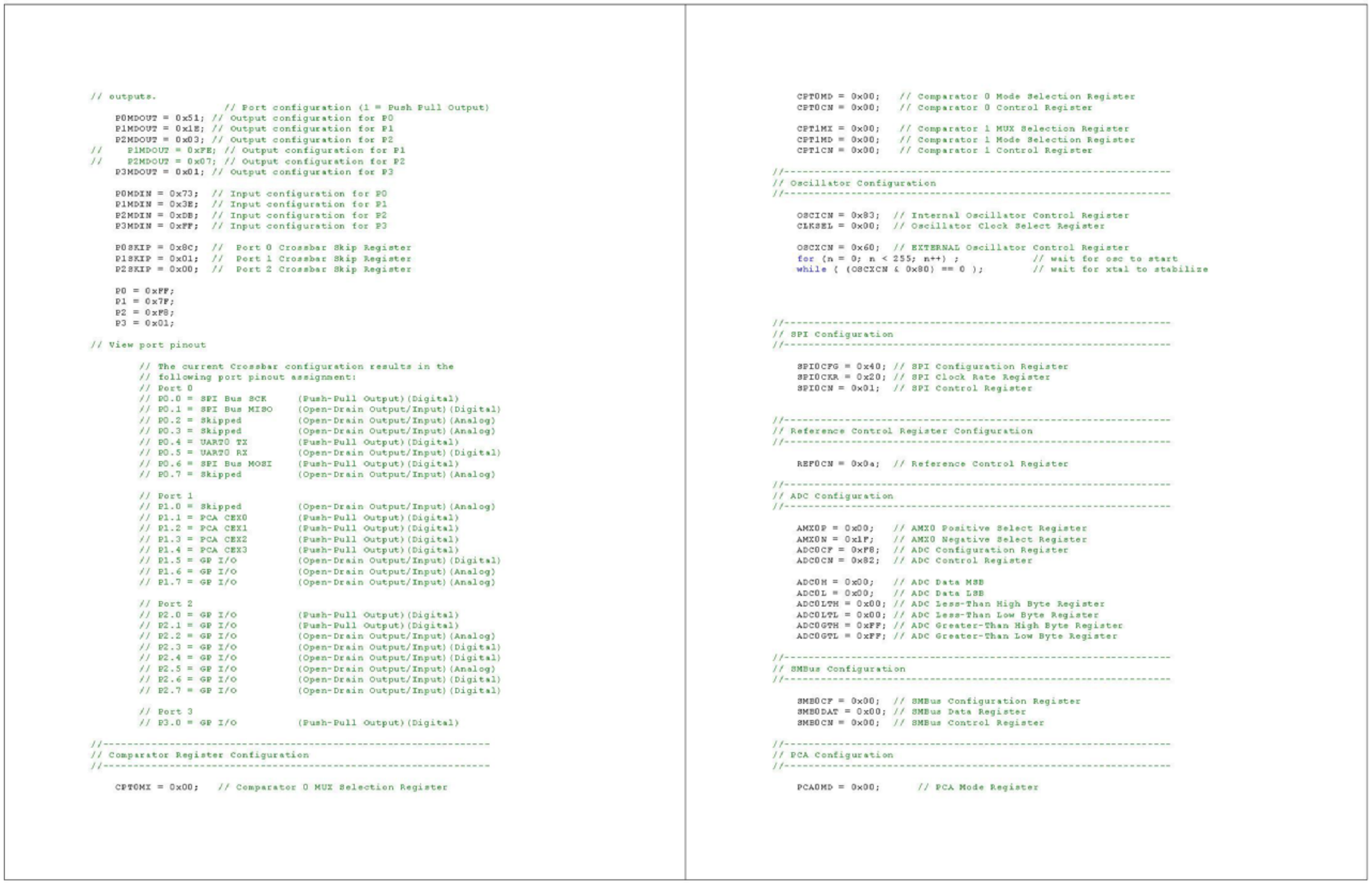
```
//-----------------------------------------------------------------------------
// FILE: paramain.c
// DESCRIPTION: Main firmware file for the parasitically mobile nodes
// AUTHOR: Mat Laibowitz
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------
#include <c8051f310.h>                    // SFR declarations
#include <intrins.h>
#include <math.h>
#include "para_defs.h"
#include "cy_flash.h"
#include "cy_delays.h"
#include "cy_hdq.h"

#define GPS                     0
#define BT_LOCATION             1
#define BT_NAME_H               0x30
#define BT_NAME_L               0x37
#define LED_BLINK               1
#define BATT_POLL               100
#define DATA_SEND               1
#define MAP_SIZE                30
#define MOTION_THRESHOLD        20
#define STILL_THRESHOLD         15
#define STILL_SAMPLES           50
#define HOPS_RESET              1
#define MOTOR_TIMEOUT           150
#define DETACH_TIMEOUT          300

// BEHAVIOR
#define POWER_THRESHOLD         0
#ifdef  BT_LOCATION
#define GOAL_X                  32
#define GOAL_Y                  0
#endif
#ifdef  GPS
#define GOAL_LAT_DIR            0
#define GOAL_LAT_DEG            0
#define GOAL_LAT_MIN            0
#define GOAL_LAT_SEC            0
#define GOAL_LON_DIR            0
#define GOAL_LON_DEG            0
#define GOAL_LON_MIN            0
#define GOAL_LON_SEC            0
#endif
#define GO_TO_GOAL              0
#define STOP_AT_GOAL            0
#define COVERAGE                0
#define LIGHT_THRESHOLD         10000
#define VIBRATION_THRESHOLD     0
#ifdef  GPS
#define ALTITUDE_THRESHOLD      0
#endif
#define TEMPERATURE_THRESHOLD   10000
#define AUDIO_THRESHOLD         10000
#define SENSE_TIME              3600
#define HOPS_PER_LOCALE         0

//-----------------------------------------------------------------------------
// Data Structures
//-----------------------------------------------------------------------------
```

```
enum { IDLE, ATTACHED, SENSING };

typedef struct {
#ifdef  BT_LOCATION
    unsigned int btX;
    unsigned int btY;
#endif
#ifdef  GPS
    unsigned char latDir;
    unsigned int latDeg;
    unsigned int latMin;
    float latSec;
    unsigned char lonDir;
    unsigned int lonDeg;
    unsigned int lonMin;
    float lonSec;
#endif
} location;

typedef struct {
    unsigned int PowerThreshold;
    location goal;
    unsigned char gotoGoal;
    unsigned char stopAtGoal;
    unsigned char coverage;
    unsigned int LightThreshold;
    unsigned int VibrationThreshold;
    unsigned int TemperatureThreshold;
    unsigned int AltitudeThreshold;
    unsigned int AudioThreshold;
    unsigned int senseTime;
    unsigned int hopsPerLocale;
} behavior;

typedef struct {
    location good[MAP_SIZE];
    location bad[MAP_SIZE];
    location visited[MAP_SIZE];
    location unvisited[MAP_SIZE];
} map;

typedef struct {
    unsigned int sensors[6];
    unsigned int baseline[6];
    unsigned char state;
    location current;
    location last;
} node;

//-----------------------------------------------------------------------------
// Global Variables
//-----------------------------------------------------------------------------
unsigned char xdata tx_uart_buf[BUFFER_SIZE];
unsigned char xdata rx_uart_buf[5];
unsigned char data tx_uart_head,tx_uart_tail;
unsigned char data cur_sensor;
unsigned char data header;
bit tx_active;
bit data_send;
bit ssaver;
bit new_location;
bit calibrated;
bit detach_sensing;
bit at_goal;
bit ext_motor;
```

```c
bit detaching;
bit data dump;
unsigned char data still_ticks;
unsigned char data red,green,blue;
unsigned char data led_ticks, data_ticks, batt_ticks;
unsigned char data byte_index;
unsigned int xdata good_index, bad_index, visited_index, unvisited_index;
unsigned int data hops_remaining;
unsigned int data sense ticks;
unsigned int data motor_ticks;
unsigned int data detach_ticks;
unsigned long data flash_address;
unsigned char idata pBuffer[14];

node idata myNode;
behavior idata myBehavior;
map xdata myMap;

//-----------------------------------------------------------------------------
// Function PROTOTYPES
//-----------------------------------------------------------------------------
void config (void);
void send uart byte (unsigned char byte);
//unsigned char check_uart_rx_buf (void);
void UART0_ISR (void);
unsigned int distance_traveled(location point1, location point2);
void write_address(unsigned long address);

//-----------------------------------------------------------------------------
// MAIN Routine
//-----------------------------------------------------------------------------
void main (void) {
    unsigned long index;

    // Configure Hardware
    config();

    // Power Up Timer;
    Timer0_ms(100);

    // Initialize Variables
    MOTOR1=0;
    tx_active = 0;
    cur_sensor = 0;
    header = 0;
    data_send = 0;
    calibrated = 0;
    detach_sensing = 0;
    at_goal = 0;
    detaching = 0;
    data_dump = 0;
    led_ticks = LED_BLINK;
    data_ticks = DATA_SEND;
    batt_ticks = BATT_POLL;
    motor_ticks = MOTOR_TIMEOUT;
    detach_ticks = DETACH_TIMEOUT;
    still_ticks = STILL_SAMPLES;
    good_index = 0;
    bad_index = 0;
    visited_index = 0;
    unvisited_index = 0;
    red = 0xFF;
    green = 0xFF;
    blue = 0xFF;
    ext_motor = 0;
```

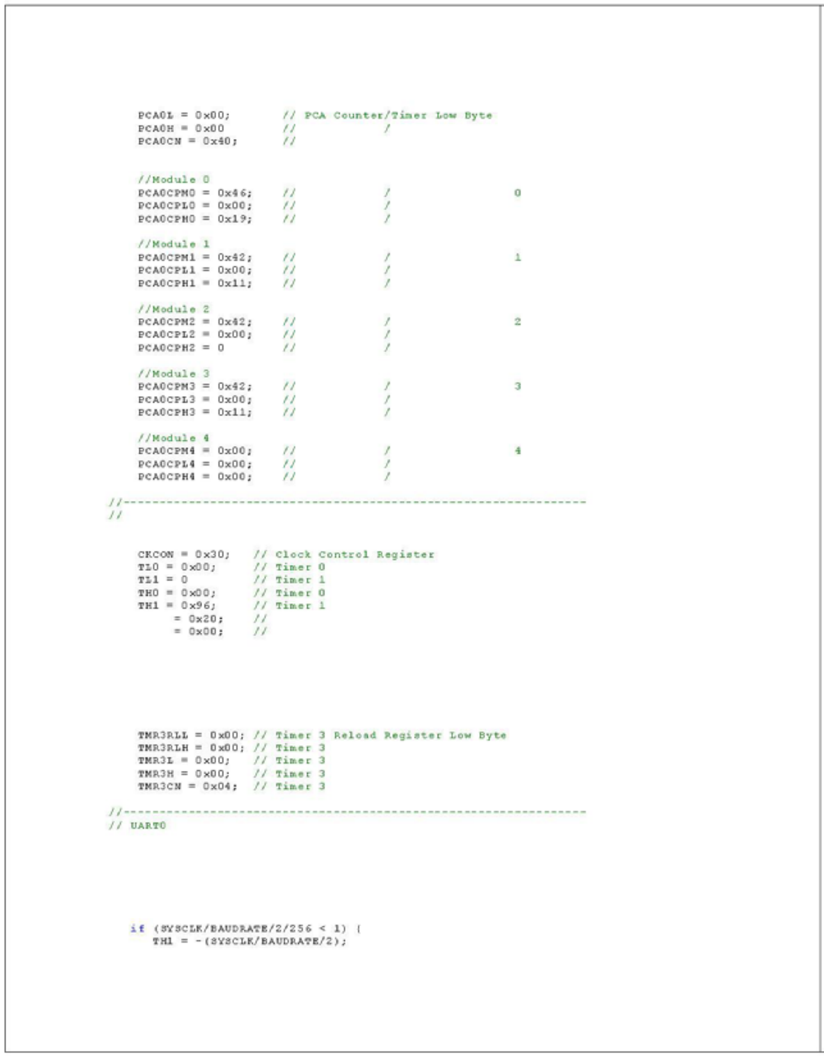
```c
    ssaver = 1;
    PCA0CPH1 = red;
    PCA0CPH2 = green;
    PCA0CPH3 = blue;

    // Initialize Flash Control
    flash_main_mem_read(pBuffer,0x00000000,5);
    if (pBuffer[0] != 0x6C) {
        flash address = 0x00000005;
        write_address(flash_address);
    } else {
        flash_address = ((unsigned long)pBuffer[1]) << 24;
        flash_address += ((unsigned long)pBuffer[2]) << 16;
        flash_address += ((unsigned long)pBuffer[3]) << 8;
        flash_address += ((unsigned long)pBuffer[4]);
    }
    pBuffer[0] = 0x50;           // POWER ON MESSAGE
    flash_main_mem_write(pBuffer,flash_address,1,BUFFER1);
    flash_address = flash_address + 1;
    write_address(flash_address);

    // Initialize behavior
    myBehavior.PowerThreshold = POWER_THRESHOLD;
#ifdef  BT_LOCATION
    myBehavior.goal.btX = GOAL_X;
    myBehavior.goal.btY = GOAL_Y;
#endif
#ifdef  GPS
    myBehavior.goal.latDir = GOAL_LAT_DIR;
    myBehavior.goal.latDeg = GOAL_LAT_DEG;
    myBehavior.goal.latMin = GOAL_LAT_MIN;
    myBehavior.goal.latSec = GOAL_LAT_SEC;
    myBehavior.goal.lonDir = GOAL_LON_DIR;
    myBehavior.goal.lonDeg = GOAL_LON_DEG;
    myBehavior.goal.lonMin = GOAL_LON_MIN;
    myBehavior.goal.lonSec = GOAL_LON_SEC;
    myBehavior.AltitudeThreshold = ALTITUDE_THRESHOLD;
#endif
    myBehavior.gotoGoal = GO_TO_GOAL;
    myBehavior.stopAtGoal = STOP_AT_GOAL;
    myBehavior.coverage = COVERAGE;
    myBehavior.LightThreshold = LIGHT_THRESHOLD;
    myBehavior.TemperatureThreshold = TEMPERATURE_THRESHOLD;
    myBehavior.VibrationThreshold = VIBRATION_THRESHOLD;
    myBehavior.AudioThreshold = AUDIO_THRESHOLD;
    myBehavior.senseTime = SENSE_TIME;
    myBehavior.hopsPerLocale = HOPS_PER_LOCALE;
    hops_remaining = myBehavior.hopsPerLocale;


    // Initialize Node State
    myNode.state = IDLE;

    EA = 1;                                    // Enable Interrupts

    // Set Bluetooth Name
    send_uart_byte(0x41);
    send_uart_byte(0x54);
    send_uart_byte(0x53);
    send_uart_byte(0x4E);
    send_uart_byte(0x2C);
    send_uart_byte(0x50);
    send_uart_byte(0x41);
    send_uart_byte(0x52);
    send_uart_byte(0x41);
send_uart_byte(0x4D);
send uart byte(0x4F);
send_uart_byte(0x52);
send_uart_byte(BT_NAME_H);
send_uart_byte(BT_NAME_L);
send_uart_byte(0x0D);
send_uart_byte(0x0A);

while (1) {                              // spin forever
    if (data_dump == 1) {
        EIE1 = 0x00;
        pBuffer[0] = "d";            // DATA DUMP MESSAGE
        pBuffer[1] = "a";            // DATA DUMP MESSAGE
        pBuffer[2] = "T";            // DATA DUMP MESSAGE
        pBuffer[3] = "a";            // DATA DUMP MESSAGE
        pBuffer[4] = "d";            // DATA DUMP MESSAGE
        pBuffer[5] = "U";            // DATA DUMP MESSAGE
        pBuffer[6] = "M";            // DATA DUMP MESSAGE
        pBuffer[7] = "P";            // DATA DUMP MESSAGE
        flash_main_mem_write(pBuffer,flash_address,8,BUFFER1);
        flash_address = flash_address + 8;
        write_address(flash_address);

        for(index=0;index<0x3FFFFF;index++) {
            flash_main_mem_read(pBuffer,index,1);
            send_uart_byte(pBuffer[0]);
            // if (index%32 == 0) Timer0_ms(30);
        }
        data_dump = 0;
        EIE1 = 0x88;
    }
    if (myNode.state == IDLE) {
        PCON |= 0x01;
    } else if (myNode.state == ATTACHED) {
        if (detaching == 0) {
            if (myNode.sensors[0] >= myBehavior.LightThreshold) {
                detach_sensing = 1;
                motor_ticks = MOTOR_TIMEOUT;
                detaching = 1;
                detach_ticks = DETACH_TIMEOUT;
                MOTOR1 = 1;
            }
            if (myNode.sensors[1] >= myBehavior.TemperatureThreshold) {
                detach_sensing = 1;
                motor_ticks = MOTOR_TIMEOUT;
                detaching = 1;
                detach_ticks = DETACH_TIMEOUT;
                MOTOR1 = 1;
            }
            if (abs(myNode.sensors[4]-myNode.baseline[4]) >= myBehavior.         ↙
AudioThreshold) {
                detach_sensing = 1;
                motor_ticks = MOTOR_TIMEOUT;
                detaching = 1;
                detach_ticks = DETACH_TIMEOUT;
                MOTOR1 = 1;
            }
            if (myBehavior.PowerThreshold > myNode.sensors[5]) {
                detach_sensing = 1;
                motor_ticks = MOTOR_TIMEOUT;
                detaching = 1;
                detach_ticks = DETACH_TIMEOUT;
                MOTOR1 = 1;
            }
        }
```

```
            if (new_location == 1) {
                if(distance traveled(myNode.current, myNode.last) >= HOPS_RESET)    ↙
    hops_remaining = myBehavior.hopsPerLocale;
                if (myBehavior.gotoGoal == 1) {
                    if (myNode.current.btX == myBehavior.goal.btX && myNode.current ↙
    .btY == myBehavior.goal.btY) {
                        at_goal = 1;
                        detach_sensing = 1;
                        motor ticks = MOTOR_TIMEOUT;
                        detaching = 1;
                        detach_ticks = DETACH_TIMEOUT;
                        MOTOR1 = 1;
                    } else {
                        if (distance_traveled(myNode.current, myBehavior.goal) >    ↙
    distance_traveled(myNode.last, myBehavior.goal)) {
                            if(hops remaining-- > 0) {
                                detach_sensing = 0;
                                motor_ticks = MOTOR_TIMEOUT;
                                detaching = 1;
                                detach_ticks = DETACH_TIMEOUT;
                                MOTOR1 = 1;
                            }
                        }
                    }
                }
                if (myBehavior.coverage == 1) {
                    myMap.visited[visited_index].btX = myNode.current.btX;
                    myMap.visited[visited_index++].btY = myNode.current.btY;
                    if (visited_index >= MAP_SIZE) visited_index--;
                }
                new_location = 0;
            }

        }


    }
}

void config (void) {

    unsigned char n = 0;

//-----------------------------------------------------------------
// CROSSBAR REGISTER CONFIGURATION
//
// NOTE: The crossbar register should be configured before any
// of the digital peripherals are enabled. The pinout of the
// device is dependent on the crossbar configuration so caution
// must be exercised when modifying the contents of the XBR0,
// XBR1 registers. For detailed information on
// Crossbar Decoder Configuration, refer to Application Note
// AN001, "Configuring the Port I/O Crossbar Decoder".
//-----------------------------------------------------------------

// Configure the XBRn Registers

    XBR0 = 0x03;     // Crossbar Register 1
    XBR1 = 0x44;     // Crossbar Register 2

// Select Pin I/O

// NOTE: Some peripheral I/O pins can function as either inputs or
// outputs, depending on the configuration of the peripheral. By default,
// the configuration utility will configure these I/O pins as push-pull
```

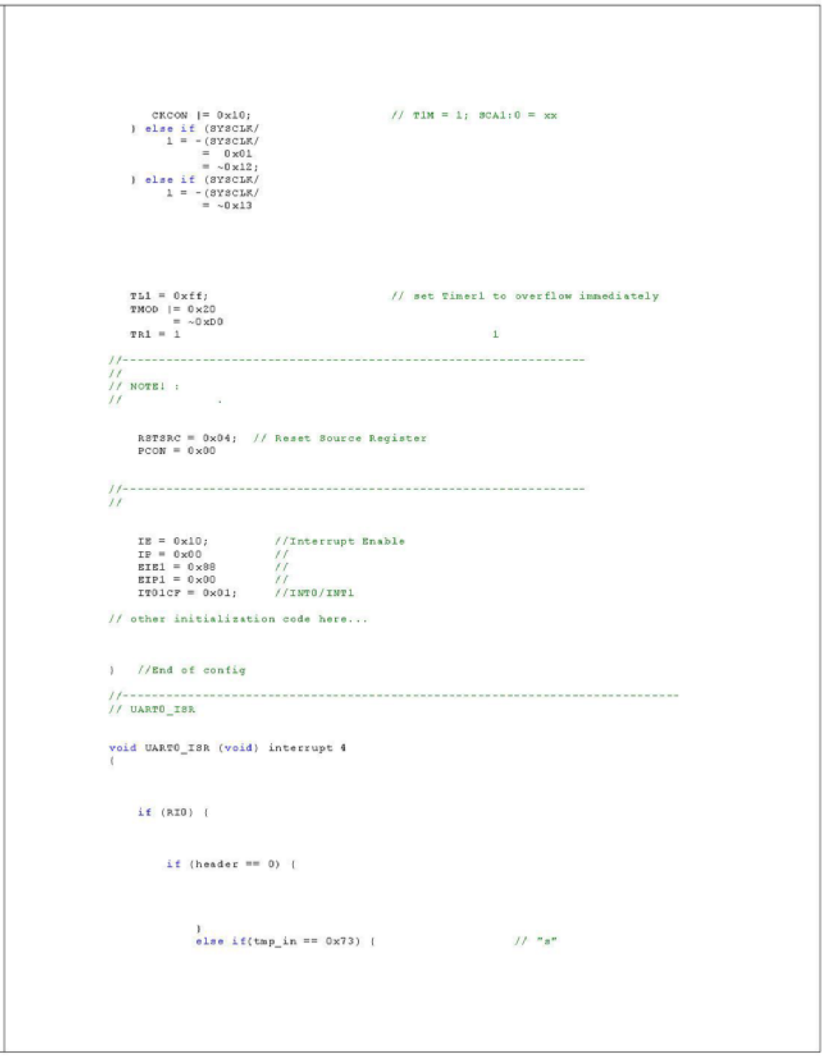


```
// outputs.
                        // Port configuration (1 = Push Pull Output)
    P0MDOUT = 0x51; // Output configuration for P0
    P1MDOUT = 0x1E; // Output configuration for P1
    P2MDOUT = 0x03; // Output configuration for P2
//     P1MDOUT = 0xFE; // Output configuration for P1
//     P2MDOUT = 0x07; // Output configuration for P2
    P3MDOUT = 0x01; // Output configuration for P3

    P0MDIN = 0x73;  // Input configuration for P0
    P1MDIN = 0x3E;  // Input configuration for P1
    P2MDIN = 0xDB;  // Input configuration for P2
    P3MDIN = 0xFF;  // Input configuration for P3

    P0SKIP = 0x8C;  //  Port 0 Crossbar Skip Register
    P1SKIP = 0x01;  //  Port 1 Crossbar Skip Register
    P2SKIP = 0x00;  //  Port 2 Crossbar Skip Register

    P0 = 0xFF;
    P1 = 0x7F;
    P2 = 0xF8;
    P3 = 0x01;

// View port pinout

        // The current Crossbar configuration results in the
        // following port pinout assignment:
        // Port 0
        // P0.0 = SPI Bus SCK      (Push-Pull Output)(Digital)
        // P0.1 = SPI Bus MISO     (Open-Drain Output/Input)(Digital)
        // P0.2 = Skipped          (Open-Drain Output/Input)(Analog)
        // P0.3 = Skipped          (Open-Drain Output/Input)(Analog)
        // P0.4 = UART0 TX         (Push-Pull Output)(Digital)
        // P0.5 = UART0 RX         (Open-Drain Output/Input)(Digital)
        // P0.6 = SPI Bus MOSI     (Push-Pull Output)(Digital)
        // P0.7 = Skipped          (Open-Drain Output/Input)(Analog)

        // Port 1
        // P1.0 = Skipped          (Open-Drain Output/Input)(Analog)
        // P1.1 = PCA CEX0         (Push-Pull Output)(Digital)
        // P1.2 = PCA CEX1         (Push-Pull Output)(Digital)
        // P1.3 = PCA CEX2         (Push-Pull Output)(Digital)
        // P1.4 = PCA CEX3         (Push-Pull Output)(Digital)
        // P1.5 = GP I/O           (Open-Drain Output/Input)(Digital)
        // P1.6 = GP I/O           (Open-Drain Output/Input)(Analog)
        // P1.7 = GP I/O           (Open-Drain Output/Input)(Analog)

        // Port 2
        // P2.0 = GP I/O           (Push-Pull Output)(Digital)
        // P2.1 = GP I/O           (Push-Pull Output)(Digital)
        // P2.2 = GP I/O           (Open-Drain Output/Input)(Analog)
        // P2.3 = GP I/O           (Open-Drain Output/Input)(Digital)
        // P2.4 = GP I/O           (Open-Drain Output/Input)(Digital)
        // P2.5 = GP I/O           (Open-Drain Output/Input)(Analog)
        // P2.6 = GP I/O           (Open-Drain Output/Input)(Digital)
        // P2.7 = GP I/O           (Open-Drain Output/Input)(Digital)

        // Port 3
        // P3.0 = GP I/O           (Push-Pull Output)(Digital)

//-----------------------------------------------------------------
// Comparator Register Configuration
//-----------------------------------------------------------------

    CPT0MX = 0x00;   // Comparator 0 MUX Selection Register
```

```
    CPT0MD = 0x00;   // Comparator 0 Mode Selection Register
    CPT0CN = 0x00;   // Comparator 0 Control Register

    CPT1MX = 0x00;   // Comparator 1 MUX Selection Register
    CPT1MD = 0x00;   // Comparator 1 Mode Selection Register
    CPT1CN = 0x00;   // Comparator 1 Control Register

//-----------------------------------------------------------------
// Oscillator Configuration
//-----------------------------------------------------------------

    OSCICN = 0x83;  // Internal Oscillator Control Register
    CLKSEL = 0x00;  // Oscillator Clock Select Register

    OSCXCN = 0x60;  // EXTERNAL Oscillator Control Register
    for (n = 0; n < 255; n++) ;              // wait for osc to start
    while ( (OSCXCN & 0x80) == 0 );          // wait for xtal to stabilize



//-----------------------------------------------------------------
// SPI Configuration
//-----------------------------------------------------------------

    SPI0CFG = 0x40; // SPI Configuration Register
    SPI0CKR = 0x20; // SPI Clock Rate Register
    SPI0CN = 0x01;  // SPI Control Register


//-----------------------------------------------------------------
// Reference Control Register Configuration
//-----------------------------------------------------------------

    REF0CN = 0x0a;  // Reference Control Register

//-----------------------------------------------------------------
// ADC Configuration
//-----------------------------------------------------------------

    AMX0P = 0x00;   // AMX0 Positive Select Register
    AMX0N = 0x1F;   // AMX0 Negative Select Register
    ADC0CF = 0xF8;  // ADC Configuration Register
    ADC0CN = 0x82;  // ADC Control Register

    ADC0H = 0x00;   // ADC Data MSB
    ADC0L = 0x00;   // ADC Data LSB
    ADC0LTH = 0x00; // ADC Less-Than High Byte Register
    ADC0LTL = 0x00; // ADC Less-Than Low Byte Register
    ADC0GTH = 0xFF; // ADC Greater-Than High Byte Register
    ADC0GTL = 0xFF; // ADC Greater-Than Low Byte Register

//-----------------------------------------------------------------
// SMBus Configuration
//-----------------------------------------------------------------

    SMB0CF = 0x00;  // SMBus Configuration Register
    SMB0DAT = 0x00; // SMBus Data Register
    SMB0CN = 0x00;  // SMBus Control Register

//-----------------------------------------------------------------
// PCA Configuration
//-----------------------------------------------------------------

    PCA0MD = 0x00;      // PCA Mode Register
```



```
    PCA0L = 0x00;       // PCA Counter/Timer Low Byte
    PCA0H = 0x00        //            /
    PCA0CN = 0x40;      //


    //Module 0
    PCA0CPM0 = 0x46;    //            /                 0
    PCA0CPL0 = 0x00;    //            /
    PCA0CPH0 = 0x19;    //            /

    //Module 1
    PCA0CPM1 = 0x42;    //            /                 1
    PCA0CPL1 = 0x00;    //            /
    PCA0CPH1 = 0x11;    //            /

    //Module 2
    PCA0CPM2 = 0x42;    //            /                 2
    PCA0CPL2 = 0x00;    //            /
    PCA0CPH2 = 0        //            /

    //Module 3
    PCA0CPM3 = 0x42;    //            /                 3
    PCA0CPL3 = 0x00;    //            /
    PCA0CPH3 = 0x11;    //            /

    //Module 4
    PCA0CPM4 = 0x00;    //            /                 4
    PCA0CPL4 = 0x00;    //            /
    PCA0CPH4 = 0x00;    //            /

//-----------------------------------------------------------------
//


    CKCON = 0x30;   // Clock Control Register
    TL0 = 0x00;     // Timer 0
    TL1 = 0         // Timer 1
    TH0 = 0x00;     // Timer 0
    TH1 = 0x96;     // Timer 1
         = 0x20;    //
         = 0x00;    //




    TMR3RLL = 0x00; // Timer 3 Reload Register Low Byte
    TMR3RLH = 0x00; // Timer 3
    TMR3L = 0x00;   // Timer 3
    TMR3H = 0x00;   // Timer 3
    TMR3CN = 0x04;  // Timer 3

//-----------------------------------------------------------------
// UART0




   if (SYSCLK/BAUDRATE/2/256 < 1) {
      TH1 = -(SYSCLK/BAUDRATE/2);
```



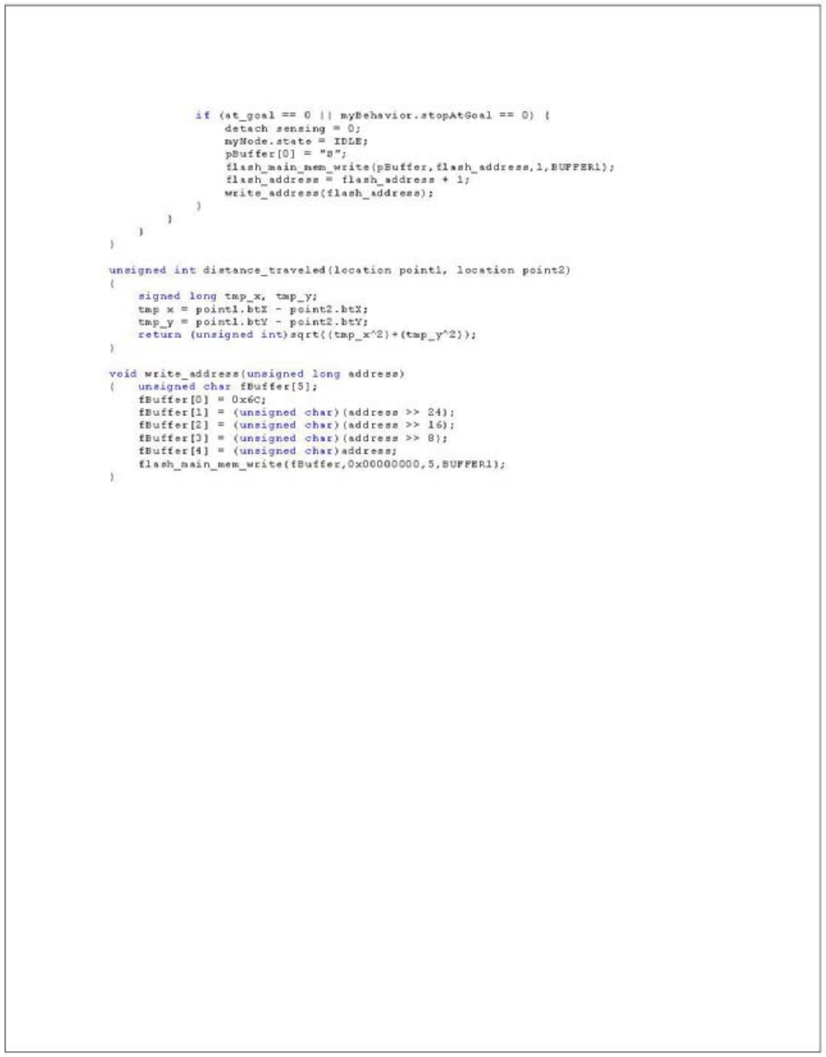
```
      CKCON |= 0x10;                    // T1M = 1; SCA1:0 = xx
   } else if (SYSCLK/
        1 = -(SYSCLK/
             =  0x01
             = ~0x12;
   } else if (SYSCLK/
        1 = -(SYSCLK/
             = ~0x13




   TL1 = 0xff;                          // set Timer1 to overflow immediately
   TMOD |= 0x20
        = ~0xD0
   TR1 = 1                                            1
//-----------------------------------------------------------------
//
// NOTE! :
//             .


    RSTSRC = 0x04;  // Reset Source Register
    PCON = 0x00


//-----------------------------------------------------------------
//


    IE = 0x10;          //Interrupt Enable
    IP = 0x00           //
    EIE1 = 0x88         //
    EIP1 = 0x00         //
    IT01CF = 0x01;      //INT0/INT1

// other initialization code here...



}   //End of config

//------------------------------------------------------------------------------
// UART0_ISR


void UART0_ISR (void) interrupt 4
{


    if (RI0) {



        if (header == 0) {



            }
            else if(tmp_in == 0x73) {                 // "s"
            MOTOR1 = 0;
            ext_motor = 0;
        }
        else if(tmp_in == 0x70) ssaver = 1;            // "p"
        else if(tmp_in == 0x72) header = 2;            // "r"
        else if(tmp_in == 0x67) header = 2;            // "g"
        else if(tmp_in == 0x62) header = 2;            // "b"
        else if(tmp_in == 0x63) data_send = 1;         // "c"
        else if(tmp in == 0x6D) data send = 0;         // "m"
        else if(tmp_in == 0x7A) calibrated = 0;        // "z"
        else if(tmp_in == 0x6C) {                      // "l"
            header = 2;
            byte_index = 1;
        }
        else if (tmp_in == 0x61) {                     // "a"
            header = 2;
            byte_index = 1;
        }
        else if (tmp_in == 0x64) {                     // "d"
            data_dump = 1;
        }
        else if (tmp_in == 0x66) {                     // "f"
            flash address = 0x00000005;
            write_address(flash_address);
        }
    } else if (header == 2) {
        if (rx_uart_buf[0] == 0x72) {
            red = tmp_in;
            ssaver = 0;
            header = 0;
        } else if (rx_uart_buf[0] == 0x67) {
            green = tmp_in;
            ssaver = 0;
            header = 0;
        } else if (rx_uart_buf[0] == 0x62) {
            blue = tmp_in;
            ssaver = 0;
            header = 0;
        } else if (rx_uart_buf[0] == 0x6C) {
            rx_uart_buf[byte_index++] = tmp_in;
            if (byte_index == 6) {
                header = 0;
                if (rx_uart_buf[1] == 0x47) {            // "G"
                    myMap.good[good_index].btX = (rx_uart_buf[3] << 8) +          ↙
rx_uart_buf[2];
                    myMap.good[good_index++].btY = (rx_uart_buf[5] << 8) +        ↙
rx_uart_buf[4];
                    if (good_index >= MAP_SIZE) good_index--;
                    flash_main_mem_write(rx_uart_buf,flash_address,6,BUFFER1);
                    flash_address = flash_address + 6;
                    write_address(flash_address);
                } else if (rx_uart_buf[1] == 0x42) {    // "B"
                    myMap.bad[bad_index].btX = (rx_uart_buf[3] << 8) +            ↙
rx_uart_buf[2];
                    myMap.bad[bad_index++].btY = (rx_uart_buf[5] << 8) +          ↙
rx_uart_buf[4];
                    if (bad_index >= MAP_SIZE) bad_index--;
                    flash_main_mem_write(rx_uart_buf,flash_address,6,BUFFER1);
                    flash_address = flash_address + 6;
                    write_address(flash_address);
                } else if (rx_uart_buf[1] == 0x56) {    // "V"
                    myMap.visited[visited_index].btX = (rx_uart_buf[3] << 8) + ↙
rx_uart_buf[2];
                    myMap.visited[visited_index++].btY = (rx_uart_buf[5] << 8) ↙
+ rx_uart_buf[4];
```

```
                    if (visited_index >= MAP_SIZE) visited_index--;
                    flash main mem write(rx uart buf,flash_address,6,BUFFER1);
                    flash_address = flash_address + 6;
                    write_address(flash_address);
                } else if (rx_uart_buf[1] == 0x55) {    // "U"
                    myMap.unvisited[unvisited_index].btX = (rx_uart_buf[3] << 8 ↙
) + rx_uart_buf[2];
                    myMap.unvisited[unvisited_index++].btY = (rx_uart_buf[5] << ↙
 8) + rx uart buf[4];
                    if (unvisited_index >= MAP_SIZE) unvisited_index--;
                    flash_main_mem_write(rx_uart_buf,flash_address,6,BUFFER1);
                    flash_address = flash_address + 6;
                    write_address(flash_address);
                } else if (rx_uart_buf[1] == 0x43) {    // "C"
                    tmp_x = (rx_uart_buf[3] << 8) + rx_uart_buf[2];
                    tmp y = (rx uart buf[5] << 8) + rx_uart_buf[4];
                    new_location = 0;
                    if (tmp_x != myNode.current.btX) {
                        myNode.last.btX = myNode.current.btX;
                        myNode.current.btX = tmp_x;
                        new_location = 1;
                        flash_main_mem_write(rx_uart_buf,flash_address,6,BUFFER ↙
1);

                        flash_address = flash_address + 6;
                        write_address(flash_address);
                    }
                    if (tmp_y != myNode.current.btY) {
                        myNode.last.btY = myNode.current.btY;
                        myNode.current.btY = tmp_y;
                        new_location = 1;
                        flash_main_mem_write(rx_uart_buf,flash_address,6,BUFFER ↙
1);

                        flash_address = flash_address + 6;
                        write_address(flash_address);
                    }
                }

            }
        } else if (rx_uart_buf[0] == 0x61) {
            rx_uart_buf[byte_index++] = tmp_in;
            if (byte_index == 4) {
                header = 0;
                if (rx_uart_buf[1] == 0x50) {            // "P"
                    myBehavior.PowerThreshold = (rx_uart_buf[2] << 8) +          ↙
rx_uart_buf[3];
                } else if (rx_uart_buf[1] == 0x58) {            // "X"
                    myBehavior.goal.btX = (rx_uart_buf[2] << 8) + rx_uart_buf[3 ↙
];
                } else if (rx_uart_buf[1] == 0x59) {            // "Y"
                    myBehavior.goal.btY = (rx_uart_buf[2] << 8) + rx_uart_buf[3 ↙
];
                } else if (rx_uart_buf[1] == 0x47) {            // "G"
                    myBehavior.gotoGoal = rx_uart_buf[3];
                } else if (rx_uart_buf[1] == 0x53) {            // "S"
                    myBehavior.stopAtGoal = rx_uart_buf[3];
                } else if (rx_uart_buf[1] == 0x43) {            // "C"
                    myBehavior.coverage = rx_uart_buf[3];
                } else if (rx_uart_buf[1] == 0x4C) {            // "L"
                    myBehavior.LightThreshold = (rx_uart_buf[2] << 8) +          ↙
rx_uart_buf[3];
                } else if (rx_uart_buf[1] == 0x56) {            // "V"
                    myBehavior.VibrationThreshold = (rx_uart_buf[2] << 8) +      ↙
rx_uart_buf[3];
                } else if (rx_uart_buf[1] == 0x54) {            // "T"
                    myBehavior.TemperatureThreshold = (rx_uart_buf[2] << 8) +    ↙
```

```
    rx_uart_buf[3];
                    } else if (rx uart buf[1] == 0x41) {         // "A"
                        myBehavior.AltitudeThreshold = (rx_uart_buf[2] << 8) +       ↙
    rx_uart_buf[3];
                    } else if (rx_uart_buf[1] == 0x4E) {         // "N"
                        myBehavior.AudioThreshold = (rx_uart_buf[2] << 8) +          ↙
    rx_uart_buf[3];
                    } else if (rx_uart_buf[1] == 0x45) {         // "E"
                        myBehavior.senseTime = (rx_uart_buf[2] << 8) + rx_uart_buf[ ↙
    3];
                    } else if (rx_uart_buf[1] == 0x48) {         // "H"
                        myBehavior.hopsPerLocale = (rx_uart_buf[2] << 8) +           ↙
    rx_uart_buf[3];
                    }
                }
            }
        }

    } else if (TI0) {
        TI0 = 0;
        if (tx_uart_tail != tx_uart_head) {
            //BUFFER IS NOT EMPTY -- SEND NEXT BYTE
            SBUF0 = tx uart buf[tx uart head++];
            if (tx_uart_head == BUFFER_SIZE) tx_uart_head = 0;
        } else {
            tx_active = 0;
        }
    }
}

void send_uart_byte (unsigned char byte)
{
  unsigned char buf_empty = 0;
  if( tx_uart_tail == tx_uart_head ) buf_empty = 1;
  tx_uart_buf[tx_uart_tail++] = byte;
  if (tx_uart_tail == BUFFER_SIZE) tx_uart_tail = 0;
  if (buf_empty && (tx_active == 0)) {
    tx_active = 1;
    TI0 = 1;                                        //MANUALLY TRIGGER INTERRUPT
  }

}

/*
unsigned char check_uart_rx_buf (void)
{
    if (rx_uart_head - rx_uart_tail == 1) {
        return (254);
    } else if ((rx_uart_head == 0) && (rx_uart_tail == (BUFFER_SIZE-1))) {
        return (254);
    } else {
        return (0);
    }
}
*/

void ADC_ISR (void) interrupt 10
{
    unsigned char adch,adcl,i,j;
    adcl = ADC0L;
    adch = ADC0H;
    if (cur_sensor == 0) {
        myNode.sensors[0] = (adch << 8) + adcl;
        AMX0P = 0x09;   // AMX0 Positive Select Register
        cur_sensor = 1;
```

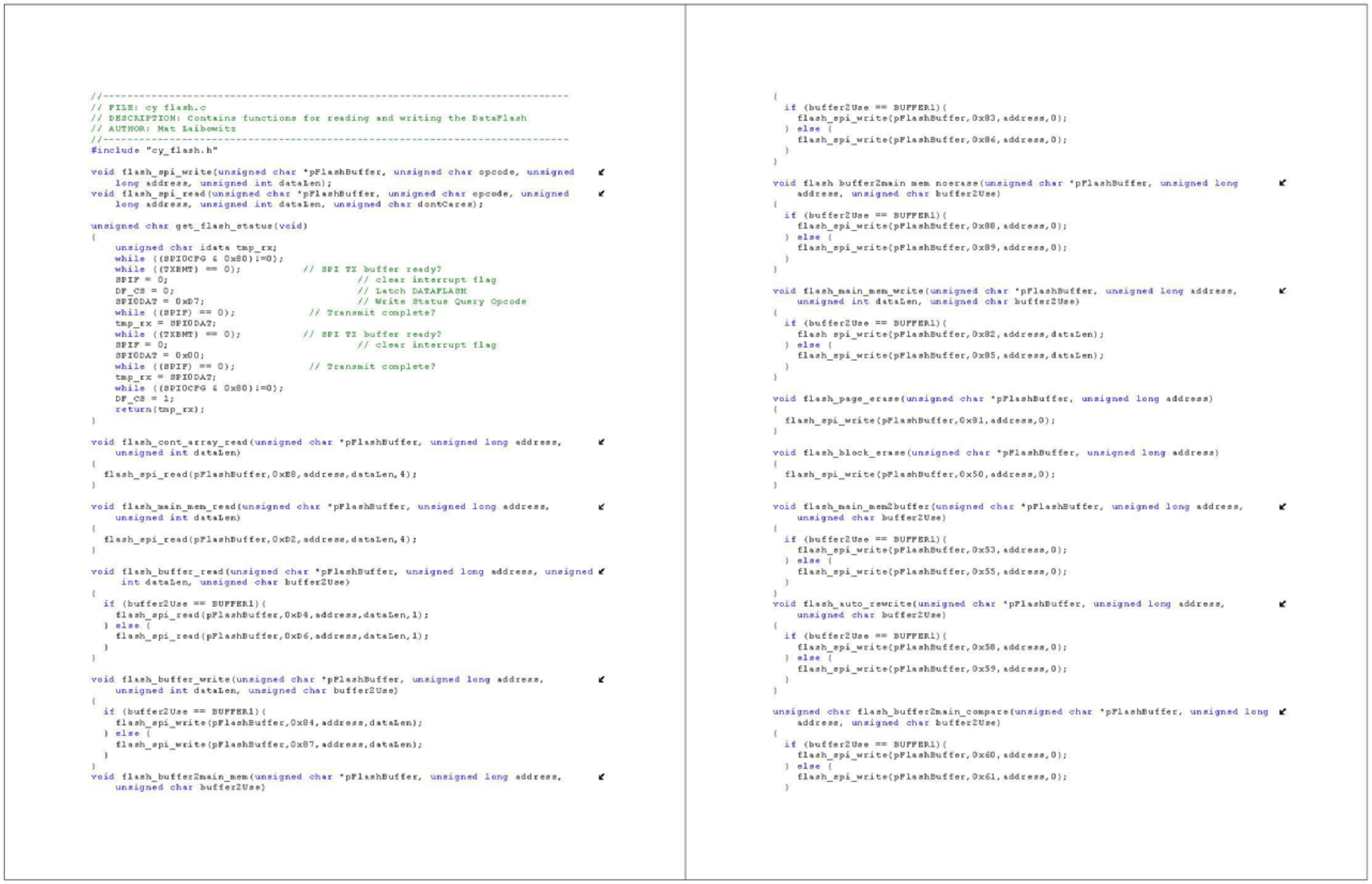
```
    } else if (cur_sensor == 1) {
        myNode.sensors[1] = (adch << 8) + adcl;
        AMX0P = 0x0A;   // AMX0 Positive Select Register
        cur_sensor = 2;
    } else if (cur_sensor == 2) {
        myNode.sensors[2] = (adch << 8) + adcl;
        AMX0P = 0x07;   // AMX0 Positive Select Register
        cur_sensor = 3;
    } else if (cur sensor == 3) {
        myNode.sensors[3] = (adch << 8) + adcl;
        AMX0P = 0x06;   // AMX0 Positive Select Register
        cur_sensor = 4;
    } else if (cur_sensor == 4) {
        myNode.sensors[4] = (adch << 8) + adcl;
        AMX0P = 0x00;   // AMX0 Positive Select Register

        j=1;
        pBuffer[0] = 0x44;
        for (i=0;i<6;i++) {
            pBuffer[j++] = (unsigned char)(myNode.sensors[i] >> 8);
            pBuffer[j++] = (unsigned char)myNode.sensors[i];
        }
        flash main mem write(pBuffer,flash address,11,BUFFER1);
        flash_address = flash_address + 11;
        write_address(flash_address);

        if (calibrated == 0) {
            for (i=0;i<6;i++) {
                myNode.baseline[i] = myNode.sensors[i];
            }
            calibrated = 1;
        } else {
            if (myNode.state == IDLE || myNode.state == SENSING) {
                if (abs(myNode.sensors[3] - myNode.baseline[3]) > MOTION_THRESHOLD) ↙
    {
                    still_ticks = STILL_SAMPLES;
                    hops_remaining = myBehavior.hopsPerLocale;
                    myNode.state = ATTACHED;
                    pBuffer[0] = "A";
                    flash_main_mem_write(pBuffer,flash_address,1,BUFFER1);
                    flash_address = flash_address + 1;
                    write_address(flash_address);
                } else {
                    myNode.baseline[3] = myNode.sensors[3];
                }
                if (abs(myNode.sensors[2] - myNode.baseline[2]) > MOTION_THRESHOLD) ↙
    {
                    still_ticks = STILL_SAMPLES;
                    hops_remaining = myBehavior.hopsPerLocale;
                    myNode.state = ATTACHED;
                    pBuffer[0] = "A";
                    flash_main_mem_write(pBuffer,flash_address,1,BUFFER1);
                    flash_address = flash_address + 1;
                    write_address(flash_address);
                } else {
                    myNode.baseline[2] = myNode.sensors[2];
                }
            } else if (myNode.state == ATTACHED) {
                if (abs(myNode.sensors[3] - myNode.baseline[3]) > STILL_THRESHOLD)  ↙
   {
                    still_ticks = STILL_SAMPLES;
                } else {
                    if (--still_ticks == 0) {
                        still_ticks = STILL_SAMPLES;
                        MOTOR1 = 0;
                        detaching = 0;
                        if (detach sensing == 1) {
                            sense_ticks = myBehavior.senseTime;
                            myNode.state = SENSING;
                            pBuffer[0] = "S";
                            flash_main_mem_write(pBuffer,flash_address,1,BUFFER1);
                            flash_address = flash_address + 1;
                            write_address(flash_address);
                        } else {
                            myNode.state = IDLE;
                            pBuffer[0] = "I";
                            flash_main_mem_write(pBuffer,flash_address,1,BUFFER1);
                            flash_address = flash_address + 1;
                            write_address(flash_address);
                        }
                    }
                }
                myNode.baseline[3] = myNode.sensors[3];
                if (abs(myNode.sensors[2] - myNode.baseline[2]) > STILL_THRESHOLD)  ↙
    {
                    still_ticks = STILL_SAMPLES;
                } else {
                    if (--still ticks == 0) {
                        still_ticks = STILL_SAMPLES;
                        MOTOR1 = 0;
                        detaching = 0;
                        if (detach_sensing == 1) {
                            sense_ticks = myBehavior.senseTime;
                            myNode.state = SENSING;
                            pBuffer[0] = "S";
                            flash_main_mem_write(pBuffer,flash_address,1,BUFFER1);
                            flash_address = flash_address + 1;
                            write_address(flash_address);
                        } else {
                            myNode.state = IDLE;
                            pBuffer[0] = "I";
                            flash_main_mem_write(pBuffer,flash_address,1,BUFFER1);
                            flash_address = flash_address + 1;
                            write_address(flash_address);
                        }
                    }
                }
                myNode.baseline[2] = myNode.sensors[2];
            }
        }
        cur_sensor = 0;
    }
    ADC0CN &= ~(0x20);
    ADC0CN |= 0x10;
}

void TIMER3_ISR (void) interrupt 14
{
    unsigned char i,vh, vl, ret_hdqh, ret_hdql;

    TMR3CN &= ~(0x80);
    if (--led_ticks == 0) {
        led_ticks = LED_BLINK;
        if (ssaver == 1) {
            if (myNode.state == IDLE) {
                if (green == 0) {
                    green = 0xFF;
                } else {
                    green = 0;
                }
```

```
                red = 0xFF;
                blue = 0xFF;
            } else if (myNode.state == ATTACHED) {
                if (red == 0x9F) {
                    red = 0xFF;
                    green = 0xFF;
                } else {
                    red = 0x9F;
                    green = 0;
                }
                blue = 0xFF;
            } else {
                if (red == 0) {
                    red = 0xFF;
                } else {
                    red = 0;
                }
                blue = 0xFF;
                green = 0xFF;
            }
            PCA0CPH1 = red;
            PCA0CPH2 = green;
            PCA0CPH3 = blue;
        }
    }

    if (--batt_ticks == 0) {
        batt_ticks = BATT_POLL;
        vh = HDQreadbyte(0x09, &ret_hdqh);
        vl = HDQreadbyte(0x08, &ret_hdql);
        if (vh == 1 && vl == 1) {
            myNode.sensors[5] = (ret_hdqh << 8) + ret_hdql;
        }
    }

    if(data_send == 1) {
        if (--data_ticks == 0) {
            data_ticks = DATA_SEND;
            for (i=0;i<6;i++) {
                send_uart_byte(myNode.sensors[i] >> 8);
                send_uart_byte((unsigned char)myNode.sensors[i]);
            }
            send_uart_byte(0x0d);
            send_uart_byte(0x0a);
        }
    }

    if (MOTOR1 == 1 && ext_motor == 0) {
        if (--motor_ticks == 0) {
            motor_ticks = MOTOR_TIMEOUT;
            MOTOR1 = 0;
            still_ticks = STILL_SAMPLES;
        }
    }

    if (detaching == 1 && MOTOR1 == 0) {
        if (--detach_ticks == 0) {
            detach_ticks = DETACH_TIMEOUT;
            detaching = 0;
        }
    }

    if (myNode.state == SENSING) {
        if (--sense_ticks == 0) {
            sense_ticks = myBehavior.senseTime;
```



```
            if (at_goal == 0 || myBehavior.stopAtGoal == 0) {
                detach sensing = 0;
                myNode.state = IDLE;
                pBuffer[0] = "S";
                flash_main_mem_write(pBuffer,flash_address,1,BUFFER1);
                flash_address = flash_address + 1;
                write_address(flash_address);
            }
        }
    }
}

unsigned int distance_traveled(location point1, location point2)
{
    signed long tmp_x, tmp_y;
    tmp x = point1.btX - point2.btX;
    tmp_y = point1.btY - point2.btY;
    return (unsigned int)sqrt((tmp_x^2)+(tmp_y^2));
}

void write_address(unsigned long address)
{   unsigned char fBuffer[5];
    fBuffer[0] = 0x6C;
    fBuffer[1] = (unsigned char)(address >> 24);
    fBuffer[2] = (unsigned char)(address >> 16);
    fBuffer[3] = (unsigned char)(address >> 8);
    fBuffer[4] = (unsigned char)address;
    flash_main_mem_write(fBuffer,0x00000000,5,BUFFER1);
}
```

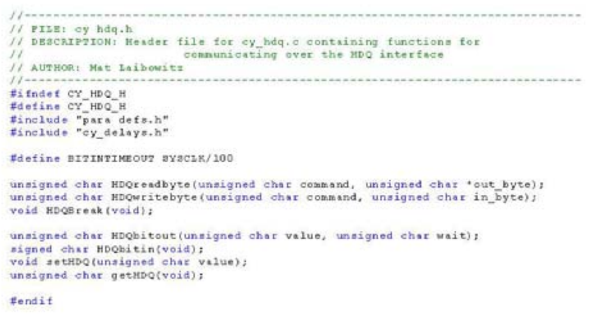
```c
//----------------------------------------------------------------------------
// FILE: cy flash.h
// DESCRIPTION: Header file for cy_flash.c containing functions for reading
//                         and writing the DataFlash
// AUTHOR: Mat Laibowitz
//----------------------------------------------------------------------------
#ifndef CY_FLASH_H
#define CY_FLASH_H
#include "para_defs.h"

#define BUFFER1 1
#define BUFFER2 2

unsigned char get_flash_status(void);
void flash_cont_array_read(unsigned char *pFlashBuffer, unsigned long address,      ↙
    unsigned int dataLen);
void flash_main_mem_read(unsigned char *pFlashBuffer, unsigned long address,        ↙
    unsigned int dataLen);
void flash_buffer_read(unsigned char *pFlashBuffer, unsigned long address, unsigned ↙
     int dataLen, unsigned char buffer2use);
void flash_buffer_write(unsigned char *pFlashBuffer, unsigned long address,         ↙
    unsigned int dataLen, unsigned char buffer2use);
void flash buffer2main mem(unsigned char *pFlashBuffer, unsigned long address,      ↙
    unsigned char buffer2use);
void flash_buffer2main_mem_noerase(unsigned char *pFlashBuffer, unsigned long       ↙
    address, unsigned char buffer2use);
void flash_main_mem_write(unsigned char *pFlashBuffer, unsigned long address,       ↙
    unsigned int dataLen, unsigned char buffer2use);
void flash_page_erase(unsigned char *pFlashBuffer, unsigned long address);
void flash_block_erase(unsigned char *pFlashBuffer, unsigned long address);
void flash_main_mem2buffer(unsigned char *pFlashBuffer, unsigned long address,      ↙
    unsigned char buffer2use);
void flash_auto_rewrite(unsigned char *pFlashBuffer, unsigned long address,         ↙
    unsigned char buffer2use);
unsigned char flash_buffer2main_compare(unsigned char *pFlashBuffer, unsigned long  ↙
    address, unsigned char buffer2use);

#endif
```



```c
//------------------------------------------------------------------------------
// FILE: cy flash.c
// DESCRIPTION: Contains functions for reading and writing the DataFlash
// AUTHOR: Mat Laibowitz
//------------------------------------------------------------------------------
#include "cy_flash.h"

void flash_spi_write(unsigned char *pFlashBuffer, unsigned char opcode, unsigned ↙
    long address, unsigned int dataLen);
void flash_spi_read(unsigned char *pFlashBuffer, unsigned char opcode, unsigned ↙
    long address, unsigned int dataLen, unsigned char dontCares);

unsigned char get_flash_status(void)
{
    unsigned char idata tmp_rx;
    while ((SPI0CFG & 0x80)!=0);
    while ((TXBMT) == 0);          // SPI TX buffer ready?
    SPIF = 0;                               // clear interrupt flag
    DF_CS = 0;                              // Latch DATAFLASH
    SPI0DAT = 0xD7;                         // Write Status Query Opcode
    while ((SPIF) == 0);            // Transmit complete?
    tmp_rx = SPI0DAT;
    while ((TXBMT) == 0);          // SPI TX buffer ready?
    SPIF = 0;                               // clear interrupt flag
    SPI0DAT = 0x00;
    while ((SPIF) == 0);            // Transmit complete?
    tmp_rx = SPI0DAT;
    while ((SPI0CFG & 0x80)!=0);
    DF_CS = 1;
    return(tmp_rx);
}

void flash_cont_array_read(unsigned char *pFlashBuffer, unsigned long address, ↙
    unsigned int dataLen)
{
  flash_spi_read(pFlashBuffer,0xE8,address,dataLen,4);
}

void flash_main_mem_read(unsigned char *pFlashBuffer, unsigned long address, ↙
    unsigned int dataLen)
{
  flash_spi_read(pFlashBuffer,0xD2,address,dataLen,4);
}

void flash_buffer_read(unsigned char *pFlashBuffer, unsigned long address, unsigned ↙
     int dataLen, unsigned char buffer2Use)
{
  if (buffer2Use == BUFFER1){
    flash_spi_read(pFlashBuffer,0xD4,address,dataLen,1);
  } else {
    flash_spi_read(pFlashBuffer,0xD6,address,dataLen,1);
  }
}

void flash_buffer_write(unsigned char *pFlashBuffer, unsigned long address, ↙
    unsigned int dataLen, unsigned char buffer2Use)
{
  if (buffer2Use == BUFFER1){
    flash_spi_write(pFlashBuffer,0x84,address,dataLen);
  } else {
    flash_spi_write(pFlashBuffer,0x87,address,dataLen);
  }
}
void flash_buffer2main_mem(unsigned char *pFlashBuffer, unsigned long address, ↙
    unsigned char buffer2Use)
{
  if (buffer2Use == BUFFER1){
    flash_spi_write(pFlashBuffer,0x83,address,0);
  } else {
    flash_spi_write(pFlashBuffer,0x86,address,0);
  }
}

void flash buffer2main mem noerase(unsigned char *pFlashBuffer, unsigned long ↙
    address, unsigned char buffer2Use)
{
  if (buffer2Use == BUFFER1){
    flash_spi_write(pFlashBuffer,0x88,address,0);
  } else {
    flash_spi_write(pFlashBuffer,0x89,address,0);
  }
}

void flash_main_mem_write(unsigned char *pFlashBuffer, unsigned long address, ↙
    unsigned int dataLen, unsigned char buffer2Use)
{
  if (buffer2Use == BUFFER1){
    flash spi_write(pFlashBuffer,0x82,address,dataLen);
  } else {
    flash_spi_write(pFlashBuffer,0x85,address,dataLen);
  }
}

void flash_page_erase(unsigned char *pFlashBuffer, unsigned long address)
{
  flash_spi_write(pFlashBuffer,0x81,address,0);
}

void flash_block_erase(unsigned char *pFlashBuffer, unsigned long address)
{
  flash_spi_write(pFlashBuffer,0x50,address,0);
}

void flash_main_mem2buffer(unsigned char *pFlashBuffer, unsigned long address, ↙
    unsigned char buffer2Use)
{
  if (buffer2Use == BUFFER1){
    flash_spi_write(pFlashBuffer,0x53,address,0);
  } else {
    flash_spi_write(pFlashBuffer,0x55,address,0);
  }
}
void flash_auto_rewrite(unsigned char *pFlashBuffer, unsigned long address, ↙
    unsigned char buffer2Use)
{
  if (buffer2Use == BUFFER1){
    flash_spi_write(pFlashBuffer,0x58,address,0);
  } else {
    flash_spi_write(pFlashBuffer,0x59,address,0);
  }
}

unsigned char flash_buffer2main_compare(unsigned char *pFlashBuffer, unsigned long ↙
    address, unsigned char buffer2Use)
{
  if (buffer2Use == BUFFER1){
    flash_spi_write(pFlashBuffer,0x60,address,0);
  } else {
    flash_spi_write(pFlashBuffer,0x61,address,0);
  }
```

```
//  while (!(P1IN & DF_RDY));
  return (get_flash_status() & 0x40) ;
}
void flash_spi_write(unsigned char *pFlashBuffer, unsigned char opcode, unsigned    ↙
    long address, unsigned int dataLen)
{
  unsigned int i;

    while((get_flash_status() & 0x80)==0);

    SPIF=0;                         // clear interrupt flag
    while ((TXBMT) == 0);              // SPI TX buffer ready?
    DF_CS = 0;                                  // Latch DATAFLASH
    SPI0DAT = opcode;                           // Write Opcode
    while ((TXBMT) == 0);              // SPI TX buffer ready?
    SPI0DAT = (address & 0x003F0000) >> 16;
    while ((TXBMT) == 0);              // SPI TX buffer ready?
    SPI0DAT = (address & 0x0000FF00) >> 8;
    while ((TXBMT) == 0);              // SPI TX buffer ready?
    SPI0DAT = (address & 0x000000FF);
    for(i=0;i<dataLen;i++){
        while ((TXBMT) == 0);              // SPI TX buffer ready?
        SPIF=0;
        SPI0DAT = pFlashBuffer[i];
    }
    while ((TXBMT) == 0);              // SPI TX buffer ready?
    while ((SPIF) == 0);                // Transmit complete?
    while ((SPI0CFG & 0x80)!=0);
    DF_CS = 1;
}


void flash_spi_read(unsigned char *pFlashBuffer, unsigned char opcode, unsigned    ↙
    long address, unsigned int dataLen, unsigned char dontCares)
{
  unsigned int i;

    while((get_flash_status() & 0x80)==0);

    SPIF = 0;                                   // clear interrupt flag
    while ((TXBMT) == 0);              // SPI TX buffer ready?
    DF_CS = 0;                                  // Latch DATAFLASH
    SPI0DAT = opcode;                           // Write Opcode
    while ((TXBMT) == 0);              // SPI TX buffer ready?
    SPI0DAT = (address & 0x003F0000) >> 16;
    while ((TXBMT) == 0);              // SPI TX buffer ready?
    SPI0DAT = (address & 0x0000FF00) >> 8;
    while ((TXBMT) == 0);              // SPI TX buffer ready?
    SPI0DAT = (address & 0x000000FF);
    for(i=0;i<dontCares;i++) {
        while ((TXBMT) == 0);          // SPI TX buffer ready?
        SPI0DAT = 0x00;
    }
    while ((TXBMT) == 0);              // SPI TX buffer ready?
    SPIF = 0;                                   // clear interrupt flag
    while ((SPIF) == 0);                // Transmit complete?

    for(i=0;i<dataLen;i++){
        while ((TXBMT) == 0);          // SPI TX buffer ready?
        SPIF = 0;                                   // clear interrupt flag
        SPI0DAT = 0x00;
        while ((SPIF) == 0);                // Transmit complete?
        pFlashBuffer[i] = SPI0DAT;
    }
    while ((TXBMT) == 0);              // SPI TX buffer ready?
```

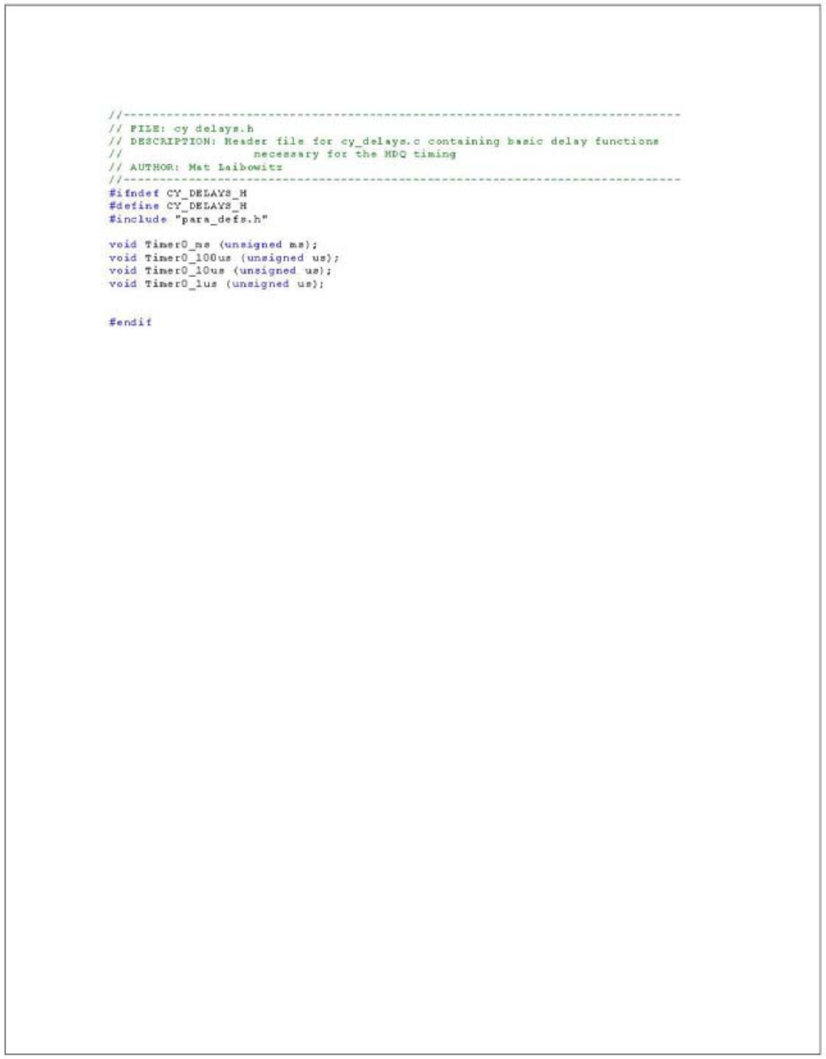
```
    while ((SPIF) == 0);                // Transmit complete?
    while ((SPI0CFG & 0x80)!=0);
    DF_CS=1;
}
```



```c
//----------------------------------------------------------------------------
// FILE: cy hdq.h
// DESCRIPTION: Header file for cy_hdq.c containing functions for
//                         communicating over the HDQ interface
// AUTHOR: Mat Laibowitz
//----------------------------------------------------------------------------
#ifndef CY_HDQ_H
#define CY_HDQ_H
#include "para defs.h"
#include "cy_delays.h"

#define BITINTIMEOUT SYSCLK/100

unsigned char HDQreadbyte(unsigned char command, unsigned char *out_byte);
unsigned char HDQwritebyte(unsigned char command, unsigned char in_byte);
void HDQBreak(void);

unsigned char HDQbitout(unsigned char value, unsigned char wait);
signed char HDQbitin(void);
void setHDQ(unsigned char value);
unsigned char getHDQ(void);

#endif
```

```
//-----------------------------------------------------------------------------
// FILE: cy delays.c
// DESCRIPTION: Containins basic delay functions necessary for the HDQ timing
// AUTHOR: Mat Laibowitz
//-----------------------------------------------------------------------------
#include <c8051f310.h>
#include <intrins.h>
#include "cy_delays.h"

//-----------------------------------------------------------------------------
// Timer0_ms
//-----------------------------------------------------------------------------
//
// Configure Timer0 to delay <ms> milliseconds before returning.
//
void Timer0_ms (unsigned ms)
{
   unsigned i;                          // millisecond counter

   TCON  &= ~0x30;                      // STOP Timer0 and clear overflow flag
   TMOD  &= ~0x0f;                      // configure Timer0 to 16-bit mode
   TMOD  |=  0x01;
   CKCON |=  0x04;                      // Timer0 counts SYSCLKs

   for (i = 0; i < ms; i++)             // count milliseconds
   {
      TR0 = 0;                          // STOP Timer0
      TH0 = (-SYSCLK/1000) >> 8;        // set Timer0 to overflow in 1ms
      TL0 = -SYSCLK/1000;
      TR0 = 1;                          // START Timer0
      while (TF0 == 0);                 // wait for overflow
      TF0 = 0;                          // clear overflow indicator
   }
}

//-----------------------------------------------------------------------------
// Timer0_100us
//-----------------------------------------------------------------------------
//
// Configure Timer0 to delay <us>*100 microseconds before returning.
//
void Timer0_100us (unsigned us)
{
   unsigned i;                          // millisecond counter

   TCON  &= ~0x30;                      // STOP Timer0 and clear overflow flag
   TMOD  &= ~0x0f;                      // configure Timer0 to 16-bit mode
   TMOD  |=  0x01;
   CKCON |=  0x04;                      // Timer0 counts SYSCLKs

   for (i = 0; i < us; i++) {           // count microseconds
     TR0 = 0;                          // STOP Timer0
      TH0 = (-SYSCLK/10000) >> 8;     // set Timer0 to overflow in 100us
      TL0 = -SYSCLK/10000;
      TR0 = 1;                          // START Timer0
      while (TF0 == 0);                 // wait for overflow
      TF0 = 0;                          // clear overflow indicator
   }
}

//-----------------------------------------------------------------------------
// Timer0_10us
//-----------------------------------------------------------------------------
//
// Configure Timer0 to delay <us>*10 microseconds before returning.
```

```
//
void Timer0_10us (unsigned us)
{
   unsigned i;                          // millisecond counter

   TCON  &= ~0x30;                      // STOP Timer0 and clear overflow flag
   TMOD  &= ~0x0f;                      // configure Timer0 to 16-bit mode
   TMOD  |=  0x01;
   CKCON |=  0x04;                      // Timer0 counts SYSCLKs

   for (i = 0; i < us; i++) {           // count microseconds
     TR0 = 0;                          // STOP Timer0
      TH0 = (-SYSCLK/100000) >> 8;    // set Timer0 to overflow in 100us
      TL0 = -SYSCLK/100000;
      TR0 = 1;                          // START Timer0
      while (TF0 == 0);                 // wait for overflow
      TF0 = 0;                          // clear overflow indicator
   }
}

//-----------------------------------------------------------------------------
// Timer0_1us
//-----------------------------------------------------------------------------
//
// Configure Timer0 to delay <us> microseconds before returning.
//
void Timer0_1us (unsigned us)
{
   unsigned i,of;
    of = us;
   for (i = 0; i < of; i++) {           // count microseconds
    _nop_();
   }
}
```



```
//---------------------------------------------------------------------------
// FILE: cy delays.h
// DESCRIPTION: Header file for cy_delays.c containing basic delay functions
//                  necessary for the MDQ timing
// AUTHOR: Mat Laibowitz
//---------------------------------------------------------------------------
#ifndef CY_DELAYS_H
#define CY_DELAYS_H
#include "para_defs.h"

void Timer0_ms (unsigned ms);
void Timer0_100us (unsigned us);
void Timer0_10us (unsigned us);
void Timer0_1us (unsigned us);


#endif
```

```c
//---------------------------------------------------------------------------------
// FILE: cy hdq.c
// DESCRIPTION: Contains functions for communicating over the HDQ interface
// AUTHOR: Mat Laibowitz
//---------------------------------------------------------------------------------
#include "cy_hdq.h"

unsigned char HDQwritebyte(unsigned char command, unsigned char in_byte)
{
    unsigned char bits;
    command |= 0x80;
    HDQBreak();
    for(bits=0;bits<7;bits++){
        HDQbitout(command&0x01,1);
        command = command >> 1;
    }
    HDQbitout(command&0x01,0);
    Timer0_100us(1);
    Timer0_10us(9);
    for(bits=0;bits<7;bits++){
        HDQbitout(in_byte&0x01,1);
        in_byte = in_byte >> 1;
    }
    HDQbitout(in_byte&0x01,0);
    return 1;
}

unsigned char HDQreadbyte(unsigned char command, unsigned char *out_byte)
{
    unsigned char bits, result;
    signed char ret_val;
    command &= ~(0x80);
    HDQBreak();
    result = 0;
    for(bits=0;bits<7;bits++){
        HDQbitout(command&0x01,1);
        command = command >> 1;
    }
    HDQbitout(command&0x01,0);
    for(bits=0;bits<7;bits++){
        result = result >> 1;
        ret_val = HDQbitin();
        if (ret_val==1) {
            result |= 0x80;
        } else if (ret_val == 0) {
            result &= ~(0x80);
        } else {
            return 0;
        }
    }

        result = result >> 1;
        ret_val = HDQbitin();
        if (ret_val==1) {
            result |= 0x80;
        } else if (ret_val == 0) {
            result &= ~(0x80);
        } else {
            return 0;
        }
    *out_byte = result;
    return 1;
}

void HDQBreak(void)
```

```c
{
    setHDQ(0);                              // Drives DQ low
    Timer0_100us(2);
    Timer0_10us(2);
    setHDQ(1);                              // Releases the bus
    Timer0_10us(6);
}

unsigned char HDQbitout(unsigned char value, unsigned char wait)
{
    unsigned char i;
    setHDQ(0);
    Timer0_10us(1);
    for(i=0;i<4;i++) _nop_();
    if (value==1) {
        setHDQ(1);
    }
    Timer0_10us(11);
    setHDQ(1);
    if (wait == 1) Timer0_100us(2);
    return 1;
}

signed char HDQbitin(void)
{
    unsigned char result;
    unsigned int ticker=0;
    setHDQ(1);
    while(getHDQ()) {
        if(ticker++ == BITINTIMEOUT) return -1;
    }
    Timer0_10us(1);
    if(getHDQ() == 1) {
        return -1;
    }
    Timer0_10us(5);
    result = getHDQ();
    Timer0_100us(1);
    Timer0_10us(5);
    return result;
}

void setHDQ(unsigned char value)
{
    if (value==0) {
        // Clear HDQ
        HDQ = 0;
        P2MDOUT |= HDQMASK;
    } else {
        HDQ = 1;
        P2MDOUT &= ~(HDQMASK);
    }
}

unsigned char getHDQ(void)
{
    return (HDQ);
}
```

Appendix C

Software Simulator Code

```
'FILE: parasim_main.vb
'CLASS: parasimMain
'AUTHOR: Mat Laibowitz
'DESCRIPTION: Main file for the Parastic Mobility Map Editor

Public Class parasimMain
    Inherits System.Windows.Forms.Form
    Public Shared myMap As psMap
    Dim myPBArray As PBArray
    Dim myTXPBArray As PBArray
    Dim curTile As Integer
    Dim curTxt As Integer
    Dim tmpX, tmpY As Integer
    Dim running As Boolean
    Dim BHTable As DataTable
    Dim behaveForm As parasim_behaviors
    Dim tile_bitmaps(6) As Bitmap
    Dim wall_bitmapN As Bitmap
    Dim wall_bitmapS As Bitmap
    Dim wall_bitmapW As Bitmap
    Dim wall_bitmapE As Bitmap
    Dim paramor_bitmap As Bitmap
    Dim host bitmap As Bitmap
    Dim portal_bitmap As Bitmap
    Dim fileName As String = ""
    Dim XDefault As Integer = 20
    Dim YDefault As Integer = 10



    '/ <summary>
    ' Main entry point of the application.
    '/ </summary>
    Public Shared Sub Main()
        Dim m As New parasimMain
        Application.Exit()
    End Sub


    Public Sub New()
        MyBase.New()
        running = False
        'This call is required by the Windows Form Designer.
        InitializeComponent()

        'Add any initialization after the InitializeComponent() call

        'Load tiles from file TODO
        tile_bitmaps(0) = Image.FromFile("tex_tile1.bmp")
        tile_bitmaps(1) = Image.FromFile("tex_tile2.bmp")
        tile_bitmaps(2) = Image.FromFile("tex_tile3.bmp")
        tile_bitmaps(3) = Image.FromFile("tex_tile4.bmp")
        tile_bitmaps(4) = Image.FromFile("tex_tile5.bmp")
        tile_bitmaps(5) = Image.FromFile("tex_tile6.bmp")
        wall_bitmapN = Image.FromFile("wall2.bmp")
        wall_bitmapE = Image.FromFile("wall2.bmp")
        wall_bitmapS = Image.FromFile("wall2.bmp")
        wall_bitmapW = Image.FromFile("wall2.bmp")
        wall_bitmapN.RotateFlip(RotateFlipType.Rotate270FlipNone)
        wall_bitmapS.RotateFlip(RotateFlipType.Rotate90FlipNone)
        wall_bitmapW.RotateFlip(RotateFlipType.Rotate180FlipNone)
        wall_bitmapN.MakeTransparent(Color.FromArgb(255, 0, 255))
        wall_bitmapS.MakeTransparent(Color.FromArgb(255, 0, 255))
        wall_bitmapE.MakeTransparent(Color.FromArgb(255, 0, 255))
        wall_bitmapW.MakeTransparent(Color.FromArgb(255, 0, 255))
```

```
        paramor_bitmap = Image.FromFile("paramor.bmp")
        host bitmap = Image.FromFile("host.bmp")
        portal_bitmap = Image.FromFile("portal.bmp")
        paramor_bitmap.MakeTransparent(Color.FromArgb(255, 0, 255))
        host_bitmap.MakeTransparent(Color.FromArgb(255, 0, 255))
        portal_bitmap.MakeTransparent(Color.FromArgb(255, 0, 255))


        myTXPBArray = New PBArray(pnlTN)
        myTXPBArray.AddNewPB(8, 8, 60, 60)
        myTXPBArray(0).Image = ResizeBitmap(tile_bitmaps(0), 60, 60)
        myTXPBArray.AddNewPB(83, 8, 60, 60)
        myTXPBArray(1).Image = ResizeBitmap(tile_bitmaps(1), 60, 60)
        myTXPBArray.AddNewPB(158, 8, 60, 60)
        myTXPBArray(2).Image = ResizeBitmap(tile_bitmaps(2), 60, 60)
        myTXPBArray.AddNewPB(8, 82, 60, 60)
        myTXPBArray(3).Image = ResizeBitmap(tile_bitmaps(3), 60, 60)
        myTXPBArray.AddNewPB(83, 82, 60, 60)
        myTXPBArray(4).Image = ResizeBitmap(tile_bitmaps(4), 60, 60)
        myTXPBArray.AddNewPB(158, 82, 60, 60)
        myTXPBArray(5).Image = ResizeBitmap(tile_bitmaps(5), 60, 60)
        curTxt = -1
        curTile = -1

        myMap = New psMap(XDefault, YDefault, pnlMap.Width / XDefault, pnlMap.Width ↙
 / XDefault)
        myPBArray = New PBArray(pnlMap)
        DrawMap(myMap, pnlMap)
        running = True
        Show()
        StartLoop()

    End Sub

    Private Sub DrawMap(ByRef mapToDraw As psMap, ByVal target As Control)
        Dim i, j As Integer
        myPBArray.ClearPBArray()
        myPBArray = New PBArray(pnlMap)
        For j = 0 To myMap.Y - 1
            For i = 0 To myMap.X - 1
                myPBArray.AddNewPB(myMap.paraMap(i, j).XPos, myMap.paraMap(i, j).  ↙
YPos, myMap.paraMap(i, j).XSize, myMap.paraMap(i, j).YSize)
                DrawTile(i, j)
            Next
        Next
    End Sub

    Private Sub StartLoop()

        Do While (Created)
            If curTile <> myPBArray.Selected Then
                curTile = myPBArray.Selected
                If curTile > -1 Then
                    Dim tmpX As Integer = myMap.GetIndexDouble(curTile).X
                    Dim tmpY As Integer = myMap.GetIndexDouble(curTile).Y
                    If curTile = -1 Then
                        lblSelected.Text = "( - , - )"
                    Else
                        lblSelected.Text = "( " & tmpX & " , " & tmpY & " )"
                    End If
                    numHosts.Value = myMap.paraMap(tmpX, tmpY).Hosts
                    If numHosts.Value > 0 Then
                        cmdHAsn.Enabled = True
                    Else
                        cmdHAsn.Enabled = False
                End If
                numParamors.Value = myMap.paraMap(tmpX, tmpY).Paramors
                If numParamors.Value > 0 Then
                    cmdPAsn.Enabled = True
                Else
                    cmdPAsn.Enabled = False
                End If
                myTXPBArray.SelectPB(myMap.paraMap(tmpX, tmpY).textureNum)
                curTxt = myMap.paraMap(tmpX, tmpY).textureNum
                cbWallN.Checked = myMap.paraMap(tmpX, tmpY).WallN
                cbWallS.Checked = myMap.paraMap(tmpX, tmpY).WallS
                cbWallE.Checked = myMap.paraMap(tmpX, tmpY).WallE
                cbWallW.Checked = myMap.paraMap(tmpX, tmpY).WallW
                cbPortal.Checked = myMap.paraMap(tmpX, tmpY).Portal
                numPower.Value = myMap.paraMap(tmpX, tmpY).Power
                numTemp.Value = myMap.paraMap(tmpX, tmpY).Temperature
                numLight.Value = myMap.paraMap(tmpX, tmpY).Light
                numAlt.Value = myMap.paraMap(tmpX, tmpY).Altitude
                numVib.Value = myMap.paraMap(tmpX, tmpY).Vibration
                numRad.Value = myMap.paraMap(tmpX, tmpY).Radiation
                numHTraffic.Value = myMap.paraMap(tmpX, tmpY).HostTraffic
            End If
        End If
        If curTxt <> myTXPBArray.Selected Then
            curTxt = myTXPBArray.Selected
            If curTxt > -1 Then
                myMap.paraMap(myMap.GetIndexDouble(curTile).X, myMap.      ↙
GetIndexDouble(curTile).Y).textureNum = curTxt
                DrawTile(myMap.GetIndexDouble(curTile).X, myMap.GetIndexDouble( ↙
curTile).Y)
            End If
        End If
        Application.DoEvents()
    Loop
End Sub

'BEGIN HIDDEN CODE - HIDE GUI CODE FROM LISTING
Windows Form Designer generated code
'END HIDDEN CODE SECTION

Private Sub MenuItem6_Click(ByVal sender As System.Object, ByVal e As System.   ↙
EventArgs) Handles MenuItem6.Click
    Application.Exit()
End Sub


Private Sub cmdResize_Click(ByVal sender As System.Object, ByVal e As System.   ↙
EventArgs) Handles cmdResize.Click
    If IsNumeric(txtMapW.Text) And IsNumeric(txtMapH.Text) Then
        myMap.changeMapSize(txtMapW.Text, txtMapH.Text)
        DrawMap(myMap, pnlMap)
    End If
End Sub

Private Sub cmdNewMap_Click(ByVal sender As System.Object, ByVal e As System.   ↙
EventArgs) Handles cmdNewMap.Click
    If IsNumeric(txtMapW.Text) And IsNumeric(txtMapH.Text) Then
        myMap = New psMap(txtMapW.Text, txtMapH.Text, pnlMap.Width / txtMapW.   ↙
Text, pnlMap.Width / txtMapW.Text)
        DrawMap(myMap, pnlMap)
        Me.MenuItem4.Enabled = False
        Me.Text = "Parasitic Mobility Simulator Map Editor"
    End If
End Sub
```

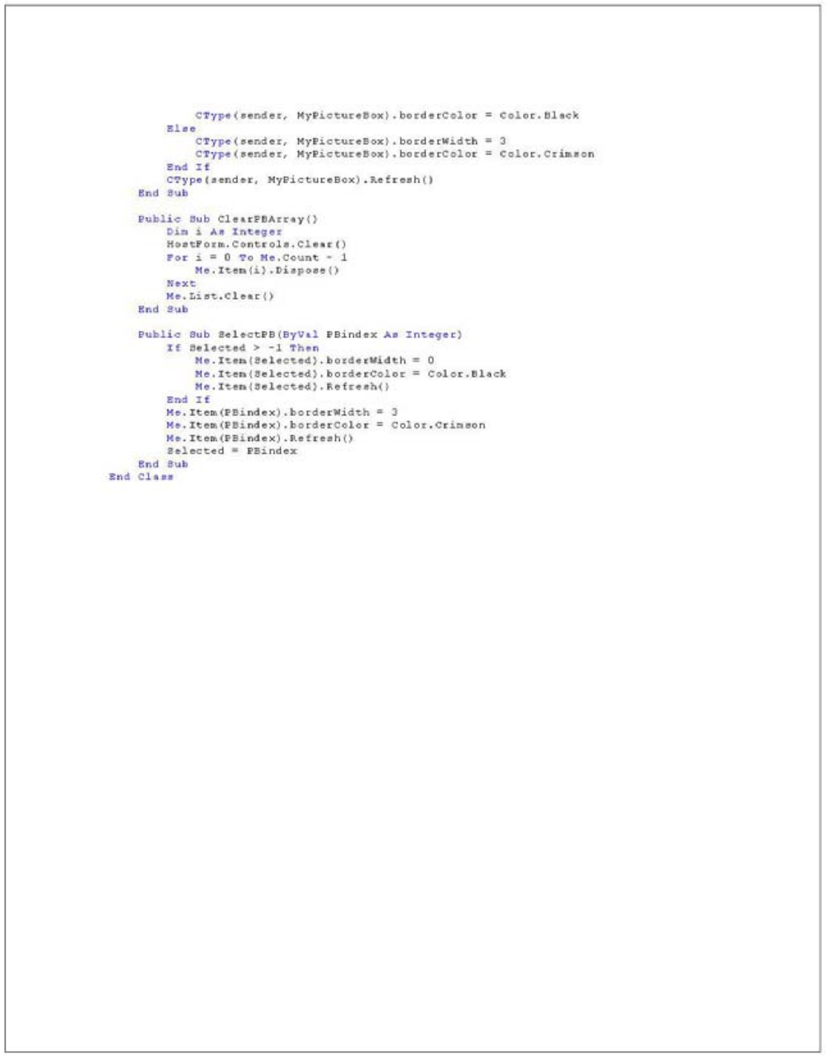
```
Private Sub cmdZoom_Click(ByVal sender As System.Object, ByVal e As System.     ↙
EventArgs) Handles cmdZoom.Click
    If IsNumeric(txtZoom.Text) Then
        myMap.zoomMap(txtZoom.Text)
        DrawMap(myMap, pnlMap)
    End If
End Sub


Private Sub numParamors_ValueChanged(ByVal sender As System.Object, ByVal e As  ↙
System.EventArgs) Handles numParamors.Click
    If running Then
        If curTile > -1 Then
            If numParamors.Value > 0 Then
                cmdPAsn.Enabled = True
            Else
                cmdPAsn.Enabled = False
            End If
            tmpX = myMap.GetIndexDouble(curTile).X
            tmpY = myMap.GetIndexDouble(curTile).Y
            If numParamors.Value > 0 Then
                ReDim Preserve myMap.paraMap(tmpX, tmpY).ParamorBehaviors(      ↙
numParamors.Value)
                For i As Integer = 0 To numParamors.Value - 1
                    If myMap.paraMap(tmpX, tmpY).ParamorBehaviors(i) = "" Then
                        myMap.paraMap(tmpX, tmpY).ParamorBehaviors(i) = "       ↙
Behavior 1"
                    End If
                Next
            Else
                ReDim myMap.paraMap(tmpX, tmpY).ParamorBehaviors(1)
            End If
            myMap.paraMap(tmpX, tmpY).Paramors = numParamors.Value
            DrawTile(tmpX, tmpY)
        End If
    End If
End Sub

Private Sub numHosts_ValueChanged(ByVal sender As System.Object, ByVal e As     ↙
System.EventArgs) Handles numHosts.Click
    If running Then
        If curTile > -1 Then
            If numHosts.Value > 0 Then
                cmdHAsn.Enabled = True
            Else
                cmdHAsn.Enabled = False
            End If
            tmpX = myMap.GetIndexDouble(curTile).X
            tmpY = myMap.GetIndexDouble(curTile).Y
            If numHosts.Value > 0 Then
                ReDim Preserve myMap.paraMap(tmpX, tmpY).HostBehaviors(numHosts ↙
.Value)
                For i As Integer = 0 To numHosts.Value - 1
                    If myMap.paraMap(tmpX, tmpY).HostBehaviors(i) = "" Then
                        myMap.paraMap(tmpX, tmpY).HostBehaviors(i) = "Behavior ↙
1"
                    End If
                Next
            Else
                ReDim myMap.paraMap(tmpX, tmpY).HostBehaviors(1)
            End If
            myMap.paraMap(tmpX, tmpY).Hosts = numHosts.Value
            DrawTile(tmpX, tmpY)
        End If
    End If
End Sub


Private Sub cbWallN_CheckedChanged(ByVal sender As System.Object, ByVal e As    ↙
System.EventArgs) Handles cbWallN.Click, cbWallE.Click, cbWallS.Click, cbWallW. ↙
Click, cbPortal.Click
    If running Then
        If curTile > -1 Then
            Dim tmpX As Integer = myMap.GetIndexDouble(curTile).X
            Dim tmpY As Integer = myMap.GetIndexDouble(curTile).Y
            myMap.paraMap(tmpX, tmpY).WallN = cbWallN.Checked
            myMap.paraMap(tmpX, tmpY).WallE = cbWallE.Checked
            myMap.paraMap(tmpX, tmpY).WallW = cbWallW.Checked
            myMap.paraMap(tmpX, tmpY).WallS = cbWallS.Checked
            myMap.paraMap(tmpX, tmpY).Portal = cbPortal.Checked
            DrawTile(tmpX, tmpY)
        End If
    End If
End Sub

Private Sub cmdPAsn_Click(ByVal sender As System.Object, ByVal e As System.     ↙
EventArgs) Handles cmdPAsn.Click
    numParamors ValueChanged(Nothing, Nothing)
    behaveForm = New parasim_behaviors(myMap.GetIndexDouble(curTile).X, myMap.  ↙
GetIndexDouble(curTile).Y, numParamors.Value, True)
    behaveForm.ShowDialog()
End Sub

Private Sub cmdHAsn_Click(ByVal sender As System.Object, ByVal e As System.     ↙
EventArgs) Handles cmdHAsn.Click
    numHosts_ValueChanged(Nothing, Nothing)
    behaveForm = New parasim_behaviors(myMap.GetIndexDouble(curTile).X, myMap.  ↙
GetIndexDouble(curTile).Y, numHosts.Value, False)
    behaveForm.ShowDialog()
End Sub

Private Function ResizeBitmap(ByVal imageBm As Bitmap, ByVal XSize As Integer,  ↙
ByVal YSize As Integer) As Bitmap
    Dim bm_tile As Bitmap
    Dim bm_graphics As Graphics
    Dim bm_rect As Rectangle
    bm_rect = New Rectangle(0, 0, XSize, YSize)
    bm_tile = New Bitmap(XSize, YSize)
    bm_graphics = Graphics.FromImage(bm_tile)
    bm_graphics.DrawImage(imageBm, bm_rect)
    Return bm_tile
End Function

Private Sub DrawTile(ByVal Xadd As Integer, ByVal Yadd As Integer)
    Dim bm_tile As Bitmap
    Dim bm_rect As Rectangle
    Dim bm_graphics As Graphics
    bm_rect = New Rectangle(0, 0, myMap.paraMap(Xadd, Yadd).XSize, myMap.       ↙
paraMap(Xadd, Yadd).YSize)
    bm_tile = New Bitmap(myMap.paraMap(Xadd, Yadd).XSize, myMap.paraMap(Xadd,   ↙
Yadd).YSize)
    bm_graphics = Graphics.FromImage(bm_tile)
    bm_graphics.DrawImage(tile_bitmaps(myMap.paraMap(Xadd, Yadd).textureNum),   ↙
bm_rect)
    If myMap.paraMap(Xadd, Yadd).WallE = True Then
        bm_graphics.DrawImage(wall_bitmapE, bm_rect)
    End If
    If myMap.paraMap(Xadd, Yadd).WallN = True Then
        bm_graphics.DrawImage(wall_bitmapN, bm_rect)
    End If
    If myMap.paraMap(Xadd, Yadd).WallS = True Then
        bm_graphics.DrawImage(wall_bitmapS, bm_rect)
    End If
    If myMap.paraMap(Xadd, Yadd).WallW = True Then
        bm_graphics.DrawImage(wall_bitmapW, bm_rect)
    End If
    If myMap.paraMap(Xadd, Yadd).Paramors > 0 Then
        bm_graphics.DrawImage(paramor_bitmap, bm_rect)
    End If
    If myMap.paraMap(Xadd, Yadd).Hosts > 0 Then
        bm_graphics.DrawImage(host_bitmap, bm_rect)
    End If
    If myMap.paraMap(Xadd, Yadd).Portal = True Then
        bm_graphics.DrawImage(portal_bitmap, bm_rect)
    End If
    myPBArray(myMap.GetIndexSingle(Xadd, Yadd)).Image = bm_tile
End Sub


Private Sub numPower_ValueChanged(ByVal sender As System.Object, ByVal e As     ↙
System.EventArgs) Handles numPower.Click
    If running Then
        If curTile > -1 Then
            Dim tmpX As Integer = myMap.GetIndexDouble(curTile).X
            Dim tmpY As Integer = myMap.GetIndexDouble(curTile).Y
            myMap.paraMap(tmpX, tmpY).Power = numPower.Value
        End If
    End If
End Sub

Private Sub cmdCancel_Click(ByVal sender As System.Object, ByVal e As System.   ↙
EventArgs) Handles cmdCancel.Click
    If running Then
        If curTile > -1 Then
            Dim tmpX As Integer = myMap.GetIndexDouble(curTile).X
            Dim tmpY As Integer = myMap.GetIndexDouble(curTile).Y
            numPower.Value = myMap.paraMap(tmpX, tmpY).Power
            numTemp.Value = myMap.paraMap(tmpX, tmpY).Temperature
            numLight.Value = myMap.paraMap(tmpX, tmpY).Light
            numAlt.Value = myMap.paraMap(tmpX, tmpY).Altitude
            numVib.Value = myMap.paraMap(tmpX, tmpY).Vibration
            numRad.Value = myMap.paraMap(tmpX, tmpY).Radiation
        End If
    End If
End Sub

Private Sub cmdUpdate_Click(ByVal sender As System.Object, ByVal e As System.   ↙
EventArgs) Handles cmdUpdate.Click
    If running Then
        If curTile > -1 Then
            Dim tmpX As Integer = myMap.GetIndexDouble(curTile).X
            Dim tmpY As Integer = myMap.GetIndexDouble(curTile).Y
            myMap.paraMap(tmpX, tmpY).Power = numPower.Value
            myMap.paraMap(tmpX, tmpY).Temperature = numTemp.Value
            myMap.paraMap(tmpX, tmpY).Light = numLight.Value
            myMap.paraMap(tmpX, tmpY).Altitude = numAlt.Value
            myMap.paraMap(tmpX, tmpY).Vibration = numVib.Value
            myMap.paraMap(tmpX, tmpY).Radiation = numRad.Value
        End If
    End If
End Sub

Private Sub MenuItem2_Click(ByVal sender As System.Object, ByVal e As System.   ↙
EventArgs) Handles MenuItem2.Click
    cmdNewMap_Click(Nothing, Nothing)
    End Sub

    Private Sub MenuItem3_Click(ByVal sender As System.Object, ByVal e As System.   ↙
    EventArgs) Handles MenuItem3.Click
        Me.OpenFileDialog1.ShowDialog()
        If OpenFileDialog1.FileName <> "" Then
            fileName = OpenFileDialog1.FileName
            Me.MenuItem4.Enabled = True
            Me.Text = "Parasitic Mobility Simulator Map Editor - " & fileName
            myMap = New psMap(fileName, pnlMap.Width / XDefault, pnlMap.Width /     ↙
    XDefault)
            DrawMap(myMap, pnlMap)
        End If
    End Sub

    Private Sub MenuItem4 Click(ByVal sender As System.Object, ByVal e As System.   ↙
    EventArgs) Handles MenuItem4.Click
        myMap.writeMapToFile(fileName)
    End Sub

    Private Sub MenuItem5_Click(ByVal sender As System.Object, ByVal e As System.   ↙
    EventArgs) Handles MenuItem5.Click
        SaveFileDialog1.ShowDialog()
        If SaveFileDialog1.FileName <> "" Then
            fileName = SaveFileDialog1.FileName
            Me.MenuItem4.Enabled = True
            Me.Text = "Parasitic Mobility Simulator Map Editor - " & fileName
            myMap.writeMapToFile(fileName)
        End If
    End Sub

    Private Sub numHTraffic_ValueChanged(ByVal sender As System.Object, ByVal e As ↙
    System.EventArgs) Handles numHTraffic.Click
        If running Then
            If curTile > -1 Then
                tmpX = myMap.GetIndexDouble(curTile).X
                tmpY = myMap.GetIndexDouble(curTile).Y
                myMap.paraMap(tmpX, tmpY).HostTraffic = numHTraffic.Value
            End If
        End If
    End Sub
End Class
```

```vb
'FILE: psMap.vb
'CLASS: psMap
'DESCRIPTION: Class for generating and control a tile based map
'CLASS: PBArray
'DESCRIPTION: Class to create a multi-instance array of picturebox controls
'CLASS: MyPictureBox
'DESCRIPTION: Class to add rollover effects to a picturebox control
'AUTHOR: Mat Laibowitz

Public Class psMap

    Public Structure paraTile
        Dim XSize As Integer
        Dim YSize As Integer
        Dim XPos As Integer
        Dim YPos As Integer
        Dim WallN As Boolean
        Dim WallE As Boolean
        Dim WallS As Boolean
        Dim WallW As Boolean
        Dim Portal As Boolean
        Dim Paramors As Integer
        Dim ParamorBehaviors() As String
        Dim Hosts As Integer
        Dim HostBehaviors() As String
        Dim Power As Integer
        Dim Temperature As Integer
        Dim Light As Integer
        Dim Altitude As Integer
        Dim Vibration As Integer
        Dim Radiation As Integer
        Dim textureNum As Integer
        Dim HostTraffic As Integer
    End Structure

    Public paraMap(,) As paraTile
    Public X As Integer
    Public Y As Integer

    Public Sub New(ByVal XDim As Integer, ByVal YDim As Integer, ByVal XWidth As ↙
    Integer, ByVal YHeight As Integer)
        Dim i, j As Integer
        X = XDim
        Y = YDim
        ReDim paraMap(X - 1, Y - 1)
        For j = 0 To YDim - 1
            For i = 0 To XDim - 1
                paraMap(i, j).Hosts = 0
                paraMap(i, j).Paramors = 0
                paraMap(i, j).Portal = False
                paraMap(i, j).textureNum = 0
                paraMap(i, j).WallE = False
                paraMap(i, j).WallN = False
                paraMap(i, j).WallW = False
                paraMap(i, j).WallE = False
                paraMap(i, j).WallS = False
                If i = 0 Then paraMap(i, j).WallW = True
                If j = 0 Then paraMap(i, j).WallN = True
                If i = X - 1 Then paraMap(i, j).WallE = True
                If j = Y - 1 Then paraMap(i, j).WallS = True
                paraMap(i, j).Power = 0
                paraMap(i, j).Temperature = 0
                paraMap(i, j).Altitude = 0
                paraMap(i, j).Light = 0
                paraMap(i, j).Radiation = 0
                paraMap(i, j).Vibration = 0
                paraMap(i, j).HostTraffic = 100
                paraMap(i, j).XSize = XWidth
                paraMap(i, j).YSize = YHeight
                paraMap(i, j).XPos = i * XWidth
                paraMap(i, j).YPos = j * YHeight
            Next
        Next
End Sub

Public Sub New(ByVal fileName As String, ByVal XWidth As Integer, ByVal YHeight ↙
 As Integer)
    Dim i, j, k As Integer
    Dim m_nodelist As XmlNodeList
    Dim m_node As XmlNode
    Dim m_xmld As XmlDocument
    m_xmld = New XmlDocument
    'Load the Xml file
    m_xmld.Load(fileName)
    m_node = m_xmld.SelectSingleNode("/Map")
    X = m_node.Attributes.GetNamedItem("width").Value
    Y = m_node.Attributes.GetNamedItem("height").Value
    ReDim paraMap(X - 1, Y - 1)
    'Get the list of name nodes
    m_nodelist = m_xmld.SelectNodes("/Map/Tile")
    'Loop through the nodes
    For Each m_node In m_nodelist
        i = m_node.Attributes.GetNamedItem("X").Value
        j = m_node.Attributes.GetNamedItem("Y").Value
        paraMap(i, j).WallN = m_node.ChildNodes.Item(0).InnerText
        paraMap(i, j).WallE = m_node.ChildNodes.Item(1).InnerText
        paraMap(i, j).WallS = m_node.ChildNodes.Item(2).InnerText
        paraMap(i, j).WallW = m_node.ChildNodes.Item(3).InnerText
        paraMap(i, j).Portal = m_node.ChildNodes.Item(4).InnerText
        paraMap(i, j).Paramors = m_node.ChildNodes.Item(5).Attributes.          ↙
GetNamedItem("num").Value
        ReDim paraMap(i, j).ParamorBehaviors(paraMap(i, j).Paramors)
        For k = 0 To paraMap(i, j).Paramors - 1
            paraMap(i, j).ParamorBehaviors(k) = m_node.ChildNodes.Item(5).      ↙
ChildNodes.Item(k).InnerText
        Next
        paraMap(i, j).Hosts = m_node.ChildNodes.Item(6).Attributes.GetNamedItem ↙
("num").Value
        ReDim paraMap(i, j).HostBehaviors(paraMap(i, j).Hosts)
        For k = 0 To paraMap(i, j).Hosts - 1
            paraMap(i, j).HostBehaviors(k) = m_node.ChildNodes.Item(6).         ↙
ChildNodes.Item(k).InnerText
        Next
        paraMap(i, j).Power = m_node.ChildNodes.Item(7).InnerText
        paraMap(i, j).Temperature = m_node.ChildNodes.Item(8).InnerText
        paraMap(i, j).Light = m_node.ChildNodes.Item(9).InnerText
        paraMap(i, j).Altitude = m_node.ChildNodes.Item(10).InnerText
        paraMap(i, j).Vibration = m_node.ChildNodes.Item(11).InnerText
        paraMap(i, j).Radiation = m_node.ChildNodes.Item(12).InnerText
        paraMap(i, j).textureNum = m_node.ChildNodes.Item(13).InnerText
        paraMap(i, j).HostTraffic = m_node.ChildNodes.Item(14).InnerText
        paraMap(i, j).XSize = XWidth
        paraMap(i, j).YSize = YHeight
        paraMap(i, j).XPos = i * XWidth
        paraMap(i, j).YPos = j * YHeight
    Next

End Sub

Public Function writeMapToFile(ByVal fileName As String)
```

```
Dim textWriter As XmlTextWriter = New XmlTextWriter(fileName, Nothing)
Dim i, j, k As Integer
textWriter.Formatting = Formatting.Indented
' Opens the document
textWriter.WriteStartDocument()
' Write comments
textWriter.WriteComment("ParaSim Map File")
Dim str As String = "Filename: " & fileName
textWriter.WriteComment(str)
str = "Created: " & System.DateTime.Now
textWriter.WriteComment(str)
textWriter.WriteStartElement("Map")
textWriter.WriteAttributeString("width", X)
textWriter.WriteAttributeString("height", Y)

For j = 0 To Y - 1
    For i = 0 To X - 1
        textWriter.WriteStartElement("Tile")
        textWriter.WriteAttributeString("X", i)
        textWriter.WriteAttributeString("Y", j)
        textWriter.WriteStartElement("WallN")
        textWriter.WriteString(paraMap(i, j).WallN)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("WallE")
        textWriter.WriteString(paraMap(i, j).WallE)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("WallS")
        textWriter.WriteString(paraMap(i, j).WallS)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("WallW")
        textWriter.WriteString(paraMap(i, j).WallW)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("Portal")
        textWriter.WriteString(paraMap(i, j).Portal)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("Paramors")
        textWriter.WriteAttributeString("num", paraMap(i, j).Paramors)
        For k = 0 To paraMap(i, j).Paramors - 1
            textWriter.WriteStartElement("B" & k)
            textWriter.WriteString(paraMap(i, j).ParamorBehaviors(k))
            textWriter.WriteEndElement()
        Next
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("Hosts")
        textWriter.WriteAttributeString("num", paraMap(i, j).Hosts)
        For k = 0 To paraMap(i, j).Hosts - 1
            textWriter.WriteStartElement("B" & k)
            textWriter.WriteString(paraMap(i, j).HostBehaviors(k))
            textWriter.WriteEndElement()
        Next
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("Power")
        textWriter.WriteString(paraMap(i, j).Power)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("Temperature")
        textWriter.WriteString(paraMap(i, j).Temperature)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("Light")
        textWriter.WriteString(paraMap(i, j).Light)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("Altitude")
        textWriter.WriteString(paraMap(i, j).Altitude)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("Vibration")
        textWriter.WriteString(paraMap(i, j).Vibration)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("Radiation")
        textWriter.WriteString(paraMap(i, j).Radiation)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("Texture")
        textWriter.WriteString(paraMap(i, j).textureNum)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("HostTraffic")
        textWriter.WriteString(paraMap(i, j).HostTraffic)
        textWriter.WriteEndElement()
        textWriter.WriteEndElement()
    Next
Next
textWriter.WriteEndElement()

' Ends the document.
textWriter.WriteEndDocument()
' close writer
textWriter.Close()

End Function

Public Function changeMapSize(ByVal XDim As Integer, ByVal YDim As Integer)
    Dim i, j As Integer
    Dim XOrig As Integer = X
    Dim YOrig As Integer = Y
    Dim paraTemp(,) As paraTile = paraMap
    ReDim paraMap(XDim - 1, YDim - 1)
    For j = 0 To YDim - 1
        For i = 0 To XDim - 1
            If i < XOrig - 1 And j < YOrig - 1 Then
                paraMap(i, j) = paraTemp(i, j)
            ElseIf i = XDim - 1 And j < YOrig Then
                paraMap(i, j) = paraTemp(XOrig - 1, j)
                paraMap(i, j).WallE = False
                paraMap(i, j).WallN = False
                paraMap(i, j).WallW = False
                paraMap(i, j).WallS = False
            ElseIf i < XOrig And j = YDim - 1 Then
                paraMap(i, j) = paraTemp(i, YOrig - 1)
                paraMap(i, j).WallE = False
                paraMap(i, j).WallN = False
                paraMap(i, j).WallW = False
                paraMap(i, j).WallS = False
            Else
                paraMap(i, j).Hosts = 0
                paraMap(i, j).Paramors = 0
                paraMap(i, j).Portal = False
                paraMap(i, j).textureNum = 0
                paraMap(i, j).WallE = False
                paraMap(i, j).WallN = False
                paraMap(i, j).WallW = False
                paraMap(i, j).WallS = False
                paraMap(i, j).Power = 0
                paraMap(i, j).Temperature = 0
                paraMap(i, j).Altitude = 0
                paraMap(i, j).Light = 0
                paraMap(i, j).Radiation = 0
                paraMap(i, j).Vibration = 0
                paraMap(i, j).HostTraffic = 100
            End If
            If i = 0 Then paraMap(i, j).WallW = True
            If j = 0 Then paraMap(i, j).WallN = True
            If i = XDim - 1 Then paraMap(i, j).WallE = True
            If j = YDim - 1 Then paraMap(i, j).WallS = True
```

```
                paraMap(i, j).XSize = paraTemp(0, 0).XSize
                paraMap(i, j).YSize = paraTemp(0, 0).YSize
                paraMap(i, j).XPos = i * paraTemp(0, 0).XSize
                paraMap(i, j).YPos = j * paraTemp(0, 0).YSize
            Next
        Next
        X = XDim
        Y = YDim
    End Function

    Public Function zoomMap(ByVal amount As Integer)
        Dim i, j As Integer
        For j = 0 To Y - 1
            For i = 0 To X - 1
                paraMap(i, j).XSize = paraMap(i, j).XSize + amount
                paraMap(i, j).YSize = paraMap(i, j).YSize + amount
                paraMap(i, j).XPos = paraMap(i, j).XPos + (i * amount)
                paraMap(i, j).YPos = paraMap(i, j).YPos + (j * amount)
            Next
        Next
    End Function

    Public Function GetIndexSingle(ByVal Xindex As Integer, ByVal Yindex As Integer ↙
    ) As Integer
        Return Xindex + (Yindex * X)
    End Function

    Public Function GetIndexDouble(ByVal index As Integer) As Point
        Dim doubleIndex As Point
        doubleIndex.X = index Mod X
        doubleIndex.Y = index \ X
        Return doubleIndex
    End Function

    Public Function GetSize() As Integer
        Return X * Y
    End Function
End Class

Public Class MyPictureBox

    Inherits PictureBox
    Public borderWidth As Integer = 1
    Public borderColor As Color = Color.Blue

    Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)

        MyBase.OnPaint(e)

        ControlPaint.DrawBorder(e.Graphics, e.ClipRectangle, borderColor,           ↙
    borderWidth, ButtonBorderStyle.Solid, borderColor, borderWidth,                  ↙
    ButtonBorderStyle.Solid, borderColor, borderWidth, ButtonBorderStyle.Solid,      ↙
    borderColor, borderWidth, ButtonBorderStyle.Solid)

    End Sub 'OnPaint

End Class

Public Class PBArray
    Inherits System.Collections.CollectionBase
    Private ReadOnly HostForm As System.Windows.Forms.Panel
    Public Selected As Integer

    Public Sub New(ByVal host As System.Windows.Forms.Panel)
        HostForm = host
```

```
    Selected = -1
End Sub

Default Public ReadOnly Property Item(ByVal Index As Integer) As MyPictureBox
    Get
        Return CType(Me.List.Item(Index), MyPictureBox)
    End Get
End Property

Public Function AddNewPB(ByVal Xpos As Integer, ByVal Ypos As Integer, ByVal     ↙
Xsize As Integer, ByVal YSize As Integer) As MyPictureBox
    Dim aPB As New MyPictureBox
    Me.List.Add(aPB)
    HostForm.Controls.Add(aPB)
    aPB.Location = New System.Drawing.Point(Xpos, Ypos)
    aPB.Name = "PB" & Me.Count.ToString
    aPB.Size = New System.Drawing.Size(Xsize, YSize)
    aPB.TabIndex = Me.Count
    aPB.TabStop = False
    aPB.borderColor = Color.Black
    aPB.borderWidth = 0
    '         aPB.Image = Image.FromFile("c:\animate.bmp").GetThumbnailImage(50, ↙
 50, Nothing, Nothing)
    aPB.Tag = Me.Count - 1
    AddHandler aPB.Click, AddressOf ClickHandler
    AddHandler aPB.MouseEnter, AddressOf MouseOverHandler
    AddHandler aPB.MouseLeave, AddressOf MouseOutHandler
    Return aPB
End Function

Public Sub Remove(ByVal index As Integer)
    If Me.Count > 0 Then
        HostForm.Controls.Remove(Me(index))
        Me.List.RemoveAt(index)
    End If
End Sub

Public Sub Remove()
    If Me.Count > 0 Then
        HostForm.Controls.Remove(Me(Me.Count - 1))
        Me.List.RemoveAt(Me.Count - 1)
    End If
End Sub

Public Sub ClickHandler(ByVal sender As Object, ByVal e As System.EventArgs)
    If Selected > -1 Then
        CType(Me.List.Item(Selected), MyPictureBox).borderWidth = 0
        CType(Me.List.Item(Selected), MyPictureBox).Refresh()
    End If
    CType(sender, MyPictureBox).borderWidth = 3
    CType(sender, MyPictureBox).borderColor = Color.Crimson
    CType(sender, MyPictureBox).Refresh()
    Selected = CType(sender, MyPictureBox).Tag
End Sub

Public Sub MouseOverHandler(ByVal sender As Object, ByVal e As System.EventArgs ↙
)
    CType(sender, MyPictureBox).borderWidth = 3
    CType(sender, MyPictureBox).borderColor = Color.Blue
    CType(sender, MyPictureBox).Refresh()
End Sub

Public Sub MouseOutHandler(ByVal sender As Object, ByVal e As System.EventArgs)
    If CType(sender, MyPictureBox).Tag <> Selected Then
        CType(sender, MyPictureBox).borderWidth = 0
```



```
            CType(sender, MyPictureBox).borderColor = Color.Black
        Else
            CType(sender, MyPictureBox).borderWidth = 3
            CType(sender, MyPictureBox).borderColor = Color.Crimson
        End If
        CType(sender, MyPictureBox).Refresh()
    End Sub

    Public Sub ClearPBArray()
        Dim i As Integer
        HostForm.Controls.Clear()
        For i = 0 To Me.Count - 1
            Me.Item(i).Dispose()
        Next
        Me.List.Clear()
    End Sub

    Public Sub SelectPB(ByVal PBindex As Integer)
        If Selected > -1 Then
            Me.Item(Selected).borderWidth = 0
            Me.Item(Selected).borderColor = Color.Black
            Me.Item(Selected).Refresh()
        End If
        Me.Item(PBindex).borderWidth = 3
        Me.Item(PBindex).borderColor = Color.Crimson
        Me.Item(PBindex).Refresh()
        Selected = PBindex
    End Sub
End Class
```

```
'FILE: frmParasimRun.vb
'CLASS: frmParasimRun
'DESCRIPTION: Main execution code for the Parasitic Mobility Simulator
'               Loads map files from the map editor, sets up behaviors,
'               executes simulations, generates 3D graphics
'AUTHOR: Mat Laibowitz

Imports Microsoft.DirectX
Imports Microsoft.DirectX.Direct3D
Imports Microsoft.DirectX.Direct3D.Geometry
Imports System.Xml
Imports System.Drawing
Imports System.Drawing.Imaging


Public Class frmParasimRun
    Inherits System.Windows.Forms.Form

    Private Const COLLISION_THRESHOLD = 1.0
    Private Const GOAL_THRESHOLD = 0.1
    Private Const PARAMOR_RANGE = 1.5

    Dim pass As Integer = 45
    Dim end_pass As Integer = 45
    Dim multipass As Boolean = False
    Dim running As Boolean
    Dim mapFileName As String
    Dim logFileName As String
    Dim mapLoaded As Boolean
    Dim LoggingActive As Boolean = False
    Dim logWriter As XmlTextWriter

    Private DXGrafx As DX9ToolsR5.DX9Graphics
    Private SphereMesh As Mesh
    Dim wallTex As Texture
    Dim groundTex As Texture
    Private vbCube As VertexBuffer
    Private DeviceLost As Boolean
    Private TxtFont As Direct3D.Font
    Private Cam As DX9ToolsR5.Camera
    Private D3DMngr As Manager

    'misc variables
    Private sDevInfo As String
    Private ToggleInfo As Boolean

    'splash text variable
    Private splashFont As New System.Drawing.Font("Arial", 12.0F, FontStyle.Bold)
    Private mesh3DText As Mesh = Nothing ' Mesh to draw 3d text
    Private splashMat As New Matrix
    Private lastFrameUpdate As Int32
    Private splashAngle As Single
    Private Const splashSpeed As Single = 50.0F

    Protected framePerSecond As Single ' Instanteous frame rate
    Private lastTime As Single = 0.0F ' The last time
    Private frames As Integer = 0 ' Number of rames since our last update
    Private lastMousePos As Point
    Private mouseClickedL As Boolean
    Private mouseClickedR As Boolean

    Dim myParamors As paramors
    Dim myHosts As hosts
    Dim myMap As psMap
```

```
Dim simRunning As Boolean = False
Dim iLastTick As Integer = 0
Dim globalCover(,) As Boolean
Dim globalPercent As Single = 0.0

Dim rando As New System.Random
Dim r_p As New System.Random
Dim videoFileName As String
Dim videoFrameCnt As Integer = 0

Public Shared Sub Main()
    Dim m As New frmParasimRun
    Application.Exit()
End Sub


Public Sub New()
    MyBase.New()
    running = False
    'This call is required by the Windows Form Designer.
    InitializeComponent()
    myHosts = New hosts
    myParamors = New paramors
    mapLoaded = False
    ToggleInfo = True
    Show()
    Focus()
    pbRender.Focus()
    If InitGraphics() Then
        Cam.SetPosition(DX9ToolsR5.SetVector.Z, -30)
        Cam.AspectRatio = CSng(pbRender.ClientSize.Width / pbRender.ClientSize. ↙
Height)
        mesh3DText = Mesh.TextFromFont(DXGrafx.Device, splashFont, "Welcome to ↙
the Parasitic Mobility Simulator", 0.001F, 0.4F)
        SphereMesh = Mesh.Sphere(DXGrafx.Device, 0.5, 5, 5)
        loadTextures(DXGrafx.PresentParameters.BackBufferFormat)
        loadGeometry()
    End If

    AddHandler Me.KeyPress, AddressOf KeyPressed

    mouseClickedL = False
    mouseClickedR = False
    running = True
    DXUtil.Timer(DirectXTimer.Start)
    MainLoop()
End Sub

Private Sub MainLoop()
    Do While (Created)
        Application.DoEvents()
        FrameUpdate()
        FrameRender()
    Loop
End Sub

Private Sub FrameUpdate()
    splashAngle += ((Environment.TickCount() - lastFrameUpdate) / 1000) *       ↙
splashSpeed
    lastFrameUpdate = Environment.TickCount()
    splashMat = Matrix.Identity()
    splashMat = Matrix.Multiply(splashMat, Matrix.Scaling(6, 10, 0.75))
    splashMat = Matrix.Multiply(splashMat, Matrix.Translation(-30, 0, 30))
    splashMat = Matrix.Multiply(splashMat, Matrix.RotationY(splashAngle * (Math ↙
.PI / 180)))
```

```
    'calculate the current framerate
    Dim time As Single = DXUtil.Timer(DirectXTimer.GetAbsoluteTime)
    frames += 1

    ' Update the scene stats once per second
    If time - lastTime > 1.0F Then
        framePerSecond = frames / (time - lastTime)
        lastTime = time
        frames = 0
    End If

    If simRunning = True Then
        Dim i As Integer
        Dim elapsed As Integer = (Environment.TickCount - iLastTick) *          ↙
numTimeScale.Value
        For i = 0 To myHosts.num - 1
            If myHosts.hosts(i).involved = True Then
                If myHosts.hosts(i).moving = True Then
                    Dim collisions As Integer = 5
                    Do While collisions > 0
                        If GetDistance(myHosts.hosts(i).position, myHosts.hosts ↙
(i).goal) < GOAL THRESHOLD Then
                            'HOST AT GOAL
                            Dim Ptile As New Point
                            Dim GTile As New Point
                            Ptile = GetTile(myHosts.hosts(i).position)
                            myHosts.hosts(i).coverage(Ptile.X, Ptile.Y) = True
                            GTile = GetGoal(Ptile, i)
                            myHosts.hosts(i).goal = GetPosition(GTile)
                            If LoggingActive = True And chkLogMTraj.Checked =   ↙
True Then

                                If myHosts.hosts(i).trajLogged = False Then
                                    logWriter.WriteStartElement("HostLocation")
                                    logWriter.WriteAttributeString("Host", i)
                                    logWriter.WriteAttributeString("X", Ptile.X ↙
)

                                    logWriter.WriteAttributeString("Y", Ptile.Y ↙
)

                                    logWriter.WriteString(Environment.TickCount ↙
)

                                    logWriter.WriteEndElement()
                                    myHosts.hosts(i).trajLogged = True
                                End If
                            End If
                        End If
                        Dim displacement As Single = myHosts.hosts(i).speed * ( ↙
elapsed / 1000)
                        Dim pVector As New Vector3
                        Dim orgVector As New Vector3
                        orgVector = myHosts.hosts(i).position
                        pVector.X = myHosts.hosts(i).goal.X - myHosts.hosts(i). ↙
position.X
                        pVector.Y = myHosts.hosts(i).goal.Y - myHosts.hosts(i). ↙
position.Y
                        pVector.Z = myHosts.hosts(i).goal.Z - myHosts.hosts(i). ↙
position.Z
                        myHosts.hosts(i).position.X = myHosts.hosts(i).position ↙
.X + (pVector.X * displacement)
                        myHosts.hosts(i).position.Y = myHosts.hosts(i).position ↙
.Y + (pVector.Y * displacement)
                        myHosts.hosts(i).position.Z = myHosts.hosts(i).position ↙
.Z + (pVector.Z * displacement)
                        Dim h As Integer
                        For h = 0 To myHosts.num - 1
```

```
                            If h <> i Then
                                If GetDistance(myHosts.hosts(i).position,       ↙
myHosts.hosts(h).position) < COLLISION_THRESHOLD Then
                                    myHosts.hosts(i).position = orgVector
                                    myHosts.hosts(i).goal = myHosts.hosts(i).   ↙
position

                                    Exit For
                                End If
                            End If
                            If h = myHosts.num - 1 Then
                                myHosts.hosts(i).trajLogged = False
                                Exit Do
                            End If
                        Next
                        collisions -= 1
                    Loop
                Else
                    myHosts.hosts(i).stayTicks -= elapsed
                    If myHosts.hosts(i).stayTicks <= 0 Then
                        myHosts.hosts(i).moving = True
                    End If
                End If
            Else
                myHosts.hosts(i).portalTicks -= elapsed
                If myHosts.hosts(i).portalTicks <= 0 Then
                    myHosts.hosts(i).involved = True
                    Dim r_pVal As Integer = r_p.Next(myHosts.numPortals)
                    myHosts.hosts(i).goal = GetPosition(myHosts.locPortals(     ↙
r_pVal))

                    myHosts.hosts(i).position = myHosts.hosts(i).goal
                End If
            End If
        Next

        For i = 0 To myParamors.num - 1
            If myParamors.paramors(i).dead = False Then
                myParamors.paramors(i).power -= (elapsed / 1000) * myParamors.  ↙
paramors(i).behavior.powerRate
                If myParamors.paramors(i).power <= 0 Then
                    myParamors.paramors(i).dead = True
                    myParamors.paramors(i).attached = False
                    myHosts.hosts(myParamors.paramors(i).hoster).numParamors -= ↙
 1
                    If LoggingActive = True And chkLogDetach.Checked = True     ↙
Then
                        logWriter.WriteStartElement("ParamorDetachment")
                        logWriter.WriteAttributeString("Paramor", i)
                        logWriter.WriteAttributeString("EventType", "Death")
                        logWriter.WriteAttributeString("X", myParamors.paramors ↙
(i).curTile.X)
                        logWriter.WriteAttributeString("Y", myParamors.paramors ↙
(i).curTile.Y)
                        logWriter.WriteString(Environment.TickCount)
                        logWriter.WriteEndElement()
                    End If
                    If LoggingActive = True And chkLogPower.Checked = True Then
                        logWriter.WriteStartElement("ParamorPowerEvent")
                        logWriter.WriteAttributeString("Paramor", i)
                        logWriter.WriteAttributeString("EventType", "Dead")
                        logWriter.WriteAttributeString("X", myParamors.paramors ↙
(i).curTile.X)
                        logWriter.WriteAttributeString("Y", myParamors.paramors ↙
(i).curTile.Y)
                        logWriter.WriteString(Environment.TickCount)
                        logWriter.WriteEndElement()
                End If
            ElseIf myParamors.paramors(i).attached = False Then
                If myParamors.paramors(i).stopped = False Then
                    If myParamors.paramors(i).sensing = False Then
                        If myParamors.paramors(i).hoppedOff <> -1 Then
                            If GetDistance(myHosts.hosts(myParamors.          ↙
paramors(i).hoster).position, myParamors.paramors(i).position) > PARAMOR_RANGE ↙
Then
                                myParamors.paramors(i).hoppedOff = -1
                            End If
                        End If
                        Dim j As Integer
                        For j = 0 To myHosts.num - 1
                            If GetDistance(myHosts.hosts(j).position,         ↙
myParamors.paramors(i).position) < PARAMOR_RANGE Then
                                If myParamors.paramors(i).hoppedOff <> j      ↙
Then
                                    If myParamors.paramors(i).behavior.       ↙
coverageGoal = False Or myHosts.hosts(j).numParamors = 0 Then

                                        myParamors.paramors(i).attached =     ↙
True

                                        myParamors.paramors(i).hoster = j
                                        myHosts.hosts(j).numParamors += 1
                                        myParamors.paramors(i).freshBite =    ↙
True

                                        myParamors.paramors(i).curTile =      ↙
GetTile(myHosts.hosts(j).position)
                                        'myParamors.paramors(i).              ↙
hopsRemaining = myParamors.paramors(i).behavior.hopsPerLocale
                                        myParamors.paramors(i).power -=       ↙
myParamors.paramors(i).behavior.attachmentPower
                                        If LoggingActive = True And           ↙
chkLogAttach.Checked = True Then
                                            logWriter.WriteStartElement("     ↙
ParamorAttachment")
                                            logWriter.WriteAttributeString    ↙
("Paramor", i)
                                            logWriter.WriteAttributeString    ↙
("Host", j)
                                            logWriter.WriteAttributeString    ↙
("X", myParamors.paramors(i).curTile.X)
                                            logWriter.WriteAttributeString    ↙
("Y", myParamors.paramors(i).curTile.Y)
                                            logWriter.WriteString(            ↙
Environment.TickCount)
                                            logWriter.WriteEndElement()
                                        End If

                                    End If
                                End If
                            End If
                        Next
                    Else
                        myParamors.paramors(i).senseTicks -= elapsed
                        If myParamors.paramors(i).senseTicks <= 0 Then
                            myParamors.paramors(i).sensing = False
                        End If
                    End If
                Else
                    myParamors.paramors(i).stoppedTick -= elapsed
                    If myParamors.paramors(i).stoppedTick <= 0 Then
                        If myParamors.paramors(i).behavior.stopAtGoal =       ↙
False Then
                            myParamors.paramors(i).stopped = False
```

```
                            End If
                        End If
                    End If
                Else
                    myParamors.paramors(i).position = myHosts.hosts(myParamors. ↙
paramors(i).hoster).position
                    Dim tmpTile As New Point
                    tmpTile = GetTile(myHosts.hosts(myParamors.paramors(i).     ↙
hoster).position)
                    Dim tmpGoal As New Point
                    tmpGoal = GetTile(myHosts.hosts(myParamors.paramors(i).     ↙
hoster).goal)
                    globalCover(tmpTile.X, tmpTile.Y) = True
                    If myParamors.paramors(i).coverage(tmpTile.X, tmpTile.Y) =  ↙
False Then
                        myParamors.paramors(i).coverage(tmpTile.X, tmpTile.Y) = ↙
 True
                        If LoggingActive = True And chkLogCover.Checked = True  ↙
Then
                            logWriter.WriteStartElement("ParamorCoverage")
                            logWriter.WriteAttributeString("Paramor", i)
                            logWriter.WriteAttributeString("X", myParamors.     ↙
paramors(i).curTile.X)
                            logWriter.WriteAttributeString("Y", myParamors.     ↙
paramors(i).curTile.Y)
                            logWriter.WriteString(Environment.TickCount)
                            logWriter.WriteEndElement()
                        End If
                    End If
                    If tmpTile.X <> myParamors.paramors(i).curTile.X Or tmpTile ↙
.Y <> myParamors.paramors(i).curTile.Y Then
                        myParamors.paramors(i).hopsRemaining = myParamors.      ↙
paramors(i).behavior.hopsPerLocale
                        myParamors.paramors(i).curTile = tmpTile
                        If LoggingActive = True And chkLogPTraj.Checked = True  ↙
Then
                            logWriter.WriteStartElement("ParamorTrajectory")
                            logWriter.WriteAttributeString("Paramor", i)
                            logWriter.WriteAttributeString("X", myParamors.     ↙
paramors(i).curTile.X)
                            logWriter.WriteAttributeString("Y", myParamors.     ↙
paramors(i).curTile.Y)
                            logWriter.WriteString(Environment.TickCount)
                            logWriter.WriteEndElement()
                        End If
                        If myParamors.paramors(i).power < myParamors.paramors(i ↙
).behavior.powerThreshold Then
                            If myMap.paraMap(tmpTile.X, tmpTile.Y).Power > 0    ↙
Then
                                myParamors.paramors(i).power -= myParamors.     ↙
paramors(i).behavior.attachmentPower
                                myParamors.paramors(i).attached = False
                                myHosts.hosts(myParamors.paramors(i).hoster).   ↙
numParamors -= 1
                                myParamors.paramors(i).sensing = True
                                myParamors.paramors(i).senseTicks = myParamors. ↙
paramors(i).behavior.senseTime
                                myParamors.paramors(i).power = myParamors.      ↙
paramors(i).behavior.batteryLife
                                myParamors.paramors(i).position = GetPosition(  ↙
tmpTile)
                                If LoggingActive = True And chkLogDetach.       ↙
Checked = True Then
                                    logWriter.WriteStartElement("               ↙
ParamorDetachment")
```

```
                                    logWriter.WriteAttributeString("Paramor", i ↙
)
                                    logWriter.WriteAttributeString("EventType", ↙
 "Charging")
                                    logWriter.WriteAttributeString("X",         ↙
myParamors.paramors(i).curTile.X)
                                    logWriter.WriteAttributeString("Y",         ↙
myParamors.paramors(i).curTile.Y)
                                    logWriter.WriteString(Environment.TickCount ↙
)
                                    logWriter.WriteEndElement()
                                End If
                                If LoggingActive = True And chkLogPower.Checked ↙
 = True Then
                                    logWriter.WriteStartElement("               ↙
ParamorPowerEvent")
                                    logWriter.WriteAttributeString("Paramor", i ↙
)
                                    logWriter.WriteAttributeString("EventType", ↙
 "Charging")
                                    logWriter.WriteAttributeString("X",         ↙
myParamors.paramors(i).curTile.X)
                                    logWriter.WriteAttributeString("Y",         ↙
myParamors.paramors(i).curTile.Y)
                                    logWriter.WriteString(Environment.TickCount ↙
)
                                    logWriter.WriteEndElement()
                                End If
                            End If
                        Else
                            If myParamors.paramors(i).behavior.gotoGoal = True  ↙
And myParamors.paramors(i).behavior.goal.X = tmpTile.X And myParamors.paramors( ↙
i).behavior.goal.Y = tmpTile.Y Then
                                myParamors.paramors(i).power -= myParamors.     ↙
paramors(i).behavior.attachmentPower
                                myParamors.paramors(i).attached = False
                                myHosts.hosts(myParamors.paramors(i).hoster).   ↙
numParamors -= 1
                                myParamors.paramors(i).stopped = True
                                myParamors.paramors(i).stoppedTick = myParamors ↙
.paramors(i).behavior.stopTime
                                myParamors.paramors(i).position = GetPosition(  ↙
tmpTile)
                                If LoggingActive = True And chkLogDetach.       ↙
Checked = True Then
                                    logWriter.WriteStartElement("               ↙
ParamorDetachment")
                                    logWriter.WriteAttributeString("Paramor", i ↙
)
                                    logWriter.WriteAttributeString("EventType", ↙
 "Goal")
                                    logWriter.WriteAttributeString("X",         ↙
myParamors.paramors(i).curTile.X)
                                    logWriter.WriteAttributeString("Y",         ↙
myParamors.paramors(i).curTile.Y)
                                    logWriter.WriteString(Environment.TickCount ↙
)
                                    logWriter.WriteEndElement()
                                End If
                                If LoggingActive = True And chkLogGoal.Checked  ↙
= True Then
                                    logWriter.WriteStartElement("               ↙
ParamorGoalEvent")
                                    logWriter.WriteAttributeString("Paramor", i ↙
)
```

```
                                    logWriter.WriteAttributeString("X",         ↙
myParamors.paramors(i).curTile.X)
                                    logWriter.WriteAttributeString("Y",         ↙
myParamors.paramors(i).curTile.Y)
                                    logWriter.WriteString(Environment.TickCount ↙
)
                                    logWriter.WriteEndElement()
                                End If
                            ElseIf myMap.paraMap(tmpTile.X, tmpTile.Y).Altitude ↙
 > myParamors.paramors(i).behavior.findAltitude Or _
                            myMap.paraMap(tmpTile.X, tmpTile.Y).Light >         ↙
myParamors.paramors(i).behavior.findLight Or _
                            myMap.paraMap(tmpTile.X, tmpTile.Y).Radiation >     ↙
myParamors.paramors(i).behavior.findRadiation Or _
                            myMap.paraMap(tmpTile.X, tmpTile.Y).Temperature >   ↙
myParamors.paramors(i).behavior.findTemperation Or
                            myMap.paraMap(tmpTile.X, tmpTile.Y).Vibration >     ↙
myParamors.paramors(i).behavior.findVibration Then
                                myParamors.paramors(i).power -= myParamors.     ↙
paramors(i).behavior.attachmentPower
                                myParamors.paramors(i).attached = False
                                myHosts.hosts(myParamors.paramors(i).hoster).   ↙
numParamors -= 1
                                myParamors.paramors(i).sensing = True
                                myParamors.paramors(i).senseTicks = myParamors. ↙
paramors(i).behavior.senseTime
                                myParamors.paramors(i).position = GetPosition(  ↙
tmpTile)
                                If LoggingActive = True And chkLogDetach.       ↙
Checked = True Then
                                    logWriter.WriteStartElement("               ↙
ParamorDetachment")
                                    logWriter.WriteAttributeString("Paramor", i ↙
)
                                    logWriter.WriteAttributeString("EventType", ↙
 "Sensing")
                                    logWriter.WriteAttributeString("X",         ↙
myParamors.paramors(i).curTile.X)
                                    logWriter.WriteAttributeString("Y",         ↙
myParamors.paramors(i).curTile.Y)
                                    logWriter.WriteString(Environment.TickCount ↙
)
                                    logWriter.WriteEndElement()
                                End If
                                If LoggingActive = True And chkLogSensor.       ↙
Checked = True Then
                                    logWriter.WriteStartElement("               ↙
ParamorSensorEvent")
                                    logWriter.WriteAttributeString("Paramor", i ↙
)
                                    logWriter.WriteAttributeString("Altitude",  ↙
myMap.paraMap(tmpTile.X, tmpTile.Y).Altitude)
                                    logWriter.WriteAttributeString("Light",     ↙
myMap.paraMap(tmpTile.X, tmpTile.Y).Light)
                                    logWriter.WriteAttributeString("Radiation", ↙
 myMap.paraMap(tmpTile.X, tmpTile.Y).Radiation)
                                    logWriter.WriteAttributeString("Temperature ↙
", myMap.paraMap(tmpTile.X, tmpTile.Y).Temperature)
                                    logWriter.WriteAttributeString("Vibration", ↙
 myMap.paraMap(tmpTile.X, tmpTile.Y).Vibration)
                                    logWriter.WriteAttributeString("X",         ↙
myParamors.paramors(i).curTile.X)
                                    logWriter.WriteAttributeString("Y",         ↙
myParamors.paramors(i).curTile.Y)
                                    logWriter.WriteString(Environment.TickCount ↙
)
                                    logWriter.WriteEndElement()
                                End If
                            Else
                                Dim xind, yind As Integer
                                Dim mapCovered As Boolean = True
                                For xind = 0 To myMap.X - 1
                                    For yind = 0 To myMap.Y - 1
                                        If myParamors.paramors(i).coverage(xind ↙
, yind) = False Then
                                            mapCovered = False
                                            Exit For
                                        End If
                                    Next
                                    If mapCovered = False Then Exit For
                                Next
                                If mapCovered = True Then
                                    myParamors.paramors(i).power -= myParamors. ↙
paramors(i).behavior.attachmentPower
                                    myParamors.paramors(i).attached = False
                                    myHosts.hosts(myParamors.paramors(i).hoster ↙
).numParamors -= 1
                                    myParamors.paramors(i).stopped = True
                                    myParamors.paramors(i).stoppedTick =        ↙
myParamors.paramors(i).behavior.stopTime
                                    myParamors.paramors(i).position =           ↙
GetPosition(tmpTile)
                                    If LoggingActive = True And chkLogDetach.   ↙
Checked = True Then
                                        logWriter.WriteStartElement("           ↙
ParamorDetachment")
                                        logWriter.WriteAttributeString("Paramor ↙
", i)
                                        logWriter.WriteAttributeString("        ↙
EventType", "CoverageComplete")
                                        logWriter.WriteAttributeString("X",     ↙
myParamors.paramors(i).curTile.X)
                                        logWriter.WriteAttributeString("Y",     ↙
myParamors.paramors(i).curTile.Y)
                                        logWriter.WriteString(Environment.      ↙
TickCount)
                                        logWriter.WriteEndElement()
                                    End If
                                    If LoggingActive = True And chkLogGoal.     ↙
Checked = True Then
                                        logWriter.WriteStartElement("           ↙
ParamorCoverageCompleteEvent")
                                        logWriter.WriteAttributeString("Paramor ↙
", i)
                                        logWriter.WriteAttributeString("X",     ↙
myParamors.paramors(i).curTile.X)
                                        logWriter.WriteAttributeString("Y",     ↙
myParamors.paramors(i).curTile.Y)
                                        logWriter.WriteString(Environment.      ↙
TickCount)
                                        logWriter.WriteEndElement()
                                    End If
                                End If
                            End If
                        End If
                        myParamors.paramors(i).freshBite = True
                    ElseIf tmpGoal.X <> myParamors.paramors(i).curTile.X Or     ↙
tmpGoal.Y <> myParamors.paramors(i).curTile.Y Then
                        If myParamors.paramors(i).hopsRemaining > 0 Then
                            If myParamors.paramors(i).behavior.gotoGoal = True  ↙
Then
                                If GetDistance(myParamors.paramors(i).position, ↙
 GetPosition(myParamors.paramors(i).behavior.goal)) < GetDistance(myHosts.hosts ↙
(myParamors.paramors(i).hoster).goal, GetPosition(myParamors.paramors(i).       ↙
behavior.goal)) Then
                                    myParamors.paramors(i).power -= myParamors. ↙
paramors(i).behavior.attachmentPower
                                    myParamors.paramors(i).hopsRemaining -= 1
                                    myParamors.paramors(i).attached = False
                                    myHosts.hosts(myParamors.paramors(i).hoster ↙
).numParamors -= 1
                                    myParamors.paramors(i).curTile = tmpTile
                                    myParamors.paramors(i).position =           ↙
GetPosition(tmpTile)
                                    myParamors.paramors(i).hoppedOff =          ↙
myParamors.paramors(i).hoster
                                    If LoggingActive = True And chkLogDetach.   ↙
Checked = True Then
                                        logWriter.WriteStartElement("           ↙
ParamorDetachment")
                                        logWriter.WriteAttributeString("Paramor ↙
", i)
                                        logWriter.WriteAttributeString("        ↙
EventType", "AwayFromGoal")
                                        logWriter.WriteAttributeString("X",     ↙
myParamors.paramors(i).curTile.X)
                                        logWriter.WriteAttributeString("Y",     ↙
myParamors.paramors(i).curTile.Y)
                                        logWriter.WriteString(Environment.      ↙
TickCount)
                                        logWriter.WriteEndElement()
                                    End If
                                End If
                            End If
                            If myParamors.paramors(i).behavior.coverageGoal =   ↙
True Then
                                Dim tmpGTile As Point = GetTile(myHosts.hosts(  ↙
myParamors.paramors(i).hoster).goal)
                                If myParamors.paramors(i).coverage(tmpGTile.X,  ↙
tmpGTile.Y) = True Then
                                    myParamors.paramors(i).power -= myParamors. ↙
paramors(i).behavior.attachmentPower
                                    myParamors.paramors(i).hopsRemaining -= 1
                                    myParamors.paramors(i).attached = False
                                    myHosts.hosts(myParamors.paramors(i).hoster ↙
).numParamors -= 1
                                    myParamors.paramors(i).position =           ↙
GetPosition(tmpTile)
                                    myParamors.paramors(i).curTile = tmpTile
                                    myParamors.paramors(i).hoppedOff =          ↙
myParamors.paramors(i).hoster
                                    If LoggingActive = True And chkLogDetach.   ↙
Checked = True Then
                                        logWriter.WriteStartElement("           ↙
ParamorDetachment")
                                        logWriter.WriteAttributeString("Paramor ↙
", i)
                                        logWriter.WriteAttributeString("        ↙
EventType", "Covered")
                                        logWriter.WriteAttributeString("X",     ↙
myParamors.paramors(i).curTile.X)
                                        logWriter.WriteAttributeString("Y",     ↙
myParamors.paramors(i).curTile.Y)
                                        logWriter.WriteString(Environment.      ↙
TickCount)
```

```
                                        logWriter.WriteEndElement()
                                    End If
                                End If
                            End If
                        End If
                    End If

                End If
            End If
            If myParamors.paramors(i).behavior.gotoGoal = True Then
                myParamors.paramors(i).lastDistance = GetDistance(myParamors.   ↙
paramors(i).position, GetPosition(myParamors.paramors(i).behavior.goal))
            End If
        Next
        Dim z As Integer
        Dim numgoaled As Integer = 0
        For z = 0 To myParamors.num - 1
            If myParamors.paramors(z).stopped = True Then
                numgoaled += 1
            End If
            If numgoaled >= myParamors.num - 1 Then
                cmdRunLog_Click(Nothing, Nothing)
                cmdRunSim Click(Nothing, Nothing)
                cmdRunLoop_Click(Nothing, Nothing)
            End If
        Next
        Dim m As Integer
        Dim allcover As Integer = 0
        For i = 0 To myMap.X - 1
            For m = 0 To myMap.Y - 1
                If globalCover(i, m) = True Then
                    allcover += 1
                End If
            Next
        Next
        globalPercent = (allcover / (myMap.Y * myMap.X)) * 100.0
        If globalPercent >= 95.0 Then
            If LoggingActive = True Then
                logWriter.WriteStartElement("ParamorCoverageCompleteEvent")
                logWriter.WriteString(Environment.TickCount)
                logWriter.WriteEndElement()
            End If
            cmdRunLog_Click(Nothing, Nothing)
            cmdRunSim_Click(Nothing, Nothing)
            'cmdRunLoop_Click(Nothing, Nothing)
        End If
        iLastTick = Environment.TickCount
    End If
End Sub

Private Function GetGoal(ByVal inTile As Point, ByVal hIndex As Integer) As   ↙
Point
    Dim RMax As Integer = 0
    Dim Weights(5) As Integer
    Dim outTile As New Point

    Weights(0) = 0
    If inTile.X + 1 < myMap.X And inTile.X + 1 >= 0 Then
        If myMap.paraMap(inTile.X, inTile.Y).WallE = False And myMap.paraMap(   ↙
inTile.X + 1, inTile.Y).WallW = False Then
            Weights(0) = myMap.paraMap(inTile.X + 1, inTile.Y).HostTraffic
            If myHosts.hosts(hIndex).coverage(inTile.X + 1, inTile.Y) = True   ↙
Then
                Weights(0) = Weights(0) * myHosts.hosts(hIndex).coveredWeight
            Else
```

```
                Weights(0) = Weights(0) * myHosts.hosts(hIndex).uncoveredWeight
            End If
        End If
    End If

    Weights(1) = 0
    If inTile.X - 1 < myMap.X And inTile.X - 1 >= 0 Then
        If myMap.paraMap(inTile.X, inTile.Y).WallW = False And myMap.paraMap(   ↙
inTile.X - 1, inTile.Y).WallE = False Then
            Weights(1) = myMap.paraMap(inTile.X - 1, inTile.Y).HostTraffic
            If myHosts.hosts(hIndex).coverage(inTile.X - 1, inTile.Y) = True   ↙
Then
                Weights(1) = Weights(1) * myHosts.hosts(hIndex).coveredWeight
            Else
                Weights(1) = Weights(1) * myHosts.hosts(hIndex).uncoveredWeight
            End If
        End If
    End If

    Weights(2) = 0
    If inTile.Y - 1 < myMap.Y And inTile.Y - 1 >= 0 Then
        If myMap.paraMap(inTile.X, inTile.Y).WallN = False And myMap.paraMap(   ↙
inTile.X, inTile.Y - 1).WallS = False Then
            Weights(2) = myMap.paraMap(inTile.X, inTile.Y - 1).HostTraffic
            If myHosts.hosts(hIndex).coverage(inTile.X, inTile.Y - 1) = True   ↙
Then
                Weights(2) = Weights(2) * myHosts.hosts(hIndex).coveredWeight
            Else
                Weights(2) = Weights(2) * myHosts.hosts(hIndex).uncoveredWeight
            End If
        End If
    End If

    Weights(3) = 0
    If inTile.Y + 1 < myMap.Y And inTile.Y + 1 >= 0 Then
        If myMap.paraMap(inTile.X, inTile.Y).WallS = False And myMap.paraMap(   ↙
inTile.X, inTile.Y + 1).WallN = False Then
            Weights(3) = myMap.paraMap(inTile.X, inTile.Y + 1).HostTraffic
            If myHosts.hosts(hIndex).coverage(inTile.X, inTile.Y + 1) = True   ↙
Then
                Weights(3) = Weights(3) * myHosts.hosts(hIndex).coveredWeight
            Else
                Weights(3) = Weights(3) * myHosts.hosts(hIndex).uncoveredWeight
            End If
        End If
    End If

    Weights(4) = myHosts.hosts(hIndex).stayWeight

    Weights(5) = 0
    If myMap.paraMap(inTile.X, inTile.Y).Portal = True Then
        Weights(5) = myHosts.hosts(hIndex).portalWeight
    End If

    Dim i As Integer
    Dim rVal As Integer = rando.Next(1, Weights(0) + Weights(1) + Weights(2) +   ↙
Weights(3) + Weights(4) + Weights(5))
    Dim rBack As Integer = rVal
    For i = 0 To 5
        rVal -= Weights(i)
        If rVal <= 0 Then
            Select Case i
                Case 0
                    outTile.X = inTile.X + 1
                    outTile.Y = inTile.Y
                    Exit For
                Case 1
                    outTile.X = inTile.X - 1
                    outTile.Y = inTile.Y
                    Exit For
                Case 2
                    outTile.X = inTile.X
                    outTile.Y = inTile.Y - 1
                    Exit For
                Case 3
                    outTile.X = inTile.X
                    outTile.Y = inTile.Y + 1
                    Exit For
                Case 4
                    outTile.X = inTile.X
                    outTile.Y = inTile.Y
                    myHosts.hosts(hIndex).moving = False
                    myHosts.hosts(hIndex).stayTicks = myHosts.hosts(hIndex).    ↙
stayDuration
                    Exit For
                Case 5
                    outTile.X = inTile.X
                    outTile.Y = inTile.Y
                    myHosts.hosts(hIndex).involved = False
                    myHosts.hosts(hIndex).portalTicks = myHosts.hosts(hIndex).  ↙
portalDuration
                    Exit For
            End Select
        End If
    Next
    If outTile.X > myMap.X - 1 Or outTile.Y > myMap.Y - 1 Then
        outTile.X = inTile.X
        outTile.Y = inTile.Y
    End If
    Return outTile
End Function

Private Function GetDistance(ByVal point1 As Vector3, ByVal point2 As Vector3)  ↙
As Single
    Return Vector3.Length(Vector3.Subtract(point1, point2))
End Function

Private Function GetTile(ByVal point1 As Vector3) As Point
    Dim tmpPt As New Point
    Dim tmpInt As Integer
    tmpInt = Int(point1.X + 2.5) / 5
    tmpPt.X = tmpInt
    tmpInt = Int(point1.Z + 2.5) / 5
    tmpPt.Y = tmpInt
    If tmpPt.X > myMap.X - 1 Then
        tmpPt.X = myMap.X - 1
    End If
    If tmpPt.Y > myMap.Y - 1 Then
        tmpPt.Y = myMap.Y - 1
    End If
    Return tmpPt
End Function

Private Function GetPosition(ByVal inTile As Point) As Vector3
    Dim tmpVector As Vector3
    tmpVector.X = inTile.X * 5
    tmpVector.Y = 2
    tmpVector.Z = inTile.Y * 5
    Return tmpVector
End Function
```

```
Private Sub FrameRender()
    Dim Mat As Matrix = Matrix.Identity
    Dim NewMat As Matrix
    Dim mtrl3d As Material = Nothing
    Dim i, j As Integer

    DXGrafx.Clear(Color.DarkBlue, 1.0F, 0)
    DXGrafx.Device.BeginScene()

    InitLights()

    If mapLoaded = False Then
        'Render Splash Text
        mtrl3d.Diffuse = Color.NavajoWhite
        DXGrafx.Device.Material = mtrl3d
        DXGrafx.Device.Transform.World = splashMat
        mesh3DText.DrawSubset(0)
    Else
        DXGrafx.Device.Lights(4).Position = New Vector3(((myMap.X / 2) - 1) * 5 ↙
, 40, ((myMap.Y / 2) - 1) * 5)
        DXGrafx.Device.Lights(4).Commit()
        mtrl3d.Diffuse = Color.NavajoWhite
        DXGrafx.Device.Material = mtrl3d
        For j = 0 To myMap.Y - 1
            For i = 0 To myMap.X - 1
                Mat = Matrix.Translation(myMap.paraMap(i, j).XPos, 0, myMap.    ↙
paraMap(i, j).YPos)
                DXGrafx.Device.SetTransform(Direct3D.TransformType.World, Mat)
                DXGrafx.Device.SetStreamSource(0, vbCube, 0)
                DXGrafx.Device.VertexFormat = CustomVertex.                     ↙
PositionNormalTextured.Format
                'render the first cube
                DXGrafx.Device.SetTexture(0, groundTex)
                DXGrafx.Device.DrawPrimitives(PrimitiveType.TriangleList, 0, 12 ↙
)

                If myMap.paraMap(i, j).WallE = True Then
                    DXGrafx.Device.SetTexture(0, wallTex)
                    NewMat = Matrix.Multiply(Matrix.Scaling(0.5, 0.8, 1), Mat)
                    NewMat = Matrix.Multiply(Matrix.RotationX(Math.PI * 0.5),   ↙
NewMat)
                    NewMat = Matrix.Multiply(Matrix.RotationZ(Math.PI * 0.5),   ↙
NewMat)
                    NewMat = Matrix.Multiply(NewMat, Matrix.Translation(2.25, 2 ↙
.5, 0))
                    DXGrafx.Device.SetTransform(Direct3D.TransformType.World,   ↙
NewMat)
                    DXGrafx.Device.DrawPrimitives(PrimitiveType.TriangleList, 0 ↙
, 12)
                End If
                If myMap.paraMap(i, j).WallW = True Then
                    DXGrafx.Device.SetTexture(0, wallTex)
                    NewMat = Matrix.Multiply(Matrix.Scaling(0.5, 0.8, 1), Mat)
                    NewMat = Matrix.Multiply(Matrix.RotationX(Math.PI * 0.5),   ↙
NewMat)
                    NewMat = Matrix.Multiply(Matrix.RotationZ(Math.PI * 0.5),   ↙
NewMat)
                    NewMat = Matrix.Multiply(NewMat, Matrix.Translation(-2.25,  ↙
2.5, 0))
                    DXGrafx.Device.SetTransform(Direct3D.TransformType.World,   ↙
NewMat)
                    DXGrafx.Device.DrawPrimitives(PrimitiveType.TriangleList, 0 ↙
, 12)
                End If
                If myMap.paraMap(i, j).WallS = True Then
                    DXGrafx.Device.SetTexture(0, wallTex)
                    NewMat = Matrix.Multiply(Matrix.Scaling(1, 0.8, 0.5), Mat)
                    NewMat = Matrix.Multiply(Matrix.RotationX(Math.PI * 0.5),    ↙
NewMat)
                    NewMat = Matrix.Multiply(NewMat, Matrix.Translation(0, 2.5, ↙
 2.25))
                    DXGrafx.Device.SetTransform(Direct3D.TransformType.World,    ↙
NewMat)
                    DXGrafx.Device.DrawPrimitives(PrimitiveType.TriangleList, 0 ↙
, 12)
                End If
                If myMap.paraMap(i, j).WallN = True Then
                    DXGrafx.Device.SetTexture(0, wallTex)
                    NewMat = Matrix.Multiply(Matrix.Scaling(1, 0.8, 0.5), Mat)
                    NewMat = Matrix.Multiply(Matrix.RotationX(Math.PI * 0.5),    ↙
NewMat)
                    NewMat = Matrix.Multiply(NewMat, Matrix.Translation(0, 2.5, ↙
 -2.25))
                    DXGrafx.Device.SetTransform(Direct3D.TransformType.World,    ↙
NewMat)
                    DXGrafx.Device.DrawPrimitives(PrimitiveType.TriangleList, 0 ↙
, 12)
                End If
            Next
        Next

        'Draw Hosts
        For i = 0 To myHosts.num - 1
            If myHosts.hosts(i).involved = True Then
                If myHosts.hosts(i).numParamors > 0 Then
                    mtrl3d.Ambient = Color.FromArgb(128, 255, 153, 0)
                Else
                    mtrl3d.Ambient = Color.FromArgb(128, 255, 0, 0)
                End If
                mtrl3d.Diffuse = mtrl3d.Ambient
                DXGrafx.Device.Material = mtrl3d
                Mat = Matrix.Translation(myHosts.hosts(i).position.X, myHosts.  ↙
hosts(i).position.Y, myHosts.hosts(i).position.Z)
                DXGrafx.Device.SetTransform(Direct3D.TransformType.World, Mat)
                SphereMesh.DrawSubset(0)
            End If
        Next

        For i = 0 To myParamors.num - 1
            If myParamors.paramors(i).attached = False Then
                If myParamors.paramors(i).dead = True Then
                    mtrl3d.Ambient = Color.FromArgb(128, 0, 0, 0)
                ElseIf myParamors.paramors(i).sensing = True Then
                    mtrl3d.Ambient = Color.FromArgb(128, 255, 255, 255)
                ElseIf myParamors.paramors(i).stopped = True Then
                    mtrl3d.Ambient = Color.FromArgb(128, 0, 255, 0)
                Else
                    mtrl3d.Ambient = Color.FromArgb(128, 255, 255, 0)
                End If
                mtrl3d.Diffuse = mtrl3d.Ambient
                DXGrafx.Device.Material = mtrl3d
                Mat = Matrix.Translation(myParamors.paramors(i).position.X,      ↙
myParamors.paramors(i).position.Y, myParamors.paramors(i).position.Z)
                DXGrafx.Device.SetTransform(Direct3D.TransformType.World, Mat)
                SphereMesh.DrawSubset(0)
            End If
        Next

    End If
```

```
    If ToggleInfo = True Then
        RenderInfo()
    End If

    DXGrafx.Device.EndScene()

    If DeviceLost Then
        Try
            ' Test the cooperative level to see if it's okay to render
            DXGrafx.Device.TestCooperativeLevel()
        Catch e As Direct3D.DeviceLostException
            ' If the device was lost, do not render until we get it back
            Exit Sub
        Catch e As Direct3D.DeviceNotResetException
            ' Reset the device and resize it
            DXGrafx.Device.Reset(DXGrafx.Device.PresentationParameters)
        End Try

        DeviceLost = False
    End If

    Try
        DXGrafx.Present()
    Catch e As Direct3D.DeviceLostException
        DeviceLost = True
    End Try

End Sub

Private Sub RenderInfo()
    Dim Msg As String = ""
    Msg &= "Device: " + sDevInfo & vbCrLf
    Msg &= "FPS: " & framePerSecond & vbCrLf   'iFrameRate.ToString & vbCrLf
    Msg &= "Lighting: " & DXGrafx.Device.RenderState.Lighting.ToString & vbCrLf
    Msg &= "FillMode: " & DXGrafx.Device.RenderState.FillMode.ToString & vbCrLf
    Msg &= "ShadeMode: " & DXGrafx.Device.RenderState.ShadeMode.ToString &      ↙
vbCrLf
    Msg &= "CullMode: " & DXGrafx.Device.RenderState.CullMode.ToString & vbCrLf ↙
 & vbCrLf
    Msg &= "Coverage: " & globalPercent & vbCrLf & vbCrLf

    TxtFont.DrawText(Msg, New Rectangle(10, 10, Me.ClientSize.Width - 10, Me.   ↙
ClientSize.Height - 10), _
                        Direct3D.DrawTextFormat.Left Or Direct3D.DrawTextFormat. ↙
Top, _
                        Color.Red)
End Sub

Private Sub InitLights()
    Dim Light0 As Direct3D.Light = DXGrafx.Device.Lights(0)
    Dim Light1 As Direct3D.Light = DXGrafx.Device.Lights(1)
    Dim Light2 As Direct3D.Light = DXGrafx.Device.Lights(2)
    Dim Light3 As Direct3D.Light = DXGrafx.Device.Lights(3)
    Dim Light4 As Direct3D.Light = DXGrafx.Device.Lights(4)

    Light0.Type = Direct3D.LightType.Directional
    Light0.Direction = New Vector3(0, -1, 1)
    Light0.Diffuse = Color.FromArgb(255, 255, 255)
    Light0.Ambient = Color.FromArgb(0, 30, 30, 200)
    Light0.Enabled = False
    Light0.Commit()

    Light1.Type = Direct3D.LightType.Directional
    Light1.Direction = New Vector3(0, 1, 1)
```

```
        Light1.Diffuse = Color.FromArgb(255, 255, 255)
        Light1.Ambient = Color.FromArgb(0, 30, 30, 200)
        Light1.Enabled = False
        Light1.Commit()

        Light2.Type = Direct3D.LightType.Directional
        Light2.Direction = New Vector3(0, 0, -1)
        Light2.Diffuse = Color.FromArgb(255, 255, 255)
        Light2.Ambient = Color.FromArgb(0, 30, 30, 200)
        Light2.Enabled = False
        Light2.Commit()

        Light3.Type = Direct3D.LightType.Directional
        Light3.Direction = New Vector3(0, -1, 0)
        Light3.Diffuse = Color.FromArgb(255, 255, 255)
        Light3.Ambient = Color.FromArgb(0, 30, 30, 200)
        Light3.Enabled = False
        Light3.Commit()

        Light4.Type = Direct3D.LightType.Point
        Light4.Direction = New Vector3(0, -1, 0)
        Light4.Position = New Vector3(10, 40, 10)
        Light4.Range = 100
        Light4.Diffuse = Color.White
        Light4.Ambient = Color.Red
        Light4.InnerConeAngle = 0.0F
        Light4.OuterConeAngle = 1.0F
        Light4.Falloff = 1.0F
        Light4.Attenuation0 = 1.0F
        Light4.Enabled = True
        Light4.Commit()
    End Sub

    Private Function InitGraphics() As Boolean
        Try
            Dim d3dPP As New PresentParameters
            DXGrafx = New DX9ToolsR5.DX9Graphics(pbRender)

            Cam = New DX9ToolsR5.Camera(DXGrafx)
            With DXGrafx.Device
                .RenderState.CullMode = Direct3D.Cull.CounterClockwise
                .RenderState.FillMode = Direct3D.FillMode.Solid
                .RenderState.ShadeMode = Direct3D.ShadeMode.Gouraud
                .RenderState.Lighting = True
                .RenderState.ZBufferEnable = True

                .RenderState.SourceBlend = Blend.SourceAlpha
                .RenderState.DestinationBlend = Blend.InvSourceAlpha

                'set up texture blending
                .SamplerState(0).MinFilter = TextureFilter.Linear
                .SamplerState(0).MagFilter = TextureFilter.Linear

                ' Blend a Texture with Vertex Color
                .TextureState(0).ColorOperation = Direct3D.TextureOperation.   ↙
    Modulate
                .TextureState(0).ColorArgument1 = Direct3D.TextureArgument.    ↙
    TextureColor
                .TextureState(0).ColorArgument2 = Direct3D.TextureArgument.Diffuse
                .TextureState(0).AlphaOperation = Direct3D.TextureOperation.Disable

                TxtFont = New Direct3D.Font(DXGrafx.Device, Me.Font)
                sDevInfo = D3DMngr.Adapters(0).Information.Description
            End With
        Catch ex As Exception
```

```
            Return False
        End Try

        Return True
    End Function

    Private Function RotateView(ByVal angle As Single, ByVal x As Single, ByVal y  ↙
    As Single, ByVal z As Single)
        Dim vView As Vector3
        Dim vNewView As Vector3
        vView.X = Cam.LookAt.X - Cam.Position.X
        vView.Y = Cam.LookAt.Y - Cam.Position.Y
        vView.Z = Cam.LookAt.Z - Cam.Position.Z
        Dim cosTheta As Single = Math.Cos(angle)
        Dim sinTheta As Single = Math.Sin(angle)
        vNewView.X = (cosTheta + (1 - cosTheta) * x * x) * vView.X
        vNewView.X += ((1 - cosTheta) * x * y - z * sinTheta) * vView.Y
        vNewView.X += ((1 - cosTheta) * x * z + y * sinTheta) * vView.Z
        vNewView.Y = ((1 - cosTheta) * x * y + z * sinTheta) * vView.X
        vNewView.Y += (cosTheta + (1 - cosTheta) * y * y) * vView.Y
        vNewView.Y += ((1 - cosTheta) * y * z - x * sinTheta) * vView.Z
        vNewView.Z = ((1 - cosTheta) * x * z - y * sinTheta) * vView.X
        vNewView.Z += ((1 - cosTheta) * y * z + x * sinTheta) * vView.Y
        vNewView.Z += (cosTheta + (1 - cosTheta) * z * z) * vView.Z
        vNewView.X = Cam.Position.X + vNewView.X
        vNewView.Y = Cam.Position.Y + vNewView.Y
        vNewView.Z = Cam.Position.Z + vNewView.Z
        Cam.SetLookAt(vNewView.X, vNewView.Y, vNewView.Z, True)
    End Function

    Private Function AdvanceCamera(ByVal speed As Single)
        Dim vVector As Vector3 = Vector3.Empty
        vVector.X = Cam.LookAt.X - Cam.Position.X
        vVector.Y = Cam.LookAt.Y - Cam.Position.Y
        vVector.Z = Cam.LookAt.Z - Cam.Position.Z

        Cam.SetPosition(DX9ToolsR5.SetVector.X, Cam.Position.X + (vVector.X * speed ↙
    ))
        Cam.SetPosition(DX9ToolsR5.SetVector.Y, Cam.Position.Y + (vVector.Y * speed ↙
    ))
        Cam.SetPosition(DX9ToolsR5.SetVector.Z, Cam.Position.Z + (vVector.Z * speed ↙
    ))
        Cam.SetLookAt(DX9ToolsR5.SetVector.X, Cam.LookAt.X + (vVector.X * speed))
        Cam.SetLookAt(DX9ToolsR5.SetVector.Y, Cam.LookAt.Y + (vVector.Y * speed))
        Cam.SetLookAt(DX9ToolsR5.SetVector.Z, Cam.LookAt.Z + (vVector.Z * speed))
        Cam.UpdateVP()
    End Function

    Private Function HStrafeCamera(ByVal speed As Single)
        Dim vStrafe As Vector3 = Vector3.Cross(Vector3.Subtract(Cam.LookAt, Cam.   ↙
    Position), Cam.Orientation)
        Cam.SetPosition(DX9ToolsR5.SetVector.X, Cam.Position.X + (vStrafe.X * speed ↙
    ))
        Cam.SetPosition(DX9ToolsR5.SetVector.Y, Cam.Position.Y + (vStrafe.Y * speed ↙
    ))
        Cam.SetPosition(DX9ToolsR5.SetVector.Z, Cam.Position.Z + (vStrafe.Z * speed ↙
    ))
        Cam.SetLookAt(DX9ToolsR5.SetVector.X, Cam.LookAt.X + (vStrafe.X * speed))
        Cam.SetLookAt(DX9ToolsR5.SetVector.Y, Cam.LookAt.Y + (vStrafe.Y * speed))
        Cam.SetLookAt(DX9ToolsR5.SetVector.Z, Cam.LookAt.Z + (vStrafe.Z * speed))
        Cam.UpdateVP()
    End Function

    Private Function VStrafeCamera(ByVal speed As Single)
        Dim vStrafe As Vector3 = Cam.Orientation
        Cam.SetPosition(DX9ToolsR5.SetVector.X, Cam.Position.X + (vStrafe.X * speed ↙
    ))
        Cam.SetPosition(DX9ToolsR5.SetVector.Y, Cam.Position.Y + (vStrafe.Y * speed ↙
    ))
        Cam.SetPosition(DX9ToolsR5.SetVector.Z, Cam.Position.Z + (vStrafe.Z * speed ↙
    ))
        Cam.SetLookAt(DX9ToolsR5.SetVector.X, Cam.LookAt.X + (vStrafe.X * speed))
        Cam.SetLookAt(DX9ToolsR5.SetVector.Y, Cam.LookAt.Y + (vStrafe.Y * speed))
        Cam.SetLookAt(DX9ToolsR5.SetVector.Z, Cam.LookAt.Z + (vStrafe.Z * speed))
        Cam.UpdateVP()
    End Function

    Private Function RollCamera(ByVal angle As Single, ByVal x As Single, ByVal y  ↙
    As Single, ByVal z As Single)
        Dim vView As Vector3
        Dim vNewView As Vector3
        vView.X = Cam.Orientation.X - Cam.Position.X
        vView.Y = Cam.Orientation.Y - Cam.Position.Y
        vView.Z = Cam.Orientation.Z - Cam.Position.Z
        Dim cosTheta As Single = Math.Cos(angle)
        Dim sinTheta As Single = Math.Sin(angle)
        vNewView.X = (cosTheta + (1 - cosTheta) * x * x) * vView.X
        vNewView.X += ((1 - cosTheta) * x * y - z * sinTheta) * vView.Y
        vNewView.X += ((1 - cosTheta) * x * z + y * sinTheta) * vView.Z
        vNewView.Y = ((1 - cosTheta) * x * y + z * sinTheta) * vView.X
        vNewView.Y += (cosTheta + (1 - cosTheta) * y * y) * vView.Y
        vNewView.Y += ((1 - cosTheta) * y * z - x * sinTheta) * vView.Z
        vNewView.Z = ((1 - cosTheta) * x * z - y * sinTheta) * vView.X
        vNewView.Z += ((1 - cosTheta) * y * z + x * sinTheta) * vView.Y
        vNewView.Z += (cosTheta + (1 - cosTheta) * z * z) * vView.Z
        vNewView.X = Cam.Position.X + vNewView.X
        vNewView.Y = Cam.Position.Y + vNewView.Y
        vNewView.Z = Cam.Position.Z + vNewView.Z
        Cam.SetOrientation(vNewView.X, vNewView.Y, vNewView.Z, True)
    End Function

Windows Form Designer generated code

    Private Sub frmParasimRun_Closed(ByVal sender As Object, ByVal e As System.  ↙
    EventArgs) Handles MyBase.Closed
        Cleanup()
    End Sub
    Private Sub Cleanup()
        TxtFont = Nothing
        DXGrafx = Nothing

        If LoggingActive = True Then
            logWriter.WriteStartElement("LogStopped")
            logWriter.WriteString(Environment.TickCount)
            logWriter.WriteEndElement()
            logWriter.WriteEndElement()
            ' Ends the document.
            logWriter.WriteEndDocument()
            ' close writer
            logWriter.Close()
        End If
        End
    End Sub


    Private Sub cmdInfo_Click(ByVal sender As System.Object, ByVal e As System.  ↙
    EventArgs) Handles cmdInfo.Click
        If ToggleInfo = False Then
            ToggleInfo = True
        Else
            ToggleInfo = False
        End If
    End Sub

    Private Sub cmdLights_Click(ByVal sender As System.Object, ByVal e As System.  ↙
    EventArgs) Handles cmdLights.Click
        If DXGrafx.Device.RenderState.Lighting = True Then
            DXGrafx.Device.RenderState.Lighting = False
        Else
            DXGrafx.Device.RenderState.Lighting = True
        End If
    End Sub

    Private Sub cmdCull_Click(ByVal sender As System.Object, ByVal e As System.  ↙
    EventArgs) Handles cmdCull.Click
        If DXGrafx.Device.RenderState.CullMode = Direct3D.Cull.None Then
            DXGrafx.Device.RenderState.CullMode = Cull.Clockwise
        ElseIf DXGrafx.Device.RenderState.CullMode = Cull.Clockwise Then
            DXGrafx.Device.RenderState.CullMode = Cull.CounterClockwise
        Else
            DXGrafx.Device.RenderState.CullMode = Cull.None
        End If
    End Sub

    Private Sub cmdFill_Click(ByVal sender As System.Object, ByVal e As System.  ↙
    EventArgs) Handles cmdFill.Click
        If DXGrafx.Device.RenderState.FillMode = Direct3D.FillMode.Solid Then
            DXGrafx.Device.RenderState.FillMode = Direct3D.FillMode.Point
        ElseIf DXGrafx.Device.RenderState.FillMode = Direct3D.FillMode.Point Then
            DXGrafx.Device.RenderState.FillMode = Direct3D.FillMode.WireFrame
        Else
            DXGrafx.Device.RenderState.FillMode = Direct3D.FillMode.Solid
        End If
    End Sub

    Private Sub cmdShader_Click(ByVal sender As System.Object, ByVal e As System.  ↙
    EventArgs) Handles cmdShader.Click
        If DXGrafx.Device.RenderState.ShadeMode = Direct3D.ShadeMode.Gouraud Then
            DXGrafx.Device.RenderState.ShadeMode = Direct3D.ShadeMode.Phong
        ElseIf DXGrafx.Device.RenderState.ShadeMode = Direct3D.ShadeMode.Phong Then
            DXGrafx.Device.RenderState.ShadeMode = Direct3D.ShadeMode.Flat
        Else
            DXGrafx.Device.RenderState.ShadeMode = Direct3D.ShadeMode.Gouraud
        End If
    End Sub


    Private Sub cmdLoadMap_Click(ByVal sender As System.Object, ByVal e As System.  ↙
    EventArgs) Handles cmdLoadMap.Click
        Me.OpenFileDialog1.ShowDialog()
        If OpenFileDialog1.FileName <> "" Then
            mapFileName = OpenFileDialog1.FileName
            Me.Text = "Parasitic Mobility Simulator - " & mapFileName
            myMap = New psMap(mapFileName, 5, 5)
            ReDim globalCover(myMap.X - 1, myMap.Y - 1)
            mapLoaded = True
            Dim pan As Single = (myMap.X / 2) - 1
            Cam.SetPosition(DX9ToolsR5.SetVector.X, pan * 5)
            Cam.SetLookAt(DX9ToolsR5.SetVector.X, pan * 5)
            Cam.SetPosition(DX9ToolsR5.SetVector.Y, 10)
            myHosts.SetMap(myMap)
            myHosts.AssignBehaviors()
            myParamors.SetMap(myMap)
            myParamors.AssignBehaviors()
        If mapLoaded = True And myHosts.behaviorsAssigned = True And myParamors ↙
.behaviorsAssigned = True Then
            cmdRunSim.Enabled = True
        End If
    End If
End Sub

Private Sub KeyPressed(ByVal sender As Object, ByVal e As KeyPressEventArgs)
    Select Case e.KeyChar
        Case "w"
            AdvanceCamera(0.02)
            e.Handled = True
        Case "s"
            AdvanceCamera(-0.02)
            e.Handled = True
        Case "a"
            HStrafeCamera(0.02)
            e.Handled = True
        Case "d"
            HStrafeCamera(-0.02)
            e.Handled = True
        Case "e"
            VStrafeCamera(0.5)
            e.Handled = True
        Case "c"
            VStrafeCamera(-0.5)
            e.Handled = True
        Case "r"
            With Cam
                .Reset()
                .SetPosition(DX9ToolsR5.SetVector.Z, -30)
                .AspectRatio = CSng(Me.ClientSize.Width / Me.ClientSize.Height)
            End With
            e.Handled = True
    End Select
End Sub

Private Sub ViewMouseLeave(ByVal sender As System.Object, ByVal e As System.    ↙
Windows.Forms.MouseEventArgs) Handles pbRender.MouseUp
    If e.Button = MouseButtons.Left Then
        mouseClickedL = False
    End If
    If e.Button = MouseButtons.Right Then
        mouseClickedR = False
    End If
End Sub

Private Sub ViewByMouse(ByVal sender As System.Object, ByVal e As System.      ↙
Windows.Forms.MouseEventArgs) Handles pbRender.MouseMove
    If e.Button = MouseButtons.Left Then
        If mouseClickedL = False Then
            mouseClickedL = True
        Else
            Dim angleX As Single = 0.0
            Dim angleY As Single = 0.0
            angleY = ((lastMousePos.X - e.X)) / 1000.0F
            angleX = ((lastMousePos.Y - e.Y)) / 1000.0F
            RotateView(angleY, 0, 1, 0)
            RotateView(angleX, 1, 0, 0)
        End If
    End If
    If e.Button = MouseButtons.Right Then
        If mouseClickedR = False Then
            mouseClickedR = True
        Else
```

```
            Dim angleZ As Single = 0.0
            angleZ = ((lastMousePos.X - e.X)) / 1000.0F
            RollCamera(angleZ, 0, 1, 1)
        End If
    End If
    lastMousePos.X = e.X
    lastMousePos.Y = e.Y

End Sub

Private Sub loadTextures(ByVal AdapterFmt As Format)


    'attempt to load the cube textures
    If D3DMngr.CheckDeviceFormat(0, _
                                 DeviceType.Hardware, _
                                 AdapterFmt, _
                                 0, _
                                 ResourceType.Textures, _
                                 Format.X8R8G8B8 _
                                 ) Then
        wallTex = TextureLoader.FromFile(DXGrafx.Device, _
                                         Application.StartupPath + "\wall.jpg" ↙
, _
                                         256, 256, 1, 0, _
                                         Format.X8R8G8B8, Pool.Managed, _
                                         Filter.Linear, Filter.Linear, 0)
        groundTex = TextureLoader.FromFile(DXGrafx.Device, _
                                         Application.StartupPath + "\tex_tile2 ↙
.bmp", _
                                         256, 256, 1, 0, _
                                         Format.X8R8G8B8, Pool.Managed, _
                                         Filter.Linear, Filter.Linear, 0)

    ElseIf D3DMngr.CheckDeviceFormat(0, _
                                     DeviceType.Hardware, _
                                     AdapterFmt, _
                                     0, _
                                     ResourceType.Textures, _
                                     Format.R5G6B5 _
                                     ) Then
        wallTex = TextureLoader.FromFile(DXGrafx.Device, _
                                         Application.StartupPath + "\wall.jpg" ↙
, _
                                         256, 256, 1, 0, _
                                         Format.R5G6B5, Pool.Managed, _
                                         Filter.Linear, Filter.Linear, 0)
        groundTex = TextureLoader.FromFile(DXGrafx.Device, _
                                         Application.StartupPath + "\tex_tile2 ↙
.bmp", _
                                         256, 256, 1, 0, _
                                         Format.R5G6B5, Pool.Managed, _
                                         Filter.Linear, Filter.Linear, 0)

    Else
        Throw New Exception("cube textures cannot be created: no support for 16 ↙
 bit (R5G6B5) or 32bit (X8R8G8B8) textures!")
    End If
End Sub


Private Sub loadGeometry()
    Try
        '[-----------------------------------]
        '   LOAD 3D GEOMETRY INTO VBUFFERS
```

```
            '[-----------------------------------]
            vbCube = New VertexBuffer(GetType(CustomVertex.PositionNormalTextured), ↙
     _
                                     36, DXGrafx.Device, 0, CustomVertex.        ↙
    PositionNormalTextured.Format, _
                                     Pool.Managed)

            'use CType() to typecast a pointer to the
            'memory into an array of position&textured vertices
            Dim vCube As CustomVertex.PositionNormalTextured() = CType(vbCube.Lock( ↙
    0, 0), _
CustomVertex.PositionNormalTextured())

            '2a. Generate top plane
            vCube(0) = New CustomVertex.PositionNormalTextured(-2.5, 0.5, 2.5, 0, 1 ↙
    , 0, 0, 0)
            vCube(1) = New CustomVertex.PositionNormalTextured(2.5, 0.5, 2.5, 0, 1, ↙
     0, 1, 0)
            vCube(2) = New CustomVertex.PositionNormalTextured(-2.5, 0.5, -2.5, 0,  ↙
    1, 0, 0, 1)

            vCube(3) = New CustomVertex.PositionNormalTextured(2.5, 0.5, -2.5, 0, 1 ↙
    , 0, 1, 1)
            vCube(4) = New CustomVertex.PositionNormalTextured(-2.5, 0.5, -2.5, 0,  ↙
    1, 0, 0, 1)
            vCube(5) = New CustomVertex.PositionNormalTextured(2.5, 0.5, 2.5, 0, 1, ↙
     0, 1, 0)

            '2b. Generate bottom plane
            vCube(6) = New CustomVertex.PositionNormalTextured(-2.5, -0.5, -2.5, 0, ↙
     -1, 0, 0, 0)
            vCube(7) = New CustomVertex.PositionNormalTextured(2.5, -0.5, 2.5, 0, - ↙
    1, 0, 1, 1)
            vCube(8) = New CustomVertex.PositionNormalTextured(-2.5, -0.5, 2.5, 0,  ↙
    -1, 0, 0, 1)

            vCube(9) = New CustomVertex.PositionNormalTextured(2.5, -0.5, 2.5, 0, - ↙
    1, 0, 1, 1)
            vCube(10) = New CustomVertex.PositionNormalTextured(-2.5, -0.5, -2.5, 0 ↙
    , -1, 0, 0, 0)
            vCube(11) = New CustomVertex.PositionNormalTextured(2.5, -0.5, -2.5, 0, ↙
     -1, 0, 1, 0)

            '2c. Generate 'left' plane
            vCube(12) = New CustomVertex.PositionNormalTextured(-2.5, 0.5, -2.5, -1 ↙
    , 0, 0, 0, 0)
            vCube(13) = New CustomVertex.PositionNormalTextured(-2.5, -0.5, -2.5, - ↙
    1, 0, 0, 1, 0)
            vCube(14) = New CustomVertex.PositionNormalTextured(-2.5, 0.5, 2.5, -1, ↙
     0, 0, 0, 1)

            vCube(15) = New CustomVertex.PositionNormalTextured(-2.5, 0.5, 2.5, -1, ↙
     0, 0, 0, 1)
            vCube(16) = New CustomVertex.PositionNormalTextured(-2.5, -0.5, -2.5, - ↙
    1, 0, 0, 1, 0)
            vCube(17) = New CustomVertex.PositionNormalTextured(-2.5, -0.5, 2.5, -1 ↙
    , 0, 0, 1, 1)

            '2d. Generate 'right' plane
            vCube(18) = New CustomVertex.PositionNormalTextured(2.5, 0.5, -2.5, 1,  ↙
    0, 0, 0, 0)
            vCube(19) = New CustomVertex.PositionNormalTextured(2.5, 0.5, 2.5, 1, 0 ↙
    , 0, 1, 0)
            vCube(20) = New CustomVertex.PositionNormalTextured(2.5, -0.5, -2.5, 1, ↙
     0, 0, 0, 1)
```

```
        vCube(21) = New CustomVertex.PositionNormalTextured(2.5, -0.5, 2.5, 1,  ↙
0, 0, 1, 1)
        vCube(22) = New CustomVertex.PositionNormalTextured(2.5, -0.5, -2.5, 1, ↙
 0, 0, 0, 1)
        vCube(23) = New CustomVertex.PositionNormalTextured(2.5, 0.5, 2.5, 1, 0 ↙
, 0, 1, 0)

        '2e. Generate 'Back' plane
        vCube(24) = New CustomVertex.PositionNormalTextured(-2.5, 0.5, -2.5, 0, ↙
 0, -1, 0, 0)
        vCube(25) = New CustomVertex.PositionNormalTextured(2.5, 0.5, -2.5, 0,  ↙
0, -1, 1, 0)
        vCube(26) = New CustomVertex.PositionNormalTextured(-2.5, -0.5, -2.5, 0 ↙
, 0, -1, 0, 1)

        vCube(27) = New CustomVertex.PositionNormalTextured(2.5, -0.5, -2.5, 0, ↙
 0, -1, 1, 1)
        vCube(28) = New CustomVertex.PositionNormalTextured(-2.5, -0.5, -2.5, 0 ↙
, 0, -1, 0, 1)
        vCube(29) = New CustomVertex.PositionNormalTextured(2.5, 0.5, -2.5, 0,  ↙
0, -1, 1, 0)

        '2f. Generate 'Front' plane
        vCube(30) = New CustomVertex.PositionNormalTextured(2.5, 0.5, 2.5, 0, 0 ↙
, 1, 1, 0)
        vCube(31) = New CustomVertex.PositionNormalTextured(-2.5, 0.5, 2.5, 0,  ↙
0, 1, 0, 0)
        vCube(32) = New CustomVertex.PositionNormalTextured(-2.5, -0.5, 2.5, 0, ↙
 0, 1, 0, 1)

        vCube(33) = New CustomVertex.PositionNormalTextured(-2.5, -0.5, 2.5, 0, ↙
 0, 1, 0, 1)
        vCube(34) = New CustomVertex.PositionNormalTextured(2.5, -0.5, 2.5, 0,  ↙
0, 1, 1, 1)
        vCube(35) = New CustomVertex.PositionNormalTextured(2.5, 0.5, 2.5, 0, 0 ↙
, 1, 1, 0)

        vbCube.Unlock()


    Catch err As Exception
        MsgBox("loadGeometry(): " + Chr(13) + Chr(13) + err.ToString())
        Throw New Exception("could not complete loadGeometry()")
    End Try
End Sub

Private Sub cmdSetupHosts_Click(ByVal sender As System.Object, ByVal e As       ↙
System.EventArgs) Handles cmdSetupHosts.Click
    Dim shbf As frmHostB = New frmHostB(myHosts)
    shbf.Show()
    'myHosts.AssignBehaviors()
End Sub

Private Sub cmdSave_Click(ByVal sender As System.Object, ByVal e As System.     ↙
EventArgs) Handles cmdSave.Click
    SaveFileDialog1.ShowDialog()
    If SaveFileDialog1.FileName <> "" Then
        writeSettingsToFile(SaveFileDialog1.FileName)
    End If

End Sub

Private Sub cmdRunSim_Click(ByVal sender As System.Object, ByVal e As System.   ↙
EventArgs) Handles cmdRunSim.Click
```

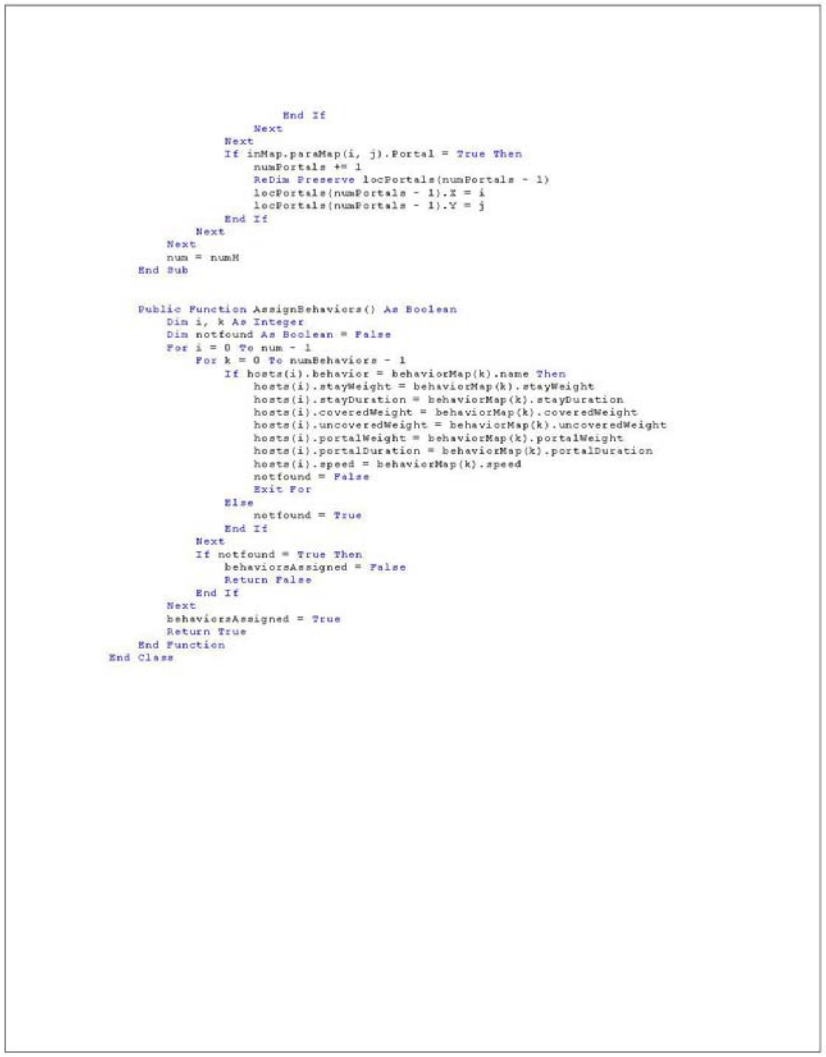
```
    If cmdRunSim.Text = "RUN!" Then
        cmdRunSim.Text = "STOP!"
        simRunning = True
        iLastTick = Environment.TickCount
    Else
        cmdRunSim.Text = "RUN!"
        simRunning = False
    End If
End Sub

Public Function writeSettingsToFile(ByVal fileName As String)
    Dim textWriter As XmlTextWriter = New XmlTextWriter(fileName, Nothing)
    Dim i, j, k As Integer
    textWriter.Formatting = Formatting.Indented
    ' Opens the document
    textWriter.WriteStartDocument()
    ' Write comments
    textWriter.WriteComment("ParaSim Settings File")
    Dim str As String = "Filename: " & fileName
    textWriter.WriteComment(str)
    str = "Created: " & System.DateTime.Now
    textWriter.WriteComment(str)
    textWriter.WriteStartElement("Settings")
    textWriter.WriteAttributeString("TimeScale", numTimeScale.Value)

    textWriter.WriteStartElement("HostBehaviorMap")
    textWriter.WriteAttributeString("numBehaviors", myHosts.numBehaviors)

    For i = 0 To myHosts.numBehaviors - 1
        textWriter.WriteStartElement("Behavior")
        textWriter.WriteAttributeString("index", i)
        textWriter.WriteStartElement("Name")
        textWriter.WriteString(myHosts.behaviorMap(i).name)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("CoveredWeight")
        textWriter.WriteString(myHosts.behaviorMap(i).coveredWeight)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("UnCoveredWeight")
        textWriter.WriteString(myHosts.behaviorMap(i).uncoveredWeight)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("PortalDuration")
        textWriter.WriteString(myHosts.behaviorMap(i).portalDuration)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("PortalWeight")
        textWriter.WriteString(myHosts.behaviorMap(i).portalWeight)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("Speed")
        textWriter.WriteString(myHosts.behaviorMap(i).speed)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("StayDuration")
        textWriter.WriteString(myHosts.behaviorMap(i).stayDuration)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("StayWeight")
        textWriter.WriteString(myHosts.behaviorMap(i).stayWeight)
        textWriter.WriteEndElement()
        textWriter.WriteEndElement()
    Next
    textWriter.WriteEndElement()

    textWriter.WriteStartElement("ParamorBehaviorMap")
    textWriter.WriteAttributeString("numBehaviors", myParamors.numBehaviors)

    For i = 0 To myParamors.numBehaviors - 1
        textWriter.WriteStartElement("Behavior")
        textWriter.WriteAttributeString("index", i)
        textWriter.WriteStartElement("Name")
        textWriter.WriteString(myParamors.behaviorMap(i).name)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("AttachmentPower")
        textWriter.WriteString(myParamors.behaviorMap(i).attachmentPower)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("CoverageGoal")
        textWriter.WriteString(myParamors.behaviorMap(i).coverageGoal)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("FindAltitude")
        textWriter.WriteString(myParamors.behaviorMap(i).findAltitude)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("FindLight")
        textWriter.WriteString(myParamors.behaviorMap(i).findLight)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("FindRadiation")
        textWriter.WriteString(myParamors.behaviorMap(i).findRadiation)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("FindTemperature")
        textWriter.WriteString(myParamors.behaviorMap(i).findTemperation)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("FindVibration")
        textWriter.WriteString(myParamors.behaviorMap(i).findVibration)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("GoalX")
        textWriter.WriteString(myParamors.behaviorMap(i).goal.X)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("GoalY")
        textWriter.WriteString(myParamors.behaviorMap(i).goal.Y)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("GoToGoal")
        textWriter.WriteString(myParamors.behaviorMap(i).gotoGoal)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("StopAtGoal")
        textWriter.WriteString(myParamors.behaviorMap(i).stopAtGoal)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("PowerRate")
        textWriter.WriteString(myParamors.behaviorMap(i).powerRate)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("SenseTime")
        textWriter.WriteString(myParamors.behaviorMap(i).senseTime)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("BatteryLife")
        textWriter.WriteString(myParamors.behaviorMap(i).batteryLife)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("HopsPerLocale")
        textWriter.WriteString(myParamors.behaviorMap(i).hopsPerLocale)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("PowerThreshold")
        textWriter.WriteString(myParamors.behaviorMap(i).powerThreshold)
        textWriter.WriteEndElement()
        textWriter.WriteStartElement("StopTime")
        textWriter.WriteString(myParamors.behaviorMap(i).stopTime)
        textWriter.WriteEndElement()
        textWriter.WriteEndElement()
    Next
    textWriter.WriteEndElement()

    textWriter.WriteEndElement()
    ' Ends the document.
    textWriter.WriteEndDocument()
    ' close writer
    textWriter.Close()

End Function
```

```
Private Sub cmdSetupPara Click(ByVal sender As System.Object, ByVal e As System ↙
.EventArgs) Handles cmdSetupPara.Click
    Dim shbf As frmParamorB = New frmParamorB(myParamors)
    shbf.Show()

End Sub

Private Sub cmdLoad Click(ByVal sender As System.Object, ByVal e As System.     ↙
EventArgs) Handles cmdLoad.Click
    Me.OpenFileDialog1.ShowDialog()
    If OpenFileDialog1.FileName <> "" Then
        LoadSettings(OpenFileDialog1.FileName)
    End If
End Sub

Private Function LoadSettings(ByVal filename As String)
    Dim i, j, k As Integer
    Dim m_nodelist As XmlNodeList
    Dim m_node As XmlNode
    Dim m_xmld As XmlDocument
    m_xmld = New XmlDocument
    'Load the Xml file
    m_xmld.Load(filename)
    m_node = m_xmld.SelectSingleNode("/Settings")
    numTimeScale.Value = m_node.Attributes.GetNamedItem("TimeScale").Value
    m_node = m_xmld.SelectSingleNode("/Settings/HostBehaviorMap")
    myHosts.numBehaviors = m_node.Attributes.GetNamedItem("numBehaviors").Value
    ReDim myHosts.behaviorMap(myHosts.numBehaviors - 1)
    'Get the list of name nodes
    m_nodelist = m_xmld.SelectNodes("/Settings/HostBehaviorMap/Behavior")
    'Loop through the nodes
    For Each m_node In m_nodelist
        i = m_node.Attributes.GetNamedItem("index").Value
        myHosts.behaviorMap(i).name = m_node.ChildNodes.Item(0).InnerText
        myHosts.behaviorMap(i).coveredWeight = m_node.ChildNodes.Item(1).       ↙
InnerText
        myHosts.behaviorMap(i).uncoveredWeight = m_node.ChildNodes.Item(2).     ↙
InnerText
        myHosts.behaviorMap(i).portalDuration = m_node.ChildNodes.Item(3).      ↙
InnerText
        myHosts.behaviorMap(i).portalWeight = m_node.ChildNodes.Item(4).        ↙
InnerText
        myHosts.behaviorMap(i).speed = m_node.ChildNodes.Item(5).InnerText
        myHosts.behaviorMap(i).stayDuration = m_node.ChildNodes.Item(6).        ↙
InnerText
        myHosts.behaviorMap(i).stayWeight = m_node.ChildNodes.Item(7).InnerText
    Next
    myHosts.AssignBehaviors()
    m_node = m_xmld.SelectSingleNode("/Settings/ParamorBehaviorMap")
    myParamors.numBehaviors = m_node.Attributes.GetNamedItem("numBehaviors").   ↙
Value
    ReDim myParamors.behaviorMap(myParamors.numBehaviors - 1)

    m_nodelist = m_xmld.SelectNodes("/Settings/ParamorBehaviorMap/Behavior")
    'Loop through the nodes
    For Each m_node In m_nodelist
        i = m_node.Attributes.GetNamedItem("index").Value
        myParamors.behaviorMap(i).name = m_node.ChildNodes.Item(0).InnerText
        myParamors.behaviorMap(i).attachmentPower = m_node.ChildNodes.Item(1).  ↙
InnerText
        myParamors.behaviorMap(i).coverageGoal = m_node.ChildNodes.Item(2).     ↙
InnerText
        myParamors.behaviorMap(i).findAltitude = m_node.ChildNodes.Item(3).     ↙
InnerText
        myParamors.behaviorMap(i).findLight = m_node.ChildNodes.Item(4).        ↙
InnerText
        myParamors.behaviorMap(i).findRadiation = m_node.ChildNodes.Item(5).    ↙
InnerText
        myParamors.behaviorMap(i).findTemperation = m_node.ChildNodes.Item(6).  ↙
InnerText
        myParamors.behaviorMap(i).findVibration = m_node.ChildNodes.Item(7).    ↙
InnerText
        myParamors.behaviorMap(i).goal.X = m node.ChildNodes.Item(8).InnerText
        myParamors.behaviorMap(i).goal.Y = m_node.ChildNodes.Item(9).InnerText
        myParamors.behaviorMap(i).gotoGoal = m_node.ChildNodes.Item(10).        ↙
InnerText
        myParamors.behaviorMap(i).stopAtGoal = m_node.ChildNodes.Item(11).      ↙
InnerText
        myParamors.behaviorMap(i).powerRate = m_node.ChildNodes.Item(12).       ↙
InnerText
        myParamors.behaviorMap(i).senseTime = m_node.ChildNodes.Item(13).       ↙
InnerText
        myParamors.behaviorMap(i).batteryLife = m_node.ChildNodes.Item(14).     ↙
InnerText
        myParamors.behaviorMap(i).hopsPerLocale = m_node.ChildNodes.Item(15).   ↙
InnerText
        myParamors.behaviorMap(i).powerThreshold = m_node.ChildNodes.Item(16).  ↙
InnerText
        myParamors.behaviorMap(i).stopTime = m_node.ChildNodes.Item(17).        ↙
InnerText
    Next
    myParamors.AssignBehaviors()

End Function

Private Sub cmdVideoFile_Click(ByVal sender As System.Object, ByVal e As System ↙
.EventArgs) Handles cmdVideoFile.Click
    Me.SaveFileDialog2.ShowDialog()
    If SaveFileDialog2.FileName <> "" Then
        videoFileName = SaveFileDialog2.FileName
        cmdRecord.Enabled = True
        videoFrameCnt = 0
    End If
End Sub

Private Sub cmdRecord_Click(ByVal sender As System.Object, ByVal e As System.   ↙
EventArgs) Handles cmdRecord.Click
    If cmdRecord.Text = "Record" Then
        cmdRecord.Text = "Stop Recording"
        tmrVideo.Enabled = True
    Else
        cmdRecord.Text = "Record"
        tmrVideo.Enabled = False
    End If
End Sub

Private Sub tmrVideo_Tick(ByVal sender As System.Object, ByVal e As System.     ↙
EventArgs) Handles tmrVideo.Tick
    Dim tmpFN As String
    Dim B As Direct3D.Surface
    tmpFN = videoFileName & "." & videoFrameCnt.ToString.PadLeft(5, "0") & ".   ↙
bmp"
    'Dim tmpBmp As New Bitmap(Me.pbRender.Image)
    'tmpBmp.Save(tmpFN)
    B = DXGrafx.Device.GetBackBuffer(0, 0, BackBufferType.Mono)
    Direct3D.SurfaceLoader.Save(tmpFN, Direct3D.ImageFileFormat.Bmp, B)
    B.Dispose()

    videoFrameCnt += 1
    If videoFrameCnt > 99999 Then
        tmrVideo.Enabled = False
        cmdRecord.Enabled = False
    End If
End Sub

Private Sub cmdLogFile_Click(ByVal sender As System.Object, ByVal e As System. ↙
EventArgs) Handles cmdLogFile.Click
    Me.SaveFileDialog3.ShowDialog()
    If SaveFileDialog3.FileName <> "" Then
        logFileName = SaveFileDialog3.FileName
        cmdRunLog.Enabled = True
        cmdRunLog.Text = "Begin Logging"
    End If

End Sub

Private Sub cmdRunLog_Click(ByVal sender As System.Object, ByVal e As System. ↙
EventArgs) Handles cmdRunLog.Click
    If cmdRunLog.Text = "Begin Logging" Then
        cmdRunLog.Text = "Stop Logging"
        LoggingActive = True
        logWriter = New XmlTextWriter(logFileName, Nothing)
        logWriter.Formatting = Formatting.Indented
        ' Opens the document
        logWriter.WriteStartDocument()
        ' Write comments
        logWriter.WriteComment("ParaSim Log File")
        Dim str As String = "Filename: " & logFileName
        logWriter.WriteComment(str)
        str = "Created: " & System.DateTime.Now
        logWriter.WriteComment(str)
        logWriter.WriteStartElement("Log")
        logWriter.WriteStartElement("LoggingStarted")
        logWriter.WriteAttributeString("TimeScale", numTimeScale.Value)
        logWriter.WriteString(Environment.TickCount)
        logWriter.WriteEndElement()

    Else
        cmdRunLog.Text = "Begin Logging"
        LoggingActive = False
        logWriter.WriteStartElement("LogStopped")
        logWriter.WriteString(Environment.TickCount)
        logWriter.WriteEndElement()
        logWriter.WriteEndElement()
        ' Ends the document.
        logWriter.WriteEndDocument()
        ' close writer
        logWriter.Close()
    End If
End Sub

Private Sub cmdRunLoop_Click(ByVal sender As System.Object, ByVal e As System. ↙
EventArgs) Handles cmdRunLoop.Click
    If pass > end_pass Then
        Exit Sub
    End If
    multipass = True
    'mapFileName = "C:\paramosim\parasim_run\parasim_run\bin\coverage\cover_set ↙
" & pass & ".xml"
    mapFileName = "C:\paramosim\parasim_run\parasim_run\bin\distance15\        ↙
distance_map_long2.xml"
    Me.Text = "Parasitic Mobility Simulator - " & mapFileName
    myMap = New psMap(mapFileName, 5, 5)
    mapLoaded = True
```

```
        ReDim globalCover(myMap.X - 1, myMap.Y - 1)
        Dim pan As Single = (myMap.X / 2) - 1
        Cam.SetPosition(DX9ToolsR5.SetVector.X, pan * 5)
        Cam.SetLookAt(DX9ToolsR5.SetVector.X, pan * 5)
        Cam.SetPosition(DX9ToolsR5.SetVector.Y, 10)
        myHosts.SetMap(myMap)
        myHosts.AssignBehaviors()
        myParamors.SetMap(myMap)
        myParamors.AssignBehaviors()
        If mapLoaded = True And myHosts.behaviorsAssigned = True And myParamors.     ↙
    behaviorsAssigned = True Then
            cmdRunSim.Enabled = True
        End If
        Dim setFileName As String = "C:\paramosim\parasim_run\parasim_run\bin\      ↙
    distance15\distance_set15.xml"
        'Dim setFileName As String = "C:\paramosim\parasim_run\parasim_run\bin\     ↙
    coverage\cover_set.xml"
        LoadSettings(setFileName)
        Dim j As Integer
        For j = 0 To myParamors.num - 1
            myParamors.paramors(j).behavior.goal.X = pass + 5
        Next
        '         logFileName = "C:\paramosim\parasim_run\parasim_run\bin\autodata\ ↙
    distance_log" & pass & ".xml"
        'myMap.paraMap(10, 1).HostTraffic = pass
        'myMap.paraMap(10, 2).HostTraffic = pass
        'myMap.paraMap(10, 3).HostTraffic = pass
        'myMap.paraMap(10, 4).HostTraffic = pass
        'myMap.paraMap(10, 5).HostTraffic = pass
        'myMap.paraMap(10, 6).HostTraffic = pass
        'myMap.paraMap(10, 7).HostTraffic = pass
        'myMap.paraMap(10, 8).HostTraffic = pass
        logFileName = "C:\paramosim\parasim_run\parasim_run\bin\distance15\         ↙
    distance_data" & pass & ".xml"
        cmdRunLog.Enabled = True
        cmdRunLog.Text = "Begin Logging"
        chkLogAttach.Checked = True
        chkLogDetach.Checked = True
        chkLogGoal.Checked = True
        '         chkLogCover.Checked = True
        cmdRunLog_Click(Nothing, Nothing)
        cmdRunSim_Click(Nothing, Nothing)
        pass += 1
    End Sub

    Private Sub cmdClear_Click(ByVal sender As System.Object, ByVal e As System.    ↙
    EventArgs) Handles cmdClear.Click
        Dim i, j, k As Integer
        For i = 0 To myParamors.num - 1
            For j = 0 To myMap.X - 1
                For k = 0 To myMap.Y - 1
                    myParamors.paramors(i).coverage(j, k) = False
                Next
            Next
        Next
    End Sub
End Class
```

```
'FILE: hosts.vb
'CLASS: hosts
'DESCRIPTION: Class that assigns host behaviors
'AUTHOR: Mat Laibowitz

Public Class hosts

    Public Structure host
        Dim position As Vector3
        Dim goal As Vector3
        Dim speed As Single
        Dim behavior As String
        Dim coverage(,) As Boolean
        Dim numParamors As Integer
        Dim involved As Boolean
        Dim moving As Boolean
        Dim stayWeight As Integer
        Dim stayDuration As Integer
        Dim stayTicks As Integer
        Dim portalDuration As Integer
        Dim portalTicks As Integer
        Dim coveredWeight As Single
        Dim uncoveredWeight As Single
        Dim portalWeight As Integer
        Dim trajLogged As Boolean
    End Structure

    Public Structure hostBehavior
        Dim name As String
        Dim stayWeight As Integer
        Dim stayDuration As Integer
        Dim coveredWeight As Single
        Dim uncoveredWeight As Single
        Dim portalWeight As Integer
        Dim portalDuration As Integer
        Dim speed As Single
    End Structure

    Public num As Integer
    Public hosts() As host
    Public numBehaviors As Integer
    Public behaviorMap() As hostBehavior
    Public tmpnumBehaviors As Integer
    Public tmpbehaviorMap() As hostBehavior
    Public behaviorsAssigned As Boolean
    Public numPortals As Integer
    Public locPortals() As Point

    Public Sub New()
        behaviorsAssigned = False
        num = 0
        numBehaviors = 0
        tmpnumBehaviors = 0
    End Sub

    Public Sub SetMap(ByVal inMap As psMap)
        Dim i, j, k, l, numH As Integer
        behaviorsAssigned = False
        numH = 0
        numPortals = 0
        For j = 0 To inMap.Y - 1
            For i = 0 To inMap.X - 1
                For k = 0 To inMap.paraMap(i, j).Hosts - 1
                    numH = numH + 1
                    ReDim Preserve hosts(numH - 1)
```

```
                ReDim hosts(numH - 1).coverage(inMap.X, inMap.Y)
                hosts(numH - 1).behavior = inMap.paraMap(i, j).HostBehaviors(k)
                hosts(numH - 1).numParamors = 0
                hosts(numH - 1).trajLogged = False
                Select Case k Mod 9
                    Case 0
                        hosts(numH - 1).position = New Vector3(inMap.paraMap(i, ↙
j).XPos - 2, 2, inMap.paraMap(i, j).YPos - 2)
                    Case 1
                        hosts(numH - 1).position = New Vector3(inMap.paraMap(i, ↙
j).XPos - 2, 2, inMap.paraMap(i, j).YPos)
                    Case 2
                        hosts(numH - 1).position = New Vector3(inMap.paraMap(i, ↙
j).XPos - 2, 2, inMap.paraMap(i, j).YPos + 2)
                    Case 3
                        hosts(numH - 1).position = New Vector3(inMap.paraMap(i, ↙
j).XPos, 2, inMap.paraMap(i, j).YPos - 2)
                    Case 4
                        hosts(numH - 1).position = New Vector3(inMap.paraMap(i, ↙
j).XPos, 2, inMap.paraMap(i, j).YPos)
                    Case 5
                        hosts(numH - 1).position = New Vector3(inMap.paraMap(i, ↙
j).XPos, 2, inMap.paraMap(i, j).YPos + 2)
                    Case 6
                        hosts(numH - 1).position = New Vector3(inMap.paraMap(i, ↙
j).XPos + 2, 2, inMap.paraMap(i, j).YPos - 2)
                    Case 7
                        hosts(numH - 1).position = New Vector3(inMap.paraMap(i, ↙
j).XPos + 2, 2, inMap.paraMap(i, j).YPos)
                    Case 8
                        hosts(numH - 1).position = New Vector3(inMap.paraMap(i, ↙
j).XPos + 2, 2, inMap.paraMap(i, j).YPos + 2)
                End Select
                hosts(numH - 1).goal = hosts(numH - 1).position
                hosts(numH - 1).involved = True
                hosts(numH - 1).moving = True
                hosts(numH - 1).speed = 5.0
                hosts(numH - 1).coveredWeight = 1.0
                hosts(numH - 1).uncoveredWeight = 1.0
                hosts(numH - 1).stayWeight = 100
                hosts(numH - 1).stayDuration = 100
                hosts(numH - 1).portalDuration = 10000
                hosts(numH - 1).portalWeight = 100
                For l = 0 To numBehaviors - 1
                    If hosts(numH - 1).behavior = behaviorMap(l).name Then
                        Exit For
                    ElseIf l = numBehaviors - 1 Then
                        numBehaviors = numBehaviors + 1
                        ReDim Preserve behaviorMap(numBehaviors - 1)
                        behaviorMap(numBehaviors - 1).name = hosts(numH - 1).    ↙
behavior
                        behaviorMap(numBehaviors - 1).coveredWeight = hosts(     ↙
numH - 1).coveredWeight
                        behaviorMap(numBehaviors - 1).portalWeight = hosts(numH ↙
 - 1).portalWeight
                        behaviorMap(numBehaviors - 1).portalDuration = hosts(    ↙
numH - 1).portalDuration
                        behaviorMap(numBehaviors - 1).stayWeight = hosts(numH - ↙
 1).stayWeight
                        behaviorMap(numBehaviors - 1).stayDuration = hosts(numH ↙
 - 1).stayDuration
                        behaviorMap(numBehaviors - 1).uncoveredWeight = hosts(   ↙
numH - 1).uncoveredWeight
                        behaviorMap(numBehaviors - 1).speed = hosts(numH - 1).   ↙
speed
```



```
                    End If
                Next
            Next
            If inMap.paraMap(i, j).Portal = True Then
                numPortals += 1
                ReDim Preserve locPortals(numPortals - 1)
                locPortals(numPortals - 1).X = i
                locPortals(numPortals - 1).Y = j
            End If
        Next
    Next
    num = numH
End Sub


Public Function AssignBehaviors() As Boolean
    Dim i, k As Integer
    Dim notfound As Boolean = False
    For i = 0 To num - 1
        For k = 0 To numBehaviors - 1
            If hosts(i).behavior = behaviorMap(k).name Then
                hosts(i).stayWeight = behaviorMap(k).stayWeight
                hosts(i).stayDuration = behaviorMap(k).stayDuration
                hosts(i).coveredWeight = behaviorMap(k).coveredWeight
                hosts(i).uncoveredWeight = behaviorMap(k).uncoveredWeight
                hosts(i).portalWeight = behaviorMap(k).portalWeight
                hosts(i).portalDuration = behaviorMap(k).portalDuration
                hosts(i).speed = behaviorMap(k).speed
                notfound = False
                Exit For
            Else
                notfound = True
            End If
        Next
        If notfound = True Then
            behaviorsAssigned = False
            Return False
        End If
    Next
    behaviorsAssigned = True
    Return True
End Function
End Class
```

```
'FILE: paramors.vb
'CLASS: paramors
'DESCRIPTION: Class that assigns paramor behaviors
'AUTHOR: Mat Laibowitz

Public Class paramors

    Public Structure paramor
        Dim position As Vector3
        Dim curTile As Point
        Dim behavior As paramorBehavior
        Dim coverage(,) As Boolean
        Dim attached As Boolean
        Dim power As Integer
        Dim hoster As Integer
        Dim stopped As Boolean
        Dim sensing As Boolean
        Dim senseTicks As Integer
        Dim freshBite As Boolean
        Dim stoppedTick As Integer
        Dim hopsRemaining As Integer
        Dim lastDistance As Single
        Dim dead As Boolean
        Dim hoppedOff As Integer
    End Structure

    Public Structure paramorBehavior
        Dim name As String
        Dim powerRate As Integer
        Dim batteryLife As Integer
        Dim attachmentPower As Integer
        Dim powerThreshold As Integer
        Dim goal As Point
        Dim gotoGoal As Boolean
        Dim stopAtGoal As Boolean
        Dim findLight As Integer
        Dim findVibration As Integer
        Dim findTemperation As Integer
        Dim findAltitude As Integer
        Dim findRadiation As Integer
        Dim senseTime As Integer
        Dim stopTime As Integer
        Dim hopsPerLocale As Integer
        Dim coverageGoal As Boolean
    End Structure

    Public num As Integer
    Public paramors() As paramor
    Public numBehaviors As Integer
    Public behaviorMap() As paramorBehavior
    Public tmpnumBehaviors As Integer
    Public tmpbehaviorMap() As paramorBehavior
    Public behaviorsAssigned As Boolean

    Public Sub New()
        behaviorsAssigned = False
        num = 0
        numBehaviors = 0
        tmpnumBehaviors = 0
    End Sub

    Public Sub SetMap(ByVal inMap As psMap)
        Dim i, j, k, l, numP As Integer
        behaviorsAssigned = False
        numP = 0
```

```
        For j = 0 To inMap.Y - 1
            For i = 0 To inMap.X - 1
                For k = 0 To inMap.paraMap(i, j).Paramors - 1
                    numP = numP + 1
                    ReDim Preserve paramors(numP - 1)
                    ReDim paramors(numP - 1).coverage(inMap.X, inMap.Y)
                    paramors(numP - 1).behavior.name = inMap.paraMap(i, j).              ↙
ParamorBehaviors(k)
                    paramors(numP - 1).attached = False
                    paramors(numP - 1).stopped = False
                    paramors(numP - 1).sensing = False
                    paramors(numP - 1).hoster = 0
                    paramors(numP - 1).power = 10000
                    paramors(numP - 1).behavior.batteryLife = 10000
                    paramors(numP - 1).curTile.X = i
                    paramors(numP - 1).curTile.Y = j
                    paramors(numP - 1).hoppedOff = -1
                    Select Case k Mod 9
                        Case 0
                            paramors(numP - 1).position = New Vector3(inMap.paraMap ↙
(i, j).XPos - 1, 2, inMap.paraMap(i, j).YPos - 1)
                        Case 1
                            paramors(numP - 1).position = New Vector3(inMap.paraMap ↙
(i, j).XPos - 1, 2, inMap.paraMap(i, j).YPos)
                        Case 2
                            paramors(numP - 1).position = New Vector3(inMap.paraMap ↙
(i, j).XPos - 1, 2, inMap.paraMap(i, j).YPos + 1)
                        Case 3
                            paramors(numP - 1).position = New Vector3(inMap.paraMap ↙
(i, j).XPos, 2, inMap.paraMap(i, j).YPos - 1)
                        Case 4
                            paramors(numP - 1).position = New Vector3(inMap.paraMap ↙
(i, j).XPos, 2, inMap.paraMap(i, j).YPos)
                        Case 5
                            paramors(numP - 1).position = New Vector3(inMap.paraMap ↙
(i, j).XPos, 2, inMap.paraMap(i, j).YPos + 1)
                        Case 6
                            paramors(numP - 1).position = New Vector3(inMap.paraMap ↙
(i, j).XPos + 1, 2, inMap.paraMap(i, j).YPos - 1)
                        Case 7
                            paramors(numP - 1).position = New Vector3(inMap.paraMap ↙
(i, j).XPos + 1, 2, inMap.paraMap(i, j).YPos)
                        Case 8
                            paramors(numP - 1).position = New Vector3(inMap.paraMap ↙
(i, j).XPos + 1, 2, inMap.paraMap(i, j).YPos + 1)
                    End Select
                    paramors(numP - 1).dead = False
                    paramors(numP - 1).freshBite = False
                    paramors(numP - 1).hopsRemaining = 2
                    paramors(numP - 1).behavior.hopsPerLocale = 2
                    paramors(numP - 1).behavior.powerThreshold = 100
                    paramors(numP - 1).behavior.stopTime = 10000
                    paramors(numP - 1).behavior.attachmentPower = 0
                    paramors(numP - 1).behavior.coverageGoal = False
                    paramors(numP - 1).behavior.findAltitude = 100
                    paramors(numP - 1).behavior.findLight = 100
                    paramors(numP - 1).behavior.findRadiation = 100
                    paramors(numP - 1).behavior.findTemperation = 100
                    paramors(numP - 1).behavior.findVibration = 100
                    paramors(numP - 1).behavior.gotoGoal = False
                    paramors(numP - 1).behavior.stopAtGoal = False
                    paramors(numP - 1).behavior.powerRate = 0
                    paramors(numP - 1).behavior.senseTime = 1000
                    For l = 0 To numBehaviors - 1
                        If paramors(numP - 1).behavior.name = behaviorMap(l).name    ↙
```

```
Then
                        Exit For
                    ElseIf l = numBehaviors - 1 Then
                        numBehaviors = numBehaviors + 1
                        ReDim Preserve behaviorMap(numBehaviors - 1)
                        behaviorMap(numBehaviors - 1).name = paramors(numP - 1) ↙
.behavior.name
                        behaviorMap(numBehaviors - 1).attachmentPower =         ↙
paramors(numP - 1).behavior.attachmentPower
                        behaviorMap(numBehaviors - 1).batteryLife = paramors(   ↙
numP - 1).behavior.batteryLife
                        behaviorMap(numBehaviors - 1).coverageGoal = paramors(  ↙
numP - 1).behavior.coverageGoal
                        behaviorMap(numBehaviors - 1).findAltitude = paramors(  ↙
numP - 1).behavior.findAltitude
                        behaviorMap(numBehaviors - 1).findLight = paramors(numP ↙
 - 1).behavior.findLight
                        behaviorMap(numBehaviors - 1).findRadiation = paramors( ↙
numP - 1).behavior.findRadiation
                        behaviorMap(numBehaviors - 1).findTemperation =         ↙
paramors(numP - 1).behavior.findTemperation
                        behaviorMap(numBehaviors - 1).findVibration = paramors( ↙
numP - 1).behavior.findVibration
                        behaviorMap(numBehaviors - 1).goal = paramors(numP - 1) ↙
.behavior.goal
                        behaviorMap(numBehaviors - 1).gotoGoal = paramors(numP  ↙
- 1).behavior.gotoGoal
                        behaviorMap(numBehaviors - 1).hopsPerLocale = paramors( ↙
numP - 1).behavior.hopsPerLocale
                        behaviorMap(numBehaviors - 1).powerRate = paramors(numP ↙
 - 1).behavior.powerRate
                        behaviorMap(numBehaviors - 1).powerThreshold = paramors ↙
(numP - 1).behavior.powerThreshold
                        behaviorMap(numBehaviors - 1).senseTime = paramors(numP ↙
 - 1).behavior.senseTime
                        behaviorMap(numBehaviors - 1).stopTime = paramors(numP  ↙
- 1).behavior.stopTime
                    End If
                Next

            Next
        Next
    Next
    num = numP
End Sub

Public Function AssignBehaviors() As Boolean
    Dim i, k As Integer
    Dim notfound As Boolean = False
    For i = 0 To num - 1
        For k = 0 To numBehaviors - 1
            If paramors(i).behavior.name = behaviorMap(k).name Then
                paramors(i).behavior.attachmentPower = behaviorMap(k).          ↙
attachmentPower
                paramors(i).behavior.coverageGoal = behaviorMap(k).coverageGoal
                paramors(i).behavior.findAltitude = behaviorMap(k).findAltitude
                paramors(i).behavior.findLight = behaviorMap(k).findLight
                paramors(i).behavior.findRadiation = behaviorMap(k).            ↙
findRadiation
                paramors(i).behavior.findTemperation = behaviorMap(k).          ↙
findTemperation
                paramors(i).behavior.findVibration = behaviorMap(k).            ↙
findVibration
                paramors(i).behavior.goal = behaviorMap(k).goal
                paramors(i).behavior.gotoGoal = behaviorMap(k).gotoGoal
```

```
                paramors(i).behavior.stopAtGoal = behaviorMap(k).stopAtGoal
                paramors(i).behavior.powerRate = behaviorMap(k).powerRate
                paramors(i).behavior.senseTime = behaviorMap(k).senseTime
                paramors(i).behavior.batteryLife = behaviorMap(k).batteryLife
                paramors(i).behavior.hopsPerLocale = behaviorMap(k).            ↙
    hopsPerLocale
                paramors(i).behavior.powerThreshold = behaviorMap(k).           ↙
    powerThreshold
                paramors(i).behavior.stopTime = behaviorMap(k).stopTime
                notfound = False
                Exit For
            Else
                notfound = True
            End If
        Next
        If notfound = True Then
            behaviorsAssigned = False
            Return False
        End If
    Next
    behaviorsAssigned = True
    Return True
    End Function

End Class
```

Appendix D

Location System Code

```
'FILE: frmPmrLocBase.vb
'CLASS: frmPmrLocBase
'DESCRIPTION: Main code for the location system, connects to beacons,
'                executes locations system
'AUTHOR: Mat Laibowitz

Imports System
Imports System.Net
Imports System.Net.Sockets
Imports System.Text
Imports System.Xml
Imports System.Threading


Public Class frmPmrLocBase
    Inherits System.Windows.Forms.Form
    Public WithEvents mySocketsClients As New SocketsClientArray
    Private Const BEACON_PORT = 10001
    Private Const MAX_INQUIRIES = 15
    Private Const MAP_SQUARE_WIDTH = 40
    Private Const MAP_SQUARE_HEIGHT = 40
    Public mtx As New System.Threading.Mutex(False, "MC")
    Private Declare Function GetTickCount Lib "kernel32.dll" () As Long
    Private Declare Sub Sleep Lib "kernel32.dll" (ByVal dwMilliseconds As Long)



    Public Structure BTNode
        Dim BTAddress As String
        Dim BTName As String
        Dim BTCOD As String
        Dim BeaconsInRange() As Boolean
    End Structure

    Public Structure Beacon
        Dim BTNodes() As Integer
        Dim Inquiring As Boolean
        Dim inquiryStr As String
        Dim XPos As Integer
        Dim YPos As Integer
    End Structure

    Friend myBTNodes() As BTNode
    Friend myBeacons() As Beacon
    Private curBeacon As Integer = 0
    Private curBTNode As Integer = 0
    Private connecting As Boolean = False
    Private dialing As Boolean = False
    Private carrier As Boolean = False
    Private hooking As Boolean = False
    Private waiting4OK As Boolean = False
    Private dialingNode As Integer
    Private curConnection As Integer = 0
    Private disconnecting As Boolean = False
    Private curDisconnection As Integer = 0
    Dim DataArriveLock As Boolean
    Dim logWriter As XmlTextWriter
    Dim logFileName As String
    Dim loggingActive As Boolean = False
    Dim mapFileName As String
    Dim frmNode As frmNodeVis = New frmNodeVis
    Dim vis As Boolean = False
    Dim visNode As Integer
    Private tmpInStr As String = ""
```

```
    Dim myPBArray As pbArray


Windows Form Designer generated code

    Private Sub mySocketsClients_onConnect(ByVal Index As Integer) Handles          ↙
    mySocketsClients.onConnect
        AT("Connected to Beacon " & Index & ".")
        If connecting = True Then
            curConnection += 1
            If curConnection = lbIPAddr.Items.Count Then
                grpBeacon.Enabled = True
                connecting = False
                ReDim myBeacons(lbIPAddr.Items.Count - 1)
                cmbBSelect.SelectedIndex = 0
            Else
                AT("Attempting connection to Beacon " & curConnection & " ...")
                mySocketsClients.AddNewSocketsClient()
                Dim remoteEP As New IPEndPoint(IPAddress.Parse(lbIPAddr.Items.Item( ↙
    curConnection)), BEACON_PORT)
                mySocketsClients(curConnection).Connect(remoteEP)
                cmbBSelect.Items.Add(curConnection)
                If loggingActive = True Then
                    logWriter.WriteStartElement("ConnectionAttempt")
                    logWriter.WriteAttributeString("beacon", curConnection)
                    logWriter.WriteAttributeString("time", Environment.TickCount)
                    logWriter.WriteEndElement()
                End If
            End If
        End If
        If loggingActive = True Then
            logWriter.WriteStartElement("Connect")
            logWriter.WriteAttributeString("beacon", Index)
            logWriter.WriteAttributeString("time", Environment.TickCount)
            logWriter.WriteEndElement()
        End If
    End Sub

    Private Sub mySocketsClients_onError(ByVal Index As Integer, ByVal Description ↙
    As String) Handles mySocketsClients.onError
        AT("Error from Beacon " & Index & ": " & Description)
        If connecting = True Then
            curConnection += 1
            If curConnection = lbIPAddr.Items.Count Then
                grpBeacon.Enabled = True
                connecting = False
                ReDim myBeacons(lbIPAddr.Items.Count - 1)
            Else
                AT("Attempting connection to Beacon " & curConnection & " ...")
                mySocketsClients.AddNewSocketsClient()
                Dim remoteEP As New IPEndPoint(IPAddress.Parse(lbIPAddr.Items.Item( ↙
    curConnection)), BEACON_PORT)
                mySocketsClients(curConnection).Connect(remoteEP)
                cmbBSelect.Items.Add(curConnection)
            End If
        End If
        If loggingActive = True Then
            logWriter.WriteStartElement("Error")
            logWriter.WriteAttributeString("beacon", Index)
            logWriter.WriteAttributeString("time", Environment.TickCount)
            logWriter.WriteString(Description)
            logWriter.WriteEndElement()
        End If
    End Sub
```

```
Private Sub mySocketsClients_onDisconnect(ByVal Index As Integer) Handles         ↙
mySocketsClients.onDisconnect
    AT("Disconnected from Beacon " & Index & ".")
    If disconnecting = True Then
        curDisconnection += 1
        If curDisconnection = lbIPAddr.Items.Count Then
            disconnecting = False
            mySocketsClients.Clear()
            cmbBSelect.Items.Clear()
            grpBeacon.Enabled = False
        Else
            mySocketsClients(curDisconnection).Disconnect()
        End If
    End If
    If loggingActive = True Then
        logWriter.WriteStartElement("Disconnect")
        logWriter.WriteAttributeString("beacon", Index)
        logWriter.WriteAttributeString("time", Environment.TickCount)
        logWriter.WriteEndElement()
    End If
End Sub

Private Sub mySocketsClients onDataArrival(ByVal Index As Integer, ByVal Data() ↙
 As Byte, ByVal totBytes As Integer) Handles mySocketsClients.onDataArrival
    mtx.WaitOne()
    Dim inData As String = Nothing '= mySocketsClients(Index).BytestoString(      ↙
Data)
    inData = System.Text.ASCIIEncoding.ASCII.GetString(Data)
    AT("Received from Beacon " & Index & ": " & inData)
    If Index = cmbBSelect.SelectedIndex Then
        ATB(inData)
    End If
    If myBeacons(Index).Inquiring = True Then
        Dim tmpData As String = StripControlChars(inData)
        myBeacons(Index).inquiryStr = myBeacons(Index).inquiryStr & tmpData
        If myBeacons(Index).inquiryStr.IndexOf("DONE") >= 0 Then
            If Index = cmbBSelect.SelectedIndex Then lbNodes.Items.Clear()
            If Not myBTNodes Is Nothing Then
                top = myBTNodes.GetLength(0)
                Dim k As Integer
                For k = 0 To myBTNodes.GetLength(0) - 1
                    myBTNodes(k).BeaconsInRange(Index) = False
                Next
            End If
            Erase myBeacons(Index).BTNodes
            Dim splitit = Split(myBeacons(Index).inquiryStr, "***", -1,               ↙
CompareMethod.Binary)
            Dim i As Integer
            For i = 0 To UBound(splitit)
                If splitit(i).IndexOf("PARAMOR") >= 0 Then
                    'CStr(splitit(i)).Trim(ControlChars.CrLf)
                    Dim tmpBTN As String
                    Dim tmpBTA As String
                    Dim badData As Boolean = False
                    Dim newsplit = Split(splitit(i), ",")
                    If UBound(newsplit) = 2 Then
                        tmpBTN = newsplit(2)
                        tmpBTA = newsplit(0)
                    ElseIf UBound(newsplit) = 1 Then
                        tmpBTN = newsplit(1)
                        If newsplit(0) <> "00000000" Then
                            If newsplit(0).IndexOf("00A09608") >= 0 Then
                                tmpBTA = newsplit(0)
                            Else
                                badData = True
                            End If
                            'ElseIf i + 1 <= UBound(splitit) Then
                            '    If splitit(i + 1).Length = 13 Then
                            '        tmpBTA = Mid(splitit(i + 1), 1, 12)
                            '    Else
                            '        badData = True
                            '    End If
                        Else
                            badData = True
                        End If
                    Else
                        badData = True
                    End If
                    If badData = False Then
                        If Index = cmbBSelect.SelectedIndex Then lbNodes.Items. ↙
Add(tmpBTN & " " & tmpBTA)
                        Dim notfound As Boolean = True
                        Dim j, btind As Integer
                        Dim top As Integer = 0
                        If Not myBTNodes Is Nothing Then
                            top = myBTNodes.GetLength(0)
                            For j = 0 To myBTNodes.GetLength(0) - 1
                                If tmpBTA = myBTNodes(j).BTAddress Or tmpBTN = ↙
myBTNodes(j).BTName Then
                                    notfound = False
                                    btind = j
                                End If
                            Next
                        End If
                        If notfound = True Then
                            ReDim Preserve myBTNodes(top)
                            ReDim Preserve myBTNodes(top).BeaconsInRange(UBound ↙
(myBeacons))

                            myBTNodes(top).BTAddress = tmpBTA
                            myBTNodes(top).BTName = tmpBTN
                            btind = top
                        End If

                        'Dim ind = AddNewElement(myBTNodes(btind).              ↙
BeaconsInRange, Index)
                        myBTNodes(btind).BeaconsInRange(Index) = True
                        Dim ind = AddNewElement(myBeacons(Index).BTNodes, btind ↙
)
                    End If
                End If
            Next
            myBeacons(Index).inquiryStr = ""
            UpdateNodeList()
            myBeacons(Index).Inquiring = False
        End If
    ElseIf dialing = True Then
        'Dim conData As String = StripControlChars(inData)
        myBeacons(Index).inquiryStr = myBeacons(Index).inquiryStr & inData
        If myBeacons(Index).inquiryStr.IndexOf("CONNECT") >= 0 Then
            myBeacons(Index).inquiryStr = ""
            carrier = True
            dialing = False
        ElseIf myBeacons(Index).inquiryStr.IndexOf("NO ANSWER") >= 0 Then
            myBeacons(Index).inquiryStr = ""
            carrier = False
            dialing = False
        End If
    ElseIf waiting4OK = True Then
        myBeacons(Index).inquiryStr = myBeacons(Index).inquiryStr & inData
        If myBeacons(Index).inquiryStr.IndexOf("OK") >= 0 Then
```

```
            waiting4OK = False
        End If
    ElseIf hooking = True Then
        myBeacons(Index).inquiryStr = myBeacons(Index).inquiryStr & inData
        If myBeacons(Index).inquiryStr.IndexOf("NO CARRIER") >= 0 Then
            waiting4OK = False
        End If
    End If
    If vis = True Then
        If myBTNodes(visNode).BeaconsInRange(Index) = True Then
            'inData = inData.Trim(ControlChars.CrLf)
            'frmNode.pUICommEventUpdate(inData)
        End If
    End If
    If loggingActive = True Then
        logWriter.WriteStartElement("DataArrival")
        logWriter.WriteAttributeString("time", Environment.TickCount)
        logWriter.WriteString(inData)
        logWriter.WriteEndElement()
    End If
    mtx.ReleaseMutex()
End Sub


Private Sub mySocketsClients_onSendComplete(ByVal Index As Integer, ByVal        ↙
dataSize As Integer) Handles mySocketsClients.onSendComplete
    AT("Sent " & dataSize & " Bytes to Beacon " & Index & ".")
    If Index = cmbBSelect.SelectedIndex Then
        ATB("Sent " & dataSize & " Bytes.")
    End If
    If loggingActive = True Then
        logWriter.WriteStartElement("SendComplete")
        logWriter.WriteAttributeString("beacon", Index)
        logWriter.WriteAttributeString("time", Environment.TickCount)
        logWriter.WriteString(dataSize)
        logWriter.WriteEndElement()
    End If

End Sub

Private Function AddNewElement(ByRef inArray() As Integer, ByVal element As     ↙
Integer) As Integer
    If inArray Is Nothing Then
        ReDim inArray(0)
        inArray(0) = element
    Else
        Dim i As Integer
        For i = 0 To UBound(inArray)
            If inArray(i) = element Then
                Return i
            End If
        Next
        ReDim Preserve inArray(UBound(inArray) + 1)
        inArray(UBound(inArray)) = element
        Return UBound(inArray)
    End If
End Function

Private Function StripControlChars(ByVal source As String) As String
    Dim index As Long
    Dim bytes = System.Text.ASCIIEncoding.ASCII.GetBytes(source)
    For index = 0 To UBound(bytes)
        ' if this is a control character
        If bytes(index) < 32 Then
            If bytes(index) = 13 Then
                bytes(index) = 42
            ElseIf bytes(index) = 10 Then
                bytes(index) = 42
            Else
                bytes(index) = 0
            End If
        End If
    Next

    ' return this string, after filtering out all null chars
    System.Text.ASCIIEncoding.ASCII.GetString(bytes)
    StripControlChars = Replace(System.Text.ASCIIEncoding.ASCII.GetString(bytes ↙
), vbNullChar, "")


End Function

Private Sub AT(ByVal Text As String)
    txtAll.AppendText(Text & vbCrLf)
    txtAll.ScrollToCaret()
End Sub

Private Sub ATB(ByVal Text As String)
    txtBeacon.AppendText(Text & vbCrLf)
    txtBeacon.ScrollToCaret()
End Sub

Private Sub txtSend_KeyPress(ByVal sender As System.Object, ByVal e As System.  ↙
Windows.Forms.KeyPressEventArgs) Handles txtSend.KeyPress
    If Asc(e.KeyChar) = 13 Then
        sendData(txtSend.Text, cmbBSelect.SelectedIndex)
        txtSend.SelectAll()
    End If
End Sub

Private Sub cmdAddIP_Click(ByVal sender As System.Object, ByVal e As System.    ↙
EventArgs) Handles cmdAddIP.Click
    If txtAddIP.Text <> "" Then
        Try
            IPAddress.Parse(txtAddIP.Text)
            lbIPAddr.Items.Add(txtAddIP.Text)
        Catch
            MsgBox("IP Address is Invalid")
        End Try
    End If
End Sub

Private Sub cmdRemoveIP_Click(ByVal sender As System.Object, ByVal e As System. ↙
EventArgs) Handles cmdRemoveIP.Click
    lbIPAddr.Items.RemoveAt(lbIPAddr.SelectedIndex)
End Sub

Private Sub cmdSaveIP_Click(ByVal sender As System.Object, ByVal e As System.   ↙
EventArgs) Handles cmdSaveIP.Click
    SaveFileDialog1.ShowDialog()
    If SaveFileDialog1.FileName <> "" Then
        Dim textWriter As XmlTextWriter = New XmlTextWriter(SaveFileDialog1.    ↙
FileName, Nothing)
        Dim i, j, k As Integer
        textWriter.Formatting = Formatting.Indented
        textWriter.WriteStartDocument()
        textWriter.WriteComment("Paramor Location System IP File")
        Dim str As String = "Filename: " & SaveFileDialog1.FileName
        textWriter.WriteComment(str)
        str = "Created: " & System.DateTime.Now
```

```
            textWriter.WriteComment(str)
            textWriter.WriteStartElement("BeaconAddresses")
            For i = 0 To lbIPAddr.Items.Count - 1
                textWriter.WriteStartElement("IP")
                textWriter.WriteAttributeString("index", i)
                textWriter.WriteString(lbIPAddr.Items.Item(i))
                textWriter.WriteEndElement()
            Next
            textWriter.WriteEndElement()
            textWriter.WriteEndDocument()
            textWriter.Close()
        End If
    End Sub

    Private Sub cmdLoadIP_Click(ByVal sender As System.Object, ByVal e As System.  ↙
EventArgs) Handles cmdLoadIP.Click
        OpenFileDialog1.ShowDialog()
        If OpenFileDialog1.FileName <> "" Then
            lbIPAddr.Items.Clear()
            Dim i, j, k As Integer
            Dim m_nodelist As XmlNodeList
            Dim m_node As XmlNode
            Dim m_xmld As XmlDocument
            m_xmld = New XmlDocument
            'Load the Xml file
            m_xmld.Load(OpenFileDialog1.FileName)
            m_nodelist = m_xmld.SelectNodes("/BeaconAddresses/IP")
            For Each m_node In m_nodelist
                lbIPAddr.Items.Add(m_node.InnerText)
            Next
        End If
    End Sub

    Private Sub cmdConnect_Click(ByVal sender As System.Object, ByVal e As System.  ↙
EventArgs) Handles cmdConnect.Click
        connecting = True
        curConnection = 0
        '      grpBeacon.Enabled = True
        AT("Attempting connection to Beacon " & curConnection & " ...")
        mySocketsClients.AddNewSocketsClient()
        Dim remoteEP As New IPEndPoint(IPAddress.Parse(lbIPAddr.Items.Item(      ↙
curConnection)), BEACON_PORT)
        mySocketsClients(curConnection).Connect(remoteEP)
        cmdDisconnect.Enabled = True
        cmdConnect.Enabled = False
        cmbBSelect.Items.Add(curConnection)
        If loggingActive = True Then
            logWriter.WriteStartElement("ConnectionAttempt")
            logWriter.WriteAttributeString("beacon", curConnection)
            logWriter.WriteAttributeString("time", Environment.TickCount)
            logWriter.WriteEndElement()
        End If
        Do While connecting = True
            Application.DoEvents()
        Loop
        DrawMap(True)
    End Sub

    Private Sub cmdDisconnect_Click(ByVal sender As System.Object, ByVal e As      ↙
System.EventArgs) Handles cmdDisconnect.Click
        disconnecting = True
        curDisconnection = 0
        mySocketsClients(curDisconnection).Disconnect()
        cmdDisconnect.Enabled = False
        cmdConnect.Enabled = True
```

```
    End Sub

    Private Sub frmPmrLocBase_Load(ByVal sender As System.Object, ByVal e As System ↙
.EventArgs) Handles MyBase.Load
        cmbBSelect.DropDownStyle = ComboBoxStyle.DropDownList
    End Sub

    Public Function sendData(ByVal data As String, ByVal Socket As Integer)
        AT("Sending to Beacon " & Socket & ": " & data)
        If Socket = cmbBSelect.SelectedIndex Then ATB("Sending: " & data)
        mySocketsClients(Socket).SendData(mySocketsClients(Socket).StringToBytes(   ↙
data & vbCrLf))
        If loggingActive = True Then
            logWriter.WriteStartElement("SendData")
            logWriter.WriteAttributeString("beacon", Socket)
            logWriter.WriteAttributeString("time", Environment.TickCount)
            logWriter.WriteString(data)
            logWriter.WriteEndElement()
        End If
    End Function

    Private Function UpdateUI(ByVal Index As Integer)
        txtBeacon.Clear()
        lbNodes.Items.Clear()
        If mySocketsClients(Index).Connected = True Then
            cmdBConnect.Enabled = False
            cmdBDisConnect.Enabled = True
        Else
            cmdBConnect.Enabled = True
            cmdBDisConnect.Enabled = False
        End If
        Dim i As Integer
        If Not myBeacons(Index).BTNodes Is Nothing Then
            For i = 0 To myBeacons(Index).BTNodes.GetLength(0) - 1
                lbNodes.Items.Add(myBTNodes(myBeacons(Index).BTNodes(i)).BTName & " ↙
 " & myBTNodes(myBeacons(Index).BTNodes(i)).BTAddress)
            Next
        End If
        cmbBSelect.SelectedIndex = Index
    End Function

    Private Function UpdateNodeList()
        Dim i As Integer
        If Not myBTNodes Is Nothing Then
            lbGNodes.Items.Clear()
            For i = 0 To myBTNodes.GetLength(0) - 1
                lbGNodes.Items.Add(myBTNodes(i).BTName & " " & myBTNodes(i).       ↙
BTAddress)
            Next
            If myBTNodes.GetLength(0) > 0 Then
                cmdControl.Enabled = True
            End If
        End If
    End Function

    Private Sub cmbBSelect_SelectedIndexChanged(ByVal sender As System.Object,     ↙
ByVal e As System.EventArgs) Handles cmbBSelect.SelectedIndexChanged
        UpdateUI(cmbBSelect.SelectedIndex)
    End Sub

    Private Sub cmbBConnect_Click(ByVal sender As System.Object, ByVal e As System. ↙
EventArgs) Handles cmdBConnect.Click
        Dim remoteEP2 As New IPEndPoint(mySocketsClients(cmbBSelect.SelectedIndex). ↙
ipAddress, mySocketsClients(cmbBSelect.SelectedIndex).port)
        mySocketsClients(cmbBSelect.SelectedIndex).Connect(remoteEP2)
    UpdateUI(cmbBSelect.SelectedIndex)
End Sub

Private Sub cmdBDisConnect_Click(ByVal sender As System.Object, ByVal e As     ↙
System.EventArgs) Handles cmdBDisConnect.Click
    mySocketsClients(cmbBSelect.SelectedIndex).Disconnect()
    UpdateUI(cmbBSelect.SelectedIndex)
End Sub

Private Sub cmdBClear_Click(ByVal sender As System.Object, ByVal e As System.    ↙
EventArgs) Handles cmdBClear.Click
    sendData("ATUCL", cmbBSelect.SelectedIndex)
End Sub

Private Sub cmdDS_Click(ByVal sender As System.Object, ByVal e As System.        ↙
EventArgs) Handles cmdDS.Click
    sendData("ATDS", cmbBSelect.SelectedIndex)
End Sub

Private Sub cmdInquiry_Click(ByVal sender As System.Object, ByVal e As System.  ↙
EventArgs) Handles cmdInquiry.Click
    myBeacons(cmbBSelect.SelectedIndex).Inquiring = True
    myBeacons(cmbBSelect.SelectedIndex).inquiryStr = ""
    sendData("ATDI," & MAX_INQUIRIES & ",00000000", cmbBSelect.SelectedIndex)
End Sub

Private Sub cmdCmd_Click(ByVal sender As System.Object, ByVal e As System.       ↙
EventArgs) Handles cmdCmd.Click
    sendData("ATMC", cmbBSelect.SelectedIndex)
End Sub

Private Sub cmdHangUp_Click(ByVal sender As System.Object, ByVal e As System.    ↙
EventArgs) Handles cmdHangUp.Click
    sendData("ATMC", cmbBSelect.SelectedIndex)
    sendData("ATDH", cmbBSelect.SelectedIndex)
End Sub

Private Sub cmdGetName_Click(ByVal sender As System.Object, ByVal e As System.  ↙
EventArgs) Handles cmdGetName.Click
    sendData("ATSI,2", cmbBSelect.SelectedIndex)
End Sub

Private Sub cmdSetName_Click(ByVal sender As System.Object, ByVal e As System.  ↙
EventArgs) Handles cmdSetName.Click
    sendData("ATSN," & txtSend.Text, cmbBSelect.SelectedIndex)
    txtSend.Clear()
End Sub

Private Sub cmdDMNode_Click(ByVal sender As System.Object, ByVal e As System.    ↙
EventArgs) Handles cmdDMNode.Click
    Dim btStr As String = myBTNodes(myBeacons(cmbBSelect.SelectedIndex).BTNodes ↙
(lbNodes.SelectedIndex)).BTAddress
    sendData("ATDM," & btStr & ",1101", cmbBSelect.SelectedIndex)
End Sub

Private Sub cmdLogFile_Click(ByVal sender As System.Object, ByVal e As System.  ↙
EventArgs) Handles cmdLogFile.Click
    SaveFileDialog1.ShowDialog()
    If SaveFileDialog1.FileName <> "" Then
        logFileName = SaveFileDialog1.FileName
        cmdLog.Enabled = True
    End If
End Sub

Private Sub cmdLog_Click(ByVal sender As System.Object, ByVal e As System.       ↙
EventArgs) Handles cmdLog.Click
    If cmdLog.Text = "Begin Logging" Then
        loggingActive = True
        cmdLog.Text = "Stop Logging"
        logWriter = New XmlTextWriter(logFileName, Nothing)
        logWriter.Formatting = Formatting.Indented
        logWriter.WriteStartDocument()
        logWriter.WriteComment("Paramor Location System Log File")
        Dim str As String = "Filename: " & SaveFileDialog1.FileName
        logWriter.WriteComment(str)
        str = "Created: " & System.DateTime.Now
        logWriter.WriteComment(str)
        logWriter.WriteStartElement("Log")
        logWriter.WriteAttributeString("TimeStarted", Environment.TickCount)
    Else
        loggingActive = False
        cmdLog.Text = "Begin Logging"
        logWriter.WriteEndElement()
        logWriter.WriteEndDocument()
        logWriter.Close()
    End If
End Sub

Private Sub cmdControl_Click(ByVal sender As System.Object, ByVal e As System.  ↙
EventArgs) Handles cmdControl.Click
    If lbGNodes.SelectedIndex >= 0 Then
        Dim btStr As String = myBTNodes(lbGNodes.SelectedIndex).BTAddress
        Dim i As Integer
        Dim btBeac As Integer
        For i = 0 To myBTNodes(lbGNodes.SelectedIndex).BeaconsInRange.GetLength ↙
(0) - 1
            If myBTNodes(lbGNodes.SelectedIndex).BeaconsInRange(i) = True Then
                btBeac = i
                Exit For
            End If
        Next
        sendData("ATDM," & btStr & ",1101", btBeac)
        visNode = lbGNodes.SelectedIndex
        vis = True
        frmNode = New frmNodeVis
        frmNode.Execute(Me, btBeac)
        vis = False
        sendData("ATMC", btBeac)
        sendData("ATDH", btBeac)
    End If
End Sub


Private Sub cmdDrawMap_Click(ByVal sender As System.Object, ByVal e As System.  ↙
EventArgs) Handles cmdDrawMap.Click
    DrawMap(True)
End Sub

Private Sub DrawMap(ByVal NewMap As Boolean)
    Dim i As Integer
    myPBArray.ClearPBArray()
    For i = 0 To myBeacons.Length - 1
        If NewMap = True Then
            myBeacons(i).XPos = (i * MAP_SQUARE_WIDTH) Mod pnlMap.Width
            myBeacons(i).YPos = (i / pnlMap.Width) * MAP_SQUARE_HEIGHT
        End If
        myPBArray.AddNewPB(myBeacons(i).XPos, myBeacons(i).YPos,                 ↙
MAP_SQUARE_HEIGHT, MAP_SQUARE_WIDTH)
        myPBArray(i).BackColor = Color.DeepSkyBlue
    Next
```

```
End Sub

Private Sub cmdExit_Click(ByVal sender As System.Object, ByVal e As System.    ↙
EventArgs) Handles cmdExit.Click
    End
End Sub

Private Sub cmdLoadMap_Click(ByVal sender As System.Object, ByVal e As System.  ↙
EventArgs) Handles cmdLoadMap.Click
    OpenFileDialog1.ShowDialog()
    If OpenFileDialog1.FileName <> "" Then
        mapFileName = OpenFileDialog1.FileName
        Dim m_nodelist As XmlNodeList
        Dim m_node As XmlNode
        Dim m_xmld As XmlDocument
        m_xmld = New XmlDocument
        'Load the Xml file
        m_xmld.Load(OpenFileDialog1.FileName)
        m_nodelist = m_xmld.SelectNodes("/Beacons/Location")
        For Each m_node In m_nodelist
            Dim index As Integer = m_node.Attributes.GetNamedItem("index").    ↙
Value
            myBeacons(index).XPos = m_node.Attributes.GetNamedItem("X").Value
            myBeacons(index).YPos = m_node.Attributes.GetNamedItem("Y").Value
        Next
        DrawMap(False)
    End If
End Sub

Private Sub cmdSaveMap_Click(ByVal sender As System.Object, ByVal e As System.  ↙
EventArgs) Handles cmdSaveMap.Click
    SaveFileDialog1.ShowDialog()
    If SaveFileDialog1.FileName <> "" Then
        Dim textWriter As XmlTextWriter = New XmlTextWriter(SaveFileDialog1.   ↙
FileName, Nothing)
        Dim i, j, k As Integer
        textWriter.Formatting = Formatting.Indented
        textWriter.WriteStartDocument()
        textWriter.WriteComment("Paramor Location System Map File")
        Dim str As String = "Filename: " & SaveFileDialog1.FileName
        textWriter.WriteComment(str)
        str = "Created: " & System.DateTime.Now
        textWriter.WriteComment(str)
        textWriter.WriteStartElement("Beacons")
        For i = 0 To myBeacons.Length - 1
            myBeacons(i).XPos = myPBArray(i).Location.X
            myBeacons(i).YPos = myPBArray(i).Location.Y
            textWriter.WriteStartElement("Location")
            textWriter.WriteAttributeString("index", i)
            textWriter.WriteAttributeString("X", myBeacons(i).XPos)
            textWriter.WriteAttributeString("Y", myBeacons(i).YPos)
            textWriter.WriteEndElement()
        Next
        textWriter.WriteEndElement()
        textWriter.WriteEndDocument()
        textWriter.Close()
    End If
End Sub

Private Sub lbGNodes_SelectedIndexChanged(ByVal sender As System.Object, ByVal  ↙
e As System.EventArgs) Handles lbGNodes.SelectedIndexChanged
    Dim i As Integer
    For i = 0 To myPBArray.Count - 1
        myPBArray(i).BackColor = Color.DeepSkyBlue
    Next
```

```
    For i = 0 To myBTNodes(lbGNodes.SelectedIndex).BeaconsInRange.Length - 1
        If myBTNodes(lbGNodes.SelectedIndex).BeaconsInRange(i) = True Then
            myPBArray(i).BackColor = Color.DarkRed
        End If
    Next
End Sub

Private Sub cmdGInquiry_Click(ByVal sender As System.Object, ByVal e As System. ↙
EventArgs) Handles cmdGInquiry.Click
    Dim i As Integer
    For i = 0 To UBound(myBeacons)
        If myBeacons(i).Inquiring = False Then
            myBeacons(i).Inquiring = True
            myBeacons(i).inquiryStr = ""
            sendData("ATDI," & MAX_INQUIRIES & ",00000000", i)
        End If
    Next
End Sub

Private Sub cmdRun_Click(ByVal sender As System.Object, ByVal e As System.      ↙
EventArgs) Handles cmdRun.Click
    If cmdRun.Text = "RUN" Then
        cmdRun.Text = "STOP"
        tmrLocRun.Enabled = True
    Else
        cmdRun.Text = "RUN"
        tmrLocRun.Enabled = False
    End If
End Sub

Private Sub tmrLocRun_Tick(ByVal sender As System.Object, ByVal e As System.    ↙
EventArgs) Handles tmrLocRun.Tick
    tmrLocRun.Enabled = False
    Dim elapsed As TimeSpan
    Static start_time As DateTime
    Static stop_time As DateTime
    Dim i As Integer
    For i = 0 To UBound(myBeacons)
        If myBeacons(i).Inquiring = False Then
            myBeacons(i).Inquiring = True
            myBeacons(i).inquiryStr = ""
            sendData("ATMC", i)
            sendData("ATDS", i)
            sendData("ATUCL", i)
            sendData("ATDI," & MAX_INQUIRIES & ",00000000", i)
        End If
    Next
    Dim inqDone As Boolean = False
    Do While inqDone = False
        For i = 0 To UBound(myBeacons)
            Application.DoEvents()
            If myBeacons(i).Inquiring = True Then
                inqDone = False
                Exit For
            Else
                inqDone = True
            End If
        Next
    Loop
    For i = 0 To UBound(myBTNodes)
        Dim tmpLoc As Long = 0
        Dim tmpBeac As Integer
        Dim j As Integer
        For j = 0 To UBound(myBTNodes(i).BeaconsInRange)
            If myBTNodes(i).BeaconsInRange(j) = True Then
```

```
                tmpLoc = tmpLoc + (2 ^ j)
                tmpBeac = j
            End If
        Next
        If tmpLoc > 0 Then
            Dim tmpBytes() As Byte
            tmpBytes = BitConverter.GetBytes(tmpLoc)
            waiting4OK = True
            sendData("ATMC", tmpBeac)
            start_time = Now
            Do While waiting4OK = True
                Application.DoEvents()
                stop_time = Now
                elapsed = stop_time.Subtract(start_time)
                If elapsed.TotalMilliseconds > 1000 Then
                    waiting4OK = False
                End If
            Loop
            hooking = True
            sendData("ATDH", tmpBeac)
            start_time = Now
            Do While hooking = True
                Application.DoEvents()
                stop_time = Now
                elapsed = stop_time.Subtract(start_time)
                If elapsed.TotalMilliseconds > 1000 Then
                    hooking = False
                End If
            Loop
            dialing = True
            carrier = False
            dialingNode = i
            sendData("ATDM," & myBTNodes(i).BTAddress & ",1101", tmpBeac)
            start_time = Now
            Do While dialing = True
                Application.DoEvents()
                stop_time = Now
                elapsed = stop_time.Subtract(start_time)
                If elapsed.TotalMilliseconds > 20000 Then
                    carrier = False
                    dialing = False
                End If
            Loop
            If carrier = True Then
                MyPause(1000)
                sendData("1C" & System.Text.ASCIIEncoding.ASCII.GetString(      ↙
tmpBytes), tmpBeac)
                MyPause(1000)
            End If
            waiting4OK = True
            sendData("ATMC", tmpBeac)
            start_time = Now
            Do While waiting4OK = True
                Application.DoEvents()
                stop_time = Now
                elapsed = stop_time.Subtract(start_time)
                If elapsed.TotalMilliseconds > 1000 Then
                    waiting4OK = False
                End If
            Loop
            hooking = True
            sendData("ATDH", tmpBeac)
            start_time = Now
            Do While hooking = True
                Application.DoEvents()
```

```
                        stop_time = Now
                        elapsed = stop time.Subtract(start time)
                        If elapsed.TotalMilliseconds > 1000 Then
                            hooking = False
                        End If
                    Loop
                End If
            Next
            If cmdRun.Text = "STOP" Then
                tmrLocRun.Enabled = True
            End If
        End Sub

        Public Sub MyPause(Optional ByVal ms As Long = 3000)
            On Error Resume Next
            Dim tc As Long
            tc = GetTickCount
            While GetTickCount < tc + ms : Sleep(1) : Application.DoEvents() : End      ↙
        While
        End Sub

    End Class
```

```
'FILE: frmNodeVis.vb
'CLASS: frmNodeVis
'DESCRIPTION: Control panel for controlling a single node
'AUTHOR: Mat Laibowitz

Imports Microsoft.DirectX
Imports Microsoft.DirectX.Direct3D
Imports Microsoft.DirectX.Direct3D.Geometry


Public Class frmNodeVis
    Inherits System.Windows.Forms.Form

    Private DXGrafx As DX9ToolsR5.DX9Graphics
    Private SphereMesh, SmallSphereMesh As Mesh
    Private DeviceLost As Boolean
    Private mintRotation As Single = 0
    Private TxtFont As Direct3D.Font
    Private Cam As DX9ToolsR5.Camera
    Private D3DMngr As Manager

    'misc variables
    Private bInitOkay As Boolean = False
    Private iLastFPSCheck As Int32
    Private Const iFPSProfileSpeed As Integer = 200 'ms
    Private iCurrCnt As Integer
    Private iFrameRate As Integer
    Private sDevInfo As String
    Private sDispInfo As String
    Private ToggleInfo As Boolean

    Protected appTime As Single ' Current time in seconds
    Protected elapsedTime As Single ' Time elapsed since last frame
    Protected framePerSecond As Single ' Instanteous frame rate
    Protected deviceStats As String ' String to hold D3D device stats
    Protected frameStats As String ' String to hold frame stats
    Private lastTime As Single = 0.0F ' The last time
    Private frames As Integer = 0 ' Number of rames since our last update
    Private lastMousePos As Point
    Private mouseClickedL As Boolean
    Private mouseClickedR As Boolean

    Private mlTicks As Long
    Private HeaderPos As Integer = 0
    Private SensorPacket(11) As Integer
    Private DataPacket(6) As Integer
    Private DataFilter(6, 5) As Integer
    Private LPIndex As Integer = 0
    Private CurIndex As Integer = 0
    Private myLight As Integer = 20
    Private myX As Single = 1.0
    Private myY As Single = 1.0
    Private running As Boolean
    Private pForm As Form
    Private pSocket As Integer



    'Public Shared Sub Main()
    '    Dim m As New frmParasimRun
    '    Application.Exit()
    'End Sub


    Public Sub New()
```

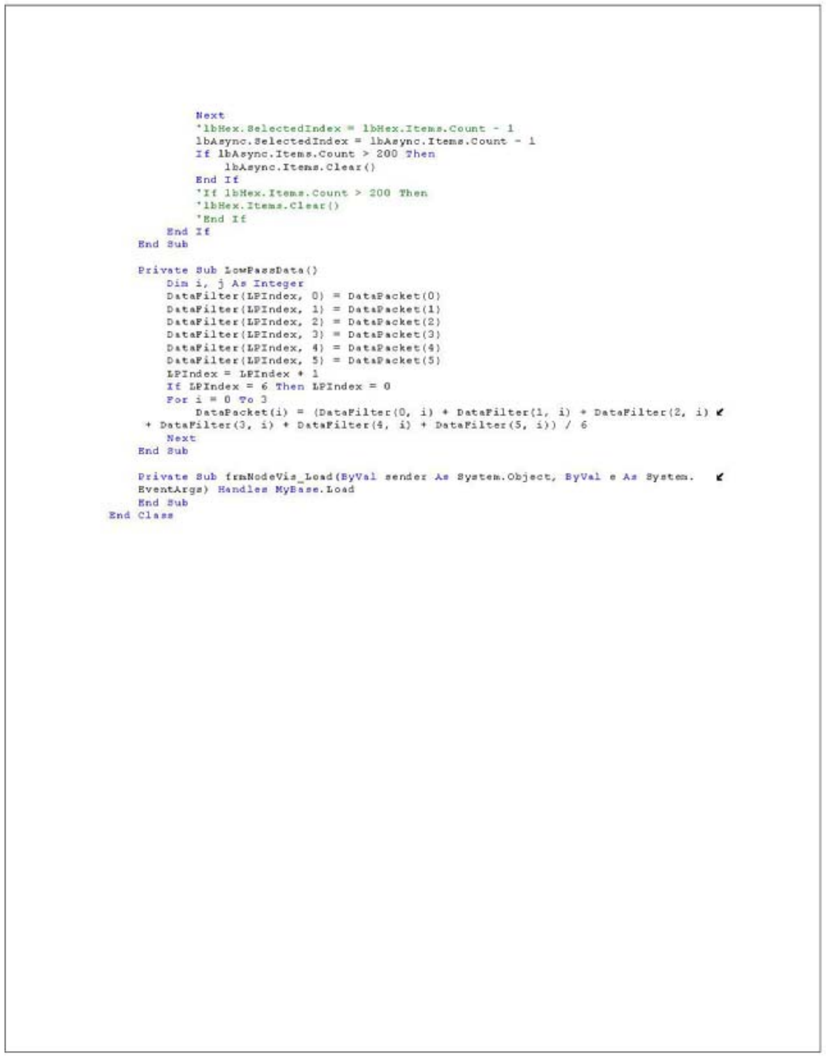
```
    MyBase.New()
End Sub

Public Sub Execute(ByRef parentForm As Form, ByVal nodesocket As Integer)
    InitializeComponent()
    ToggleInfo = True
    If InitGraphics() Then
        Cam.SetPosition(DX9ToolsR5.SetVector.Z, -30)
        Cam.AspectRatio = CSng(pbRender.ClientSize.Width / pbRender.ClientSize. ↙
Height)
        SphereMesh = Mesh.Sphere(DXGrafx.Device, 10, 40, 40)
    End If
    ' Application.AddMessageFilter(Me)
    AddHandler Me.KeyPress, AddressOf KeyPressed

    mouseClickedL = False
    mouseClickedR = False
    DXUtil.Timer(DirectXTimer.Start)
    pForm = parentForm
    pSocket = nodesocket
    Show()
    Focus()
    pbRender.Focus()
    MainLoop()
End Sub

Private Sub MainLoop()
    running = True
    Do While (Created = True And running = True)
        FrameUpdate()
        FrameRender()
        Application.DoEvents()
    Loop
End Sub

Private Sub FrameUpdate()


    'calculate the current framerate
    Dim time As Single = DXUtil.Timer(DirectXTimer.GetAbsoluteTime)
    frames += 1

    ' Update the scene stats once per second
    If time - lastTime > 1.0F Then
        framePerSecond = frames / (time - lastTime)
        lastTime = time
        frames = 0
    End If

    If (Environment.TickCount() - iLastFPSCheck >= iFPSProfileSpeed) Then
        iLastFPSCheck = Environment.TickCount()
        iFrameRate = iCurrCnt * (1000 / iFPSProfileSpeed)
        iCurrCnt = 0
    End If
    iCurrCnt += 1
End Sub

Private Sub FrameRender()
    Dim Mat As Matrix = Matrix.Identity
    Dim NewMat As Matrix
    Dim mtrl3d As Material = Nothing
    Dim i, j As Integer

    DXGrafx.Clear(Color.DarkGreen, 1.0F, 0)
    DXGrafx.Device.BeginScene()
    InitLights()

    mtrl3d.Ambient = Color.FromArgb(128, 255, 0, 255)
    mtrl3d.Diffuse = Color.FromArgb(128, 250, 0, 250)
    mtrl3d.Specular = Color.White
    mtrl3d.SpecularSharpness = 100
    DXGrafx.Device.Material = mtrl3d
    Mat = Matrix.Multiply(Mat, Matrix.Translation(myX, myY, 1.0))
    DXGrafx.Device.SetTransform(Direct3D.TransformType.World, Mat)
    SphereMesh.DrawSubset(0)

    If ToggleInfo = True Then
        RenderInfo()
    End If

    DXGrafx.Device.EndScene()

    If DeviceLost Then
        Try
            ' Test the cooperative level to see if it's okay to render
            DXGrafx.Device.TestCooperativeLevel()
        Catch e As Direct3D.DeviceLostException
            ' If the device was lost, do not render until we get it back
            Exit Sub
        Catch e As Direct3D.DeviceNotResetException
            ' Reset the device and resize it
            DXGrafx.Device.Reset(DXGrafx.Device.PresentationParameters)
        End Try

        DeviceLost = False
    End If

    Try
        DXGrafx.Present()
    Catch e As Direct3D.DeviceLostException
        DeviceLost = True
    End Try

End Sub

Private Sub RenderInfo()
    Dim Msg As String = ""
    Msg &= "Device: " + sDevInfo & vbCrLf
    Msg &= "FPS: " & framePerSecond & vbCrLf    'iFrameRate.ToString & vbCrLf
    Msg &= "Lighting: " & DXGrafx.Device.RenderState.Lighting.ToString & vbCrLf
    Msg &= "FillMode: " & DXGrafx.Device.RenderState.FillMode.ToString & vbCrLf
    Msg &= "ShadeMode: " & DXGrafx.Device.RenderState.ShadeMode.ToString &      ↙
vbCrLf
    Msg &= "CullMode: " & DXGrafx.Device.RenderState.CullMode.ToString & vbCrLf ↙
 & vbCrLf

    TxtFont.DrawText(Msg, New Rectangle(10, 10, Me.ClientSize.Width - 10, Me.   ↙
ClientSize.Height - 10), _
                      Direct3D.DrawTextFormat.Left Or Direct3D.DrawTextFormat.  ↙
Top, _
                      Color.Red)
End Sub

Private Sub InitLights()
    Dim Light4 As Direct3D.Light = DXGrafx.Device.Lights(0)


    Light4.Type = Direct3D.LightType.Point
    Light4.Direction = New Vector3(0, -1, 0)
```

```
    'Light4.Position = New Vector3(1, myLight, 1)
    Light4.Position = New Vector3(myX, myLight + myY, 1)
    Light4.Range = 100
    Light4.Diffuse = Color.White
    Light4.Ambient = Color.Red
    Light4.InnerConeAngle = 0.0F
    Light4.OuterConeAngle = 1.0F
    Light4.Falloff = 1.0F
    Light4.Attenuation0 = 1.0F
    Light4.Enabled = True
    Light4.Commit()
End Sub

Private Function InitGraphics() As Boolean
    Try
        Dim d3dPP As New PresentParameters
        DXGrafx = New DX9ToolsR5.DX9Graphics(pbRender)

        Cam = New DX9ToolsR5.Camera(DXGrafx)
        With DXGrafx.Device
            .RenderState.CullMode = Direct3D.Cull.CounterClockwise
            .RenderState.FillMode = Direct3D.FillMode.Solid
            .RenderState.ShadeMode = Direct3D.ShadeMode.Gouraud
            .RenderState.Lighting = True
            .RenderState.ZBufferEnable = True

            .RenderState.SourceBlend = Blend.SourceAlpha
            .RenderState.DestinationBlend = Blend.InvSourceAlpha

            'set up texture blending
            .SamplerState(0).MinFilter = TextureFilter.Linear
            .SamplerState(0).MagFilter = TextureFilter.Linear

            ' Blend a Texture with Vertex Color
            .TextureState(0).ColorOperation = Direct3D.TextureOperation.        ↙
Modulate
            .TextureState(0).ColorArgument1 = Direct3D.TextureArgument.         ↙
TextureColor
            .TextureState(0).ColorArgument2 = Direct3D.TextureArgument.Diffuse
            .TextureState(0).AlphaOperation = Direct3D.TextureOperation.Disable

            TxtFont = New Direct3D.Font(DXGrafx.Device, Me.Font)
            sDevInfo = D3DMngr.Adapters(0).Information.Description
        End With
    Catch ex As Exception
        Return False
    End Try

    Return True
End Function

Private Function RotateView(ByVal angle As Single, ByVal x As Single, ByVal y   ↙
As Single, ByVal z As Single)
    Dim vView As Vector3
    Dim vNewView As Vector3
    vView.X = Cam.LookAt.X - Cam.Position.X
    vView.Y = Cam.LookAt.Y - Cam.Position.Y
    vView.Z = Cam.LookAt.Z - Cam.Position.Z
    Dim cosTheta As Single = Math.Cos(angle)
    Dim sinTheta As Single = Math.Sin(angle)
    vNewView.X = (cosTheta + (1 - cosTheta) * x * x) * vView.X
    vNewView.X += ((1 - cosTheta) * x * y - z * sinTheta) * vView.Y
    vNewView.X += ((1 - cosTheta) * x * z + y * sinTheta) * vView.Z
    vNewView.Y = ((1 - cosTheta) * x * y + z * sinTheta) * vView.X
    vNewView.Y += (cosTheta + (1 - cosTheta) * y * y) * vView.Y
    vNewView.Y += ((1 - cosTheta) * y * z - x * sinTheta) * vView.Z
    vNewView.Z = ((1 - cosTheta) * x * z - y * sinTheta) * vView.X
    vNewView.Z += ((1 - cosTheta) * y * z + x * sinTheta) * vView.Y
    vNewView.Z += (cosTheta + (1 - cosTheta) * z * z) * vView.Z
    vNewView.X = Cam.Position.X + vNewView.X
    vNewView.Y = Cam.Position.Y + vNewView.Y
    vNewView.Z = Cam.Position.Z + vNewView.Z
    Cam.SetLookAt(vNewView.X, vNewView.Y, vNewView.Z, True)
End Function

Private Function AdvanceCamera(ByVal speed As Single)
    Dim vVector As Vector3 = Vector3.Empty
    vVector.X = Cam.LookAt.X - Cam.Position.X
    vVector.Y = Cam.LookAt.Y - Cam.Position.Y
    vVector.Z = Cam.LookAt.Z - Cam.Position.Z

    Cam.SetPosition(DX9ToolsR5.SetVector.X, Cam.Position.X + (vVector.X * speed ↙
))
    Cam.SetPosition(DX9ToolsR5.SetVector.Y, Cam.Position.Y + (vVector.Y * speed ↙
))
    Cam.SetPosition(DX9ToolsR5.SetVector.Z, Cam.Position.Z + (vVector.Z * speed ↙
))
    Cam.SetLookAt(DX9ToolsR5.SetVector.X, Cam.LookAt.X + (vVector.X * speed))
    Cam.SetLookAt(DX9ToolsR5.SetVector.Y, Cam.LookAt.Y + (vVector.Y * speed))
    Cam.SetLookAt(DX9ToolsR5.SetVector.Z, Cam.LookAt.Z + (vVector.Z * speed))
    Cam.UpdateVP()
End Function

Private Function HStrafeCamera(ByVal speed As Single)
    Dim vStrafe As Vector3 = Vector3.Cross(Vector3.Subtract(Cam.LookAt, Cam.    ↙
Position), Cam.Orientation)
    Cam.SetPosition(DX9ToolsR5.SetVector.X, Cam.Position.X + (vStrafe.X * speed ↙
))
    Cam.SetPosition(DX9ToolsR5.SetVector.Y, Cam.Position.Y + (vStrafe.Y * speed ↙
))
    Cam.SetPosition(DX9ToolsR5.SetVector.Z, Cam.Position.Z + (vStrafe.Z * speed ↙
))
    Cam.SetLookAt(DX9ToolsR5.SetVector.X, Cam.LookAt.X + (vStrafe.X * speed))
    Cam.SetLookAt(DX9ToolsR5.SetVector.Y, Cam.LookAt.Y + (vStrafe.Y * speed))
    Cam.SetLookAt(DX9ToolsR5.SetVector.Z, Cam.LookAt.Z + (vStrafe.Z * speed))
    Cam.UpdateVP()
End Function

Private Function VStrafeCamera(ByVal speed As Single)
    Dim vStrafe As Vector3 = Cam.Orientation
    Cam.SetPosition(DX9ToolsR5.SetVector.X, Cam.Position.X + (vStrafe.X * speed ↙
))
    Cam.SetPosition(DX9ToolsR5.SetVector.Y, Cam.Position.Y + (vStrafe.Y * speed ↙
))
    Cam.SetPosition(DX9ToolsR5.SetVector.Z, Cam.Position.Z + (vStrafe.Z * speed ↙
))
    Cam.SetLookAt(DX9ToolsR5.SetVector.X, Cam.LookAt.X + (vStrafe.X * speed))
    Cam.SetLookAt(DX9ToolsR5.SetVector.Y, Cam.LookAt.Y + (vStrafe.Y * speed))
    Cam.SetLookAt(DX9ToolsR5.SetVector.Z, Cam.LookAt.Z + (vStrafe.Z * speed))
    Cam.UpdateVP()
End Function

Private Function RollCamera(ByVal angle As Single, ByVal x As Single, ByVal y   ↙
As Single, ByVal z As Single)
    Dim vView As Vector3
    Dim vNewView As Vector3
    vView.X = Cam.Orientation.X - Cam.Position.X
    vView.Y = Cam.Orientation.Y - Cam.Position.Y
    vView.Z = Cam.Orientation.Z - Cam.Position.Z
    Dim cosTheta As Single = Math.Cos(angle)
```

```
        Dim sinTheta As Single = Math.Sin(angle)
        vNewView.X = (cosTheta + (1 - cosTheta) * x * x) * vView.X
        vNewView.X += ((1 - cosTheta) * x * y - z * sinTheta) * vView.Y
        vNewView.X += ((1 - cosTheta) * x * z + y * sinTheta) * vView.Z
        vNewView.Y = ((1 - cosTheta) * x * y + z * sinTheta) * vView.X
        vNewView.Y += (cosTheta + (1 - cosTheta) * y * y) * vView.Y
        vNewView.Y += ((1 - cosTheta) * y * z - x * sinTheta) * vView.Z
        vNewView.Z = ((1 - cosTheta) * x * z - y * sinTheta) * vView.X
        vNewView.Z += ((1 - cosTheta) * y * z + x * sinTheta) * vView.Y
        vNewView.Z += (cosTheta + (1 - cosTheta) * z * z) * vView.Z
        vNewView.X = Cam.Position.X + vNewView.X
        vNewView.Y = Cam.Position.Y + vNewView.Y
        vNewView.Z = Cam.Position.Z + vNewView.Z
        Cam.SetOrientation(vNewView.X, vNewView.Y, vNewView.Z, True)
    End Function

Windows Form Designer generated code

    Private Sub frmParasimRun_Closed(ByVal sender As Object, ByVal e As System.     ↙
    EventArgs) Handles MyBase.Closed
        Cleanup()
    End Sub

    Private Sub Cleanup()
        TxtFont = Nothing
        DXGrafx = Nothing
        running = False
    End Sub



    Private Sub KeyPressed(ByVal sender As Object, ByVal e As KeyPressEventArgs)
        Select Case e.KeyChar
            Case "w"
                AdvanceCamera(0.02)
                e.Handled = True
            Case "s"
                AdvanceCamera(-0.02)
                e.Handled = True
            Case "a"
                HStrafeCamera(0.02)
                e.Handled = True
            Case "d"
                HStrafeCamera(-0.02)
                e.Handled = True
            Case "e"
                VStrafeCamera(0.5)
                e.Handled = True
            Case "c"
                VStrafeCamera(-0.5)
                e.Handled = True
            Case "r"
                With Cam
                    .Reset()
                    .SetPosition(DX9ToolsR5.SetVector.Z, -30)
                    .AspectRatio = CSng(Me.ClientSize.Width / Me.ClientSize.Height)
                End With
                e.Handled = True
        End Select
    End Sub

    Private Sub ViewMouseLeave(ByVal sender As Object, ByVal e As MouseEventArgs)    ↙
```

```
Handles pbRender.MouseUp
    If e.Button = MouseButtons.Left Then
        mouseClickedL = False
    End If
    If e.Button = MouseButtons.Right Then
        mouseClickedR = False
    End If
End Sub

Private Sub ViewByMouse(ByVal sender As Object, ByVal e As MouseEventArgs)      ↙
Handles pbRender.MouseMove
    If e.Button = MouseButtons.Left Then
        If mouseClickedL = False Then
            mouseClickedL = True
        Else
            Dim angleX As Single = 0.0
            Dim angleY As Single = 0.0
            angleY = ((lastMousePos.X - e.X)) / 1000.0F
            angleX = ((lastMousePos.Y - e.Y)) / 1000.0F
            RotateView(angleY, 0, 1, 0)
            RotateView(angleX, 1, 0, 0)
        End If
    End If
    If e.Button = MouseButtons.Right Then
        If mouseClickedR = False Then
            mouseClickedR = True
        Else
            Dim angleZ As Single = 0.0
            angleZ = ((lastMousePos.X - e.X)) / 1000.0F
            RollCamera(angleZ, 0, 1, 1)
        End If
    End If
    lastMousePos.X = e.X
    lastMousePos.Y = e.Y

End Sub

Private Sub cmdMotorOn_Click_1(ByVal sender As System.Object, ByVal e As System ↙
.EventArgs) Handles cmdMotorOn.Click
    Dim sTx As String
    sTx = "v"
    CType(pForm, frmPmrLocBase).sendData(sTx, pSocket)
End Sub

Private Sub cmdMotorOff_Click_1(ByVal sender As System.Object, ByVal e As       ↙
System.EventArgs) Handles cmdMotorOff.Click
    Dim sTx As String
    sTx = "s"
    CType(pForm, frmPmrLocBase).sendData(sTx, pSocket)
End Sub

Private Sub cmdRelLED_Click(ByVal sender As System.Object, ByVal e As System.   ↙
EventArgs) Handles cmdRelLED.Click
    Dim sTx As String
    sTx = "p"
    CType(pForm, frmPmrLocBase).sendData(sTx, pSocket)
End Sub

Private Sub cmdSetLED_Click(ByVal sender As System.Object, ByVal e As System.   ↙
EventArgs) Handles cmdSetLED.Click
    ColorDialog1.ShowDialog()
    Dim sTx As String
    sTx = "r" & Chr(255 - ColorDialog1.Color.R) & "g" & Chr(255 - ColorDialog1. ↙
Color.G) & "b" & Chr(255 - ColorDialog1.Color.B)
    CType(pForm, frmPmrLocBase).sendData(sTx, pSocket)
```

```
End Sub

Private Sub cmdDataOn_Click(ByVal sender As System.Object, ByVal e As System.   ↙
EventArgs) Handles cmdDataOn.Click
    Dim sTx As String
    sTx = "c"
    CType(pForm, frmPmrLocBase).sendData(sTx, pSocket)
End Sub

Private Sub cmdDataOff_Click(ByVal sender As System.Object, ByVal e As System.  ↙
EventArgs) Handles cmdDataOff.Click
    Dim sTx As String
    sTx = "m"
    CType(pForm, frmPmrLocBase).sendData(sTx, pSocket)
End Sub


Public Sub pUICommEventUpdate(ByVal source As String)
    Dim iPnt As Int32, sBuf As String, Buffer() As Byte
    If (source.Length) > 0 Then
        'lbHex.Items.Add("Received data: " & source)
        Dim oEncoder As New System.Text.ASCIIEncoding
        Buffer = oEncoder.GetBytes(source)
        For iPnt = 0 To Buffer.Length - 1
            '    lbHex.Items.Add(iPnt.ToString & ControlChars.Tab & String.      ↙
Format("0x{0}", Buffer(iPnt).ToString("X")))
            If HeaderPos = 0 Then
                If Buffer(iPnt) = 13 Then
                    HeaderPos = 1
                End If
            ElseIf HeaderPos = 1 Then
                If Buffer(iPnt) = 10 Then
                    HeaderPos = 2
                Else
                    HeaderPos = 0
                End If
            Else
                SensorPacket(HeaderPos - 2) = Buffer(iPnt)
                HeaderPos = HeaderPos + 1
                If HeaderPos = 12 Then
                    HeaderPos = 0
                    'Handle Packet
                    DataPacket(0) = (SensorPacket(0) * 256) + SensorPacket(1)
                    DataPacket(0) = 1023 - DataPacket(0)
                    DataPacket(1) = (SensorPacket(2) * 256) + SensorPacket(3)
                    DataPacket(2) = (SensorPacket(4) * 256) + SensorPacket(5)
                    DataPacket(3) = (SensorPacket(6) * 256) + SensorPacket(7)
                    DataPacket(4) = (SensorPacket(8) * 256) + SensorPacket(9)
                    DataPacket(5) = (SensorPacket(10) * 256) + SensorPacket(11)
                    'lbAsync.Items.Add("Packet: " & SensorPacket(0) & " " &   ↙
SensorPacket(1) & " " & SensorPacket(2) & " " & SensorPacket(3) & " " &        ↙
SensorPacket(4) & " " & SensorPacket(5) & " " & SensorPacket(6) & " " &        ↙
SensorPacket(7) & " " & SensorPacket(8) & " " & SensorPacket(9) & " " &        ↙
SensorPacket(10) & " " & SensorPacket(11))
                    lbAsync.Items.Add("Packet: " & DataPacket(0) & " " &      ↙
DataPacket(1) & " " & DataPacket(2) & " " & DataPacket(3) & " " & DataPacket(4) ↙
 & " " & DataPacket(5))
                    If DataPacket(0) < 200 Then DataPacket(0) = 200
                    If DataPacket(0) > 840 Then DataPacket(0) = 840
                    LowPassData()
                    myLight = ((DataPacket(0) - 200) / 10) + 20
                    myX = (500 - DataPacket(3)) / 3
                    myY = (440 - DataPacket(2)) / 3
                End If
            End If
```



```
            Next
            'lbHex.SelectedIndex = lbHex.Items.Count - 1
            lbAsync.SelectedIndex = lbAsync.Items.Count - 1
            If lbAsync.Items.Count > 200 Then
                lbAsync.Items.Clear()
            End If
            'If lbHex.Items.Count > 200 Then
            'lbHex.Items.Clear()
            'End If
        End If
    End Sub

    Private Sub LowPassData()
        Dim i, j As Integer
        DataFilter(LPIndex, 0) = DataPacket(0)
        DataFilter(LPIndex, 1) = DataPacket(1)
        DataFilter(LPIndex, 2) = DataPacket(2)
        DataFilter(LPIndex, 3) = DataPacket(3)
        DataFilter(LPIndex, 4) = DataPacket(4)
        DataFilter(LPIndex, 5) = DataPacket(5)
        LPIndex = LPIndex + 1
        If LPIndex = 6 Then LPIndex = 0
        For i = 0 To 3
            DataPacket(i) = (DataFilter(0, i) + DataFilter(1, i) + DataFilter(2, i) ↙
     + DataFilter(3, i) + DataFilter(4, i) + DataFilter(5, i)) / 6
        Next
    End Sub

    Private Sub frmNodeVis_Load(ByVal sender As System.Object, ByVal e As System.   ↙
    EventArgs) Handles MyBase.Load
    End Sub
End Class
```

References

[1] Meguerdichian, S., S. Slijepcevic, V. Karayan, and M. Potkonjak. Localized algorithms in wireless ad-hoc networks: location discovery and sensor exposure. In *MOBIHOC 2001. Proceedings of the 2001 ACM International Symposium on Mobile Ad Hoc Networking and Computing*. 2001. Long Beach, CA, USA: ACM. p. 106.

[2] Grabowski, R., L. E. Navarro-Serment, and P. K. Khosla. Small is beautiful: an army of small robots. *Scientific American (International Edition)*, 2003. 289(5): p. 42.

[3] Howard, A., M. J. Mataric, and G. S. Sukhatme. An incremental self-deployment algorithm for mobile sensor networks. *Autonomous Robots*, 2002. 13(2): p. 113.

[4] LaMarca, A., W. Brunette, D. Koizumi, M. Lease, S. B. Sigurdsson, K. Sikorski, D. Fox, and G. Borriello. Making sensor networks practical with robots. In *Pervasive Computing. First International Conference, Pervasive 2002. Proceedings (Lecture Notes in Computer Science Vol.2414)*. 2002. Zurich, Switzerland: Springer-Verlag. p. 152.

[5] Sinha, A. and A. Chandrakasan. Dynamic power management in wireless sensor networks. *IEEE Design & Test of Computers*, 2001. 18(2): p. 62.

[6] Rahimi, M., H. Shah, G. S. Sukhatme, J. Heideman, and D. Estrin. Studying the feasibility of energy harvesting in a mobile sensor network. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*. 2003. Taipei, Taiwan: IEEE. p. 19.

[7] Chee-Yee, Chong and S. P. Kumar. Sensor networks: evolution, opportunities, and challenges. *Proceedings of the IEEE*, 2003. 91(8): p. 1247.

[8] Abelson, H., D. Allen, D. Coore, C. Hanson, G. Homsy, T. F. Knight, Jr., R. Nagpa, E. Rauch, G. J. Sussman, and R. Weiss. Amorphous computing. *Communications of the ACM*, 2000. 43(5): p. 74.

[9] Butera, William Joseph. *Programming a Paintable Computer*. Ph.D. thesis, Program in Media Arts and Sciences, Massachusetts Institute of Technology, February 2002.

[10] Lifton, J., Seetharam Deva, M. Broxton, and J. Paradiso. Pushpin computing system overview: a platform for distributed, embedded, ubiquitous sensor networks. In *Pervasive Computing. First International Conference, Pervasive 2002. Proceedings (Lecture Notes in Computer Science Vol.2414)*. 2002. Zurich, Switzerland: Springer-Verlag. p. 139.

[11] Paradiso, Joseph A., Joshua Lifton, and Michael Broxton. Sensate Media: Multimodal Skins as Dense Sensor Networks. *BT Technology Journal*, To Be Published 2004.

[12] Lifton, J., M. Broxton, and J. A. Paradiso. Distributed sensor networks as sensate skin. In *Proceedings of IEEE Sensors 2003 (IEEE Cat. No.03CH37498)*. 2003. Toronto, Ont., Canada: IEEE. p. 743.

[13] Kahn, J. M., R. H. Katz, and K. S. Pister. Next century challenges: mobile networking for "Smart Dust". In *MobiCom'99. Proceedings of Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking*. 1999. Seattle, WA, USA: ACM. p. 271.

[14] Warneke, B., B. Atwood, and K. S. J. Pister. Smart dust mote forerunners. In *Technical Digest. MEMS 2001. 14th IEEE International Conference on Micro Electro Mechanical Systems (Cat. No.01CH37090)*. 2001. Interlaken, Switzerland: IEEE. p. 357.

[15] Hill, Jason. Spec takes the next step toward the vision of true smart dust. 2003. `http://www.jlhlabs.com/jhill_cs/`

[16] Barton, J., K. Delaney, S. Bellis, C. O'Mathuna, J. A. Paradiso, and A. Benbasat. Development of distributed sensing systems of autonomous micro-modules. In *53rd Electronic Components and Technology Conference. Proceedings (Cat. No.03CH37438)*. 2003. New Orleans, LA, USA: IEEE. p. 1112.

[17] Thad Starner, Joseph A. Paradiso. *Human Generated Power for Mobile Electronics*. 2004.

[18] Rahimi, Mohammad, Richard Pon, William J. Kaiser, Gaurav S. Sukhatme, Deborah Estrin, and Mani Srivastava. Adaptive Sampling for Environmental

Robots, in *UCLA Center for Embedded Networked Sensing Technical Report Number 29*, 2003; University of California at Los Angeles, Los Angeles, CA.

[19] Benbasat, A. Y., S. J. Morris, and J. A. Paradiso. A wireless modular sensor architecture and its application in on-shoe gait analysis. In *Proceedings of IEEE Sensors 2003 (IEEE Cat. No.03CH37498)*. 2003. Toronto, Ont., Canada: IEEE. p. 1086.

[20] Kaiser, William J., Gregory J. Pottie, Mani Srivastava, Gaurav S. Sukhatme, John Villasenor, and Deborah Estrin. Networked Infomechanical Systems (NIMS) for Ambient Intelligence, in *UCLA Center for Embedded Networked Sensing Technical Report Number 31*, 2003; University of California at Los Angeles, Los Angeles, CA.

[21] Law, C., A. K. Mehta, and K. Y. Siu. A new Bluetooth scatternet formation protocol. *Mobile Networks and Applications*, 2003. 8(5): p. 485.

[22] Heinzelman, W. R., A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*. 2000. Maui, HI, USA: IEEE Comput. Soc. p. 10 pp. vol.2.

[23] Houston, Kenneth M. and Kent R. Engebretson. The Intelligent Sonobuoy System - A Concept for Mapping of Target Fields. Draper Laboratories, Issue CSDL-P-3429.

[24] Irish, James D., Walter Paul, J. N. Shaumeyer, Carl C. Gaither III, and John M. Borden. The Next-Generation Ocean Observing Buoy in Support of NASA's Earth Science Enterprise. *Sea Technology*, May 1999. (40): p. 37-43.

[25] Kerzhanovich, Viktor V., James A. Cutts, and Jeffery L. Hall. Low-cost balloon missions to mars and venus. In *European Space Agency, (Special Publication) ESA SP*. 2003. p. 285.

[26] Bush, Albert O., Jacqueline C. Fernandez, Gerald W. Esch, and J. Richard Seed. *Parasitism: The Diversity and Ecology of Animal Parasites*. 2001, Cambridge University Press: Cambridge, UK. p. 391-399.

[27] Bush, Albert O., Jacqueline C. Fernandez, Gerald W. Esch, and J. Richard Seed. *Parasitism: The Diversity and Ecology of Animal Parasites*. 2001, Cambridge University Press: Cambridge, UK. p. 160-196.

[28] Bush, Albert O., Jacqueline C. Fernandez, Gerald W. Esch, and J. Richard Seed. *Parasitism: The Diversity and Ecology of Animal Parasites*. 2001, Cambridge University Press: Cambridge, UK. p. 306-310.

[29] Bush, Albert O., Jacqueline C. Fernandez, Gerald W. Esch, and J. Richard Seed. *Parasitism: The Diversity and Ecology of Animal Parasites*. 2001, Cambridge University Press: Cambridge, UK. p. 6-9.

[30] Stevenson, Neal. *Snow Crash*. 1992, Bantam Books.

[31] de Bont, Jan. *Twister*. 1996, Warner Bros., USA.

[32] The National Severe Storms Laboratory FAQ. `http://www.nssl.noaa.gov/faq/vortex.shtml`

[33] Tolkien, J.R.R. *The Lord of the Rings*. 1954-1955, London, George Allen & Unwin.

[34] Batalin, M. A. and G. S. Sukhatme. Sensor coverage using mobile robots and stationary nodes. *Proceedings of the SPIE - The International Society for Optical Engineering*, 2002. 4868: p. 269.

[35] Matthies, L., Y. Xiong, R. Hogg, D. Zhu, A. Rankin, B. Kennedy, M. Hebert, R. Maclachlan, C. Won, T. Frost, G. Sukhatme, M. McHenry, and S. Goldberg. A portable, autonomous, urban reconnaissance robot. *Robotics and Autonomous Systems*, 2002. 40(2-3): p. 163.

[36] Sibley, G. T., M. H. Rahimi, and G. S. Sukhatme. Robomote: a tiny mobile robot platform for large-scale ad-hoc sensor networks. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*. 2002. Washington, DC, USA: IEEE. p. 1143.

[37] Texas Instruments. HDQ Communication Basics for TI's Battery Monitor ICs, in *Application Report Number SLVA101*, 2001.

[38] BlueRadios, Inc. `http://www.blueradios.com`

[39] Benbasat, A. and J.P. Paradiso. Design of Real-Time Adaptive Power Optimal Sensor Systems. *IEEE Sensors*, To be published 2004.

[40] Hightower, J. and G. Borriello. Location systems for ubiquitous computing. *IEEE Computer*, 2001. 34(8): p. 57.