SELF SENSING SPACES

By

HICHAM EL ZABADANI

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2006

To my wife Layal.

ACKNOWLEDGMENTS

I would like to first thank my advisor, Professor Abdelsalam Helal, whose guidance, support, patience, and friendship have allowed me to finish this work. To Professor Mark Schmalz I owe a great deal of thanks for his interest in this work and use of his expertise. I would also like to thank the members of my committee, Professors Joachim Hammer, Gerhard Ritter, Shigang Chen, and William Mann, for their time and effort.

I have been very fortunate to work with many outstanding graduate students in the Pervasive Computing Laboratory. To Youssef Kaddoura, Raja Bose, Jeff King, Hen-I Yang, Erwin Jansen, I would like to extend heartfelt thanks for their friendship, for the many discussions we have had, and the help they have given to me over the years. I would also like to thank Steven Pickles and Ed Koush for many useful help and discussions related to the sensor platform.

Finally, I would like to acknowledge the sources of financial support for this research, the National Institute on Disability and Rehabilitation of the Department of Education.

TABLE OF CONTENTS

page

LIST OF TABLES

LIST OF FIGURES

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

SELF SENSING SPACES

By

Hicham El Zabadani

August 2006

Chair: Professor Abdelsalam Helal
Major Department: Computer and Information Science and Engineering

In the past few years the field of pervasive computing has expanded and infiltrated down to everyday consumer activities. Smartness of devices is constantly increasing. Smart products are widespread, not just limited to appliances and electronics. With the availability of such products, there becomes an increased need for scalable smart environments that are capable of integrating them to harness their combined benefits. Sensing a space to find out about its devices and opportunities they offer is a particularly important capability in smart spaces. For example, it is very useful to be able to discover the location of a certain device, reason about its role and possible participation in a certain application, and finally be able to control it remotely. Such self sensing and delegation of control is needed by many applications like remotely pinpointing the location of a certain device that is causing some kind of threat. Self sensing requires world modeling when physical world entities are mapped into model artifacts. In this dissertation, we introduce three approaches to creating useful world models. First, we present SensoBot, a mobile sensor platform that has the ability of mapping the space,

creating the respective floor plan, locating important landmarks, and approximately locating furniture. In addition, we discuss the implementation of smart plugs, a novel way to locate and control smart devices within a real smart house.

We then propose a novel approach in which classical computer vision algorithms are empowered by opportunities presented by the pervasive space. Our approach, which we call PerVision, extends classical object recognition and tracking algorithms by adding a self-assessment/adjustment loop in which sensors and actuators of the pervasive space are used to vary scene parameters to minimize errors in the recognition process. We present the PerVision concept and algorithms in the context of locating and tracking dumb objects such as furniture in a smart house. Collectively, SensoBot, Smart Plugs, and PerVision take us a few steps closer to realizing the ambitious vision of completely self sensing spaces.

CHAPTER 1
INTRODUCTION

Nowadays, the emergence of new technologies in the field of computer science and engineering gives us the opportunity to achieve our goals in the field of self sensing spaces. Pervasive systems are becoming part of our daily life, people are dealing with technologies such as wearable computers, smart homes (which can control temperature gauges, control lighting, or program a home theatre system), speech and gesture recognition sensors, optical switching devices, and embedded sensor networks [1, 2]. Combining new technologies like world modeling and location tracking changes the way remote monitoring is being used. For example, caregivers will be able to locate a senior resident and check any suspected problems in the case of emergency. Despite the progress made so far with respect to world modeling, many challenges, like managing complexity and scalability, transient environments and aggregation of sensor data, and privacy and security, must be overcome [3].

What is a Self Sensing Space?

The primary goal of our research is to explore a synthesis of several disciplines including (1) sensor network research, (2) world modeling, (3) Computer Vision, and (4) self-organizing system principles, to realize the vision of self sensing spaces (SSS). Our vision is to create new capabilities in which smart spaces such as homes sense themselves and their residents, create their respective world models, and enact an accurate mapping between the world model elements and the physical world. If realized, this vision will enable many applications that rely on remote monitoring and intervention. By enabling

spaces to automatically detect their main aspects (e.g., floor plans, type of rooms, etc.), and for objects (e.g., furniture pieces, appliances, etc.) to also be identified automatically, it will be possible to create on-the fly, incremental, real-world models of the pervasive space. The identification and self sensing encompass sensing of the software components needed to interact—within the model—with the model elements. This renders an automatically created, interactive world model of the physical space. The model can be used in many applications, most notably in remote monitoring and intervention in the case of older people and those with disability who need assistance. In this application, intervention occurs by activating actuators, which are mapped in the world model to actual objects in the user's home. Intervention is limited however to objects equipped with actuators. For instance, while it is possible to monitor a chair and its current location in a house, it will not be possible to remotely move the chair from one place to another. It will be possible however to monitor the location and status (on/off) of a set of stove burners, as well as intervene to turn them on or off.

Pervasive Computing and Self Sensing Spaces

Pervasive computing research provides users with the ability to perform computations that are available at any time, any means, and any place. A pervasive space consists of a collection of devices and objects. Within this space there are "sensors," devices that sense the environment, "actuators," devices that can change the environment, and static objects that never change (like furniture, walls, etc.). Many of the devices and applications in the pervasive space require some kind of remote monitoring and intervention. Objects of importance like humans, patients, bombs, and disasters need to be monitored from time to time. To this end, we envision a smart space, such as a home, that is capable of sensing the residents and itself. The space has a model of the world that

can be used by remote monitoring and intervention services. The model of the world must at all times reflect the real world as closely as possible. With current technology, this is however not feasible. The obstacles we face are the following:

- **Scalability**: This is a major problem when it comes to smart spaces. It is still impossible to upgrade a smart space easily. Introducing a new device requires a significant amount of engineering. A smart home resident for example should not call a technician every time he/she wants to install a new lamp. The smart home should be smart enough to know that there is a lamp being installed and it should know exactly how to control it.

- **Lack of interface:** Most of the smart appliances available in the market today do not contain a controllable interface.

- **Protocols:** Different protocols used for interaction with devices. Many devices that do have some form of interface use their own proprietary protocol. X10 for example is an easy, affordable way to control simple appliances. However, what happens if you bring a device that is not X10 enabled? Regardless of the technology used, a smart space should be able to communicate with any new smart device.

To overcome those obstacles, we try to incorporate computational devices in our physical environment to assist the user to accomplish his/her daily tasks. Our end goals are self-integration of new devices, locating and identifying objects of interest to the monitoring/intervention application, and creating privacy-preserving world models. The idea is similar to the PC's plug-and-play system (PnP) which matches up device drivers with the various devices that are available in a computer. After discovering these devices, it initializes the devices to allocate resources. Once this has been taken care of, it is up to the operating system to load the appropriate device drivers.

Scope

In this project we are designing prototypes and algorithms and using different devices to achieve the following:

- First, Smart spaces should be able to sense themselves and their residents.

- Second, respective world models of those smart spaces should be created.
- Finally, there should be an accurate mapping between the physical world and the world model. This mapping will allow for easy remote monitoring and intervention.

We will achieve these goals by implementing the following three sub-projects:

- Sensing the main aspects of the space like floor plan, power outlet location, windows, and doors.
- Sensing active objects that can be queried and controlled by the space like electrical appliances.
- Sensing dumb objects like furniture.

Organization

The remainder of this dissertation is organized into the following chapters:

- Chapter 2 gives an overview of related research in the fields of self sensing spaces.
- Chapter 3 presents an overview of the requirements and approach.
- Chapter 4 discusses the use of a mobile sensor platform to map the space and locate specific landmarks.
- Chapter 5 explains how the smart plug could be used to identify, locate and control electrical appliances.
- Chapter 6 discusses how SensoBot can be used to locate furniture.
- Chapter 7 describes the Pervision approach to locating furniture.
- Chapter 8 presents the conclusion of this project.

CHAPTER 2
RELATED WORK

Although research on self sensing spaces is relatively new, work on scene analysis and world modeling has been the laborious task of many researchers. Much of the current research has centered on location information since this provides a rich source of context. Sensing who is standing before a particular machine allows automatic reconfiguration to suit preferences; tracking people allows for new security measures; tracking objects permits for a more controlled and efficient working environment. In this chapter we will overview the main aspects of the technology used in our research, in addition to brief descriptions of some related works.

Simultaneous Localization and Mapping (SLAM)

Simultaneous localization and mapping (SLAM) is a technique used by robots and autonomous vehicles to build up a map within an unknown environment while at the same time keeping track of its current position [4, 5]. This is not as straightforward as it might sound due to inherent uncertainties in discerning the robot's relative movement from its various sensors.

If at the next iteration of map building the measured distance and direction traveled has a slight inaccuracy, then any features being added to the map will contain corresponding errors. If unchecked, these positional errors build cumulatively, grossly distorting the map and therefore the robot's ability to know its precise location. There are various techniques to compensate for this such as recognizing features that it has come across previously and re-skewing recent parts of the map to make sure the two instances

of that feature become one. Some of the statistical techniques used in SLAM include Kalman filters, particle filters (aka. Monte Carlo methods) and scan matching of range data [6]. SLAM has been a fundamental problem in robotics. It is necessary for safe and efficient navigation of intelligent wheelchair, home-helper or floor-cleaning robots, etc. Previous SLAM methods may work successfully in stationary environments, but they fail in dynamic environments due to map uncertainty as well as environment changes [7].

There has been a great deal of work in Simultaneous Localization and Mapping. One common approach is the use of Kalman filters in combination with landmarks [4]. This approach has been shown to be very successful, but it often requires an environment containing a reasonable set of easily identified landmarks. Recent work has overcome some of the shortcomings of the Kalman filter approach. FastSLAM [8] improves the running time of the Kalman filter techniques and improves its ability to recover from errors. There are also examples of vision-based SLAM work revolving around finding landmarks [9] and vision-based localization only [10]. DP-SLAM [11] is a recent probabilistic technique that does not require landmarks. It uses particle filters to track a large set of potential robot configurations. In essence DP-SLAM works by simultaneously managing many potential configurations of the world and robot based on observations. The most reasonable configurations are preserved and those that contradict are discarded. DP-SLAM was demonstrated using a laser range-finder; however, the general algorithm is not limited to that type of sensor. Specifically, DP-SLAM iterates through a sequence of steps. First the robot takes an observation of the world from its current location. Each particle (configuration) is then weighted based on how consistent the observation was with previous observations and the new observation is added to each

particle's world model. At this point the particles are replaced by a new generation of particles. Particles with higher weights are more likely to be selected as the starting point for one of the new generation particles; however, parents are selected probabilistically so even low weight particles have a chance of having descendants. The next state is created using a Markovian transition model: $P(s'|s)$. Intuitively, the new state is guessed based on the old state, the measured change (odometer readings) and a predictive probability model. In this way particles with high weights will potentially have many descendants. These descendants will only vary based on the probabilistic motion model that is designed to account for error in the odometer readings. The DP-SLAM implementation uses a uniform occupancy grid for its model. Its biggest contribution to related particle filter techniques was a novel method of managing the large number of particle states efficiently. The fundamental problem was that each particle has its own notion of the world state. This resulted in a naive approach that required that there be m copies of the world map for m particles. In practice there may be thousands of particles and therefore the need to manage such a large number of maps was prohibitive.

DP-SLAM is a specific application of particle filters. It demonstrates that particle filters can be used to perform SLAM in a robust and computationally efficient manner. Importantly, it also avoids the need for landmarks.

Computer Vision

Computer vision is the branch of artificial intelligence that focuses on providing computers with the functions typical of human vision. To date, computer vision has produced important applications in fields such as industrial automation, robotics, biomedicine, and satellite observation of Earth. In the field of industrial automation alone, its applications include guidance for robots to correctly pick up and place

manufactured parts, nondestructive quality and integrity inspection, and on-line measurements [11-16].

Computer vision is also playing a very important role in the domain of elder care. This is done by utilizing pervasive technologies, including vision, to develop systems for well-being monitoring, where behavioral data is used to determine trends in the quality of life of frail and disabled older persons. The ideas of well-being and well-being monitoring cover a potentially huge area, involving a range of conceptual, methodological and instrumentation issues [17, 18].

Computer vision is the study of methods that can be used for allowing computers to understand images. The term “understand” refers to extracting specific information from images for a specific purpose, either to present it to a human operator, or for controlling a process. The image data used in a computer vision system is often represented as gray-scale or color digital image in either 2-D or 3-D format.

Since a camera can be seen as a light sensor, there are various methods in computer vision that deal with light and images. For example, it is possible to extract information about motion in fluids and waves by analyzing images of these phenomena. Moreover, a subfield of computer vision deals with the physical process which forms a camera image given a scene of objects, light sources, and lenses in the camera.

A third field which plays an important role is biology, or biological vision system. Over the last century, large amount of studies on eyes, neurons, and the brain structures devoted to processing of visual stimuli in both humans and various animals. These studies have led to a broad, yet complicated, description of how "real" vision systems operate in order to solve certain vision related tasks.

Beside the above mentioned views on computer vision, many of the related research topics can also be studied from a purely mathematical point of view. Finally, a significant part of the field is devoted to the implementation aspect of computer vision; how can existing methods be implemented in various combinations of software and hardware, or how these methods can be modified in order to gain processing speed without loosing too much performance.

The typical tasks of computer vision could be characterized as:

- **Object recognition:** detecting the presence and/or position of known objects in an image. (e.g., searching for digital images by their content or recognizing human faces and their location in photographs.)
- **Tracking:** tracking known objects through an image sequence. (e.g., tracking a single person walking through a shopping center.)
- **Scene interpretation:** creating a model from an image/video. (e.g., creating a model of the surrounding terrain from images, which are being taken by a robot-mounted camera.)
- **Ego positioning:** determining position and motion of the camera itself. (e.g., navigating a robot through a museum.)

The major functions of a typical computer vision system are image acquisition, preprocessing, feature extraction, and registration. The image or image sequence is acquired with an imaging system. Often the imaging system has to be calibrated before being used. In the preprocessing step, the image is being treated with "low-level" operations. The aim of this step is to do noise reduction on the image and to reduce the overall amount of data. Feature extraction means further reducing the data to a set of features, which ought to be invariant to disturbances such as lighting conditions, camera position, noise and distortion. The aim of the registration step is to establish correspondence between the features in the acquired set and the features of known

objects in a model-database and/or the features of the preceding image. The registration step has to bring up a final hypothesis.

What is Color?

While most of us have been taught in school that color is the wavelength of light, this is only true when we are talking about the color of a monochromatic light source. The color of an object is something much more indirect and intricate.

Let us assume that the scene is illuminated by a light source with a spectral composition described by $I(\lambda)$, where λ ranges over the wavelengths of visible light (approximately 400-700 nm) and I is the intensity of the light at each wavelength. We now need to consider how the illuminated light is transformed when it is reflected at a surface patch on an object in the scene. Here, we will make the simplifying (but generally incorrect) assumption that the surface patch is *Lambertian*, that is, it looks equally bright independent of the viewing angle. How the surface patch reflects each wavelength of the illuminating light can be described with the function $S(\lambda)$. This means that the spectral content of the light reaching the eye or camera is given by $I(\lambda)S(\lambda)$. However, this spectral composition is generally not available directly (unless a photo-spectrometer is used), but is measured by the number of photoreceptors (in the human eye) or photo-sensors (in a video camera).

The photoreceptors of the human eye are tuned approximately to red, green and blue light (564, 533 and 437 nm) [19]. As a consequence, a similar tuning is also often used in color cameras, for example 650, 530, 470 nm [20]. It is important to realize that the use of red, green and blue has nothing to do with the nature of light, it is simply a way to reduce the infinite dimensional spectrum to three measurements in an arbitrary way

that parallels the way our eyes code colors at the receptor level. Some color cameras use a different coding internally, which is subsequently converted to an RGB representation.

The reason RGB coding is used on computer screens is that the three light sources on the screen tap directly into the three receptor types in our eye. When we see the same colors on the computer screen as in the real scene, it is not because they are identical, but a consequence of the limitation of the human color vision system. Other animals have different and sometimes even more types of receptors tuned to different wavelengths and would not see the same colors in the two cases. For example, the stomatopod *Odontodactylus scyllarius* have more than ten different types of photoreceptors tuned to varying wavelengths [21].

The reaction of a photoreceptor or output from a sensor in a camera can be modeled in the following way [22]. Let i be the specific type of sensor and let $Ri(\lambda)$ be the spectral sensitivity of the sensor.

The output from the sensor qi is described by

$$q_i = \int_{400nm}^{700nm} I(\lambda) R_i(\lambda) S(\lambda) d\lambda \qquad (1)$$

If $i = R;G;B$, the above equation converts the spectrum of the light that reaches the camera to a three dimensional vector $<q_R;q_G;q_B>$. How does this vector represent the color or the surface patch? According to the equation above it does not since the output from the photosensors depends on the three factors I, S and R, only one of which is related to the surface patch. Only in the ideal case when the illumination is perfectly white, that is when $I(\lambda) = 1$, does the camera give an unique estimation of the color of the surface [23].

Basic Color Definitions

Commonly used well-known color spaces include: (for display and printing processes) RGB, CMY; (for television and video) YIQ, YUV; (standard set of primary colors) XY Z; (uncorrelated features) $I_1I_2I_3$; (normalized color) rgb, xyz; (perceptual uniform spaces) U*V*W*, L*a*b*, Luv; and (for humans) HSI. Although, the number of existing color spaces is large, a number of these color models are correlated to intensity I: Y , L*and W*; are linear combinations of RGB: CMY , XY Z and $I_1I_2I_3$; or normalized with respect to intensity rgb: IQ, xyz, UV, U*V* , a*b*, uv.

We need to be precise on the definitions of intensity I, RGB, normalized color rgb, saturation S, and hue H. To that end, in this section, we offer a quick overview of well-known facts from color theory [12].

Let R, G and B, obtained by a color camera, represent the 3-D sensor space

$$C = \int_{\lambda} p(\lambda) f_c(\lambda) d(\lambda) \qquad (2)$$

for $C \in (R; G; B)$, where p(λ) is the radiance spectrum and $f_c(\lambda)$ are the three color filter transmission functions.

To represent the RGB-sensor space, a cube can be defined on the R, G, and B axes. White is produced when all three primary colors are at M, where M is the maximum light intensity, say M = 255. The main diagonal-axis connecting the black and white corners defines the intensity

$$I(R, G, B) = R + G + B \qquad (3)$$

All points in a plane perpendicular to the grey axis of the color cube have the same intensity. The plane through the color cube at points R = G = B = M is one such plane. This plane cuts out an equilateral triangle which is the standard rgb chromaticity triangle

$$r(R,G,B)=\frac{R}{R+G+B} \quad (4)$$

$$g(R,G,B)=\frac{G}{R+G+B} \quad (5)$$

$$b(R,G,B)=\frac{B}{R+G+B} \quad (6)$$

The transformation from RGB used here to describe the color impression hue H is given by

$$H(R,G,B)=\arctan\left(\frac{\sqrt{3(G-B)}}{(R-G)+(R-B)}\right) \quad (7)$$

and saturation S measuring the relative white content of a color as having a particular hue by

$$S(R,G,B)=1-\frac{\min(R,G,B)}{R+G+B} \quad (8)$$

In this way, all color features can be calculated from the original R, G, B values from the corresponding red, green, and blue images provided by the color camera.

RFID Technology

Radio Frequency Identification (RFID) is an automatic identification method, relying on storing and remotely retrieving data using devices called RFID tags or transponders which can be attached to or embedded within objects. Tags contain silicon chips and antennas to enable them to receive and respond to radio-frequency queries from an RFID transceiver. Passive tags require no internal power source, whereas active tags require a power source [24].

RFID System

An RFID system may consist of several components: tags, tag readers, edge servers, middleware, and application software.

The purpose of an RFID system is to enable data to be transmitted by a mobile device, or the tag, which is read by an RFID reader and processed according to the needs of a particular application. The data transmitted by the tag may provide identification or location information, or specifics about the product tagged, such as price, color, date of purchase, etc. The use of RFID in tracking and access applications first appeared during the 1980s. RFID quickly gained attention because of its ability to track moving objects. As the technology is refined, more pervasive and possibly invasive uses for RFID tags are in the works.

In a typical RFID system, individual objects are equipped with a small, inexpensive tag. The tag contains a transponder with a digital memory chip that is given a unique electronic product code. The interrogator, an antenna packaged with a transceiver and decoder, emits a signal activating the RFID tag so it can read and write data to it. When an RFID tag passes through the electromagnetic zone, it detects the reader's activation signal. The reader decodes the data encoded in the tag's integrated circuit (silicon chip) and the data is passed to the host computer. The application software on the host processes the data.

RFID Usage

Take the example of books in a library. Security gates can detect whether or not a book has been properly checked out of the library. When users return items, the security bit is re-set and the item record in the integrated library system is automatically updated. In some RFID solutions a return receipt can be generated. At this point, materials can be

roughly sorted into bins by the return equipment. Inventory wands provide a finer detail of sorting. This tool can be used to put books into shelf-ready order.

Open Services Gateway Initiative (OSGi)

OSGi, the Open Services Gateway Initiative, is the specification for an execution environment that provides a managed, extensible framework to connect various devices in a local network. The system contains a standardized collection of service interfaces, and promotes the dynamic discovery and collaboration of devices and services from different sources [25]. The framework also supports connectivity from outside the local network, allowing for remote control, diagnosis, and management.

The OSGi Alliance [26] manages the OSGi specification and its associated licenses. Various open-source implementations of the framework exist, such as Oscar, Knopflerfish, and Equinox.

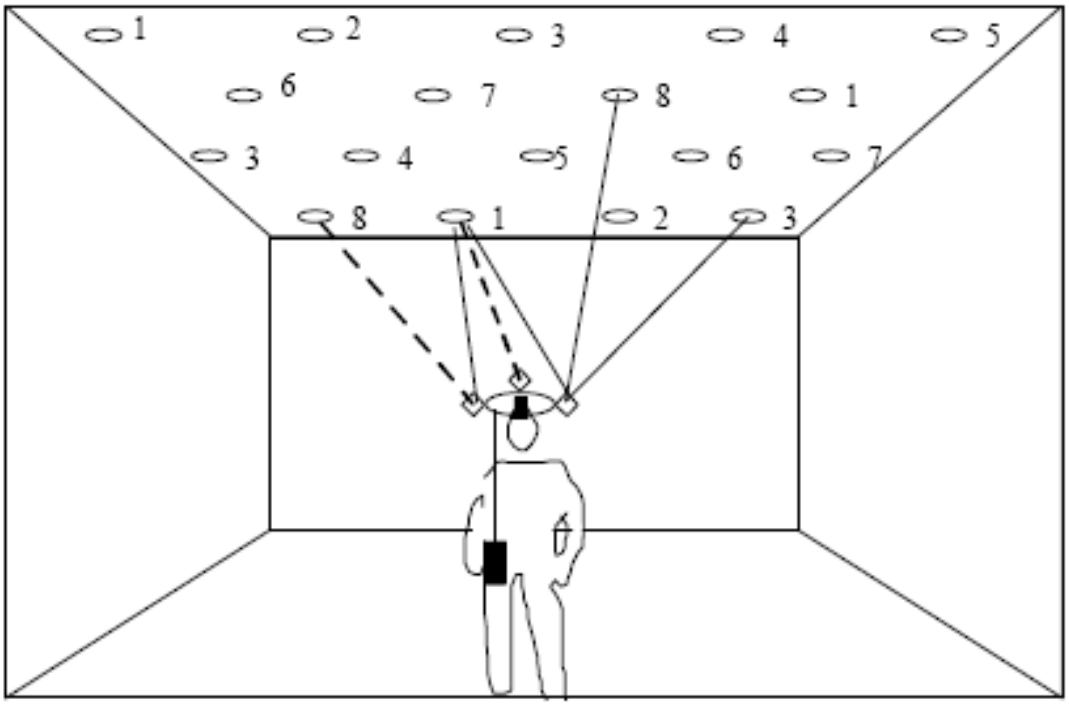


Figure 1. General idea of the constellation system

World Modeling and Location Systems

The Constellation System

The Constellation system [27] is a positioning system designed for accurate tracking in augmented reality systems. It uses a series of ultrasonic sensors worn on the belt and head of a user. Ultrasonic transmitters are installed in the environment and periodically emit signals which can be measured by the mobile sensors. Multilateration is used to calculate position to a precision of approximately 0.005m.

The CONSTELLATION tracking system is similar in its basic principles of operation to an aided inertial navigation system (INS), except that it operates indoors, has much finer resolution and accuracy, and uses acoustic rather than RF technology for range measurements. Figure 1 illustrates the system, configured for tracking an HMD in a wide-range VR or AR application. The head mounted display (HMD) is equipped with an integrated inertial sensing instrument called the InertiaCube™ and, in this example, 3 ultrasonic rangefinder modules (URMs). The rangefinder modules communicate with a constellation of transponder beacons which may be mounted at any known locations in the environment. The beacons are activated one-at-a-time by infrared trigger codes emitted by the rangefinder modules. As each beacon receives its own unique code, it responds by emitting an ultrasonic pulse. The rangefinders count the time-of-flight (TOF) until the pulse arrives, and use the speed of sound to convert the TOF into a distance. These range measurements are fed into an extended Kalman filter (EKF) which makes small adjustments to the position and orientation trajectory which is being update at a high rate by the strapdown INS. At least 6 range measurements, connecting between at least 3 HMD-mounted microphones and at least 3 fixed transponder beacons, are required to completely determine the position and orientation of the HMD. Figure 1 shows an

example of 6 suitable ranges, which illustrates that multiple nearly simultaneous measurements from each triggered beacon can be used if available, but are not required. Two degrees of freedom can be resolved by stabilizing with respect to gravity, so only 4 of the myriad potential lines-of-sight need to be open to continue tracking indefinitely, and fewer than 4 can be sufficient to sustain reasonable tracking for a while.

The Bat System

The Bat system [28] provides accurate positioning for powered tags by multilateration of the time-of-flights of ultrasonic signals. An active mobile transmitter is worn by personnel or affixed to objects. Radio signals are used to synchronize the emission of ultrasonic pulses, which are detected by a matrix of receivers installed at known locations in the ceiling. The system is designed for tracking personnel and has a precision of approximately 0.03m.

Figure 2. A mobile transmitter

A small, wireless transmitter is attached to every object that is to be located. The devices, shown in Figure 2 consist of a microprocessor, a 418MHz radio transceiver, a Xilinx FPGA (Field-Programmable Gate Array) and a hemispherical array of five

ultrasonic transducers. Each prototype mobile device has a unique 16-bit address, is powered by two lithium cells, and measures 100mm×60mm×20mm. A matrix of receiver elements is mounted on the ceiling of the room to be instrumented. Receivers are placed in an array, 1.2m apart—the prototype system has 16 receivers in a four-by-four square grid. Each receiver has an ultrasonic detector, whose output is passed through an amplifier, rectifier and smoothing filter before being digitized at 20 KHz by an ADC. The ADC is controlled by a Xilinx FPGA, which can monitor the digitized signal levels. Receivers also have a serial network interface, through which they are individually addressable, and are connected in a daisy-chain to a controlling PC. Every 200ms, a radio message consisting of a preamble and 16-bit address is transmitted in the 418MHz band by a controller also connected to the PC. The PC dictates which address is sent in each message. The radio signals are picked up by the transceiver on each mobile device and decoded by the on-board FPGA. The single addressed device then drives its transducers for 50ms at 40 KHz, and an ultrasonic pulse is broadcast in a roughly hemispherical pattern around the top of the unit. After receiving a message, mobile devices enter a power-saving state, activating themselves 195ms later, ready for the next message. The controlling PC sends a reset signal to the receivers over the serial network at the same time as each radio message is broadcast. The FPGAs on each receiver then monitor the digitized signals from the ultrasonic detector for 20ms, calculating the moment at which the received signals peak for the first time. The short width of the ultrasonic pulse ensures that receivers detect a sharp signal peak. The controlling PC then polls the receivers on the network, retrieving from them the time interval between the reset signal and detection of the first signal peak (if any signal was detected).

The Cricket System

The Cricket system [29] uses *beacons* to disseminate information about a geographic space to *listeners*. A beacon is a small device attached to some location within the geographic space it advertises. Typically, it is obtained by the "owner" of the location (e.g., the occupant of a room in an office or home, or a building administrator) and placed at an unobtrusive location like a ceiling or wall. Cricket does not attach any semantics to the space advertised by the beacon; any short string can be disseminated, such as the name of a server to contact to learn more about the space or a name resolver for the space to discover resources. Cricket beacons are inexpensive and more than one of them can be used in any space for fault-tolerance and better coverage.

To obtain information about a space, every mobile and static node has a listener attached to it. A listener is a small device that listens to messages from beacons, and uses these messages to infer the space it is currently in. The listener provides an API to programs running on the node that allow them to learn where they are, so that they can use this information to appropriately advertise themselves and their location to a resource discovery service.

The listener can be attached to both static and mobile nodes. For example, when a user attaches a new static service to the network (e.g., a printer), she does not need to configure it with a location or other any attribute; all she does is attach a listener to it. Within a few seconds, the listener infers its current location from the set of beacons it hears, and informs the device software about this via the API. This information can then be used in its own service advertisements. When a mobile computer has a listener attached to it, the listener constantly listens to beacons to infer its location. As the computer (e.g., a hand-held computer carried by a person) moves in a building, the

navigation software running on it uses the listener API to update its current location. Then, by sending this information securely to a map server (for example), it can obtain updates to the map displayed to the user. Furthermore, services appear as icons on the map that are a function of the user's current location. The services themselves learn their location information using their own listener devices, avoiding the need for any per-node configuration.

The EasyLiving Project

The EasyLiving project at Microsoft uses video cameras to track a small number of users around an environment [30]. Microsoft also developed the RADAR system [31] which uses radio propagation in environments equipped with wireless LAN systems to track mobile devices. RADAR has since inspired a number of location systems based on wireless LAN [32-34], but none exhibit the same accuracy properties of current ultrasonic systems. Their demo displays a number of different ubiquitous computing capabilities:

- **Perception and the world model:** The EasyLiving Lab has an "awareness" of how many people are in the room and where they are. This is provided by a set of stereo camera heads mounted around the walls of the room. Each stereo (3-head) camera is attached to its own dedicated PC, which calculates a depth image. The depth image is subtracted from a background image of the empty room to quickly detect moving objects, i.e. people. The outputs from all the person-detectors are fed into a person-tracker which matches the people seen by different cameras from different angles. Based on this information, the lights can be turned on and off automatically. Another camera looks down from overhead and tracks a wireless keyboard as it is picked up, put down, and handed from person to person. The information about people's locations and the keyboard location is sent to a central database called the World Model. The World Model is implemented using a commercial SQL database program. It has been augmented with the ability to perform geometric reasoning about point and polygon intersections. The world model also stores information about devices and their regions of service. The lights serve the entire room. Therefore, when a person enters the room, their location (point) in the world model intersects the light service region, and the lights are turned on.

- **Personal identity:** Most consumer electronics has no concept of personal identity; all people get the same set of controls and they work the same way. On the other hand, most computers require that you log on before you can do anything at all. In EasyLiving, you can work both ways. When you enter the room, you are given a "Person ID Number" because the system does not know who you are. You can browse the web and control the lights. Therefore, these actions are available to everyone even if they do not log onto a computer account. If you then identify yourself, by using a login password or touching a thumb scanner in the room, the system determines who you are. It then gives you the ability to call up your personal desktop onto any computer screen in the room. If you then move to another computer, the person-tracker keeps track of your movement so that the new computer will automatically know who you are as well. Then, you can access your personal information on the new computer without logging in again.

- **Disaggregated computing:** In EasyLiving, the display, keyboard, and mouse do not have to be attached to the same computer. You can use any keyboard and mouse with any display. The software you are running believes that you have "a computer", and EasyLiving transmits the input and output across the network as necessary to use the devices you indicate. This capability also works for laptop computers. Therefore, if you bring a laptop into the EasyLiving Lab, you can tell the system to move the display up onto the big wall screen. You continue to use the keyboard and pointer on your laptop. In this way, if you go to a meeting, you can show your presentation on the wall screen without having to hook up any cables or move your file onto the "room" computer that controls the screen.

- **Automatic behavior:** The EasyLiving system has "behavior rules" that cause things to happen automatically when certain relationships are satisfied in the world model. One rule causes you to be logged out of a computer when you move out of the service area of that computer. Another rule causes you to be automatically logged onto a new computer when you move into its service area. In this way, as you move from place to place, you can always use the nearby computer. These automatic behavior rules are also used to turn the lights on and off as people enter and leave the room.

Ray-Tracing

Ray-Tracing is another method to create world models. It uses rays within positioning systems to discover and characterize static objects. A ray is established between a mobile transmitter and a receiver when the ranging measurement (time-of-flight or equivalent) for a signal propagating between them is not rejected by the positioning algorithm. Rays describe straight pathways which are unobstructed to the positioning medium. Given a diverse series of transmitter locations over time, and a

similarly diverse distribution of receivers, rays penetrate into the environment and obstructions become apparent from low densities of rays. This is the premise for using ray-tracing to create and maintain the world model [35].

Large numbers of rays are best stored within an accumulation grid. This segments the volume of the environment into a regular three-dimensional grid, each cell having an associated voting count (initially zero). A new ray is quantized onto the grid by incrementing the voting count of each and every cell it intercepts. After a period of time, the voting grid can be converted to a binary occupancy grid via thresholding. By introducing the voting stage it is possible to account for any noise in the system. More complex approaches using probabilistic beams are possible, but can involve greater processing requirements and are not necessary to demonstrate the underlying methodology. Once an occupancy grid is established analysis is necessary to autonomously identify the presence of objects. Three-dimensional region growing can be used to extract containment volumes and shapes. It is also possible to use techniques specialized to the particular object. For example, the shape and height of horizontal surfaces has been extracted using profile plots. These plot the ratio of perimeter and area against vertical height.

The Ultra-Wideband Radio System

Other positioning systems are on the horizon. In particular, ultra-wideband radio systems for indoor location tracking (UWB) [36].

UWB technology provides an excellent means for wireless positioning due to its high resolution capability in the time domain. Its ability to resolve multi-path components makes it possible to obtain accurate location estimates without the need for complex estimation algorithms. This precise location estimation capability facilitates many

applications such as medical monitoring, security, and asset tracking. Standardization efforts are underway in the IEEE 802.15.4a PAN standard, which will make use of the unique features of the UWB technology for location-aware sensor networking. There exist some theoretical limits for time-of-arrival (TOA) estimation and TOA-based location estimation for UWB systems. Due to the complexity of the optimal schemes, suboptimal but practical alternatives have been emphasized. Performance limits for hybrid time-of-arrival/signal-strength (TOA/SS) and time-difference-of-arrival/signal-strength (TDOA/SS) schemes have also been considered.

CHAPTER 3
ASSUMPTIONS AND APPROACH

In this chapter we will first cover the basic assumptions needed by the self sensing spaces project. We will then describe the approach taken to implement the various elements in this project.

Reference Space

The first requirement in building our project is the need for a pervasive space. A pervasive space contains a network of more or less specialized computational devices that interact among themselves and with the users. These devices, which are distributed in the physical space, have a high degree of autonomy than we are used to, and they increasingly serve as a medium for cooperation and communication among humans. Some of these devices are moveable and possibly collocated with humans, while others are stationary. Since the devices are located in many places and some of them are able to move, they must be sensitive to their context or environments.

Our pervasive space is called the Gator Tech Smart House (Shown in Figure 3.), which is the culmination of more than five years of research in pervasive and mobile computing [15, 37]. The Gator Tech Smart House (http://www.icta.ufl.edu/gt.htm) is a 2500 sq. ft. house located in the Oak Hammock retirement community in Gainesville, FL. The primary goal of the house is to create an assistive environment for seniors, allowing residents to lead reasonably independent lives even if they require some level of managed care. The GTSH contains a myriad of sensors and actuators to monitor and control the pervasive space [38].

Figure 3. Outside the Gator Tech Smart House

The Approach

The self sensing spaces project consists of three main sub-projects: Sensing the main aspects of the space (floor plan, doors, windows, etc.), locating active objects like electrical appliances, and finally, locating passive objects like furniture.

Sensing the Main Aspects of the Space

The first task that a self sensing space should really accomplish is to discover itself in terms of boundaries and shape. Possible questions that might be answered when this task is done are: What is the floor plan of this space? How many rooms are there? How many windows and doors exist in each room? What is the location of each power outlet in the house?

Since the floor plan of a space can rarely change, this process might be done only once. In order to complete this task we use SensoBot which is a mobile sensor platform that uses on-board RFID modules to locate RFID tags embedded in the carpet in the form of a grid. While driving around, SensoBot writes to the tags the corresponding coordinates which can be used to create the corresponding floor plan. In addition,

SensoBot is used to locate important landmarks like doors, windows, and power outlets. SensoBot is discussed in details in chapter 4.

Content Sensing

Now that the space knows its floor plan and boundaries, it will then try to sense its content. Every space (house, office, etc.) contains 2 types of objects. The first type consists of static (powerless) objects like tables, chairs, entertainment center, etc. We call them static because their status never changes. The other type consists of dynamic (powered) objects like lamps, radios, TVs, etc. We call them dynamic because their status changes frequently. The only property that is shared between the two types is location. Locating those objects is one of the major goals that we are trying to achieve in the self sensing spaces project.

Sensing active objects

In order to sense active objects, we came out with a novice idea that uses power outlets and RFID. We all know that all electrical appliances use electricity to operate. Electricity is provided by hooking the power cord to a power outlet. If the appliance's plug contains an RFID tag that describes the object being used, and the power outlet contains an RFID reader, then by knowing the location of the power outlet, we can locate the object. This technology is called the smart plug and will be described in more details in chapter 5.

Sensing passive objects

The final step before we have a complete self sensing space is locating passive objects. We will assume that furniture is the only type of passive objects that can exist in a space. The approach used to complete this task includes two main subtasks: PerVision, which is a combination of pervasive computing and computer vision, and SensoBot,

which is the mobile sensor platform. PerVision uses computer vision to first locate furniture then it uses the pervasive computing technology to improve the recognition results. SensoBot uses the on-board RFID modules to locate RFID tags embedded in furniture. It then uses the RFID tag grid embedded in the carpet to locate those objects. Locating furniture using Pervision and SensoBot is described in more details in chapter 6.

CHAPTER 4
MAPPING PERVASIVE SPACES

Smart Spaces are being inhabited by many cooperating agents of different types. As well as people there are likely to be fixed sensors embedded in the environment and mobile agents such as robots [39]. By enabling spaces to automatically detect their main aspects (e.g., floor plans, type of rooms etc.), and for objects (e.g., furniture pieces, appliances, etc.) to also be identified automatically, it will be possible to create on-the-fly, incremental, world models of the pervasive space [40, 41].

To achieve this vision we use a robot equipped with different sensors and actuators to enable automatic space mapping and content sensing. The goal is to design a mobile platform that is able to automatically create a floor plan of a space. In addition, the robot should locate and identify important landmarks and objects in that space.

As we previously discussed, SLAM is a technique used by robots and autonomous vehicles to build up a map within an unknown environment while at the same time keeping track of its current position. However, this is not as straightforward as it might sound due to inherent uncertainties in discerning the robot's relative movement from its various sensors. To eliminate errors produced while building the map, we will use RFID technology. The idea is to have a grid of RFID tags embedded in the carpet. The tags initially hold no data other than the serial number. The robot’s job is to fill each tag with the corresponding coordinates. We will discuss later in this chapter the techniques used to achieve this goal. One important task will be locating important landmarks and objects like furniture. The robot will also use RFID tags embedded in those objects to perform

the identification and location process. We will discuss locating landmarks in this chapter; however, locating furniture will be detailed in chapter 6.

SensoBot: The Mobile Sensing Platform

In this project we needed to build a robot that will carry our sensing platform around. The robot's main job is to perform simple actions like move forward, turn 90 degrees clockwise, etc. Instead of building this robot from scratch, we used IRobot's famous vacuum cleaner Roomba to perform this task [42]. The reason why we chose Roomba is the fact that it allows for external control using its Serial Command Interface (SCI) [43]. Roomba SCI is a serial protocol that allows users to control a Roomba through its external serial port (Mini-DIN connector). The SCI includes commands to control all of Roomba's actuators (motors, LEDs, and speaker) and also to request sensor data from all of Roomba's sensors. Using the SCI, we can add functionality to the normal Roomba behavior and we can also create completely new operating instructions for Roomba. In addition, we needed two RFID readers and one digital compass all controlled by our Atlas sensor platform [44]. The mobile platform in its entirety is called SensoBot.

Roomba Specifications

Roomba is a robotic vacuum cleaner that uses onboard processing to perform the cleaning task. Integrated sensors detect dirt and increase the focus and intensity of cleaning in a specific area, while infra-red cliff sensors in the non-marring bumper detect stairs to keep the unit from falling, as well as furniture and walls for cleaning right up to the edge.

As shown in Figure 4, there are four buttons on the top. The buttons are from left to right: Power, Spot, Clean, and Max. The last three buttons perform different cleaning cycles when pressed. There is a bump sensor in the front of Roomba used to detect any

obstacles. The bottom view shows the two wheels that drive Roomba. The battery (in yellow) is also shown. There also exist four cliff sensors that detect edges such as stairs and keep Roomba from falling.

Figure 4. Roomba's top and bottom

To use the SCI, a processor capable of generating serial commands such as a PC or a microcontroller must be connected to the external Mini-DIN connector on Roomba. The Mini-DIN connector provides two way serial communication at TTL Levels as well as a Device Detect input line that can be used to wake Roomba from sleep. The connector also provides an unregulated direct connection to Roomba's battery which we can use to power any external devices.

One of the best features existing in Roomba is the self-charging home base. Using infrared sensors, Roomba can find its way back to the home base and can dock to recharge itself. Since we can read all sensors, including the battery level, we can send Roomba to its home base as soon as battery power drops below a certain level.

Atlas Platform

In order to control Roomba, we need to use an external microprocessor which will be the brain of SensoBot. We chose the Atlas platform to do this job. Atlas, as shown in

Figure 5, is built around an Atmel Atmega128L microcontroller. It has a modular design and it is designed for low power usage. It allows multiple types and number of sensors to be connected to it and it has a small form factor. Most important, it has WI-FI capabilities.

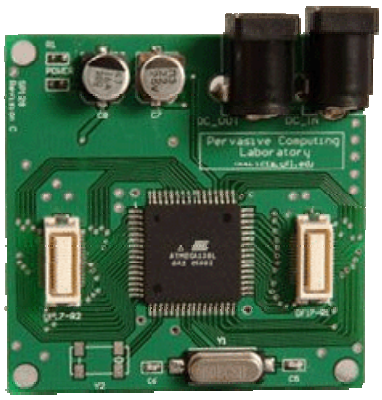


Figure 5. Atlas platform

The main job of Atlas is to act as the brain that controls the entire operation. First, it will connect to Roomba using one serial port. It will also connect to two RFID readers using the second serial port. It will also use a digital compass that will be connected using SPI. Finally, it will communicate with a computer using WI-FI. We will discuss the main use of each of the devices listed above later in this report.

RFID Modules

SensoBot uses two RFID readers each having an external antenna pointing to a different location. The first reader's antenna is located in the center of SensoBot and is pointing downward as shown in Figure 6 (right). This reader will be used to read tags embedded in the carpet.

The second reader's antenna is located in the front pointing forward as shown in Figure 6 (left). This reader will be used to read tags located on either objects or walls. We used Skyetek's M1-mini RFID reader which is a multi-protocol 13.56MHz OEM

module capable of reading and writing to transponders based on ISO 15693, ISO 14443A, and ISO18000-3 air-interface protocols . The M1 features an on-board antenna as well as the ability to attach a standard 50 Ohm external antenna for improved read-range. Four interface options are available to provide communication to a variety of host systems: RS232 or TTL/ serial, I2C, and SPI. With its on-board power regulator circuit, the M1 can operate from 1.8-5.0V; while the power management intelligence allows current to be set as low as 60 µA (Sleep Mode) making it ideal for use in battery operated devices.

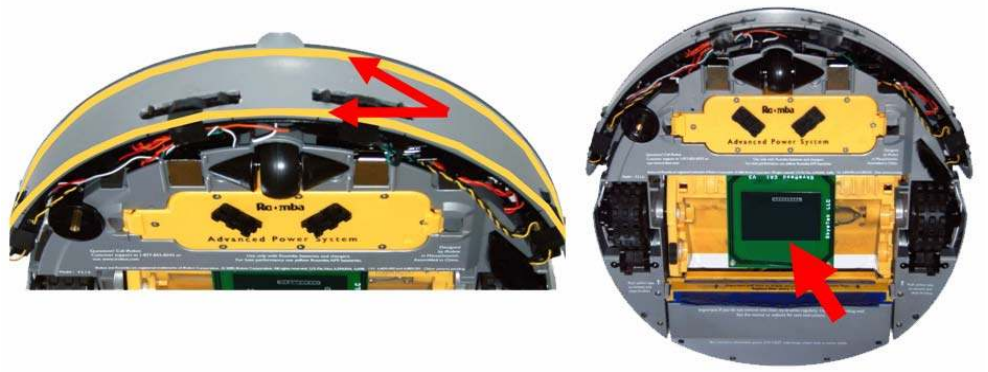


Figure 6. Front RFID reader (Left), Bottom RFID reader (right)

Digital Compass

SensoBot uses a digital compass to find directions. It is critical to have a high resolution compass so that errors in getting to a specific destination can be minimal. The compass used is PNI Corp's Vector 2Xe which is shown in Figure 7.

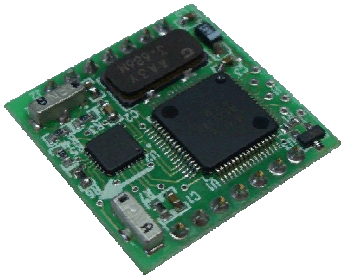


Figure 7. Vector 2Xe 2-axis compass

Applications

In this chapter we will discuss two main tasks: automatically create a two-dimensional floor plan of the space using SensoBot, and locating important landmarks using the floor plan found.

Creating Floor Plan

Mapping a space using SensoBot could not be achieved without the use of RFID technology. To solve the SLAM problem, we will use RFID tags to create landmarks that SensoBot can use to create the corresponding floor plan.

The main requirement in this project is to have a grid based on passive, low-cost, High Frequency RFID tags installed under the flooring. A single passive RFID tag represents a single grid point in the system. Carpet manufacturers could integrate the RFID tags as part of the weaving process or the RFID tags could be integrated into a thin layer of material that is applied under the carpet or hard surface flooring. Rooms that have existing carpeting could be easily upgraded by rolling up the carpet, applying the RFID flooring material and then reinstalling the existing carpet. Figure 8 shows the carpet tiles used in the Gator Tech Smart House. Each tile is a 20 square inches and holds a tag in the middle.

Initially, each tag holds no data other than a unique serial number assigned by the manufacturer. It is the job of SensoBot to fill the corresponding data for each tag. Starting from a specific location in the space, SensoBot will start moving until it finds the first tag. This tag will be the origin with coordinates [0,0]. The task now is to find a neighbor of this tag. Two tiles are called neighbors if they share the same side. After finding the neighbor, SensoBot starts looking for the neighbors of the previous ones and so on until

we find all tags. We will discuss the algorithm used to find the neighbors later in this chapter.

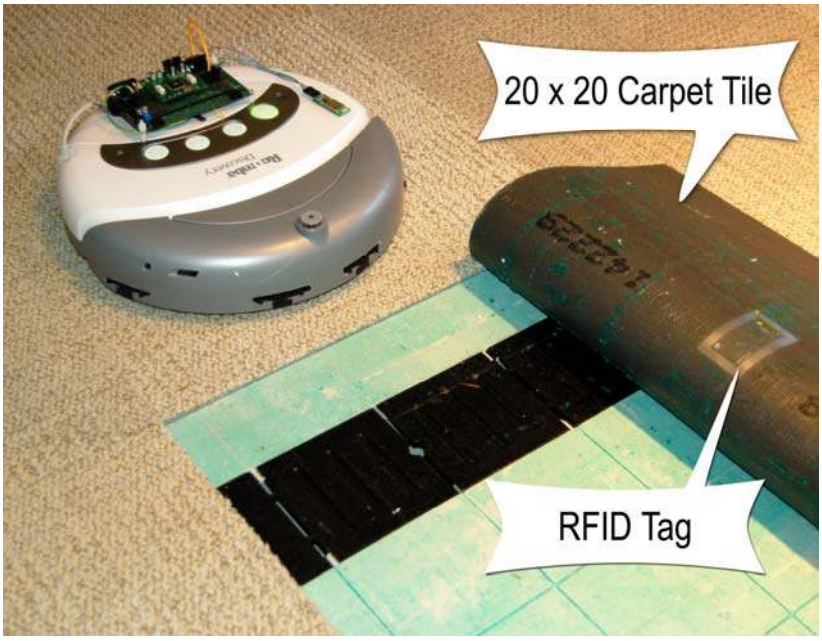


Figure 8. RFID tag in the middle of a carpet tile

Locating Important Landmarks

In the previous section we discussed how SensoBot can produce a 2D floor plan of the space. It is clear that such an operation will take time due to the fact that SensoBot runs on batteries and it needs to be recharged from time to time. The other reason is the fact that such an operation is not error free. There will be a certain amount of error that needs to be fixed over time.

Figure 9 shows a sample floor plan that should be produced by SensoBot. The white area represents the area where SensoBot was able to read the tags.

This map will be used to locate important landmarks and objects in the space. Landmarks are fixed and cannot be moved from one place to another like doors, windows, and power outlets. Recall that SensoBot has two RFID modules. The one pointing forward is the one used to complete this task. The idea is to place RFID tags on

the wall base below the landmarks that we want to describe. For example, Figure 10 shows two RFID tags located on the wall base of a door, one on the right side and the other on the left side of the door. When SensoBot reads the tags, it will know that this is a door side and it will also know which one is the opening side. Using the RFID tags in the carpet, SensoBot knows its approximate location, hence the door's location.

Figure 9. Floor plan produced

Tag-it HF RFID tags are used in this project. Each tag can hold up to 256 bits (32 bytes) of user data. There are 7 user blocks 32 bits (4 bytes) each. Each tag has a unique 32 bits ID that cannot be changed by the user.

SensoBot can identify different landmarks using the format shown in table 1 (the complete table is shown in table 7 in appendix A). The first column describes the landmark and the second and third column describes the data stored on the first and second byte respectively. For example, a front door can be located if SensoBot encounters either [81,1] or [81,0] where 81 is the first byte in block 0 and 0/1 is the second byte in block 0.

Figure 10. RFID tag describing a door

Table 1. Landmark Identification

Landmark	**Block 0 First Byte**	**Block 0 Second Byte**
Door (Opening Side)	00000001	00000001
Door (Fixed Side)	00000001	00000000
Power Outlet	00000011	-
Dynamic Landmark	11111111	-

Algorithms

In the previous section we discussed the two applications that SensoBot is able to run. The first one is creating a floor plan of the space. The second application is finding important landmarks. Although we discussed how SensoBot will perform such a task, we did not talk in details about the algorithms that guarantee a successful outcome. In this section we will discuss the algorithms for both applications.

Space Mapping Algorithms

Choosing the best algorithm is not an easy task due to the fact that we are not dealing with pure software algorithms anymore. Many external factors have to be included in determining the efficiency of the algorithms. Factors include time to complete, power consumed, and maximum area covered.

We will use the term tag to represent a tile in the algorithms discussed later. Each tag has four adjacent tags that we call neighbors as shown in Figure 11. Let us assume that all tags are initially colored white. A tag is colored gray if it is visited but not all neighbors are visited yet. A tag becomes black when all its neighbors are visited. We start by discussing a naive algorithm in the following section. Later we will discuss some improvement that can be made to achieve better results.

Naive space mapping algorithm

The first algorithm is based on the Breadth First Traversal algorithm. We will call it Breadth First Mapping (BFM). We assume in BFM that SensoBot is able to get from one tag to another without errors. We start at a tag t and mark it as having been reached. The tag t is at this time said to be unexplored. A tag is said to have been explored when SensoBot has visited all tags adjacent from it. All unvisited tags adjacent from t are visited next. These are new unexplored tags. Tag t has now been explored. The newly visited tags have not been explored and are put onto the end of a list of unexplored tags. The first tag on this list is the next to be explored. Exploration continues until no unexplored tag is left. The list of unexplored tags operates as a queue and can be represented using any of the standard queue representations. Figure 11 shows the four adjacent tags (neighbors) of the origin in the middle. The numbers on each tile shows the sequence that the algorithm follows to traverse all the adjacent tags.

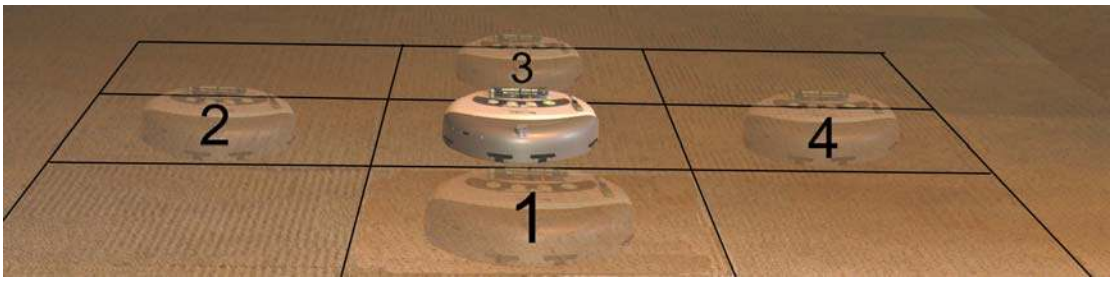


Figure 11. Finding the 4 neighbors of each tag

Following is a pseudocode of our first algorithm:

Algorithm 1 BFM

- Look for a tag and make it the origin, mark it visited, and place it in a queue.
- **While the queue is not empty**
  - Take a tag, t, from the queue, and visit it
  - For all neighbors, n, of t
    - If n has not been visited or explored
      - Mark it as visited
      - Insert n into the queue
  - t is now explored

Although BFM will guarantee that SensoBot visits all tags it is an expensive and time consuming one. The fact that this algorithm is based on the traditional breadth first search algorithm makes it time consuming and repetitive. SensoBot has to visit all neighbors of a tag before moving to the next one. If a tag has only one neighbor that is not visited yet, SensoBot has to visit all four tags instead of only visiting the not-visited tag. The second disadvantage in BFM is the fact that SensoBot runs on batteries. If, for example, this algorithm reaches level five, the distance that SensoBot has to travel back and forth could reach up to 20 feet.

It is obvious by now that we need a more efficient algorithm that can eliminate the repetition and minimize unnecessary traveling.

Improved space mapping algorithm

To eliminate revisiting tags, we need to look at a more efficient algorithm. The new algorithm is based on the Depth First Search Traversal and we call it Depth First Mapping (DFM).

Algorithm 2 DFM

- Mark the current tag t visited, and push it to the stack
- **While the stack is not empty**
  - Drive to the next tag
  - If path is open
    - Push the tag to the stack and mark it visited
  - Else if path is blocked
    - Check for open path in different direction
    - If none exist, pop the stack and backtrack.

In this algorithm too, we assume SensoBot is able to move from a tag to its neighbor with no errors. In DFM, SensoBot finds the first tag and marks it as origin. Next it moves in one direction and pushes each visited tag into a stack. When blocked, it changes direction choosing the one that might lead outwards. If all three neighbor tags are either inaccessible or visited, SensoBot pops a tag from the stack and backtracks to that tag and tries to find an open path. SensoBot finishes when the stack is empty.

DFM is faster and more efficient than BFM. This is due to the fact that SensoBot does not look for the neighbors of each tag before moving to the next one. On the contrary, it keeps on moving in one direction. By default, using this approach, SensoBot will discover the neighbors of most of the visited tags automatically.

RFID hint tags

Using previously mentioned algorithms, SensoBot is able to successfully map a space in a timely manner. This process can be made faster and more efficient by using what we call **RFID hint tags.** An RFID hint tag is simply a tag that stores information useful to SensoBot to determine some basic information about the space. Following are some examples of RFID hint tags:

- Room type: This is an informative tag that gives information about the room type SensoBot is in. Values could be Kitchen, Bedroom, Hallway, etc.
- Wall: A tag that tells SensoBot that there is a wall on the right or left side.
- Corner: SensoBot knows that it is already in a corner. The tag could specify whether the corner is on the left or right side.
- Room Size: Using this tag, SensoBot will know how big the room is. Values are width x length.

One might use none of the hint tags mentioned here, or use all and even add more. For simplicity purposes and to make things easy to install, the best scenario is not to use any. In case time is an important factor, we can use those hint tags to minimize the time needed to map the space.

More improvements

SensoBot, as mentioned before, uses a digital compass to determine orientation. Digital compasses are able to give exact heading most of the times. However, under certain circumstances, the compass might give false headings when soft or hard-iron distortions are present. **Hard-iron** distortions are caused by permanent magnets and magnetized steel or iron object within close proximity to the compass. **Soft-iron**

distortions are the result of interactions between the earth's magnetic field and any magnetically "soft" material within close proximity to the compass. How can SensoBot find the exact direction when faced with such a distortion? There is one type of compasses that can solve this problem. This compass has a built-in gyroscope that has the ability to compensate for short term transient magnetic disturbances, keeping the heading output accurate, even in the presence of these unexpected magnetic fields.

SensoBot uses a different approach to fix any errors. If lost, SensoBot simply starts looking for a tag; if the tag is already visited, (SensoBot knows from the tag's unique serial number) it resumes running the algorithm. However, if the tag is new (not visited before), SensoBot restarts the algorithm starting from the new tag as the origin. If encountered with a previously visited tag, SensoBot combines the two occupancy grids and continues filling the new combined grid.

The improved algorithm is called IDFM and is described in Algorithm 3. There is one extra improvement over DFM which saves more time during the backtracking process. In DFM, SensoBot used to backtrack to the popped tag before checking for an open path from this specific tag. In IDFM, each time SensoBot pops a tag, it checks for an open path before driving to that tag. If it happens to be an explored tag (all neighbors are visited), it pops another tag. SensoBot repeats this process until it finds a tag that is not explored. At this moment, it backtracks to that tag. In small areas this change will not significantly improve the performance; however, in large areas, a big percentage of tags in the stack become explored before they get popped. Checking the status of those tags before driving to them saves a significant amount of time and power.

Algorithm 3 IDFM

- Mark the current tag t visited, and push it to the stack
- **While the stack is not empty or not lost**
  - Drive to the next tag
  - If path is open
    - Push the tag to the stack and mark it visited. Increase S_T
  - Else if path is blocked
    - Check for open path in different direction
    - If none exist, keep popping the stack until you find a tag that has an open path.
    - Backtrack to the tag found. Increase S_T
- If lost, save old data and restart. When old tag is found, merge two grids and continue.

Backtracking

Whenever a path is totally blocked, meaning that all four neighbors are visited, SensoBot has to backtrack to the tag on top of the stack. SensoBot uses Dijkstra's algorithm to find the shortest open path between the origin (SensoBot's location) and the destination (the tag on top of the stack).

Locating Landmarks

In the previous section the main task was to map a space and create a corresponding floor plan that can be used in different applications. One of the applications that uses this floor plan is the one that locates landmarks. Locating landmarks is, like mapping the space, done only once.

As we mentioned earlier, SensoBot is equipped with two RFID readers. The reader pointing forward is used to perform the task of locating landmarks.

There are several types of landmarks that exist in a space. Currently we are interested in only the following ones: Doors, Windows, Closets, and Power outlets. Locating power outlets is useful to the smart plug, which we will discuss in chapter 5, where a smart space can automatically identify what is plugged in a specific power outlet. Knowing the location of the outlets, the space will be able to locate the device plugged.

Each landmark to be located uses one tag installed at the wall base below or next to the landmark. For example, below each power outlet, we stick one RFID tag on the wall base. Using the floor plan created, SensoBot can move along the walls and look for those tags. Each time SensoBot finds a tag, it will use the tags in the carpet to find the corresponding location.

When complete, the smart space will be capable of using a digital floor plan that can be used in many applications like remote monitoring, emergency evacuation, and many more.

Performance Evaluation

We evaluated SensoBot's performance in the Gator Tech Smart House which is the perfect environment to do such evaluation. The fact that it is a real house with real furniture brings to this research accurate results that can be used later to decide the feasibility of this approach. All three algorithms, BFM, DFM, and IDFM, were used during the evaluation process.

We installed tags in the living and dining areas, which consist of about half the size of the house. Figure 12 shows the dining area with part of the living room area. There are 207 carpet tiles in both areas, with one RFID tag in each tile. The size of the carpet tile is 20 inches. As shown in Figure 12, the yellow shaded area represents one carpet tile with one RFID tag in the middle.

Figure 12. RFID grid in the GTSH

We removed all furniture from the two areas except for three items (two are shown in Figure 12). SensoBot started from one corner where the home base is located. The first tag it encountered was used as the origin ([1,1] coordinates).

Figure 13 shows the chart were results from all three algorithms were plotted. The pink line represents BFM which was able to cover the 207 tags in about 69 minutes. The green line represents DFM. Using this algorithm SensoBot was able to find all 207 tags in about 42 minutes. The brown line represents IDFM. SensoBot covered all 207 tags using this algorithm in about 26 minutes.

IDFM did indeed improve the execution time from 69 minutes, which is BFM completion time, down to 26 minutes, the time taken by IDFM to cover the entire RFID grid.

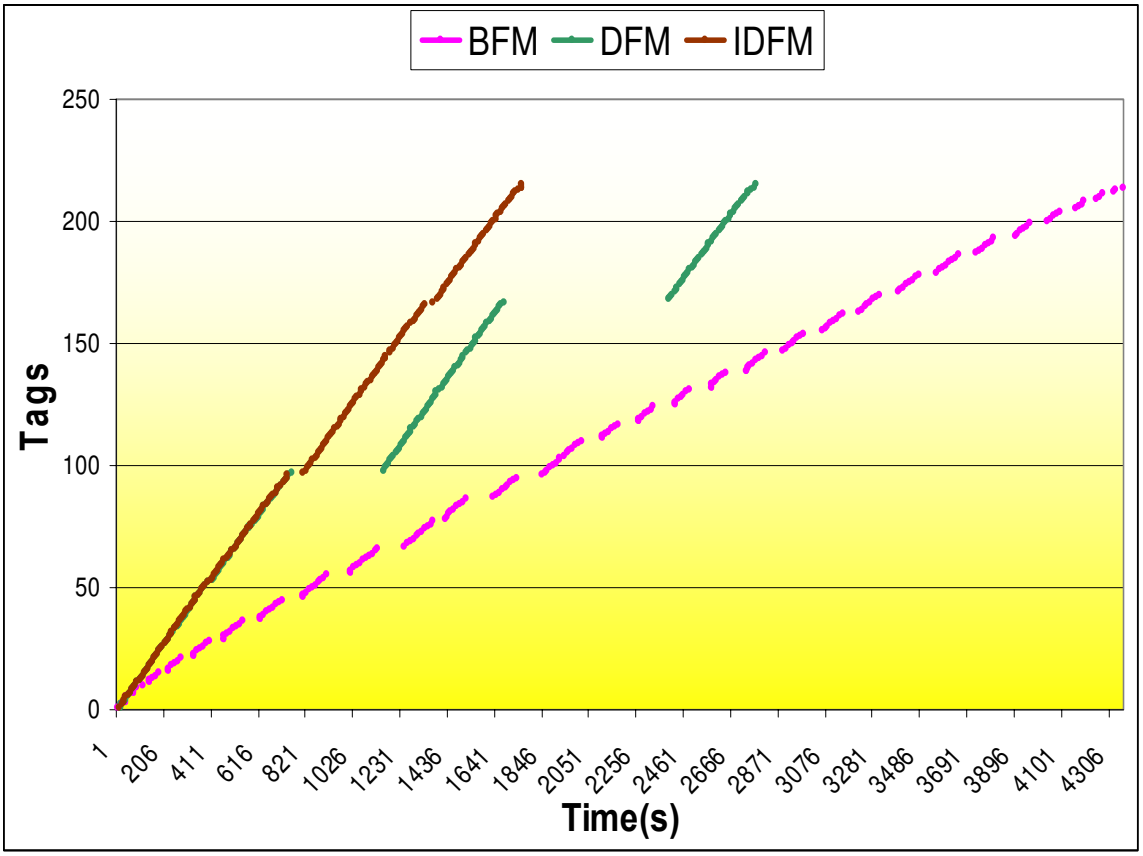


Figure 13. Performance Evaluation

SensoBot uses WI-FI to send all results to a desktop application that keeps track of all tags found and draws the floor plan on the fly. The progress of mapping the living and dining areas is shown in Figure 14. Notice how SensoBot starts from the corner and uses the improved algorithm discussed earlier to draw the floor plan of the house. The result is shown in the bottom right part of Figure 14.

Figure 15 shows the resultant floor plan superimposed over the real floor plan of the house. Notice that some areas, especially on the borders, were not recognized by SensoBot. This is caused by either the furniture which blocked the way, or because the tile was too small to have a tag on it (border tiles). Later, in chapter 6, we will discuss how furniture can be located using SensoBot.

Figure 14. Mapping progress

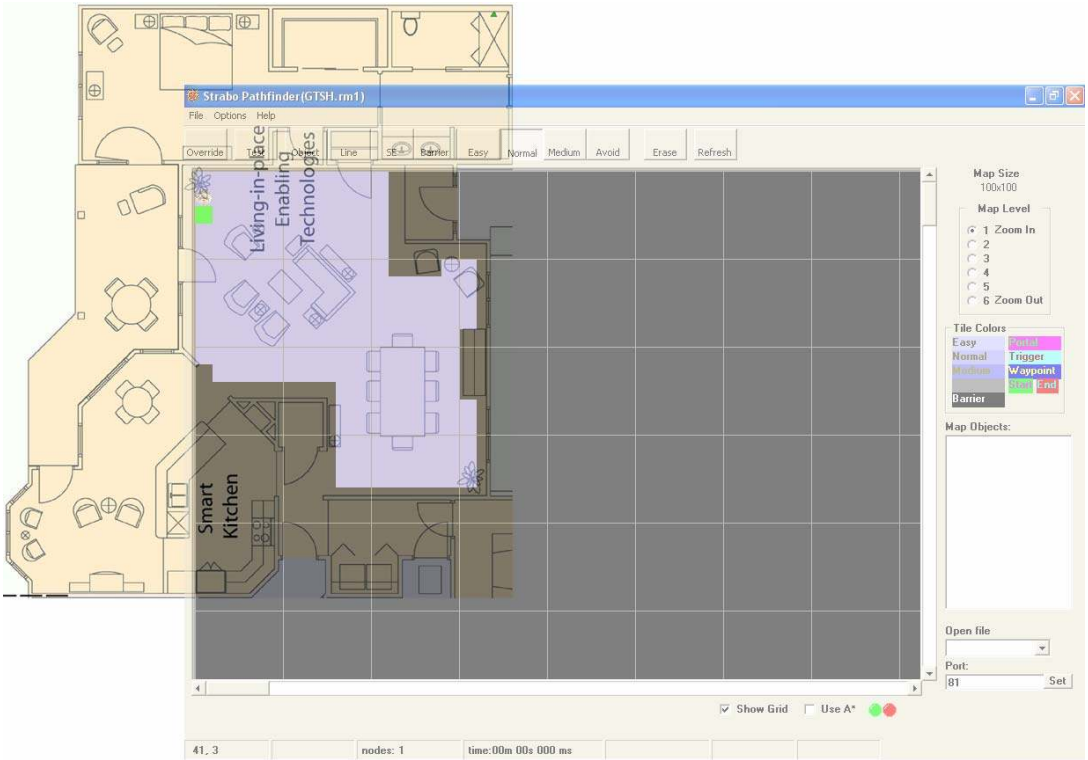


Figure 15. Floor plan produced superimposed over the real floor plan of the GTSH

Analysis

We discussed in previous sections several mapping algorithms. However, based on our experiments, IDFM proved to be the fastest among those algorithms. In this section we will discuss the theoretical formulation of IDFM and analyze its properties. For purposes of analysis we will assign a value S_T for each tag t representing the number of times SensoBot read tag t ($S_T = 2$ means that SensoBot have seen t twice). Initially, $\forall$ t $S_T = 0$. Each time SensoBot drives over a tag, it increases S_T as shown in Algorithm 3 in steps 5 and 9. S_T is thus purely an aid to our analysis of the algorithm. In particular we discuss the following two properties

1. Completeness of IDFM (eventually terminates)
2. Performance of IDFM (when used with SensoBot)

The Graph Model

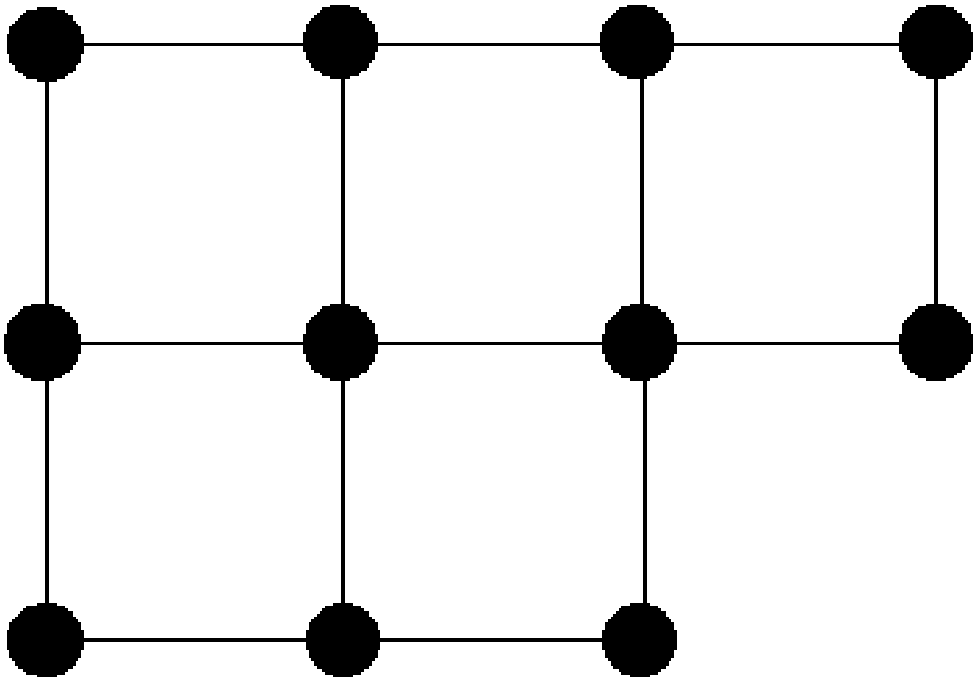


Figure 16. Example graph

For purposes of analysis, consider a bounded environment with or without obstacles. In this case, given our mapping algorithm IDFM described previously, we can model the steady spatial configuration of the tags as a finite graph $G = (V,E)$, where V is a set of vertices (RFID tags) and E is a set of edges such that $\forall i, j \in V$ there is an edge between i and j iff there is an open path between i and j.

In addition, a vertex can have at most 4 edges connected to 4 separate vertices as shown in Figure 16. Before discussing the theoretical properties of IDFM we provide the working definition for coverage on graphs and corresponding performance metric.

Definition 1: *Coverage on a graph is the act of visiting every vertex of a graph*.

We say SensoBot covered all tags when it visits each tag at least once. The performance of a coverage algorithm is measured using the cover time metric defined as follows:

Definition 2: *Cover time is measured in terms of the number of vertices traversed such that every vertex of a graph is visited at least once, i.e. the graph is covered.*

Note that in order to cover a graph, SensoBot needs to traverse at least one edge per tag.

IDFM Completeness and Cover Time

Given the graph model we formally exhibit one important property of IDFM in which we show that IDFM is complete.

Completeness of IDFM on finite graphs

Theorem 1 (Completeness): *The coverage time of IDFM on a finite graph is finite.*

Proof: The goal is to show that IDFM traverses every node of any finite graph in finite time. Since IDFM is based on DFS and since the graph is finite and connected, we conclude that IDFM will terminate in a finite amount of time.

Performance of IDFM

First let us define L as the lower bound cover time of a hypothetical optimal algorithm in this context. The first property of such an algorithm is the ability to explore a graph $G = (V,E)$ in $O(|V|)$ time. In addition, this algorithm traverses to each node once and only once and explores the graph without backtracking. In this section we will compare the performance of IDFM to L. For the purpose of this analysis we conducted simulation experiments running IDFM on a total number of vertices (tags) equal to 324. For each experiment two parameters where supplied:

- B_p = percentage number of blocked vertices.
- B_d = distribution of blocked vertices in the graph.

To clarify both B_p and B_d, consider the example of a room where the floor is covered with the RFID grid. Suppose the number of tags in this grid is 324. if $B_p = 10\%$ then there are 32 tags on the floor covered by some furniture object which would make it impossible for SensoBot to reach those tags. If $B_d = 1$ then those 32 tags are covered by either one object or multiple, close to each other, objects. However, if $B_d = 2$ then there are two separate, away from each other, areas occupied by those objects covering those tags.

The performance of IDFM is given by $(1+\varepsilon)L$ where ε, which is the performance factor difference between IDFM and the lower bound, was derived as follows: For each B_d and B_p combination we executed the simulation algorithm 1000 times. S_T of each node

is computed as described in Algorithm 3. After each run we get the sum of all S_Ts and divide by the total number of nodes as shown in Equation 9 where S_T is the number of times SensoBot reads tag t.

$$\varepsilon_m(B_p, B_d) = \frac{\sum S_T}{NumofTags} - 1 \qquad (9)$$

In equation 9, m is the permutation number of the simulation for each B_p and B_d combination where $1 \le m \le n$ (n=1000).

$$\varepsilon(B_p, B_d) = \frac{\sum_{1}^{n} \varepsilon_i}{n} \qquad (10)$$

To compute ε, we use Equation 10 where the sum of nε is divided by n.

Back to our experiments, the list of possible values of B_p is (5,10,15,20,25,30) and the list of possible values of B_d is (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16). The reason why we did not go further beyond 30% for B_p is because we assumed that a typical room will have 70% of its floor accessible (considering that SensoBot could reach below some furniture objects). For each combination, we executed the simulation algorithm 1000 times, each time starting from a random position and random direction.

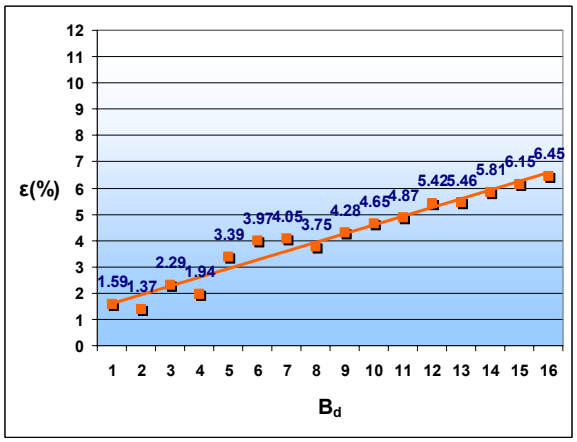


Figure 17. B_p = 5%

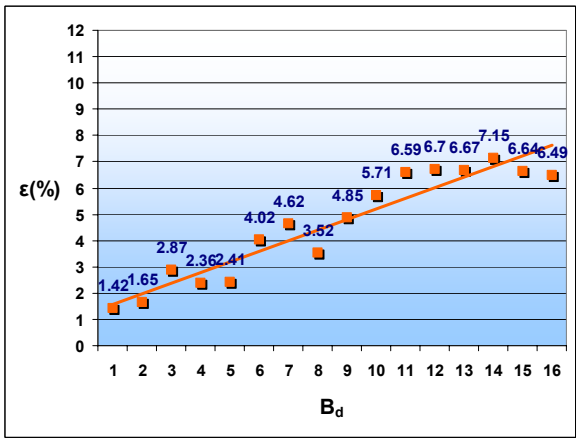


Figure 18. B_p = 10%

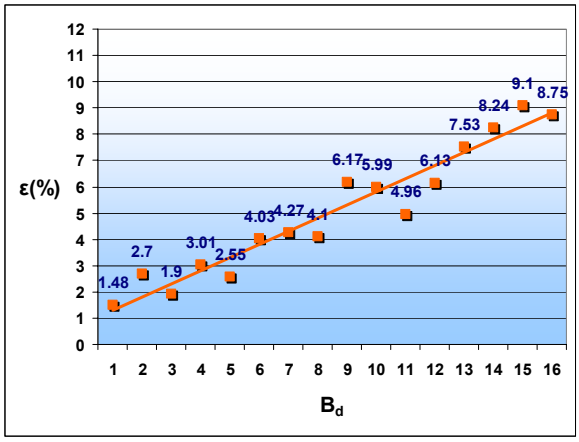


Figure 19. B_p = 15%

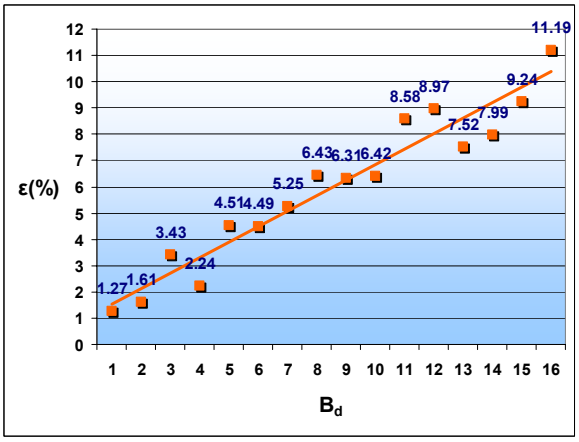


Figure 21. B_p = 20%

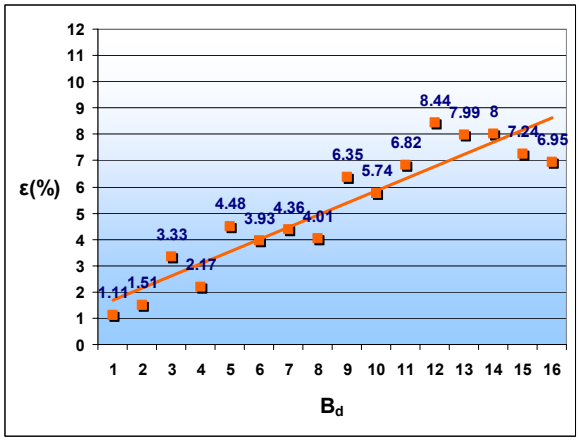


Figure 20. B_p = 25%

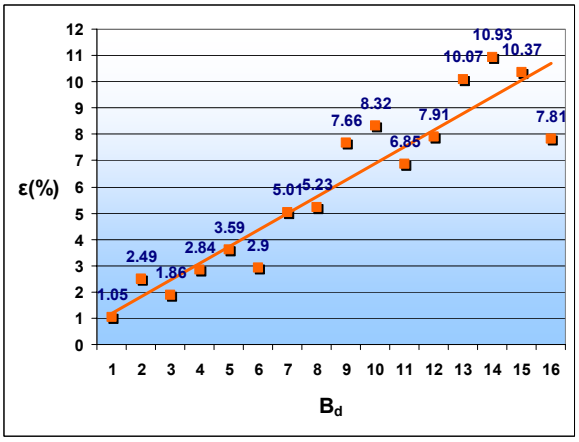


Figure 22. B_p = 30%

The results of our simulation are shown in figures 17 to 22. Starting from B_p = 5% in Figure 17 to B_p = 30% in Figure 22. The charts show the value of ε affected by the change of B_p and B_d. Back to the charts showing in figures 17 to 22, $\varepsilon(B_p,B_d)$ is plotted and the linear trendline is displayed. It is clear from the displayed results that ε increases with the increase of B_p and B_d. Notice how the starting value of ε is almost the same independent of B_p. To further characterize B_d and B_p, the value of B_d defines the amount by which ε increases while B_p defines the slope. Figure 23 shows all the results combined in one chart.

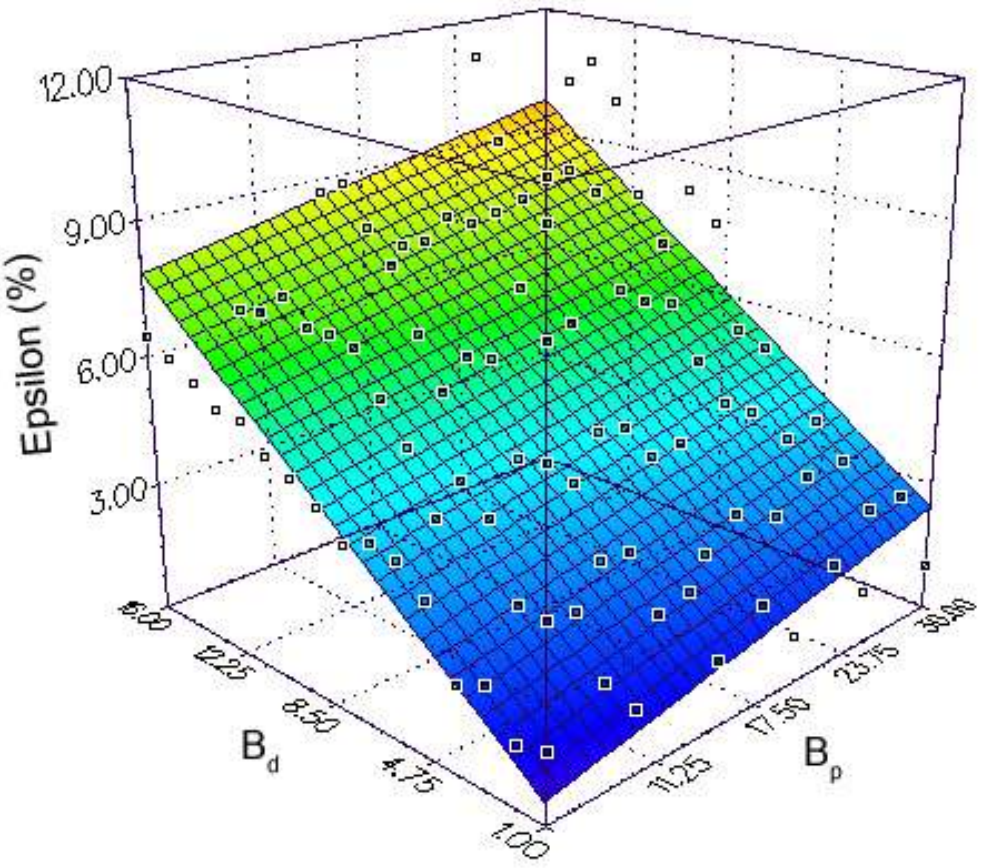


Figure 23. Combined results

As we mentioned before, the complexity of IDFM is $O(|V|)$ and the performance is given by (1+ ε)L. Whenever $\varepsilon = 0$, IDFM performance will be equal to the lower bound L due to the fact that $\varepsilon = 0$ means that IDFM managed to visit all nodes once and only once. As ε increases, IDFM performance decreases by εL. The maximum value of ε (ε = 11.19% or 0.11) produced by the simulation experiments was when B_p = 25% and B_d = 16.

We used MATLAB to compute a linear equation that best fits the set of data produced by the simulation experiments. The method used is curve fitting where the equation produced would be of the form $z = a + bx + cy$. The results were as follows:

Coefficients for Simplified Linear

$\varepsilon = a + b*B_p + c*B_d$

Fitting target is lowest sum of squared (SSQ) absolute error

sum of squared absolute error: 78.8684417139

a = −3.4372916666666542E−01

b = 7.6467857142857198E−02

c = 4.8691421568627452E−01

Based on the results produced by MATLAB, we formulate the following linear equation

$$\varepsilon = 0.076467 B_p + 0.486914 B_d - 0.343729 \quad (11)$$

Equation 11 is used to approximately predict how IDFM will behave in the presence of a certain B_d and B_p. For example, assuming $B_d = 7$ and $B_p = 15$, using Equation 3 we get $\varepsilon = 0.0421$ or 4.21%. Looking at the results from our previous experiments we notice that $\varepsilon = 4.27$ which is pretty close to the result produced by the equation.

CHAPTER 5
LOCATING ELECTRICAL DEVICES USING THE SMART PLUG

Our primary goal in the field of self-sensing spaces is to be able to sense any changes made in the space. For example, the smart space should be able to recognize that a new lamp is being installed; moreover, it should know the lamp's location and finally it should be able to control it. Since most electrical appliances require electricity, we use their power plug to locate them. We call it the smart plug and it works as follows: First we locate all power outlets in the space, then we sense when a new device is plugged in any of the registered outlets. We recognize a plugged device using RFID. Finally we use OSGi to register these devices in the framework [45-47]. In the next sections we discuss the methods and techniques used to implement the smart plug idea.

Smart Plugs for Self-Sensing Spaces

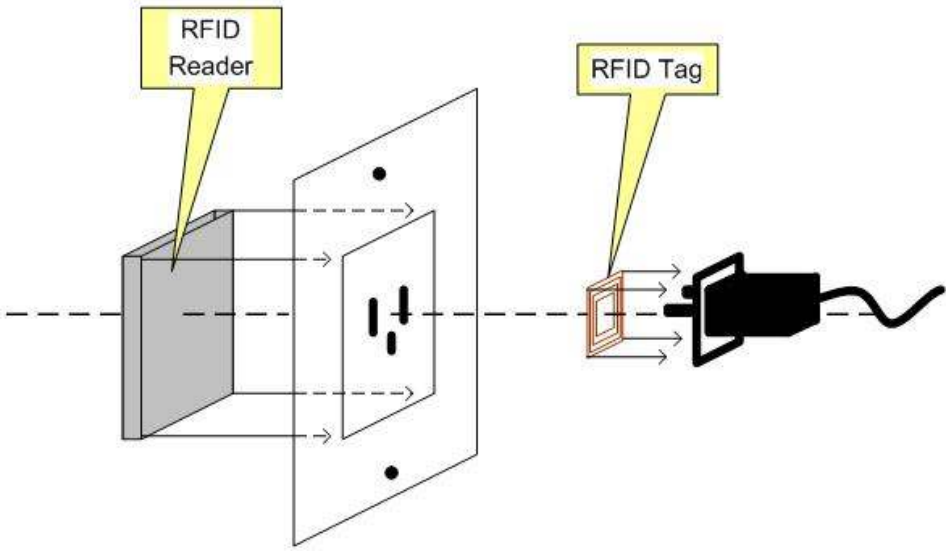


Figure 24. Outlet equipped with a low cost RFID reader and plug with a RFID tag.

Power outlets in self-sensing spaces are equipped with low cost RFID readers connected to a main computer. Electrical devices (equipped with power cords) like lamps or radios have an RFID tag on the plug. This tag contains information about the device (Figure 24). Whenever plugged into the outlet, the reader reads the tag, gets its contents, and sends it to the main computer which is an OSGi platform. Using this information, the main computer can directly identify this device.

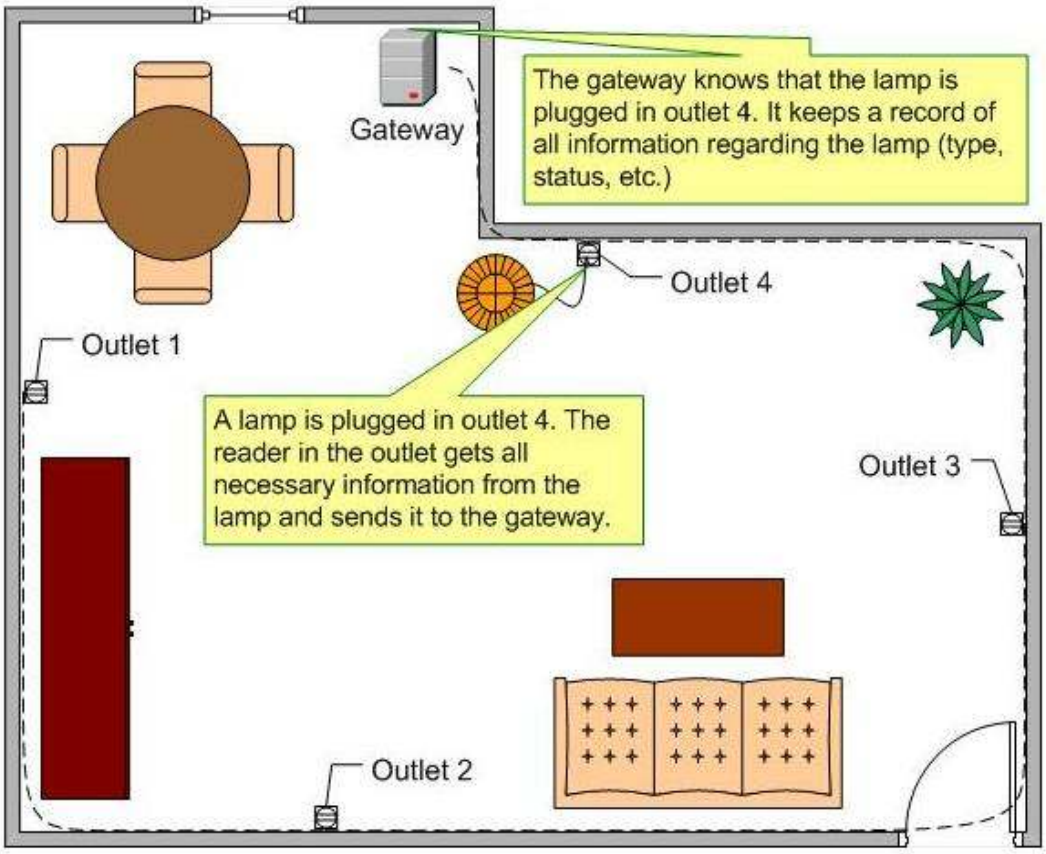


Figure 25**.** Detection and localization of a new device is done by getting the ID of the power outlet where the device was plugged.

As we discussed in previous chapter, one of the main tasks of SensoBot is to locate power outlets. Knowing the location of each power outlet, the computer will know approximately where the new device is plugged. Moreover, the computer will be able to control the new device using the information stored on the tag (Figure 25).

As we presented, the smart plug concept consists of two major parts: OSGi and RFID.

OSGI (Middleware)

We have developed a middleware layer (based on OSGi) that consists of several bundles that provide a framework for pervasive computing. The OSGi framework allows us to represent every physical device (sensor as well as actuator) as an individual bundle.

New devices installed, in the smart space, are represented by OSGi bundles. A bundle is simply a Java archive file (jar) containing interfaces, implementations for those interfaces, and a special Activator class. The manifest file contained within the jar file includes special OSGi specific headers which control how the bundle will be used within the framework.

Apart from bundles for physical devices, we have the following additional bundles:

- Floor plan: This bundle contains all the location information of the house. It contains a floor plan of all the individual rooms and position information of every device available. It is able to map coordinates to rooms and vice versa. This allows us to perform location based service discovery. For example, we can request every device in the kitchen.

- Sensor: This bundle contains a framework for automated installation of sensors. We make extensive use of the wire admin API. Every sensor is a producer that can produce a value.

- Context: The context bundle interprets sensory data and turns this into a high level representation. The context framework can also (de)activate behavior upon a change of observed state. For example, if the temperature changes from warm to cold we can turn on the heater.

- X10Controller: This bundle gives us basic access to X10 controlled devices.

- Resolution: This bundle takes care of installing a bundle associated with a smart plug. It also registers the new bundle with the floor plan.

Radio Frequency Identification (RFID)

In self sensing spaces, whenever an RFID reader, which represents a proxy between the device and the gateway, associated with the smart plug reads a tag, the context framework activates the resolution bundle. The resolution bundle will then inform the OSGi framework to load the bundle and activate it. When the bundle is activated it is its job to register a service that allows access to the device that has been installed.

Depending on the size of the RFID tag's memory, the bundle representing the new device to be installed in the smart space could be stored on the tag itself. If the OSGi bundle is too large, it is possible to just store a referral URL to where the gateway software can download it from. The referral URL can use any protocol, which the OSGi framework server has access to, such as http and ftp. Size might also not be the only reason why the referral concept would be used.

By having the software on a web server, upgrading the bundle is as easy as replacing the software. All the required downloading and installation of the gateway software, for the individual bundles, are performed by the gateway bundles installed in the framework. In case a URL is supplied on the tag, the framework can access bundle data using the bundle.

Installation in the Gator Tech Smart House

Converting regular houses into smart ones is still not an easy task that any resident can do. People need to hire engineers to do the job. That is why we are working on creating a "smart house in a box" where anyone can buy a smart house kit and either installs it or hires a technician to do the job.

In order to use the smart plug in the Gator Tech Smart House, we had to create a middleware device that will transform a regular plug into a smart one as shown in Figure

26. Each power outlet in the Gator Tech Smart House is equipped with a Phidget RFID reader [48], all connected through USB to the main computer.

Figure 26. The smart plug used in the Gator Tech Smart House.

The main computer contains the OSGi framework, which sits on top of a Java virtual machine, and is the execution environment for services. This computer could be an OSGi set-top box that plays the role of a gateway between the smart space and the external world. Each bundle will be downloaded and registered in the OSGi framework when the new device is installed in the smart space. The framework will be able to report the list of installed devices upon request. Any device within that list could be controlled via methods available in the bundle. Suppose that a new table lamp is brought to the house, all what need to be done is to attach a smart plug into the lamp's plug and plug it into any outlet in the house. The RFID reader of that specific outlet will read the content of the RFID tag and send it to the main computer.

Remote Control and Intervention

The user can control the lamp remotely by a simple interaction with a graphical interface (Figure 27). A click on the lamp will download all available methods associated with this lamp. If the user clicks on a method, the remote application will send a request to the gateway to execute the action. On his side, the gateway will translate the request and execute the corresponding method.

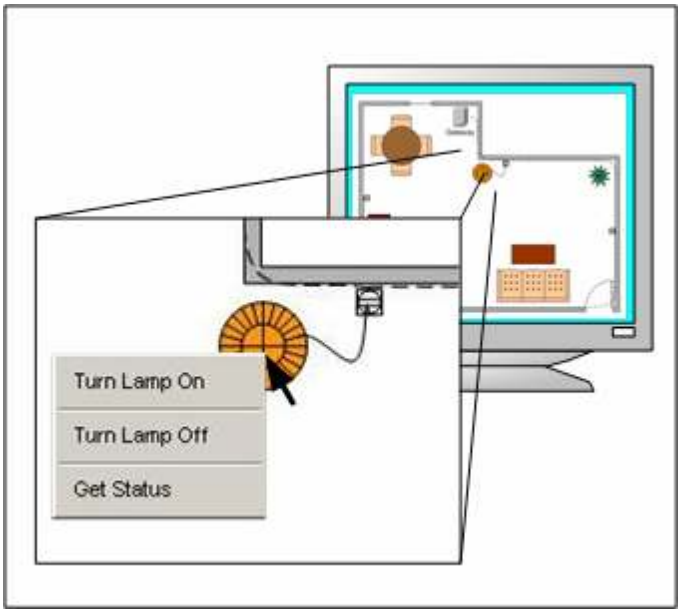


Figure 27. Information will be downloaded to the remote application on the fly. A click on the lamp will import all the available methods from the gateway.

Remote Monitoring Center

One of our goals is to enable self sensing spaces to connect to a remote center in case of emergency (Figure 28). A gateway will act as a proxy between the smart space and the center. This center will monitor and intervene in case of an emergency. Operators at the center will be able to better differentiate between false alarms and real emergencies which will reduce the number of unnecessary visits to the house.

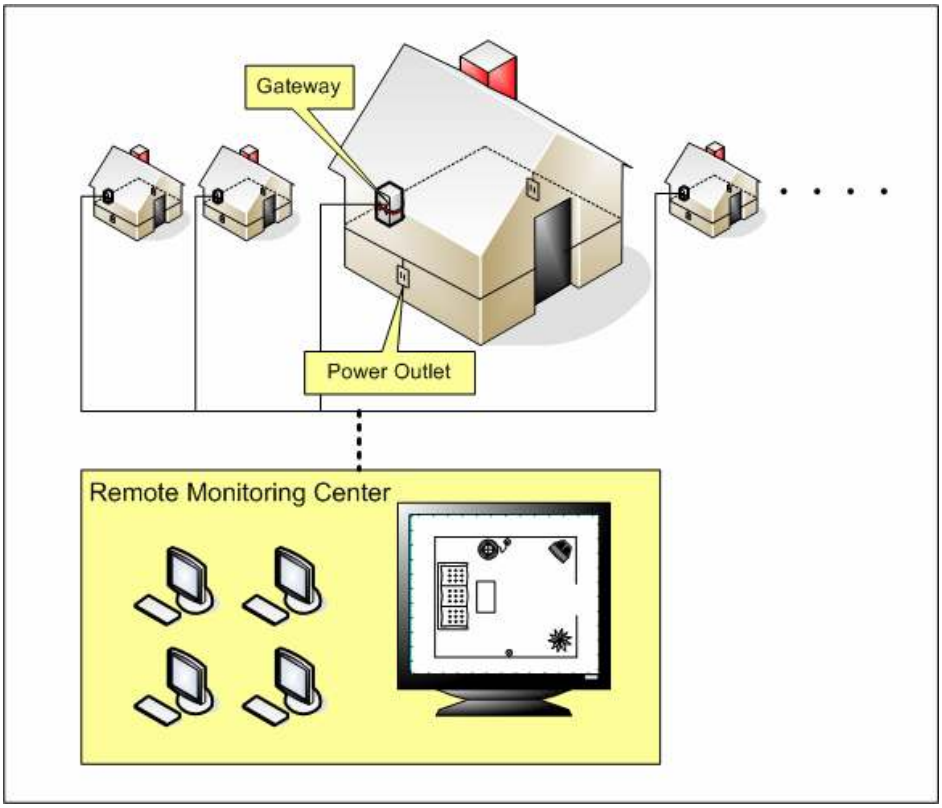


Figure 28. Self Sensing Spaces shown in a remote monitoring center concept.

CHAPTER 6
LOCATING FURNITURE USING SENSOBOT

We have seen in chapter 4 how SensoBot was able to use the RFID grid embedded in the carpet to map the space and create the corresponding floor plan. The floor plan created, as we mentioned before, will enable many applications like smart vacuum cleaning, remote emergency identification, etc. One of the applications that will also use this floor plan and described in this chapter is locating furniture.

To locate furniture we will be using RFID tags embedded in the furniture to be located. SensoBot will use an onboard RFID reader attached to an antenna pointing forward (see Figure 6 in chapter 4). Using the floor plan, whenever SensoBot reads a furniture RFID tag, it will refer to the carpet tags to approximately locate that specific object.

RFID Tags

Initially, furniture objects that need to be located should be equipped with special type of tags. Those tags are pre-configured with specific data either by the manufacturer or by a third-party. For now, we perform the pre-configuration process manually, but ultimately, we envision a barcode-like approach where all items, including furniture, come equipped with those RFID tags. How does SensoBot differentiate between carpet tags, landmark tags, and furniture tags? As we mentioned earlier in chapter 4, the first step in locating any type of landmark, be it doors or tables, is to check the tag's first byte of block 0. If the byte is equal to 11111111, SensoBot knows that this is a furniture tag. It

refers to Table 2 to know the type of this object (Table 2 is described in more details in Appendix A.)

Table 2. Furniture Tag Content

Name	**Address**
Category	Block 1, byte 0
Length	Block 1, byte 1
Width	Block 1, byte 2
Color	Block 1, byte 3
Leg Count	Block 2, byte 0
Leg Pos	Block 2, byte 1
Image	Block 3 – Block n

There are four types of information that can be extracted from the tag. The category of the object is described in byte 0 of block 1. The category could be a table, chair, bed, etc. Table 3 (Shown in more details in Appendix A) presents the possible values of the category field.

Table 3. Possible Category Values.

Name	**Value**
Table	00000001
Chair	00000010
Entertainment Center	00001000
…	…

If, for example, SensoBot reads the value 00000010 in byte 0 of block 1, it knows that it found a chair. The second type of information it can extract from the tag is the length and width found in bytes 1 and 2 of block 1 respectively. The values of both the length and the width can range from 0 to 255 inches. Finally, the color of the object (if its single colored or has a dominant color) could be found in byte 3 of block 1. The color value ranges from 0 to 255 and it represents the HUE value. An image of the object could be downloaded for later use in the desktop application. This image field stores the URL of the image starting at block 3. Finally, the Leg field stores information about which leg

of the object this tag belongs to. Values are in the range of 0 to 3 corresponding to “Front Left”, “Front Right”, “Rear Left”, and “”Rear Right” respectively.

Locating Furniture

Using all information described in the first section of this chapter, SensoBot will be able to identify any type of furniture if equipped with a tag that follows this standard. Moreover, it will be able to locate the furniture object using the tags in the carpet. Figure 29 shows the desktop tool that we used earlier to create the floor plan. At this stage, the tool is used to display the type and location of each furniture object that SensoBot finds. The tool either uses a default image to represent the object or it can download an image from the URL stored on the tag itself.

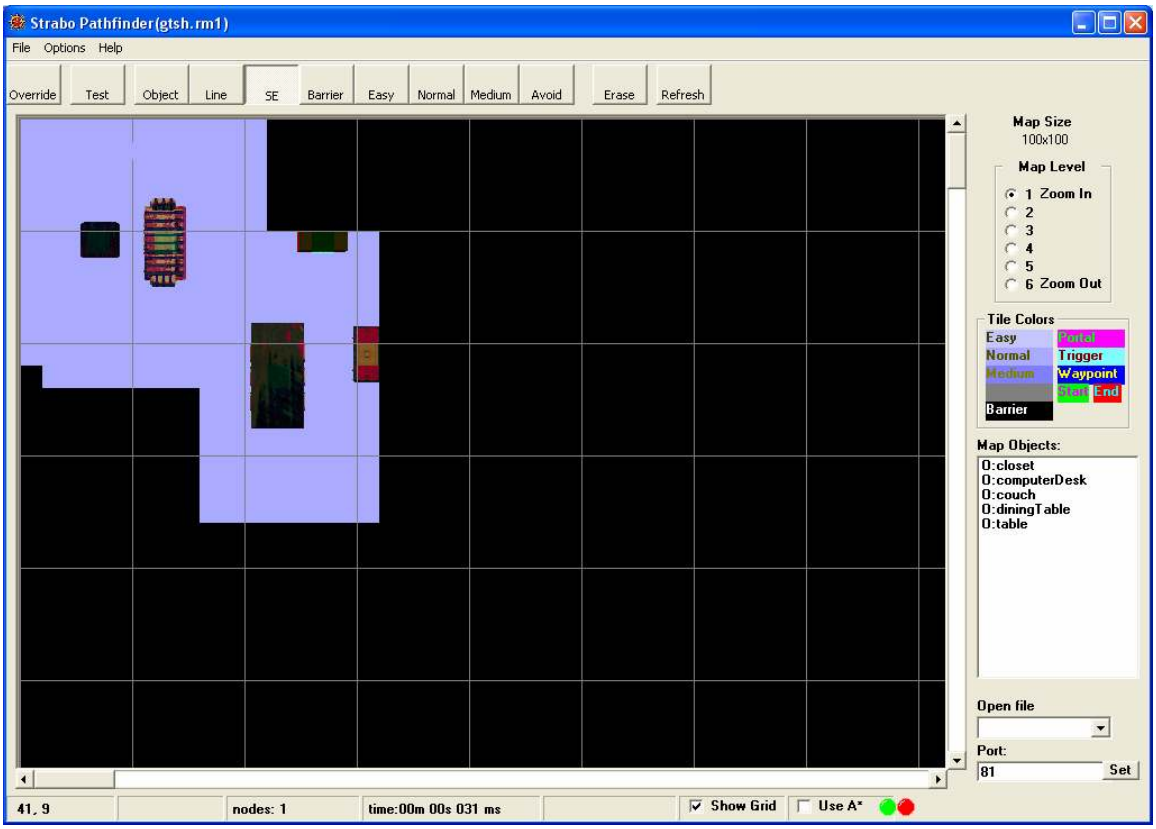


Figure 29. Displaying furniture location

One tag is enough for SensoBot to identify an object and approximately locate it. However, to know the exact location and orientation, SensoBot has to find more than one tag that belongs to the same object. For example, locating the front left and front right legs and knowing the width and length of an object, SensoBot will know the exact position of that object.

Drawbacks

SensoBot might sound an excellent, cheap, and effective way to locate furniture. However, there exist some drawbacks to this approach:

- **Availability:** SensoBot is a robot and like all other robots it operates on battery power which means it is not available all the time. Recharging might take up to three hours during which SensoBot will be unavailable.
- **Errors:** Robots are known to be reliable to a certain degree. There is a fair amount of error that can be produced during its normal operation. Even with the use of digital compasses, robots tend to lose their heading.
- **Communication:** SensoBot communicates all its data to a desktop application using WI-FI. If this communication is lost, the data will never get to the desktop application.

Until we find solutions to the mentioned drawbacks, we need to find alternate approaches to sensing dumb objects like furniture. This alternative is called Pervision and is discussed in the next chapter.

CHAPTER 7
PERVASIVE ASSISTED COMPUTER VISION

Locating furniture using SensoBot has some drawbacks as mentioned previously. To overcome those drawbacks we looked at alternative approaches that can replace SensoBot in case of its unavailability. That is why we realized we need to use some artificial intelligence techniques to sense dumb objects. Of all the branches of artificial intelligence, perhaps one of the most diverse is that of computerized vision systems. The range of applications for this relatively new, and still growing technology, is vast. From mundane tasks like production line quality control and video surveillance to exciting new technologies such as self sensing spaces needed by many applications in pervasive systems.

However, our early prototypes of vision based prototypes for sensing furniture in the Gator Tech Smart House delivered highly inconsistent performance and to a large extent proven to be unreliable. We observed though that the context in which vision algorithms are being used is unique and provides a rare opportunity to make the best out of what computer vision is able to provide. We observed that the pervasive space itself can offer assistance and can guide the basic vision algorithms to achieve the recognition and tracking goals. This is possible because the pervasive space is able to control the sensor/actuator network in the Gator Tech Smart House. By modifying the basic vision algorithms to self-score the resultant error in the recognition process, and to accordingly attempt to correct the score (lower the error below a predefined threshold) by issuing

commands to the sensor/actuator network, it was possible to adjust scene parameter and optimize conditions for effective recognition. We call this modified algorithm PerVision.

Pervasive Spaces for Computer Vision

The goal of our research is to be able to create a real time snapshot of the smart house at anytime that reflects the actual layout of salient furniture items in the house. We use this snapshot to create a realistic world model of the house. The only condition is that this process should be automatic when it comes to adding new furniture to the house. We can not rely on any human intervention every time a new furniture object is brought to the house. The smart house should be able to find this new object and track it automatically. We found it to be very difficult, if not impossible, to rely on computer vision alone to achieve this goal. We also realized that by integrating the vision system with the smart house, it will be possible to automate the recognition and tracking processes.

In our pervasive space, there exist many assistive technologies used to help the resident live safely. However, only two are related to this project: (1) the smart floor and (2) RFID technologies.

RFID Tags Describing Dumb Objects

The main use of RFID in the Gator Tech Smart House is to enable the house to know exactly when and what is being brought into the house through the front door.

As shown in Figure 30, the smart house is equipped with a gate RFID system that detects any object having an RFID tag. Figure 30 shows a chair entering the house where the reader reads the XML information stored on the tag and sends it to the smart house computer.

Figure 30. A chair detected by the RFID system in the smart house.

```
- <RFID>
  <Id>17830928EF498</Id>
  <Name>Dinning Chair</Name>
  <Description>Dining table chair with
   two arms</Description>
  <Height>35</Height>
  <Width>12</Width>
  <Length>14</Length>
  <Weight>21</Weight>
  <Area>168</Area>
  <Model>URL of the chair's model</Model>
  <MinHUE>9</MinHUE>
  <MaxHue>16</MaxHUE>
</RFID>
```

Figure 31. XML representation of a dining chair stored on the RFID tag.

The RFID tag attached to the furniture entering the house (e.g., dining chair) contains information related to that piece of furniture. This information will be used later to assist in locating the new furniture. Figure 31 shows an example XML description of a dining chair. The weight value stored in the RFID tag of the chair will be used to detect

its initial approximate location using the smart floor, which we will discuss in the next section. The MinHUE and MaxHue values will be used to give the vision system an idea of the expected color range of the chair. The model is simply an image representing the chair that will be used in the remote monitoring application.

The Smart Floor

The Gator Tech Smart House has a 2-inch residential-grade raised floor comprised of a set of blocks, each approximately 1.5 square feet in area. This raised surface simplifies the process of running cables, wires, and devices throughout the house. In addition, each block is equipped with a pressure sensor in the center as shown in Figure 32. The main benefit of such a smart floor is that it can detect any slight pressure anywhere on that block. In fact, we had to add resistors to the sensor cables to reduce sensitivity and eliminate fluctuations in the readings [40].

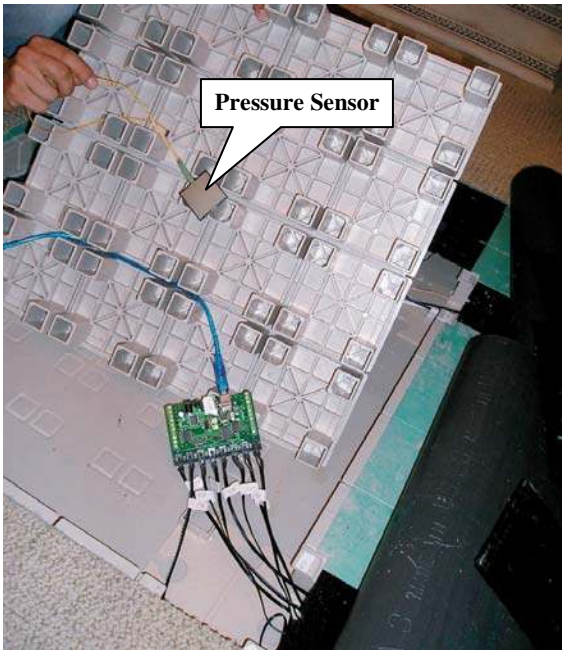
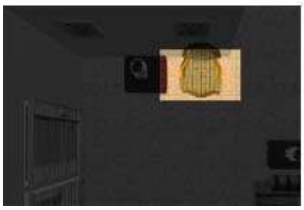


Figure 32. The pressure sensor used with each block in the raised floor.

The smart floor in the Gator Tech Smart House is used to locate the resident and help her/him in her/his daily activities. However, in this project we use the smart floor to assist the vision system with approximate location of the new piece of furniture when it first enters the smart house. Since we can detect any changes in the pressure on the smart floor tiles, we can select the tiles where the object is placed.

a- The chair is placed in the desired location

b- The smart floor locates the tiles where pressure is changed



c- Having the coordinates of the pressed tiles, the camera can now zoom in to take a picture of what lies above those tiles

Figure 33. Using the smart floor to approximately locate a new furniture object.

Let us assume that two tiles experienced changes in the pressure after the front door RFID reader detects the advent of the chair to the house. The system knows the location of each tile in the smart floor. Knowing the weight of the chair (from the XML data stored on the RFID tag), we can make sure that the change on those two tiles is the result of placing that specific chair on top of them. As shown in Figure 33-a, someone brings the chair into the house. The RFID reader is the first device that discovers that new chair. After a period of time, the system runs an algorithm that will search for the chair guided by the smart floor. Figure 33-b shows how two tiles where the chair is placed are found.

Finally, the ceiling-mounted camera in the center of the room where those two tiles are located takes an image, crops it using the coordinates of each tile, then shows only what lies above those two tiles as shown in Figure 33-c. This image will be used subsequently by the vision algorithm to locate the chair anytime it is moved. The smart house is also equipped with photo sensors that determine the level of light in each room. After taking the image, the value of the photo sensor in the room where the chair is located is saved. This value will be used later for intensity correction and error analysis.

The PerVision Algorithm

The preceding section described the first step towards locating any furniture in the house. This is done every time a new object is brought to the house. The idea is to get a top view image of that object using the same camera under the same lighting condition. We will call this operation creating a signature of that object. The object's signature will be used in the PerVision algorithm described later in this chapter.

Creating Signature

Figure 34 shows the algorithm applied to the resultant image from the previous section:

1. After locating any new furniture object using the smart floor, the camera takes an image which we call a source image.
2. It then crops it using the coordinates from the smart floor to obtain a top view for that furniture object alone.
3. Blurring this image is useful because it reduces variance at spatial frequencies below those that characterize a gross object, yet can preserve color information throughout the image.
4. The next step is to extract the Hue values of each pixel.
5. We then create a Hue histogram of that image.

6. Finally, we use the Gaussian distribution to get the min and max Hue values which correspond to: μ-2σ and μ+2σ where μ is the mean value and σ is the standard deviation. These operations are performed only once, and then saved with the profile of the furniture object.

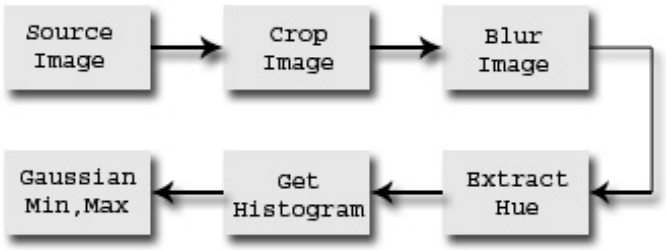


Figure 34. Performing one-time operations on the result image

Locating Furniture Using Color Restriction

In order to locate furniture in the smart house, we had to use color restriction algorithms. We can let a color (multispectral or hyperspectral) image **a** have *L* spectral bands, which causes each pixel value a(**x**) to have *L* spectral values, i.e., a(x) = (a_1, a_2,…,a_L), where x is a point in an *M*x*N*-pixel domain X. If each of the L band images $a_K \in \mathbf{Z}_m^{\mathbf{x}}$ has *m* graylevels, then **a** can be expressed in terms of the Cartesian product ∏, as follows

$$\mathbf{a} = \left\{ (\mathbf{x}, \mathbf{a}(\mathbf{x})) : \mathbf{a}(\mathbf{x}) \in \prod_{k=1}^{L} \mathbf{Z}_m, \mathbf{x} \in \mathbf{X} \right\} \qquad (12)$$

This formalism provides powerful insights that support design of pixel-level segmentation algorithms for multispectral imagery. For example, one can employ the characteristic function χ to compare each pixel value in each band with the extrema of the corresponding interval in an *L*-element spectral signature **s**=([s_{1L},s_{1U}], [s_{2L},s_{2U}], ,[s_{LL},s_{LU}]). Note that the symbol *L* in s_{kL} denotes the lower bound of the real-valued interval, and should not be confused with the number of spectral bands *L*, which is in italic typeface. Since one can also express s as the Cartesian product

$$\mathbf{s} \in \prod_{k-1}^{L} \mathbf{Z}_m \qquad (13)$$

it is possible to produce a segmentation b of image a constrained by signature s, which portrays the number of bands in each pixel value a(x) that match the signature s. For example, given the set of test values G and the characteristic function

$$\mathbf{b} = \chi_{\mathrm{G}}(\mathrm{a}) = \left\{ (\mathbf{x}, \mathbf{b}(\mathbf{x})) : \mathbf{b}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{a}(\mathbf{x}) \in \mathbf{G} \\ 0 & \text{otherwise} \end{cases}, \mathbf{x} \in \mathbf{X} \right\} \qquad (14)$$

Consider the following expression

$$\mathbf{b}(\mathbf{x}) = \sum_{k=1}^{L} \chi_{[s_{k\mathrm{L}}, s_{k\mathrm{U}}]}(\mathbf{a}(\mathbf{x})), \mathbf{x} \in \mathbf{X} \qquad (15)$$

where each pixel $0 < b(x) < L$, with a higher number indicating a better match. It is also possible to combine the band-specific matches described above by performing logical operations such as ‘and’, or, ‘xor’ (exclusive or). For example, logical ‘and’ can be implemented as

$$\mathbf{b}(\mathbf{x}) = \prod_{k=1}^{L} \chi_{[s_{k\mathrm{L}}, s_{k\mathrm{U}}]}(\mathbf{a}(\mathbf{x})), \mathbf{x} \in \mathbf{X} \qquad (16)$$

since the characteristic function χ returns a one or zero.

Let us discuss how color restriction can be implemented in practice. Firstly, note that the RGB representation of a digital color image is not intuitive: customarily, humans do not consciously think of colors in terms of how much red, green, or blue comprises a particular color. Rather, we learn colors by their names, such as red, brown, yellow, etc. In scientific terminology, these color names are called hues.

This brings us to our second observation, namely, that there is a more convenient color representation called hue, saturation, value (HSV) that makes color selection more

human friendly. Because HSV and RGB representations are equivalent, it is relatively easy to convert between them.

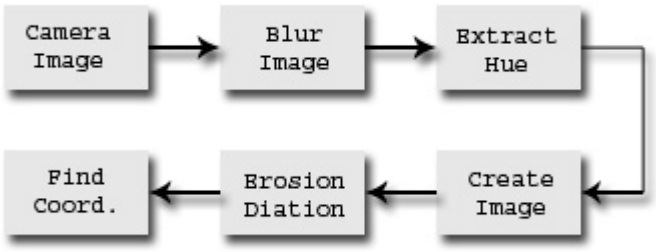


Figure 35. Locating a furniture object using PerVision

Now that we have a unique signature for each furniture item in the house, we can start locating those objects one after the other. To be able to locate those objects we will use the PerVision algorithm depicted in Figure 35.

1. The first step is to get real-time image from the ceiling mounted camera.
2. Blurring is also applied to this image.
3. The next step is to extract the Hue values of each pixel in that image.
4. Using the Min and Max values from the previous section, we now create a new image where every pixel in the old image outside the range [Min, Max] is not included. This will result in a black and white image where the white pixels corresponds to the furniture object we are looking for.
5. There will be some noise in the image which will be removed using Erosion and Dilation.
6. Finally we can get the coordinates of that polygon which will correspond to the real location of that furniture object.

PerVision in the Gator Tech Smart House

Based on the PerVision algorithm described in the previous section, we built an application that locates furniture in the Gator Tech Smart House. This application is used to test the algorithm's efficiency, precision, and usability. Figure 36 shows a snapshot of the application, where four windows can be seen. The top-left image is the live feedback

from the camera. In this example, the application is trying to locate the table. Since the table is already in the house, the application now knows its signature given by the RFID tag.

The bottom-right window shows the result of the algorithm where a round white area represents the table in this image. This is the result of color restriction and morphological operations such as erosion and dilation. The top-right window shows the location of the table in terms of x and y coordinates. Finally, the bottom-left window shows the remote monitoring application view where a model of the table is shown in the exact location where the real table is.

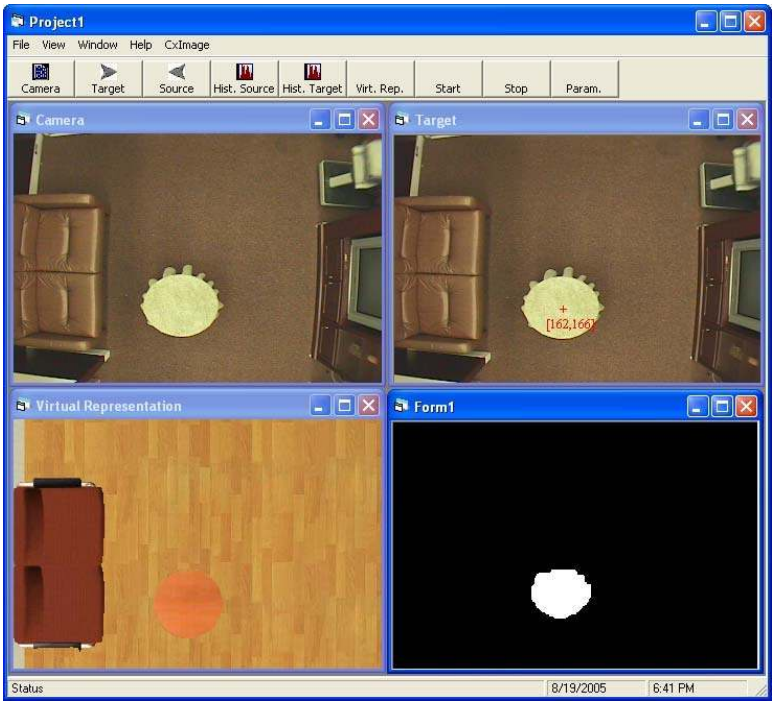


Figure 36. Demo application that locates furniture in the Gator Tech Smart House

Self Assessment

Like any vision algorithm, the probability of getting erroneous results is nontrivial. In this project we try to lower this probability by maximizing the use of the sensors and actuators in our pervasive space. First, we should note that the demo application not only

locates furniture, but also assigns a score each time the recognition is done. The score's value ranges from 0 to 100 with 100 being the best score. This score is computed based on several factors. The first one is the area of the white shape corresponding to the furniture item we are trying to locate. The closer that area is to the area of the furniture item from the XML information stored on the RFID tag, the higher the score is.

$$S_1 = 100 - \left(\frac{\left| A_{result} - A_{original} \right|}{A_{original}} \times 100 \right) \qquad (17)$$

As shown in Equation 17, S_1 is the result of comparing the two areas. A_{result} is the area of the white shape and $A_{original}$ is the area stored on the RFID tag.

The second factor is based on the accuracy of the results of our location algorithm. The system compares the result with readings from the smart floor. The maximum score occurs when the result of the location algorithm is the same as the reading from the smart floor.

$$S_2 = 100 - \left(\frac{\sqrt{(X_s - X_v)^2 + (Y_s - Y_v)^2}}{MAX} \times 100 \right) \qquad (18)$$

In Equation 18, S_2 is the score resulting from comparing the new coordinates with the approximate ones obtained from the smart floor readings. X_s and Y_s are the coordinates obtained from the smart floor and X_v,Y_v are the ones obtained from the vision algorithm. MAX is the maximum distance that those two coordinates can have between them.

$$S = (0.7 \times S_1) + (0.3 \times S_2) \qquad (19)$$

Finally, we compute S which is the final score obtained by weight averaging both S_1 and S_2 as shown in Equation 19. The weight of S2 is lower than S1 due to the fact that the smart floor only gives an approximate result.

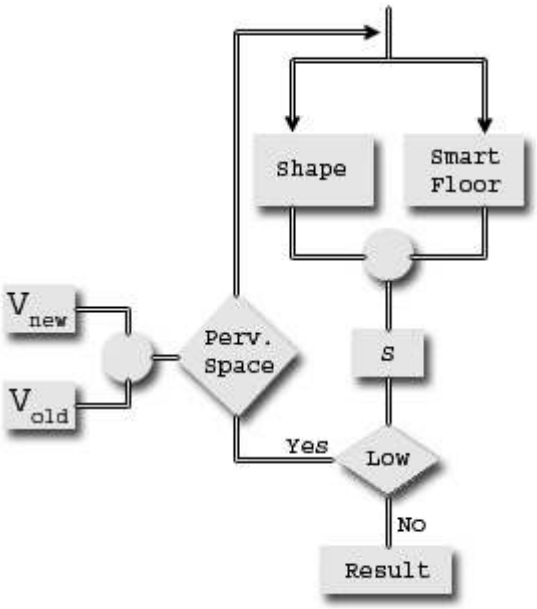


Figure 37. Self assessment and correction diagram

What happens when the score is low? Recall from previous sections that after taking an image of the chair for the first time in the smart house, the application stored the value V_{old} of the photo sensor in that room. In the case of a low score value, the application compares the current photo sensor value V_{new} with V_{old}. If there is any significant difference, the application uses the pervasive space to change the current luminance in that room so that it matches V_{old}. This is done by controlling the blinds (either opening or closing) or controlling the lights (on or off). Figure 37 shows the diagram that explains the process of computing the score by checking the shape and comparing the result to the smart floor readings. If the score (S) is low, then V_{old} is compared to V_{new} and the pervasive space is used to change luminance until V_{old} and V_{new} are similar. This process is executed iteratively until we get a score equal to or higher than a predefined threshold.

Error Analysis and Experimental Results

We tested this system in the Gator Tech Smart House using the previous demo application. The goal of this experiment is to show how a pervasive space could improve the recognition and tracking of various objects in the house. We used three objects: two chairs and one table. Figure 38 shows the floor plan of the kitchen where we did the experiments. The red shapes correspond to the blinds and the blue shapes correspond to light sources. We used one CCTV camera with a 2.8 mm wide angle lens located in the center of the kitchen.

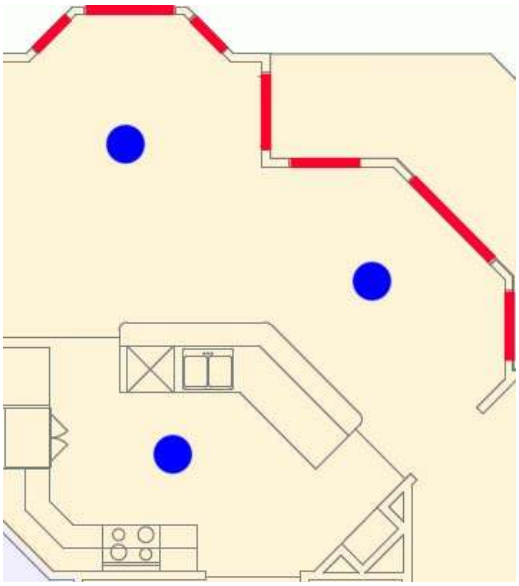


Figure 38. Floor plan of the kitchen

In this experiment, the system looks for the three objects one at a time. Since these objects are already in the house, the system knows their signatures (created when first brought into the house). By running the PerVision algorithm, we obtained the results shown in the following tables:

Tables 4, 5 and 6 show the results of recognizing and locating chair1, chair2, and table1 respectively. To get the best possible recognition result (score), the system tries

four different combinations: (1) lights off/blinds open, (2) lights off/blinds closed, (3) lights on/blinds open, and (4) lights on/blinds closed. The system at this point chooses the result where the score is the highest. For chair1, the score was the highest when lights were on and blinds were closed. However, chair2 got the highest score when lights were on and blinds were open. Notice that the system could not recognize chair2 when the lights were off. Finally, table1 got the highest score when lights were on and blinds were open. The recognition system does not need to go over all combinations. The only reason we did all combinations here is just to show how controlling a pervasive space could improve the result of the recognition algorithm.

Table 4. Results for Chair1

Lights	Blinds	Photo Sensor	Score
Off	Open	375	68
Off	Closed	370	76
On	Open	612	91
On	Closed	612	97

Table 5. Results for Chair2

Lights	Blinds	Photo Sensor	Score
Off	Open	370	X
Off	Closed	358	X
On	Open	605	86
On	Closed	594	83

Table 6. Results for Table1

Lights	Blinds	Photo Sensor	Score
Off	Open	370	70
Off	Closed	350	74
On	Open	597	95
On	Closed	585	81

Limitations and Observations

In this project, we try hard to achieve the best results that guarantee a reliable location tracking algorithm. However, this algorithm still faces some limitations which we list them below:

- **Covered objects:** In this case, a furniture object is covered with another object. For example, a table covered with a table cloth. This will block the PerVision algorithm from identifying this object. A solution could be to rely solely on the smart floor result with a low score assigned to the result.

- **Multi-colored objects:** The best result achieved using this algorithm is when looking for single color objects. The more colors used in objects, the harder the recognition would become.

- **Hidden objects:** Of course, it would be impossible to locate hidden objects. Consider a chair lying below a table. The vision algorithm will not be able to locate the chair. The smart floor can help in this case to identify the chair.

- **Split objects:** In some cases, an object could be captured by two different cameras with half in one camera and the other half in the other. This problem could be solved if the cameras' viewing areas are intersected.

The following observations were made while experimenting with the system and the PerVision algorithm described in this paper:

- Fluorescent light works better than the regular incandescent light. The colors are more vivid under fluorescent light.

- Sun light negatively affects the performance of the recognition algorithm even when blinds are closed. The optimal result the algorithm gets is when sun light is totally blocked.

The carpet color should be carefully picked. Dark single color carpets work best. This is due to the fact that it is easier for the algorithm to distinguish between a dark carpet and the furniture.

CHAPTER 8
CONCLUSION AND FUTURE WORK

In this dissertation we presented the three fundamental aspects of a complete self sensing space: Mapping a space, sensing electrical appliances, and sensing furniture.

We described a new approach to mapping spaces using an Atlas sensor platform. We called our mobile platform SensoBot. A collection of sensors were used to move SensoBot in an intelligent way and read RFID tags embedded in the carpet in a form of a grid. Several algorithms were presented and test results were analyzed.

We also presented smart plugs for self sensing spaces. The use of RFID technology in smart spaces makes installing new devices a simple, automatic and scalable task. There is no need to reprogram a smart space anymore every time we need to install a new device. A smart space will be able to automatically integrate the new device upon installation.

Finally, we described two approaches to sensing furniture: SensoBot and Pervision. SensoBot uses RFID tags embedded in furniture objects to locate them. The Pervision approach uses computer vision to recognize and track dumb objects in a pervasive space. We have shown how RFID technology and smart floors could be used to assist recognition algorithm locate furniture in a smart house. We described the extended vision algorithm we call PerVision (pervasive-assisted Vision) and showed its self-assessment features. We have conducted an experimental evaluation of PerVision in a real setting, which is the Gator Tech Smart House at the University of Florida. We have shown how

the algorithm performance is significantly improved. We discussed current challenges and limitations and summarized our experience in implementing dumb object tracking.

As part of our future work we include the following:

7. We discussed the use of one mobile sensor platform to map a space that communicates with a desktop application. However, this approach should not be limited to one mobile sensor platform. Having multiple SensoBots working in parallel in ad-hoc mode is on our list of future enhancements. In addition, we are investigating the use of more intelligent algorithms that might give error free results.

8. Currently we are using RFID tags and readers to identify new devices. We could use the same idea with different, possibly cheaper, technology. For example, every smart plug could send a specific code using the X10 protocol when the device is plugged in. Another option would be to make use of Near Field Communication (NFC). NFC is a short-range wireless connectivity standard that uses magnetic field induction to enable communication between devices when they are attached together, or brought within a few centimeters of each other.

9. Enhancing the remote monitoring and intervention application to support multiple self sensing spaces.

10. Improving the Pervision system and engaging other sensors and actuators in the self-assessment/adjustment loop of the algorithm.

Finally, this work proves that advances in technology make it more feasible to have smart spaces that can sense their selves and their contents, keep track of any changes, and report to a remote monitoring application any emergencies that might occur.

APPENDIX A
REFERENCE RFID TAGS

Table 7. Landmark RFID Tags

Landmark	**Block 0, First Byte**	**Block 0, Second Byte**
Door (Opening Side)	00000001	00000001
Door (Fixed Side)	00000001	00000000
Front Door (Opening Side)	10000001	00000001
Front Door (Fixed Side)	10000001	00000000
Window (Left)	00000010	00000001
Window (Right)	00000010	00000000
Power Outlet	00000011	-
Closet (Left)	00000100	00000001
Closet (Right)	00000100	00000000
Dynamic Landmark	11111111	-
…	…	…

Table 8. Object RFID Tags

Name	**Address**	**Comments**
Category	Block 1, byte 0	See table 9.
Length	Block 1, byte 1	0-255 inches
Width	Block 1, byte 2	0-255 inches
Color	Block 1, byte 3	0-255 representing Hue values
Leg Count	Block 2, byte 0	0-N representing the number of legs
Leg Pos	Block 2, byte 1	[0,1,2,3]
Image	Block 3 – Block n	URL

Table 9. Possible Category Values

Name	**Value**
Table	00000001
Chair	00000010
Desk	00000011
Book Shelf	00000100
Bed	00000101
Couch	00000110
Commode	00000111
Entertainment Center	00001000
…	…

APPENDIX B
SENSOBOT MAIN CODE

```
while(!(sensors[SenButton] & 0x0F)) //Start the main loop
{
    //Check if the next neighbor is not visited before

    switch (Status)
    {
        case 0:
            switch (direction)
                {
                    case 'E':
                        if (oGrid[X+1][Y] == 0)
                            Status = 2;
                        else
                            Status = 5;
                    break;
                    case 'W':
                        if (oGrid[X-1][Y] == 0)
                            Status = 2;
                        else
                            Status = 5;
                    break;
                    case 'N':
                        if (oGrid[X][Y-1] == 0)
                            Status = 2;
                        else
                            Status = 5;
                    break;
                    case 'S':
                        if (oGrid[X][Y+1] == 0)
                            Status = 2;
                        else
                            Status = 5;
                    break;
                }

        break;
        case 2:
            //Keep driving straight until blocked
            //While driving, push the tags you read to the stack
            Go(speedd); //Drive straight
            SendRFIDByte(SelectTag1, SelectTagLen); //Read RFID
            //if Tag is found
            if ((res == 0x14) && (!compareTags(Tag, TagOld)))
            {
                Stop(); //Stop
                testDone(); //Beep
                driveAndStop(45,50); //Move to the center of the tag
                delay(100);

                //Send data to the desktop computer
                temp1[0] ='\0';
                utoa(heading,temp1,10);
                SPI_puts(temp1);
                delay(100);
                copyTag(Tag, TagOld);
                Status = 0;
                switch (direction) //Increment X or Y based on direction
                {
                    case 'E':
```

```
                    X++;
               break;
               case 'W':
                    X--;
               break;
               case 'N':
                    Y--;
               break;
               case 'S':
                    Y++;
               break;
          }
          //push to the stack
          push(&cStack, X, Y);
          oGrid[X][Y] = 1;
          delay(100);

          //Send data to the desktop computer
          temp[0] ='\0';
          sX1[0] ='\0';
          sY1[0] ='\0';
          utoa(X,sX1,10);
          utoa(Y,sY1,10);
          strcat(temp, "xy;");
          strcat(temp, sX1);
          strcat(temp, ";");
          strcat(temp, sY1);
          SPI_puts(temp);
          delay(100);

     }
break;
case 3:
     //IF path is blocked turn 180 degrees
     switch (direction)
          {
               case 'E':
                    oGrid[X+1][Y] = 1;
                    direction = 'W';
               break;
               case 'W':
                    oGrid[X-1][Y] = 1;
                    direction = 'E';
               break;
               case 'N':
                    oGrid[X][Y-1] = 1;
                    direction = 'S';
               break;
               case 'S':
                    oGrid[X][Y+1] = 1;
                    direction = 'N';
               break;
          }
     turnAndStop(oneeighty, 0);
     copyTag(TagNotFound, TagCld);
     Status = 4;
break;
case 4:
     //Go back to the last tag
     Go(speedd); //Drive Straight
     SendRFIDByte(SelectTag1, SelectTagLen); //Read RFID
     //if Tag is found
     if ((res == 0x14) && (!compareTags(Tag, TagCld)))
     {
          //stop when you get to the last tag
          Stop();
          testDone();
          driveAndStop(45,50);
          copyTag(Tag, TagCld);
          Status = 5;
     }
break;
```

```
case 5:
    //If blocked, SensoBot will try to find a different open path
    if (oGrid[X][Y-1] == 0)
    {
        //north is open
        switch (direction)
        {
            case 'S':
                //Direction is South, turn 180 degrees
                turnAndStop(oneeighty, 0);
            break;
            case 'W':
                //Direction is West, turn 90 degrees CW
                turnAndStop(ninetyd, 0);
            break;
            case 'E':
                //Direction is East, turn 90 degrees CCW
                turnAndStop(ninetyd, 1);
            break;
        }
        direction = 'N';
        Status = 2;
    }
    else if (oGrid[X-1][Y] == 0)
    {
        //west is open
        switch (direction)
            {
                case 'N':
                    //Direction is North, turn 90 degrees CCW
                    turnAndStop(ninetyd, 1);
                break;
                case 'S':
                    //Direction is South, turn 90 degrees CW
                    turnAndStop(ninetyd, 0);
                break;
                case 'E':
                    //Direction is East. turn 180 degrees
                    turnAndStop(oneeighty, 0);
                break;
            }
            direction = 'W';
            Status = 2;
    }
    else if (oGrid[X+1][Y] == 0)
    {
        //East is open
        switch (direction)
            {
                case 'N':
                    //Direction is North, turn 90 degrees CW
                    turnAndStop(ninetyd, 0);
                break;
                case 'S':
                    //Direction is South, turn 90 degrees CCW
                    turnAndStop(ninetyd, 1);
                break;
                case 'W':
                    //Direction is West, turn 180 degrees
                    turnAndStop(oneeighty, 0);
                break;

            }
            direction = 'E';
            Status = 2;
    }
    else if (oGrid[X][Y+1] == 0)
    {
        //South is open
        switch (direction)
            {
                case 'N':
```

```
                                //Direction is North, turn 180 degrees
                                turnAndStop(oneeighty, 0);
                            break;
                            case 'E':
                                //Direction is East, turn 90 degrees CW
                                turnAndStop(ninetyd, 0);
                            break;
                            case 'W':
                                //Direction is West, turn 90 degrees CCW
                                turnAndStop(ninetyd, 1);
                            break;
                        }
                        direction = 'S';
                        Status = 2;
                    }
                    else
                    {
                        //No open path available, this tag is discovered
                        //Pop from the stack and backtrack
                        oGrid[X][Y] = 1; //mark tag discovered
                        if (pop(&cStack)) //pop from stack
                        {
                            //Check if there is an open path
                            if (checkIfOpen(xDestination, yDestination))
                            {
                                while (!goToDestination(X, Y, xDestination,__
                                yDestination)); //backtrack to the tag
                                X = xDestination;
                                Y = yDestination;
                                Status = 5;
                            }
                            else
                            {
                                oGrid[xDestination][yDestination] = 1;
                            }
                        }
                        else
                        {
                            Stop(); //Mapping completed
                            // Play the end song
                            delay(500);
                            byteTx(CmdPlay);
                            byteTx(3);
                            delay(1000);

                            // Turn off the leds
                            byteTx(CmdLeds);
                            byteTx(0x00);
                            byteTx(0);
                            byteTx(0);

                            // Turn Roomba off
                            byteTx(CmdPower);
                            while(1);
                        }

                    }
                break;
            } // end of switch case
            delaySensors(20);
        }
        Stop();
```

LIST OF REFERENCES

[1] A. Schmidt, M. Beigl, and H. Gellersen, "Adding some smartness to devices and everyday things," in *the Third IEEE Workshop on Mobile Computing Systems and Applications*, Monterey, CA, 2000, pp. 3-10.

[2] H. Gellersen, G. Kortuem, A. Schmidt, and M. Beigl, "Physical prototyping with smart-its," *IEEE Pervasive Computing,* vol. 3, pp. 74-82, July-September 2004.

[3] A. Ferscha and J. Johnson, "Distributed Interaction in virtual spaces," in *the Third IEEE International Workshop on Distributed Interactive Simulation and Real-Time Applications*, Greenbelt, MD, 1999, pp. 5-13.

[4] M. Montemerlo and S. Thrun, "Simultaneous localization and mapping with unknown data association using FastSLAM," in *the IEEE International Conference on Robotics and Automation*, Taipei, Taiwan, 2003.

[5] G. Dissanayake, P. Newman, S. Clark, H. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localisation and map building (slam) problem," *IEEE Transactions of Robotics and Automation,* vol. 17, pp. 229-241, 2001.

[6] D. Hähnel, R. Triebel, W. Burgard, and S. Thrun, "Map building with mobile robots in dynamic environments," in *the International Conference on Robotics and Automation*, Taipei, Taiwan, 2003.

[7] T. Kanji, H. Tsutomu, Z. Hongbin, K. Eiji, and O. Nobuhiro, "Mobile robot localization with an incomplete map in non-stationary environments," in *the IEEE International Conference on Robotics and Automation*, Taipei, Taiwan, 2003.

[8] C. Jennings, D. Murray, and J. Little, "Cooperative robot localization with vision-based mapping," in *the IEEE International Conference on Robotics and Automation*, Detroit, MI, 1999.

[9] F. Dellaert, W. Burgard, D. Fox, and S. Thrun, "Using the condensation algorithm for robust vision-based mobile robot localization," in *the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Ft. Collins, CO, 1999.

[10] A. Eliazar and R. Parr, "Dp-slam: fast, robust simultaneous localization and mapping without predetermined landmarks," in *the 18th International Joint Conference on Artificial Intelligence*, Acapulco, Mexico, 2003.

[11] Y. Tsin, R. Collins, V. Ramesh, and T. Kanade, "Bayesian color constancy for outdoor object recognition," in *the IEEE Conference on Computer Vision and Pattern Recognition*, Hawaii, 2001.

[12] T. Gevers and A. Smeulders, "Color-based object recognition " *Pattern Recognition,* vol. 32, pp. 453-464, 1999.

[13] D. Forsyth and J. Ponce, *Computer vision: a modern approach* Upper Saddle River, NJ: Prentice Hall, 2002.

[14] A. Torralba, K. Murphy, W. Freeman, and M. Rubin, "Context-based vision system for place and object recognition," in *the Ninth International Conference on Computer Vision*, Nice, France, 2003.

[15] P. Chang and J. Krumm, "Object recognition with color cooccurrence histograms," in *the International Conference in Computer Vision and Pattern Recognition*, Ft. Collins, CO, 1999.

[16] C. Papageorgiou and T. Poggio, "A trainable system for object detection," *International Journal of Computer Vision,* vol. 38, pp. 15–33, 2000.

[17] S. Giroux, A. Ayers, and H. Pigot, "An Infrastructure for Assisted Cognition and Telemonitoring," in *the International Conference on Aging, Disability and Independence*, Washington, DC, 2003.

[18] A. Sixsmith, "Pervasive computing and the well-being of older persons," in *the International Conference on Aging, Disability and Independence*, Washington, DC, 2003.

[19] J. Dowling, *The retina: an approachable part of the brain*. Cambridge, MA: Harvard University Press, 1987.

[20] Kodak, "Device performance specification," *Kodak KAC-1310 Image sensor,* vol. 4, pp. 12-35, 2002.

[21] D. Osorio, N. Marshall, and T. Cronin, "Stomatopod photoreceptor spectral tuning as an adaptation for colour constancy in water," *Vision Research,* vol. 37, pp. 3299–3309, 1997.

[22] R. Mausfeld, "Color perception: from grassman codes to a dual code for object and illumination colors," *Color Vision,* vol. 3, pp. 219-250, 1998.

[23] A. Johansson, C. Balkenius, and A. Balkenius, "Color constancy in visual scene perception," *Lund University Cognitive Studies,* vol. 61, pp. 23-42, 2003.

[24] K. Romer, T. Schoch, F. Mattern, and T. Dubendorfer, "Smart identification frameworks for ubiquitous computing applications," in *the First IEEE International Conference on Pervasive Computing and Communications*, Fort Worth, TX, 2003, pp. 253-262.

[25] C. Lee, D. Nordstedt, and S. Helal, "Enabling smart spaces with OSGi," *IEEE Pervasive Computing,* vol. 2, pp. 89-94, 2003.

[26] OSGi, "OSGi alliance," http://www.osgi.org, 1999 [Access Date: May 2006].

[27] E. Foxlin, M. Harrington, and G. Pfeifer, "Constellation: A wide-range wireless motion-tracking system for augmented reality and virtual set application," in *the 26th International Conference on Computer Graphics and Interactive Techniques*, Los Angeles, CA, 1999.

[28] A. Ward, "Sensor-driven computing." vol. Ph.D. Cambridge, MA: Cambridge University, 1998.

[29] N. Priyantha, A. Chakraborty, and H. Balakrishnan, "The cricket location-support system," in *the sixth Annual ACM International Conference on Mobile Computing Networking*, Boston, MA, 2000.

[30] B. Brumitt, B. Meyers, J. Krumm, A. Kern, and S. Shafer, "Easyliving: technologies for intelligent environments," *Handheld and Ubiquitous Computing,* vol. 3, pp. 12–29, September 2000.

[31] P. Bahl and V. Padmanabhan, "RADAR: an in-building RF-based user location and tracking system," in *the 19th Annual Joint Conference of the IEEE Computer and Communications Societies*, Tel-Aviv, Israel, 2000.

[32] A. Byström, T. Dahlroth, J. Eriksson, L. Fernandez, D. Holmgren, T. Johansson, P. Larsson, I. Styf, M. Ståhl, N. Varillas, and M. Wu, "Advanced wavelan positioning system," Luleå, Sweden 2001.

[33] A. Smailagic and D. Kogan, "Location sensing and privacy in a context-aware computing environment," *IEEE Wireless Communications,* vol. 9, pp. 10–17, October 2002.

[34] M. Youssef, A. Agrawala, and A. Shankar, "WLAN location determination via clustering and probabuility distributions," in *the First IEEE International Conference on Pervasive Computing and Communications*, Fort Worth, TX, 2003.

[35] R. Harle and A. Hopper, "Building world models by raytracing within ceiling-mounted positioning systems," in *the Fifth International Conference on Ubiquitous Computing*, Seattle, Washington, 2003.

[36] E. McGrath, A. Ranganathan, R. Campbell, and M. Mickunas, "Use of ontologies in pervasive computing environments," University of Illinois at Urbana-Champaign Computer Science Department, Urbana, IL UIUCDCS-R-2003-2332, 2003.

[37] A. Helal, W. Man, H. Elzabadani, Y. Kaddoura, E. Jansen, and J. King, "Gator tech smart house: a programmable pervasive space," *IEEE Computer Magazine,* pp. 64-74, March 2005.

[38] A. Helal, "Programming pervasive spaces," *IEEE Pervasive Computing,* vol. 4, pp. 84-87, January 2005.

[39] R. Want, "Enabling ubiquitous sensing with RFID," *IEEE Computer,* vol. 37, pp. 84-86, April 2004.

[40] Y. Kaddoura, A. Helal, and J. King, "Cost-precision tradeoffs in unencumbered floor-based indoor location tracking," in *the Third International Conference On Smart Homes and Health Telematic*, Sherbrooke, Canada, 2005.

[41] S. Intille, J. Davis, and A. Bobick, "Real-time closed-world tracking," in *the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, San Juan, Puerto Rico, 1997, pp. 697-703.

[42] iRobot, "Roomba," http://www.irobot.com, 2003 [Access Date: June 2006].

[43] iRobot, "iRobot roomba serial command interface specification," http://www.irobot.com/images/consumer/hacker/Roomba_SCI_Spec_Manual.pdf, 2006 [Access Date: March 2006].

[44] R. Bose, J. King, H. El-zabadani, S. Pickles, and A. Helal, "Building plug-and-play smart homes using the atlas platform," in *the Fourth International Conference on Smart Homes and Health Telematic*, Belfast, Ireland, 2006.

[45] D. Marples and P. Kriens, "Open services gateway initiative: an introductory overview," *IEEE Communications Magazine,* vol. 39, pp. 110-114, 2001.

[46] OSGi, *Osgi service platform, release 2*. Fairfax, VA: IOS Press, Inc., 2002.

[47] Phidgets, "Phidgets," http://www.phidgetsusa.com, 2001 [Access Date: May 2006].

[48] D. Cook and S. Das, *Smart environments: technology, protocols, and applications*. Indianapolis, IN: John Wiley & Sons, Inc., 2005.

BIOGRAPHICAL SKETCH

Hicham El Zabadani received a Bachelor of Science degree in computer science from the Lebanese American University (Beirut, Lebanon) in 2000. In 2002, he received a Master of Science degree in computer science from the same university. During the same year, he was admitted to the Ph.D. program at the University of Florida.

His major research is in the areas of mobile and pervasive computing with particular interest in smart spaces and assisted living technologies.