A FRAMEWORK FOR LEVEL-OF-DETAIL TRACKING AND SURVEILLANCE IN WIRELESS SENSOR NETWORKS

by

Jianfeng Peng

A Dissertation Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

In Partial Fulfillment of the Requirements
For the Degree of

DOCTOR OF Philosophy

In the Graduate College

THE UNIVERSITY OF ARIZONA

2007

THE UNIVERSITY OF ARIZONA
GRADUATE COLLEGE

As members of the Dissertation Committee, we certify that we have read the dissertation

prepared by Jianfeng Peng

entitled A Framework for Level-of-Detail Tracking and Surveillance in Wireless Sensor Networks

and recommend that it be accepted as fulfilling the dissertation requirement for the

Degree of Doctor of Philosophy

____ Date: 4/25/07
Dr. Jerzy Rozenblit

____ Date: 4/25/07
Dr. Roman Lysecky

____ Date: 4/25/07
Dr. Michael Marcellin

____ Date: 4/25/07
Dr. Ferenc Szidarovszky

____ Date:

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copies of the dissertation to the Graduate College.

I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

Dr. Jerzy Rozenblit
____ Date: 4/25/07
Dissertation Director:

STATEMENT BY AUTHOR

SIGNED: Jianfeng Peng

ACKNOWLEDGEMENTS

I am greatly indebted to my advisor, Dr. Jerzy Rozenblit, who has continually provided guidance and support with great flexibility and understanding. His deep interest in the research and the students he teaches make him an excellent mentor. I appreciate the time and efforts of my other committee members: Dr. Michael Marcellin, Dr. Roman Lysecky, and Dr. Ferenc Szidarovszky.

I would also like to acknowledge the other members of the Model-Based Design Laboratory, who worked with me on a daily basis and who created a positive working and studying environment. I consider myself very privileged to have been able to collaborate with this group of intelligent and interested researchers who welcomed and encouraged me to work on every phase of this project with them.

Special thanks go to Rozanne Canizales and Tami Whelan, who made all obstacles surmountable.

Last, I would like to thank my wife, Rong, for providing me with a seemingly endless supply of love and support.

TABLE OF CONTENTS

TABLE OF CONTENTS - *Continued*

TABLE OF CONTENTS - *Continued*

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

API	Application Programming Interface
FPS	Frame Per Second
GPS	Global Positioning System
HCI	Human-Computer Interaction
LOD	Level-of-Detail
MEMS	Micro Electro Mechanical Systems
MFC	Microsoft Foundation Class Library
PIR	Passive Infrared
ROI	Region of Interest
TDOA	Time Difference of Arrival
TOI	Time of Interest
UCC	User Controlled Character
UDP	User Datagram Protocol
WSN	Wireless Sensor Network

ABSTRACT

In the recent years, wireless sensor networks have become a very active research area. Technological advances paved the way for wireless networks being used in many military and civilian applications, including object tracking, security surveillance, habitat monitoring, and traffic control. However, there are intrinsic constraints on design and optimization of sensor networks. Current research mainly focuses on minimizing energy consumption when collecting and transferring data, but improvement is limited by the current sensing paradigm in which users are passive receivers and interpreters of sensory data. A shift is needed from a supplier-initiated, data-centric approach to a consumer-initiated, demand-driven paradigm. This dissertation proposes a Level-of-Detail (LOD) sensing framework, in which sensing activities are fully determined and adjusted by the user's need for sensory data in real-time. This information-on-demand approach eliminates unnecessary data collection and transfer, and provides an ideal balance between system lifetime, response time, and attention economics. A software test-bed is created for concept proof purpose. The test-bed employs a client-server architecture and allows users to carry out various experiments in an interactive, three-dimensional virtual environment. Experiments results demonstrate substantial reduction in power consumption using the proposed LOD technology.

CHAPTER 1. INTRODUCTION

1.1 Motivation

In recent years, wireless sensor networks (WSNs) have become a very active research area. Device-level research in wireless communication, large scale circuit integration, and Micro Electro Mechanical Systems (MEMS) has greatly advanced the state of art of autonomous wireless sensing devices. As a result, the unit cost and size of wireless sensors are expected to drop to a level suitable for mass production and deployment [1]. Meanwhile, network level work such as routing schemes, and prediction based sensor management techniques have to some extent reduced energy consumption during acquiring and transmitting unit data. All these technological advances have paved the way for WSNs to be used in many military and civilian applications, including object tracking, security surveillance, habitat monitoring, and traffic control.

However, two main issues have remained. First is the problem of energy consumption, which impacts overall system lifetime. In fact, while technological advances have improved the energy efficiency of acquiring and transferring unit sensory data, the effectiveness is approaching a limit under current technological conditions. Different methods of improving the system lifetime must be found. Given that total energy consumption is determined by the product of the unit energy consumed by acquiring data, and the amount of sensory data required, the solution is to reduce the amount of sensory data collected and transferred as much as possible.

Second, attention economics is aggravated with the increase of sensing capability. Wireless sensor network users are getting increasingly troubled by information overload. The sheer amount of sensor data produced by the sensor network places a great burden not only on the long range communication link between the sensor field and control center, but also on the information processing capability of the control center. Useful information is diluted by the increasing base of incoming data; this compromises the information/cost ratio.

Furthermore, current sensing architecture does not provide a mechanism for multi-resolution tracking activities. All objects are treated with equal attention. Similarly, all events are processed with same effort regardless of the nature, place, or time of the event. Since response time is reversely proportional to the amount of sensing activities - the more sensing activities and processing involved, the slower the response. Although response time can be improved by including nodes that are powered all the time, this approach contradicts the power saving principle.

The currently existing paradigm of provider initiated sensing architectures cannot sufficiently address the abovementioned problems by design. These problems severely limit a sensor network's capability and effectiveness in a wide range of real-world applications. A new paradigm governing the way sensors collect and transfer data, and the way users interpret data, is needed. The goal is an optimal balance among the tri-axial system metrics of lifetime, quality of service, and attention economics.

1.2 Contribution

This research proposes a new paradigm that uses LOD based technology for wireless sensor network tracking and surveillance applications. It allows the user to manage the sensing system from the highest system level. Under the new paradigm, the user initiates a tracking process and determines what information he needs for tracking purpose. Consequently, optimization can be performed at the top level of the tracking application. This results in substantial savings in terms of communication and computation.

The LOD based application framework is compatible with current research in wireless sensor networks. The framework provides a middleware layer that connects a traditional sensor network and a user who wishes to utilize the LOD capability in his applications. The middleware layer uses network messages to interface with other layers in the application framework. Such a message passing scheme provides the necessary flexibility and scalability for the future.

In addition, a software test-bed has been built to prove the LOD concept. The test-bed creates a virtual sensing system, based on which a series of experiments have been performed to verify the effectiveness of the LOD methodology. The software test-bed can serve as also platform for further experiments in wireless sensor networks, including power analysis, sensor deployment, tracking algorithm, etc.

1.3 Organization

This dissertation is organized into seven chapters. Chapter 2 briefly reviews current research in tracking and power saving issues. Chapter 3 presents the basic theory of LOD based tracking process. Chapter 4 discusses architectural design issues, functional components and work flow, and proposes some optimization techniques for better power-efficiency in both computation and communication. Chapter 5 describes the client-server architecture based test-bed and experimental setup, and Chapter 6 provides an analysis of the experiment results. Chapter 7 offers a conclusion with outlines for future work.

CHAPTER 2. BACKGROUND OVERVIEW AND METHODOLOGIES

2.1 Characteristics of Wireless Sensor Networks

2.1.1 Sensor Device and Constraints

A wireless sensor network (WSN) consists of massive number of autonomous sensor nodes to cooperatively monitor physical or environmental conditions, such as temperature, sound, vibration, pressure, motion or pollutants. These sensor nodes are spatially distributed and communicate with each other through short range radio. The position of sensor nodes need not be engineered or predetermined. This allows random deployment in inaccessible terrains or disaster relief operations. WSNs are gaining increasing popularity in both military and civilian applications [1] including:

- Habitat and environmental monitoring
- Life science studies
- Target tracking
- Security surveillance
- Reconnaissance
- Border control
- Emergency service & Disaster relief operations
- Manufacturing control
- Smart home appliance

Although applications of WSNs vary widely, the underlying mechanism remains the same: the sensors are deployed to collect readings from a set of points in an environment over a period of time in order to detect trends and interdependencies. The sensors are designed to work in all conditions, including wilderness, and to cooperate with each other to collect and fuse data with minimal human intervention.

One of the most important constraints on sensor nodes is the low power consumption requirement. As physical size decreases, so does energy capacity. Sensor nodes carry limited, generally irreplaceable, power sources. This fundamental restriction on energy resource ends up creating computational and storage limitations that lead to a new set of issues. For example, while traditional networks aim to achieve high quality of service (QoS) provisions, sensor network protocols must focus primarily on power conservation. They must have built-in trade-off mechanisms that give the end user the option of prolonging network lifetime at the cost of lower throughput or higher transmission delay.

Besides the intrinsic limitation of power constraint in wireless sensor networks, there is another significant issue regarding communication with an external facility, typically a control center geographically separated from the sensor sites. This communication link becomes a bottleneck when the sensor networks are in remote, unattended sites without power lines or internet facilities. The sensor networks must resort to long-range wireless communication technologies such as satellite link or GPRS to transfer data back and forth with the control center. This is usually the case in such applications as environment monitoring and border control.

All these constraints impact the way wireless sensor networks are designed, deployed, managed and optimized. Many researchers are currently engaged in developing schemes to improve WSNs, with the primary focus on reducing overall power consumption.

2.1.2 Energy Consumption Analysis

Power by Operations

The radio subsystem is the most important system on a wireless sensor node since it is the primary energy consumer in our application scenarios. Modern low power, short range transceivers consume between 15 and 300 milliwatts of power when sending and receiving. A key hardware observation is that low power radios consume approximately the same amount of energy when in receive or transmit mode. This energy is consumed if the radio is on, whether or not it is receiving actual data. The actual power emitted out of the antenna only accounts for a small fraction of the transceiver's energy consumption. A significant portion goes to internal operation. Because of this, the overall cost of radio communication can easily be dominated by receiver power consumption – a metric often ignored in wireless studies. Additionally, if the receiver is left on 100% of the time during periods of intermittent communication, the receiver will be the single largest energy consumer in the system. Many people erroneously assume that reception is free.

On the other hand, computation in WSNs is typically light-weight processing of raw data and consumes much less power as compared to radio communication. According to Akyildiz, the energy cost of transmitting 1 KB a distance of 100 m is approximately the same as that for executing 3 million instructions by a 100 million instructions per second (MIPS)/W processor [2].To further understand how different types of operation consume power, Mainwaring et. Al. listed the costs of various basic operations as shown in Table 1 [8].

Table 1 Power Consumption Decomposition

Operation	nAh
Transmitting a packet	20.00
Receiving a packet	8.00
Radio listening for a millisecond	1.25
Operating sensor for 1 sample (analog)	1.08
Operating sensor for 1 sample (digital)	0.35
Reading a sample from the ADC	0.01
Flash Read Data	1.11
Flash Write / Erase Data	83.33

Besides the power consumed when the sensor is active, a certain amount of power consumption also occurs when the sensor is in sleep mode, due to leakage. Since the baseline life of the sensor is determined by the current drawn in the sleep state, it is important to reduce power consumption when the sensors are inactive. Turning off the sensors and radio, thus driving the processor into a deep sleep mode, will help to minimize power consumption in sleep mode.

Power by Sensing Modalities

A wireless sensor network typically consists of multiple types of sensing modalities. These different types of sensor vary widely in power consumption as shown in the following table [4].

Table 2 Power Consumption by Sensors

Modality	Current	Voltage Requirement (v)
Photo	1.9 mA	2.7-5.5
Temperature	1 mA	2.5-5.5
Humidity	550 uA	2.4-5.5
Pressure	1 mA	2.2-3.6
Acoustic	0.5 mA	2.0-10
Passive IR	0 mA	3.3-5

As the table shows, passive IR motion detectors use far less power than photo sensors or acoustics sensors. In a power aware sensor network, high power consumption sensors are normally kept in sleep state while lower power sensors, such as passive IR motion detectors, are on all the time. High power sensors are only activated when needed. By doing this, the operation cost of the entire network is lowered.

For many types of application, WSNs are expected to run in the sensing field for a long period of time. Because power supplies are usually unavailable in such sensing fields (e.g. instance, desert, forest, caves), the sensors rely heavily on the batteries carried on board. Although various types of energy harvesting technologies exist, such as solar power, inducted current, or mechanical vibration to electrical energy conversion methods [5], these energy harvesters either impose considerable overhead on size and cost, have environmental implications, or require sources of energy (such as magnetic field, vibration, seismic activity) that are typically unavailable in the operating environment. Rahimi et al. [7] propose to use autonomously mobile sensors (provider) to search, recharge and deliver energy to immobile, energy-depleted nodes (consumer). However, this will greatly increase the complexity of consumer sensor node design, and the maintenance of the entire network, with added power consumption needed to move the mobile nodes. While this method can be used in mission-critical systems, it's too expensive to recharge sensor nodes in a wireless sensor network with high degree of redundancy.

As a result, current sensor devices still use an on board battery as a power source. The following table shows the power capacity of typical batteries.

Table 3 Power Source Analysis

Battery Type	Capacity	Lifetime1
Button cell	10~100mah	1 week
9V battery	100~200ma	3 weeks
Li-Polymer	1000mah	4 months
NiMH AA	2000mah	10 months

Depending on the type of application, the average power consumption per sensor per day can range from around 10 mAh [8] to a few hundred mAh. This poses a great challenge to designers of WSNs because in many types of application, the sensor networks require a lifetime of at least a few months. In habitat applications, the sensors need to run for no less than nine months, the length of a single field season.

[1] Calculation is based on minimum average power consumption of 8mAh per day

2.1.3 System Metrics

Traditionally, a designer must take into account, system lifetime, hardware cost, ease of deployment, sensing coverage, response time, tracking accuracy, security, and robustness [4]. With hardware costs continuing to drop and massive deployment becoming affordable, some of these metrics are less important especially when compared to the key metrics: lifetime, response time, tracking accuracy, and attention economics.

Lifetime

The foremost concern for WSN-based tracking applications is life time. The goal of many types of application, including environmental monitoring, security and target tracking in wilderness, all require nodes placed out in the field and left unattended for a certain period of time. Normally an application has strict requirement for minimal system lifetime. A sensing system that fails to sustain its functionality throughout the minimal lifespan will result in lost information, degraded quality of service, and perhaps failure of the entire mission. Since in most application scenarios, a majority of the nodes will be self powered by onboard batteries, the primary limiting factor for the lifetime thus is the capacity of batteries carried by the sensors. Designers of a sensing system must ensure that onboard power source is sufficient to sustain the minimal lifespan.

<u>Response Time</u>

Many types of WSN-based applications are time sensitive. For applications such as battlefield target tracking, security surveillance, disaster relief operations, etc, it is essential that the sensor network provide the user with timely result from the sensors. From the sensor's perspective, an event of interest must immediately trigger sensor reaction. For instance, an object crossing a sensing zone should immediately trigger line-breaking sensors; the movement of potential targets should be quickly tracked by positioning sensors; while activities of unauthorized personnel in restricted areas should be closely monitored by vision sensors.

From the user's perspective, the WNS should respond immediately react to the user's commands, such as a request for further details on a specific object, within a specific geographical area, for a specific period of time. The WSN's capability to quickly adapt to the user's dynamic need for data fidelity is a primary component of the quality of service.

<u>Attention Economics</u>

Sensors are becoming very affordable and the scale of mass production continues to escalate. As a result, it makes sense to increase the number of sensors used for an application, not only because they are affordable, but also as a means to increase redundancy, improve coverage, and prolong system lifetime. Unfortunately, increasing the number of sensor also has implications for attention economics.

Nowadays, it is increasingly common to hear domain experts complaining about information overload when they are working with massive wireless sensor networks. The most useful and critical information is often buried in an overwhelming flood of incoming data, making it difficult to take proper action in a timely manner. Herbert Simon wrote, "What information consumes is rather obvious: it consumes the attention of its recipients. Hence a wealth of information creates a poverty of attention, and a need to allocate that attention efficiently among the overabundance of information sources that might consume it." (Simon 1996).

2.2 Current Research

Wireless sensor network research has been a hot topic over the last decade. Research areas range from device physics, network protocols to object tracking algorithms. A primary goal of the current research is to improve the lifespan of sensor networks without seriously undermining the overall quality of the tracking results. Major research areas are enumerated in the following list. Each of the major research areas comprises a number of sub areas, such as protocols, deployment, etc.

- Device Physics: voltage scaling, low power radio [9, 10].
- Protocols: MAC, routing [11, 12].
- Naming and Addressing: low communication overhead, self-identification [13].
- Deployment: self-deployment, coverage [14,15].
- Localization: network calibration, determining sensor locations [16,17,18,19,20].
- Sensor Management: energy efficiency [22, 23].
- Fault Tolerance: reliability of individual sensor and entire network [26,25].
- Tracking Algorithm: movement prediction, adaptive tracking [26,29].

These research efforts have to a certain extent improved usability, robustness, security, and lifetime. However, under current technological conditions system lifetime has been pushed to the limit [30]. Although technological advancement can perhaps produce lower power sensors, the more important task is to discover a more power efficient way for the sensors to collaborate with the users. In the next section, we examine the factors limiting further improvement in WSN power efficiency under the current paradigm.

2.3 Limitations of Current Paradigm

2.3.1 Bottleneck of Power Saving

To analyze the limitations of the current paradigm, it is necessary to define the operational cost of a wireless sensor network. As previously mentioned, energy consumption is the most critical constraint, so it is convenient to use energy consumption as an indication of operational cost for WSNs using the following equation:

$$E_{Total} = C_{CPB} \times M_{Col} + C_{iTPB} \times M_{iTran} + C_{eTPB} \times M_{eTran} \qquad \text{Equation 2-1}$$

Total energy consumption is equal to the sum of energy spent on collecting and transmitting data. C_{CPB} is the average cost of collection for one byte of data; it is defined as total energy consumption of the sensor network divided by the amount of sensory data collected by all sensors. The actual value of C_{CPB} relies heavily on the sensing modalities, device specifications, deployment scheme, and tracking algorithm, as well as external conditions such as temperature, background noise, etc. It indicates how effective the sensor network is, as a whole, at collecting sensor data for the application.

$$C_{CPB} = \frac{\sum_{i=1}^{N} E_i}{\sum_{i=1}^{N} D_{Ci}} \qquad \text{Equation 2-2}$$

Similarly, C_{iTPB} is the average cost of transmitting one byte of data inside the sensor network from source node to sink node, and C_{eTPB} is the average cost of transmitting one byte of data from the sink node/gateway to the control center. Accordingly, M_{Col}, M_{iTran} and M_{eTran} are defined as the amount of data to be actually collected by the sensors, transmitted internally, and transmitted from the sensor site to the control center during a specific time period.

The most straightforward way to reduce C_{CPB} is to cut down power consumption over the sensor network, assuming the amount of data to be collected remains constant. Most WSN research takes this approach, e.g., device level work [22], sensor management [26], and optimized tracking algorithms [29].

Note that the amount of data collected across the sensor network (M_{Col}) may differ from that transmitted to the control center (M_{eTran}) due to in-network processing, data fusion or data aggregation processes. Under normal conditions $M_{Col} \geq M_{iTran} \geq M_{eTran}$ holds true. For sake of simplicity, let's assume $\tilde{M} = M_{Col} = M_{iTran} = M_{eTran}$. We then compute approximate total energy consumption of collecting and transmitting $\tilde{M}$ bytes of sensor data:

$$\tilde{E}_{Total} = (C_{CPB} + C_{iTPB} + C_{eTPB}) \times \tilde{M} \qquad \text{Equation 2-3}$$

Although current research has successfully cut down C_{CPB} using different types of technologies, but unfortunately there is a limit on how low $\tilde{E}_{Total}$ can be if other parameters remain the same in the above equation. Let's assume that using a new optimization technique, we have a new $C'_{CPB} = \frac{C_{CPB}}{n}$ (n>1) which is one nth of the original C_{CPB}. Then the amount of power saving is given by Amdahl's law:

$$\frac{\tilde{E}'_{Total}}{\tilde{E}_{Total}} = \frac{C'_{CPB} + C_{iTPB} + C_{eTPB}}{C_{CPB} + C_{iTPB} + C_{eTPB}} = \frac{\frac{1}{n} + \frac{C_{iTPB} + C_{eTPB}}{C_{CPB}}}{1 + \frac{C_{iTPB} + C_{eTPB}}{C_{CPB}}} = \frac{\frac{1}{n} + K}{1 + K} \longrightarrow 1 \qquad \text{Equation 2-4}$$

$$(\text{where } K = \frac{C_{iTPB} + C_{eTPB}}{C_{CPB}} >> 1)$$

We can see that with the technological advancement, $\tilde{E}'_{Total}$ is smaller than $\tilde{E}_{Total}$. We can use Table 1 to get an estimate of K. According to Table 1, C_{iTPB} is an order of magnitude higher than C_{CPB}. And we all know that C_{eTPB} is much larger than C_{iTPB} because of the data transmission distance. For the sake of simplicity, we assume $C_{eTPB} = C_{iTPB} = 10 \times C_{CPB}$; then K=20. Now we substitute K back to Equation 2-4. We can see that the effort we spent to improve n results in almost negligible improvement on $\tilde{E}'_{Total}$.

2.3.2 Response Time

The current sensing architecture does not have a multiple-resolution tracking mechanism that can improve response time. All objects are treated with equal attention. Similarly, all events are processed with same effort regardless of the nature, place, or time of the event. Response time is in reverse proportion to the amount of sensing activities: the more sensing activities and processing involved, the slower the response. Although response time can be improved by including nodes that are powered all the time, this approach contradicts with the power saving principle.

2.3.3 Attention Economics: the Cost of Getting Useful Information

In 2.3.1, we showed that optimizing C_{CPB} brings very limited benefit in the long term. The current paradigm is also insufficient in addressing the user's need for valuable information. Under the current paradigm, users use the incoming data to get the tracking results. This works well enough when the user has sufficient storage and processing power to processing the incoming information. However, technological advances have made it cost-effective to use massively distributed sensor networks become the mainstream in sensing applications. The result is that information storage and processing resources on the user's side are barely adequate. This situation is even more severe for mobile information access applications, in which the user uses portable devices such as cell phones, personal digital assistants, or mobile computers to access and manage the sensor data.

Many designers of information systems incorrectly regarded their design problem as information scarcity rather than attention scarcity. As a result they built systems that excelled at providing more and more information to people, when what were really needed were systems that excelled at filtering out unimportant or irrelevant information. Truly useful information is often buried in tremendous amount of incoming data resulting in no proper response being made in a timely manner. It also results in compromised information accuracy due to susceptibility to data fluctuation.

Although the control center can utilize high-performance parallel computers to process and evaluate the incoming data, this is only a supplemental fix that adds to the total cost of distilling the information that the users need.

2.4 Need for a Paradigm Shift

Why is the current paradigm facing the problems just described in 2.4.1 and 2.4.2? It is because this supply-driven, provider-initiated paradigm does not allow the user to control the actual data being collected and transmitted. The user is a passive consumer of data, whose demand for data is not taken as an input for the supplier/provider – the wireless sensor network. With unit cost of wireless sensors continuing to drop, it will soon become economically feasible to deploy sensor networks with high redundancy. In face of this trend, the traditional paradigm is insufficient in resolving the dilemmas of unsatisfactory system lifetime and information overload.

Under the proposed new paradigm, the user can manage the tracking process from the application level while accepting the inherent limitations of individual sensor nodes. The new paradigm will allow us to control and manage M_{Col} , M_{iTran} and M_{eTran} .The new paradigm is not an incremental optimization method such as data fusion, data aggregation, or in-network processing, but is meant to reduce unnecessary data collection and transmission altogether. The goal of this new paradigm is to maximize the most important metrics of the WSN-based sensing system: life time, attention economics, and response time.

CHAPTER 3. BASIC THEORY OF LOD-BASED TRACKING PROCESS

As mentioned in Chapter 2, users are passive information receivers in the provider-initiated paradigm. Incoming data is not controlled by the users, but by the events that are occurring at the sensor site, which often results in excessive instantaneous data influx. On the other hand, what users need most is sustained high quality tracking service and maximal service time. The provider-initiated paradigm cannot resolve the conflict between these two. To solve the problem, we need an accessible and manageable sensing architecture that can deliver satisfactory tracking results while eliminating all unnecessary sensor activities, including data collection and communication. According to Equation 2-3, reduction in data volume linearly reduces power consumption. It provides a new dimension of power conservation in addition to traditional optimization work on reducing C_{CPB}, C_{iTPB} and C_{eTPB}.

The key of the proposed LOD based tracking process is to define a set of states or levels at which the sensor network operates. Each state corresponds to a particular level of sensing activities and energy consumption. The user receives different resolution and accuracy when the sensor network operates at different levels. By choosing to operate at a suitable level of detail, the system can achieve an optimal balance between system life time, attention economics, and response time.

Although such a level-based mechanism bears some resemblance to conventional levels of abstraction commonly used in 3D graphics rendering [26], there are several major differences. Conventional 3D graphics LOD involves representing (rendering) multiresolution mesh and texture data in an effort to lessen the load on local CPU and GPU. Since mesh and texture data of the highest resolution already exists locally in the main memory/video memory, what the LOD algorithm in 3D graphics simply drops unnecessary detail in real-time rendering based on a set of criteria (distance, view angle, object occlusion, etc) [27]. In contrast, the LOD based tracking process does not have any data stored beforehand. It is a dynamic process aimed at retrieving, transporting, and representing data at the user's demand at minimal cost. Consequently such an LOD process would enable optimization for a much wider spectrum of devices, including onboard MCU of sensors, onboard RF module of sensors, long range communication device equipped at the sensor site, as well as local CPU and GPU on the user's end.

The proposed LOD based tracking process is relevant to a set of real-world problems that share a similar underlying mechanism. These problems involve tracking targets in a moderately occupied wide-area environment. Typical applications include, but are not limited to, border control, pipeline monitoring, emergency services such as disaster relief operations, etc. The remainder of this chapter is focused on defining key elements for the LOD based tracking process.

3.1 Defining Levels

The first step in building the LOD based tracking process is to define the available level of details. Since there are only limited types of sensing modalities under current technology, including motion detectors, positioning sensors, and vision sensors, the sensor network can only operate at a discrete level L_i, where $i = 1,2,...K$. Any specific level is characterized by a unique combination of sensors' states: *OFF*, *STANDBY*, and *ON*. At any time, the user can dictate the sensor network, or a portion of the network to operate on a level that simultaneously satisfies the need for tracking accuracy and minimizes power consumption.

Definitions of levels also depend on target applications. The class of applications this research work addresses is tracking the movement of objects in a moderately occupied wide-area wilderness environment, hence the level classification shall satisfy the users' need in answering the following questions: is an object present in the tracking field, where is it, who or what is it, and what is it doing [28], [33]. The levels are formally defined as entity, object refinement, and situation refinement.

Target tracking in the WSN environment is a data intensive process. Throughout the entire process of tracking, the sensors need to provide enough data to determine the presence of objects, and their precise positions. They should also provide assistance in identifying the objects and even assessing the situation. These three types of activities involve different kinds of sensors, as well as different types of data acquisition and processing operations. When energy consumption is used to measure the operational cost, different degrees of sensor activities translate into different amount of energy consumption. By adopting such a unified measurement, we will be able to model and analyze the operational cost of the entire sensor network in subsequent chapters. Section 3.1.1 through 3.1.3 will discuss the definition and representation of each of these levels, as well as sensing activities that are associated with each level.

3.1.1 Level 1: Entity Existence

Level Definition

This is the lowest level in the tracking process. Detecting the presence of an entity requires the least computation and power consumption of all sensing activities. An entity has a distinct material existence. An entity could be a human being, a non-human animal, a non-thinking life-form such as a plant or fungus, a lifeless object such as a vehicle. Although this level does not tell the user more than the presence of an object or a group of objects in the target area, the information serves several other purposes.

First, detection of an object is the basis for other sensing activities and operations at higher levels of detail. This is due to considerations of power consumption. According to Table 2, passive IR motion sensors use the least power compared to other sensors. Thus it is desirable to keep these IR motion sensors in active mode and others in sleep mode, so that power consumption across the entire network is maintained at the lowest level possible. Once an object triggers an event with the IR sensor, it is up to the user to determine whether it is necessary to perform higher level tracking operations.

Second, the entity level can serve as a low resolution positioning method. When the sensor network is deployed in the sensing field, the entire network goes through an initialization process to identify the location of each individual sensor. A common method is to equip the sensor network with a limited number of GPS enabled nodes, and to use triangulation to locate the other sensor nodes. Assuming that the sensor network uses IR line-breaking sensors to detect existence of objects, once an object triggers an event with the IR sensor, it is simple to narrow down the object's position to the sensing range of the IR sensor. This low accuracy positioning capability is sufficient for tracking object existence and movement when the object is far away from ROIs.

Sensing Activities

This level requires IR line-breaking/motion sensors to be in active mode while others are in sleep mode. These sensors are widely deployed in the sensing field to form a grid of cells. Each cell is guarded by a number of such sensors. Once an object triggers an event with the IR sensor, it is reported to the user, who will then determine whether it is necessary to perform higher level tracking operations.

Representation

At this level, the entity is represented by a particle (a solid sphere) in the tracking application. The particle is assigned to the cell in which it is spotted. When the object moves from one cell to another, the particle is positioned in the new cell accordingly. Such assignment gives the user the rough information about the object's location. However, no further information regarding specific location, velocity, identity, etc is available.

3.1.2 Level 2: Object Refinement

Level Definition

This is the second level in the LOD hierarchy. It is meant to give the user much more information than the mere existence of an object in the sensing field. The information provided at this level refines the user's understanding of the object's status and activities, including accurate position of the object, motion, and physical characteristics such as shape, size, etc.

Knowing the object's physical characteristics has another important use in the tracking process itself. Many researchers [28, 29, 32] propose tracking algorithm based on prediction of the object's next move. Knowledge of the object's physical characteristic can help the tracking algorithm to more accurately predict its next move without activating unnecessary sensors, thus further saves power consumption.

Sensing Activities

Locating the object's position is the basic operation at this level. This involves sensors such as ultrasonic range finders. In order to compute the object's position, we need at least three sensors within sensing range to triangulate the object. Normally only one object will appear in one place at one time because of the low occupancy of the application. Sometimes there are several objects presenting in one place and triangulation algorithm will fail because of multiple echoes. Vaidya et al [28] propose a method of tracking multiple targets without losing track of any individual target. Once the position of the object is known, it is rather simple to derive the object's velocity.

The second aspect of the level is to gain a basic understanding of the object's physical characteristics. This requires some basic image processing on the imagery data. Assuming there is no object in the target area when the sensors are first deployed, the vision sensors can take pictures and save them as background reference. Once an object is detected and notified by IR motion sensors at level 1, the video sensor in sensing range can take another picture with the incoming object inside the view. By performing pixel level subtraction, we can extract the image of the foreground object.

The following three images illustrate such a foreground object extraction sequence using a feature-tracking program [Appendix A: FeatureTracker Program]. The program first uses a web camera (vision sensor) to take a picture of the field and saves it as background reference; second, the program takes another picture with a toy car added to the scene; third, the image of the foreground toy car is extracted by subtracting the background image from the second picture.

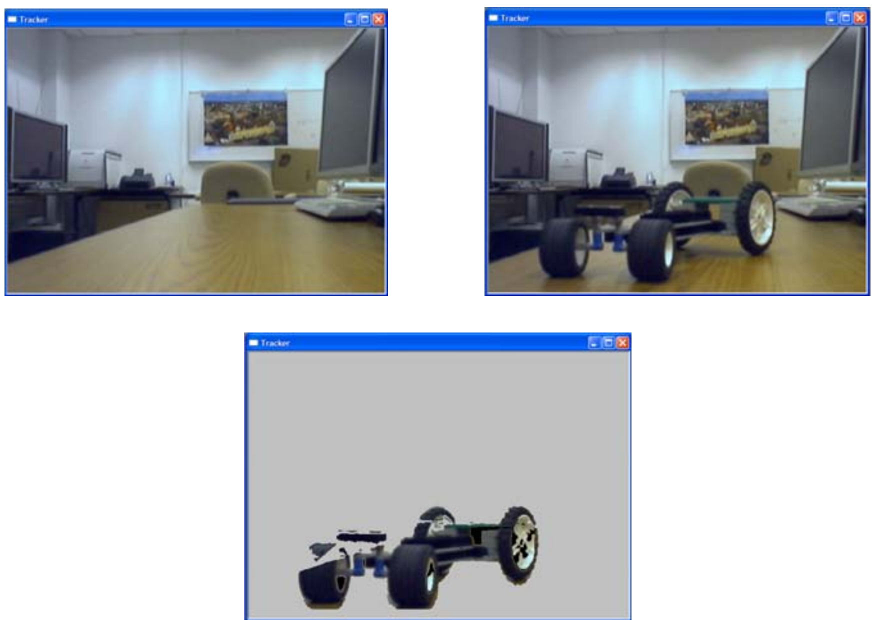

Figure 1 (a). Background (b). Object in Foreground

(c). Extracted Foreground Object

Normally, the image processing result is sent back to the control center. The user will use the picture of the foreground to construct an abstract symbolic model as shown in the following Figure 2. At the same time, the user can use the picture to determine the type of the object, and select an appropriate model from a library of 3D models. The selected model is the basis for subsequent pose estimation and motion analysis. Such a 3D model library is created by artists and contains 3D representations of objects (human being, animals, vehicles, etc) that can potentially appear in the target application.

Representation

At this level, the target is represented by abstract symbology which carries additional information such as approximate height, size, and head-shoulder ratio, provided by the data fusion engine. Abstract symbology is a novel way of representing information. By grafting the 2D unit symbols onto 3D abstractions such as the cube shown in Figure 2, the system allows users to apply existing domain knowledge to understanding the three-dimensional tracking space representation. Furthermore, additional information can be displayed on the other sides of the cube [32].

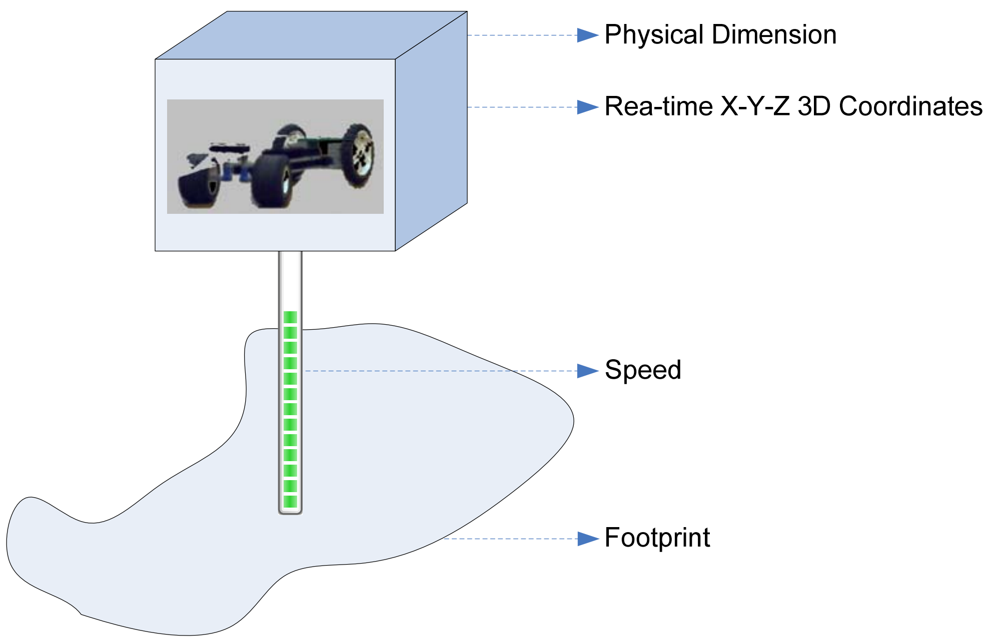

Figure 2 Abstract Symbology

The 3D symbol can take any form, as long as it is convenient for rendering and data representation. For instance, in the above proposed symbology, the footprint could morph to a real ground trace using GPS data. The stem could show actual locations and could be used as a speedometer display for moving speed. The surfaces of the cube may be used to abstract various types of information (e.g., side one: the strength of the force; side two: estimated time to destination, etc.). It is a semantically rich vehicle for information representation.

3.1.3 Level 3: Situation Assessment

Level Definition

This is the highest level in the tracking operation. Situation assessment is the process of gathering and analyzing the information to make an explicit evaluation of an entity or organization in its environment. This level needs to provide enough information for the user to assess the current situation with certain degree of confidence. For a typical tracking process, this requires accurate position information of the object together with frequent visual data of the scene.

Besides instantaneous information of events in the sensing field, historical data can greatly assist the user in assessing the situation and making correct decisions. Such historical data include movement paths of objects, previous appearances of an object, and records of interaction between objects. All of these allow the user to predict object movement tendencies, analyze intentions of objects, correlate events at different locations, and evaluate the impact of ongoing events.

Sensing Activities

This level requires the sensor network to closely monitor the object's motion. It relies on positioning sensors and vision sensors. The positioning sensors in the vicinity of the object are kept on to track the position and movement path of the object, and the vision sensors are turned on to take pictures of it. The visual data can either be sent to the control center directly, or can be processed inside the sensor network to distill useful information. Internal processing also requires a memory of the object's previous states (position, action, etc) to facilitate analysis of the object's movement path, interaction and interconnection with other objects.

Representation

Traditionally, video streaming is a popular technique to give the user continuous information of the object states. However, video streaming is not an ideal option in wireless sensor networks for several reasons. First, video streaming requires intensive use of vision sensors, which consume far more energy than other types of sensors. Second, video streaming requires a large amount of data communication, which consumes more power than data collection. Third, a video stream is not interactive. The user cannot directly operate on objects contained in the video frames.

Computer-animated characters and scenes are a better alternative for representing the data at level 3 tracking operations. Computer animation does not require constant video data transfer from the sensor site to the control center. A carefully optimized motion analysis and scene reconstruction system can greatly reduce the amount of data transfer necessary, while providing the user with unparalleled flexibility to interact with the objects in the virtual tracking environment.

A CG generated virtual tracking environment consists of two parts, the sensing field and the objects, whether human beings, non-human animals, or lifeless objects. The sensing field is constructed based on the real-world physical environment, including terrain, plants, buildings, etc. Information about these natural elements can be obtained from various sources, e.g. GIS database, satellite images, and aerial photos. On the other hand, movable objects in the tracking environment are the results of object recognition at level 2 combined with human intervention.

Depending on what the object is, the virtual tracking environment can use either a static or dynamic model from the library to represent the object. Both models provide a richness of information about the object's anatomical characteristics and help the user to discern the object's identity. The static model is well suited for representing rigid objects whose movement of body parts is neglected. A typical example is an automobile in the sensing field. Accurate tracking of the rigid object's 3D-pose, although achievable on a real-time basis [35], is rarely performed on a frame by frame basis due to power consumption consideration and relaxed tracking requirements. As a result, in most tracking scenarios the position and direction of movement are sufficient. On the other hand, human beings or other life forms can only be represented by dynamic models with key-frame or skeletal animation capabilities. Chapter 5 briefly discusses the mechanism of motion analysis and object pose estimation techniques in the current literature.

However, the actual algorithms of motion analysis are up to the designers of sensor devices and/or embedded operating systems, and beyond the scope of this research.

3.2 Determining LOD in Tracking

We have defined three levels of detail at which the sensor network will operate. These levels correspond to the processes of human perception, cognition and action. The next question in real-time LOD-based tracking applications is how to determine which level to operate. When an object first appears in the sensing field, it triggers an event in the sensor network and consequently draws a certain level of attention from the user. We need a set of objective measures to determine the significance of the event, and combining that with knowledge about the object, to determine the right level of operation. The objective measures to be used are spatial and temporal significance factors.

3.2.1 Spatial Significance Factor

In wide-area tracking applications, the movement speed of object of interest is very small compared to the geographical area of the sensing field. Assuming that the number of Regions of Interest (ROIs) in the sensing field is a finite integer M, we can denote these ROIs as $\Psi_{ROI} = \{\psi_j \mid j = 1,...,M\}$. Each element ψ_j of the set is a user defined, application specific geographical location. For instance, an ROI can be the position of critical infrastructure, access to a restricted building, a bridge, a cave, etc. If the sensing field is large enough (normally the case for wide-area tracking applications), the total area of these ROIs constitutes only a very small partition of the entire sensing field. As a result, object activities outside of Ψ_{ROI} are of less significance to the users.

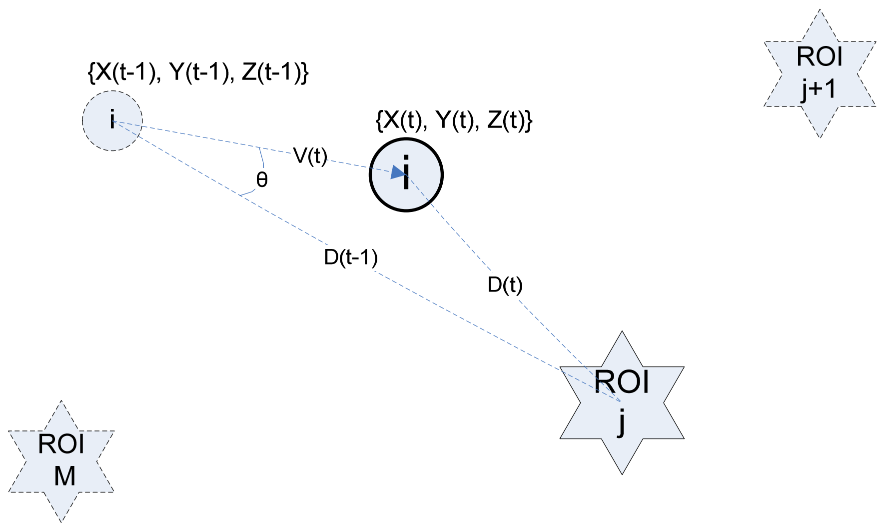

Figure 3 Sensing Field

Consider the sensing environment as shown in Figure 3. The sensing field has M independent ROIs defined. Object i moves from location $\{x_i(t-1), y_i(t-1), z_i(t-1)\}$ to the current location $\{x_i(t), y_i(t), z_i(t)\}$ during time period Δt. We claim the following parameters determines $S_{ij}(t)$, significance factor with regard to object i and ψ_j.

- $D_{ij}(t)$: the current distance between object i and ψ_j.
- $\theta_i(t)$: directional angle
- $\vec{V}_i(t)$: velocity
- W_j : weight of ψ_j.

From Figure 3, it is clear that $\theta_i(t)$ and $\vec{V}_i(t)$ can be calculated using current location and previous location at time instance (t-1. We then have the following:

$$S_{ij}(t) = W_j \times F_{ij}(x_i(t), y_i(t), z_i(t), x_i(t-1), y_i(t-1), z_i(t-1), \psi_j) \text{ Equation 3-1}$$

This equation is essentially a function that maps the object's current position, previous position, and ψ_j to a number. Since Ψ_{ROI} is a set of ROIs for the current tracking application, $S_i(t)$ is thus the weighted sum of $S_{ij}(t)$, the significance factor of object *i* to the jth ROI ψ_j. That is:

$$S_i(t) = \sum_{j=1}^{M} S_{ij}(t) = \sum_{j=1}^{M} W_j \times F_{ij}(x_i(t), y_i(t), z_i(t), x_i(t-1), y_i(t-1), z_i(t-1), \psi_j)$$

Equation 3-2

$S_i(t)$ determines how important the object *i* is, based on a set of criteria. The actual form of F_{ij} heavily depends on the user's real-time need for tracking quality, the nature of ψ_j, and the sensing environment setup. A deterministic expression is proposed in Chapter 6 where all these parameters are known when setting up the simulation.

3.2.2 Temporal Significance Factor

Another dimension of measurement also needs to be taken into consideration for the calculation in the LOD tracking process. This dimension is temporal information. A given event can be of significantly different impact if it happens at different times. Similar to the ROIs discussed above, it is possible to define a set of Time of Interest (TOIs) as this: $\Phi_{TOI} = \{\tau_k \mid k = 1,2,3,...P\}$.

Each element τ_k in the TOIs defines a specific time slot. If an event happens inside τ_k then there is a specific significance factor associated with it. Compared to ROIs, calculation of the temporal significance factor is a much simpler as it depends only on the current time *t*. In fact, in most cases the user needs only to define a time function G that maps t to a specific number. This number serves as a coefficient for the calculation of $S_i(t)$. The following is the revised equation to compute $S_i(t)$ that has taken into account both ROIs and TOIs.

$$S_i(t) = \sum_{j=1}^{M} S_{ij}(t) = \sum_{j=1}^{M} W_j \times F_{ij}(x_i(t), y_i(t), z_i(t), x_i(t-1), y_i(t-1), z_i(t-1), \psi_j) \times G(t, \Phi)$$

Equation 3-3

The actual form G heavily depends on the user's real-time need for tracking quality, the nature of the application, and the sensing environment setup. For instance, if the application is intended to study the behavior of a nocturnal animal in its habitat, then the sensor network would naturally pay more attention to activities happening at night. In other words, G will yield a higher number if t falls in the range of the animal's most active time window. The following figure shows a sample staircase function of G that maps the twenty-four hours in a day into a unique normalized weight.

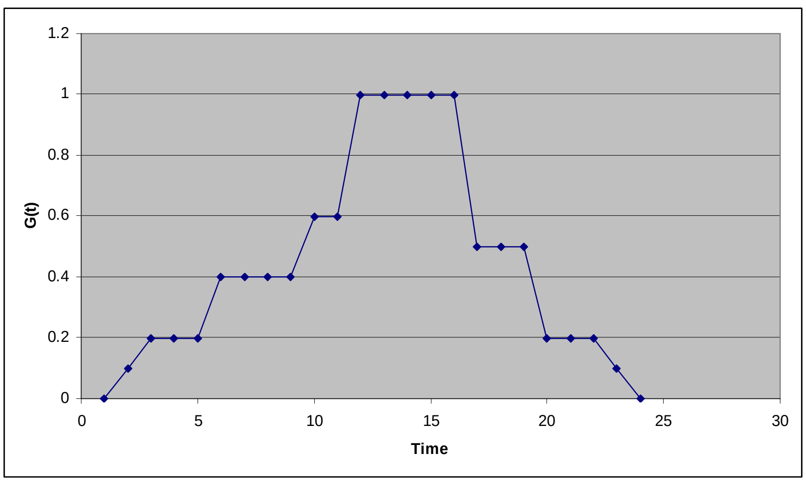

Figure 4 Sample Function of G

3.2.3 Objective Measure and Subjective Judgment

At any time, an object *i* poses a certain significance to the user as defined in Equation 3-3, based on both spatial and temporal considerations. The significance factor is an objective measure, telling the user to what level of attention the object should receive, based on the calculated values of F_{ij} and G. However, the actual level of details the sensors operate depends on another factor as well: the user's input, $INS_i(t)$

$$L_i(t) = S_i(t) \times INS_i(t) \qquad \text{Equation 3-4}$$

$INS_i(t)$ is an indication of how much the user is interested in what is happening in the sensing field with regard to object *i*. $INS_i(t)$ is usually affected by a priori information regarding object *i*. For instance, in a habitat monitoring application, if the type of species that the scientists are interested in are birds, they will pay much less attention to the activities of other wild animals. Similarly, in an object tracking and surveillance scenario, the user will be most interested in detecting activities of unauthorized personnel in restricted areas, while much less concerned about authorized personnel in the same area. It is clear that such a priori information about an object is crucial in determining $INS_i(t)$.

In the definition of Level 2 Object Refinement we have mentioned that when an object is first discovered, a picture of the object is sent to the user for object recognition purpose. After receiving the new object's picture, the user has a good understanding of what type of object is being tracked and $INS_i(t)$ can be determined accordingly.

As a result, the actual level of detail at which that the sensor network operates, depends on both the objective measure of the significance factor and the subjective judgment of the user. By incorporating these two aspects into real-time tracking applications, we have a mechanism that allows us to dynamically adjust the sensing activities of the sensor network temporally, spatially and objectwise.

3.3 LOD in Operation

3.3.1 Workflow

LOD is a mechanism that dynamically adjusts sensor activities based on the real-time value of $L_i(t)$. It adds the following process to current existing sensor networks:

1. Compute $S_i(t)$
2. User determines the minimal level of details $L_i(t) = S_i(t) \times INS_i(t)$ needed for real-time tracking
3. Send user command $U_c(t)$ to sensor management layer.
4. User command $U_c(t)$ is received and interpreted by sensor management layer
5. Sensors are activated and deactivated based on $U_c(t)$
6. Data is collected and transferred back to the user.

Now by inserting the LOD process into a traditional sensor network work flow, we have the following process:

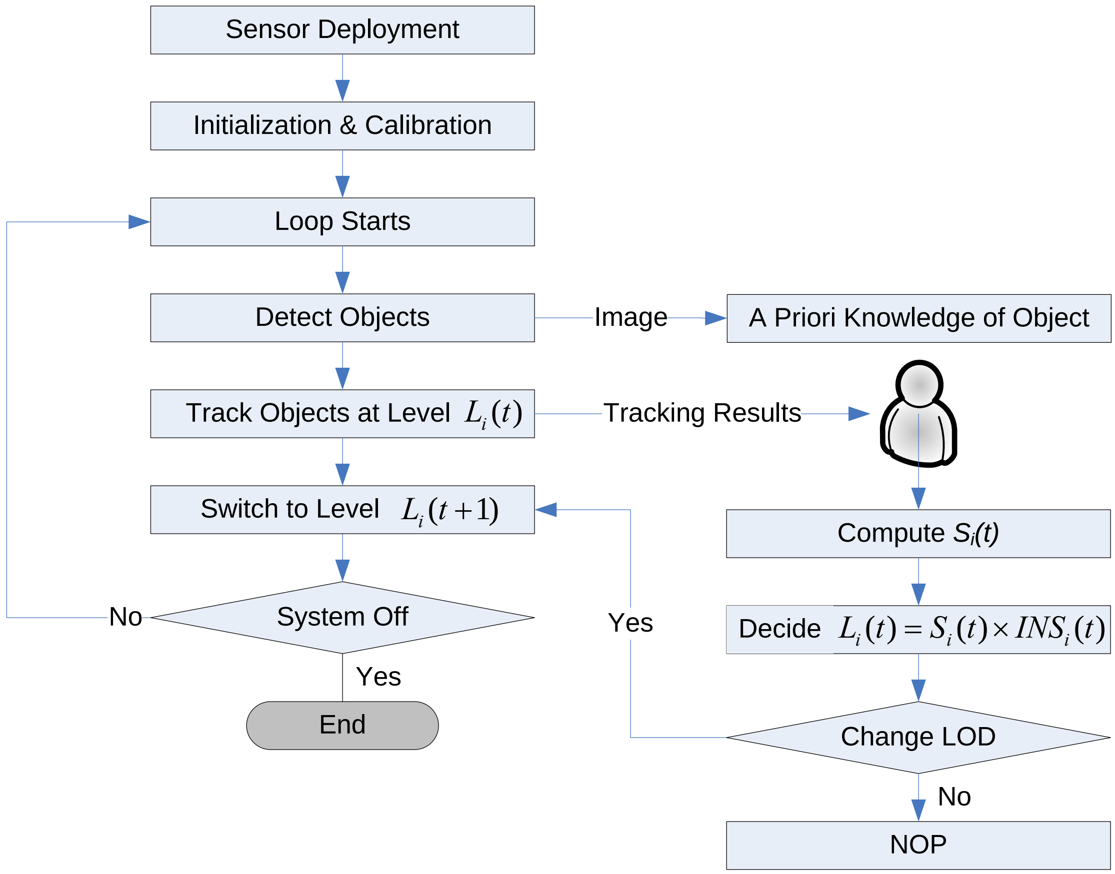

Figure 5 LOD Tracking Process

At each iteration, the user's demand for information is taken into consideration in the tracking process. In essence, LOD allows both wide-area surveillance and precision tracking by a multi-target multi-resolution tracking mechanism. The resolution of data obtained from the target scanning region is adjusted spatially and temporally to provide a tracker the desired quality of service at minimal operational cost.

3.3.2 Seamless Switching Between Levels

Switching between different levels of detail is a remote controlled process. First the user combines the significance factor of the current object with his own interest in the object, then he decides on a level $L_i(t)$ and issues a command from the control center. The command is wirelessly sent to sink node using a predetermined message definition. Upon receiving the command packet, the sink node will broadcast the message to the sensor network. The sensors will continue to operate at level $L_i(t)$ until a command to switch levels arrives. When the sink node broadcasts the LOD messages to the sensor network, there are several possible scenarios: per object, per cell, or global.

Adjusting LOD per object is ideal for cases where objects are sparsely distributed in the sensing field and far from each other. Under these conditions, there is very little interaction between objects, and one object can be easily separated from others based on its location. Thus it is possible to issue LOD commands to the sensors in the vicinity of the target, yet changing the level of these nearby sensors will not affect the tracking results of other objects in the field.

Adjusting LOD per cell is ideal for cases where that objects are relatively close to each other (for instance, within the same cell), and have similar significance. Under such conditions, the sensors in the same cell can be set to the same operational level without other complications. What the sink node needs to do is multicast the LOD command to the sensors in the cell where the objects reside. Other parts of the network are unaffected.

Adjust LOD globally is ideal for tracking applications in which object occurrence is very low. Under such conditions, there is usually one object or even no object at any time instance. All the sensing activities are associated with that specific object. The sink node needs only to broadcast the LOD command to the entire network and every sensor switch tracking level accordingly.

CHAPTER 4. LOD BASED APPLICATION FRAMEWORK

A complete sensing system has three major components: a network of sensors that are deployed in the remote site, a local control center, and a communication link that connects these two sites. Figure 6 shows a typical tiered architecture with these components. Many research groups have worked on different aspects or layers of sensing systems, from tiny operation system (TinyOS) on the sensors, sensor network protocols, to higher level tracking algorithms and applications. Each layer has its own architecture and sub-components. An LOD-based application framework needs to glue together these components while adding the capability to adjust the level of detail in tracking. The added LOD capability will not interfere with the internal structure of the components. To satisfy these requirements, an independent layer needs to be added to the currently existing sensing system architecture.

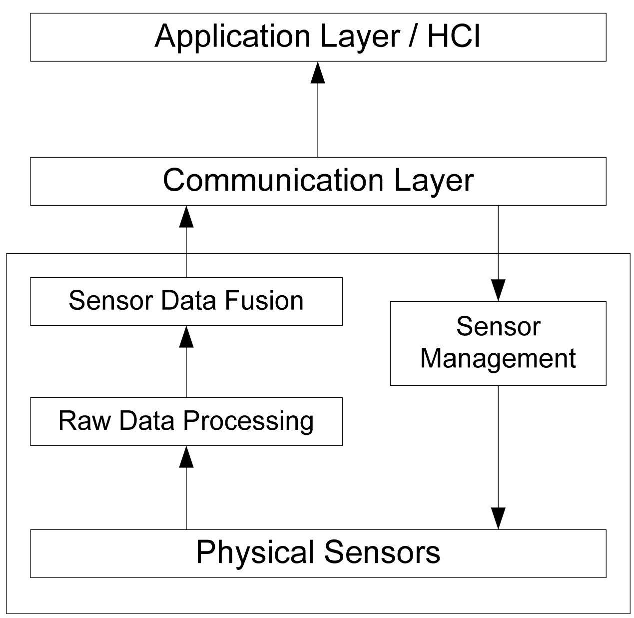

Figure 6 Common Tiered Architecture

4.1 The LOD Enabled Sensing Architecture

Adding LOD capability to the sensor network requires a new architecture that incorporates LOD related functionalities. Compared to conventional tiered architectures, the LOD layer is a major architectural change, requiring some minor changes to the Sensor Management module to adjust the sensor tracking level based on $L_i(t)$. The new module is highlighted in Figure 7.

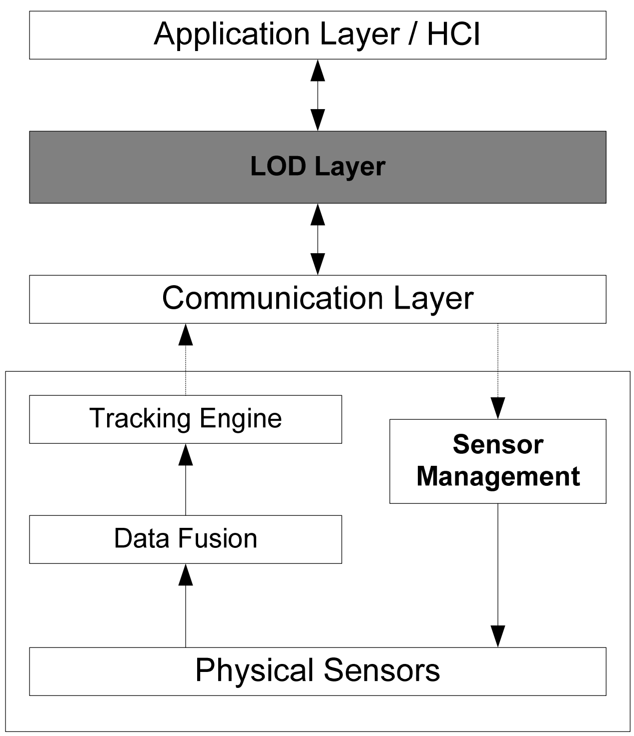

Figure 7 Revised Sensing Architecture

The remainder of this chapter discusses the functionality of each layer in the LOD based application framework.

4.1.1 Physical Sensors Layer

Sensing architecture contains a group of physical sensing devices responsible for collecting sensory data from the environment. For wide area tracking applications, we use the following type of sensors:

- Event-detecting sensors: these sensors consume very little battery power and have a much longer lifetime than other types of sensors. They are switched on all the time. Once a sensor detects an event such as temperature or light change, it notifies the head nodes in the sensing zone. A sensor network with only event-detecting sensors working is considered to be at Level 1.
- Positioning sensors: Because of the relatively high power consumption, these sensors will sleep most of the time to minimize power consumption. Once they receive an activation command from the head nodes, they will wake up from sleep mode and start to track the position of the target. A sensor network with position sensors working is considered to be at Level 2.
- Vision sensors: To reduce power consumption, vision sensors are normally kept in sleep mode unless they receive an activation command from the head nodes. The vision sensors collect the largest amount of data including sound, image, and even real-time video. A sensor network with vision sensors working is considered to be at Level 3.

Using these three types of sensor nodes, users can not only build heterogeneous sensor networks for different tracking and surveillance applications, but more importantly, they can get the data needed for modeling moving objects based on the three levels of modeling requirements of LOD.

When the sensors are deployed in the field, all sensor nodes are registered to their regional head nodes and the sensor gateway during the initialization and calibration stage. After that, only the event-detecting sensors are switched on to minimize the system's power consumption. Once event-detecting sensors detect an event, the sensor network immediately sends a message to the control center, where the level of operation for the next moment is decided.

4.1.2 Data Fusion Layer

The sensor data fusion module in a heterogeneous sensor network plays an important role. It is responsible for combining sensory data or data derived from sensory data from disparate sources. Fused data are sent to another node or layer. The intention of data fusion in a sensor network is multiple. First, by fusing data from multiple sensors of the same or different types, the resulting information is in some sense better than these sources used individually. Second, the traffic load is reduced when multiple sensory data are fused. Third, better energy efficiency is achieved by reducing the amount of data transfer.

Generally, a fusion engine has three design issues: the reporting mechanism, the fusion decision and the fusion architecture. Fused data are sent to the tracking engine, where the object's current position, movement trajectory, can be derived.

4.1.3 Tracking Engine

Reliable tracking of targets using sensors placed throughout the sensing field involves determining the locations of any targets present in the space. Over the years, many location estimation approaches have been introduced using sensors such as range finders, sonar, radar sensors, cameras, etc. Although vision sensors can also provide location information with image processing techniques, they require too much power to operate. Hence sonar sensors remain the mainstream positioning sensors.

Generally, tracking an object requires three sonar sensors within the sensing range to work together and triangulate the object at time instance t. More power-aware tracking algorithms try to predict the object's movement for the next time instance and schedule activation and deactivation of sonar sensors to save power.

Let there be N objects present in a confined space. The location of each object at time t is given by $X_{si}(t)$, $Y_{si}(t)$, where $i = 1$ to N. The next location of each object at time $(t+\Delta t)$ is predicted to be $X_{pi}(t+\Delta t)$, $Y_{pi}(t+\Delta t)$. From the previous location, $(X_s(t-\Delta t), Y_s(t-\Delta t))$, and the current location $(X_s(t), Y_s(t))$, we can predict the next location of the target at $t + \Delta t$ as : $(X_p(t + \Delta t), Y_p(t + \Delta t))$ using the velocity of the object. Consider two-dimensional movement. The velocity is given by:

$$v = \frac{\sqrt{[X_s(t) - X_s(t-\Delta t)]^2 + [Y_s(t) - Y_s(t-\Delta t)]^2}}{t-(t-\Delta t)} \qquad \text{Equation 4-1}$$

and the direction by:

$$\theta = \tan^{-1}[(Y_s(t) - Y_s(t-\Delta t))/(S_s(t) - S_s(t-\Delta t))] \qquad \text{Equation 4-2}$$

We can calculate:

$$X_p(t+\Delta t) = X_s(t) + v * \cos\theta * \Delta t \qquad \text{Equation 4-3}$$

$$Y_p(t+\Delta t) = Y_s(t) + v * \sin\theta * \Delta t \qquad \text{Equation 4-4}$$

Given $X_p(t+\Delta t)$, $Y_p(t+\Delta t)$ and a sensing range r for the sonar sensors, the next step is to find the next set of k sonar sensors to cover the predicted location. Based on the prediction, the sensor management module schedules activation of the neighboring sensors. If the target indeed moves to the predicted position, sensors around its previous position can be sent to sleep mode. If the object does not move to the predicted location, the sensor management module needs to activate more sensors in the vicinity and expand the search area to rediscover the object. The search process continues until the lost target has been rediscovered. Such a process will consume extra power. The situation can be mitigated by combining inputs from IR line-breaking sensors into the prediction stage and improving prediction accuracy. IR-line breaking sensors tell us which cell the target is currently in. If there are multiple objects present within the sensing range of the sensors, then the sensor managing system will keep track of each object.

During the tracking process, when multiple objects are close to each other they become ambiguous to track. In conventional tracking algorithms, this ambiguity is caused only by objects being too close to each other. However, in a LOD based tracking process, the situation is more complex. There are several degrees of ambiguity associated with each tracking level. The next section (4.1.4) will discuss such ambiguities for every tracking level.

4.1.4 LOD Layer

This software layer is the core of the LOD based application framework. It encapsulates all the LOD related functionalities defined in Chapter 3. Specifically, the LOD layer is responsible for three tasks: calculating the significance factor based on the tracking results, providing the tracking results to the user, and translating the user's inputs into formal LOD commands. As mentioned in Chapter 3, significance is a function of time, object location, and a set of ROIs and TOIs. The actual calculation of the significance factor depends on the application, and is often up for change in real-time. For instance, experiments discussed in Chapter 6 use normalized linear distance to calculate the significance factor and uniform time function $G\{t, \Phi\}$.

These calculation and operations are all performed in the software layer. Since the tracking layer is only responsible for finding where the physical objects are, while the application/HCI layer is only responsible for representing the objects in a virtual 3D environment, it is the LOD layer's responsibility to link the physical objects with the virtual software objects. This requires modeling objects with different levels of abstraction and maintaining an appropriate data structure for all objects. For this purpose, the LOD layer has a linked list of nodes instantiated from the CNode class, as shown in Figure 8. Each CNode object of the list corresponds to a physical object in the sensing field.

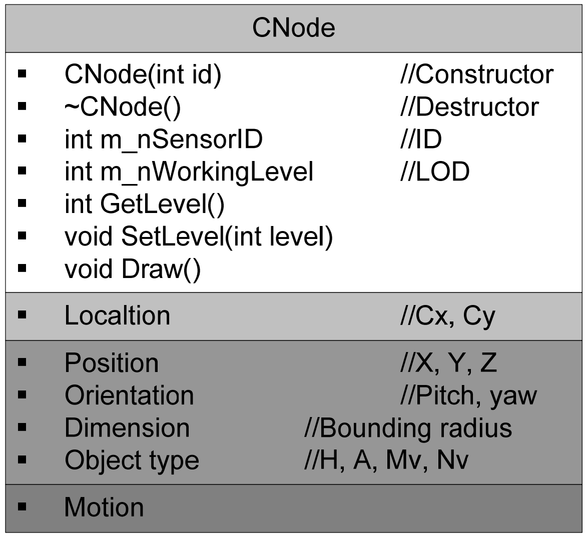

Figure 8 CNode class carrying different level of information

The LOD layer is also responsible for resolving tracking results from the tracking layer and mapping the results of physical objects to the CNode objects in its data structure. The mapping process involves searching through the list and finding the corresponding objects. When there is only one object, or there are multiple objects but no ambiguity, this mapping process is straightforward. However, when there is ambiguity, the one-to-one mapping fails and the LOD layer cannot link the physical object with its corresponding CNode software object. This usually requires escalating the level of detail, or even bringing the user into the loop to resolve the ambiguity.

The CNode class has a set of basic member variables and operations, including the node ID, and the current tracking level for the node, as well as methods to draw the node, get the current tracking level of the object, and the change tracking level of the object. More importantly, the class has member variables that maintain different levels of information (existence, object, and motion), highlighted with different grayscales in the above figure. These different levels of information correspond to the three level of detail in the tracking process. They are retrieved upon request of the user. During the tracking process, the CNode object dynamically updates the member variables that are relevant to its current tracking level and to lower levels. For instance, tracking at level 2 will update both Existence and Object information, but not motion information. The remainder of this section describes in more details how these levels of information are updated in real-time.

Existence Information

It provides rudimentary information of a newly discovered entity whose identity is unknown. An entity's status is updated when it triggers any of the IR line-breaking/motion sensors in the field, which indicates its location has changed. There are three possibilities for such an event. First, an empty (previously with no entity present) perimeter cell detects an entity. This means an entity has crossed the boundary and entered the sensing field (shown in Figure 9a). Since this is a newly discovered object, the LOD layer will instantiate a new CNode object and add the node to its existing data structure. When this new CNode object is created, a unique ID is assigned to it and the entrance event E_i is immediately reported to the control center.

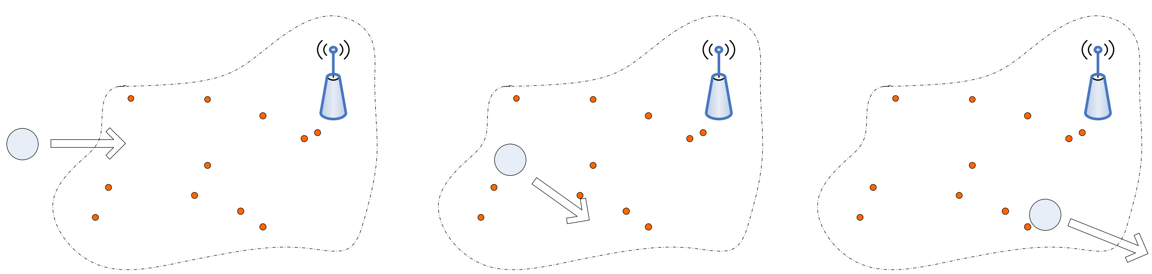

Figure 9 Entity Status Change:

a). Entering Sensing Field b). Moving in Sensing Field c). Leaving Sensing Field

The second case is an internal cell detects an object crossing its cell boundary (as shown in Figure 9b). This indicates an entity has moved inside the sensing field from one cell to the current cell. Depending on whether or not this cell was originally empty, two types of actions may be taken. If the cell was originally empty, then a simple entity moving event M_i is reported to the control center and no further action is needed. But if the cell had other objects in it, entity ambiguity will arise between the incoming object and those already present in the same cell. The objects cannot be differentiated because they are in the same cell and have the same (Cx, Cy) coordinates. This is classified as first degree ambiguity, as shown in the following figure (the orange dots are sensors).

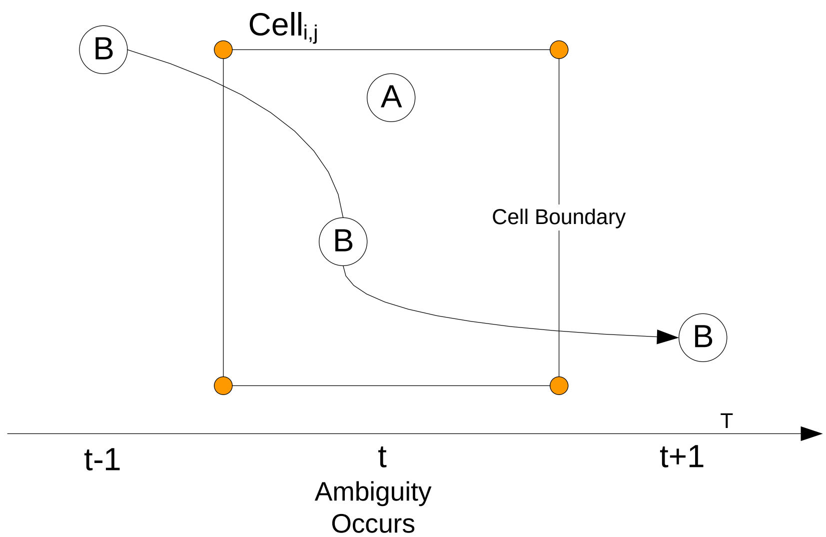

Figure 10 First Degree Ambiguity: Coexistence in Cell

This problem can be solved by escalating the sensors of the cell to a higher level of detail, for instance level 2. At level 2, sonar sensors are able to detect an accurate position for each object. The position information provides a higher resolution for discerning different objects. Changing the sensors to level 3 will achieve the same result; however, this would incur additional power consumption. Hence the user will do it only when level 2 cannot successfully resolve the ambiguity.

The third case is when a perimeter cell detects an entity moving from the cell towards the outside of the sensing field. Such event indicates the object is leaving the sensing field (as shown in Figure 9c). The CNode object associated with this entity is deconstructed, and the ID is released and becomes available for future use. The exit event X_i is reported to the control center where the user understands that the entity has left the sensing field. The same entity may return to the sensing field at a future time. When it does, an event like the first case is triggered and the tracking process will go through the above mentioned process again.

Object Information

It provides high-level information regarding the physical object itself. This high level information is useful in answering the user's questions regarding what the object is and where the object is. When the user first receives an entrance event E_i from the sensor network, the user requests an initial image for that newly discovered entity. Vision sensors will use the foreground segmentation technique described in Chapter 3 to get a picture of the object. Once the user receives the picture, a series of analyses of the object's characteristics can be conducted, including object type, object dimension, etc.

An object can be any of these types: human being, non-human animal, non-thinking life-form, or lifeless object. Under most circumstance, the initial image of an object is sufficient for the user to determine its type. Once the object's type is determined, the LOD layer will fill in the related fields in the CNode.

If the user decides to track the object at level 2, then no matter where it goes inside the sensing field, it will be traced by sonar sensors in its vicinity. The tracking layer provides the object's position to the LOD layer, which updates the CNode object's member variables. If an object moves too close to other objects, then object ambiguity arises. The sonar sensors have difficulty differentiating one object from another if they are too close to each other. The reason why sonar sensors are unable difference is due to multiple echoes they receive. Recall that a sonar sensor calculates object distance by measuring the time difference of arrival (TDOA). When there are multiple objects inside its detection range, the sensor will receive multiple echoes of the ultrasonic sound wave it transmits and get confused. This situation is classified as second degree ambiguity, as shown in the following Figure 11.

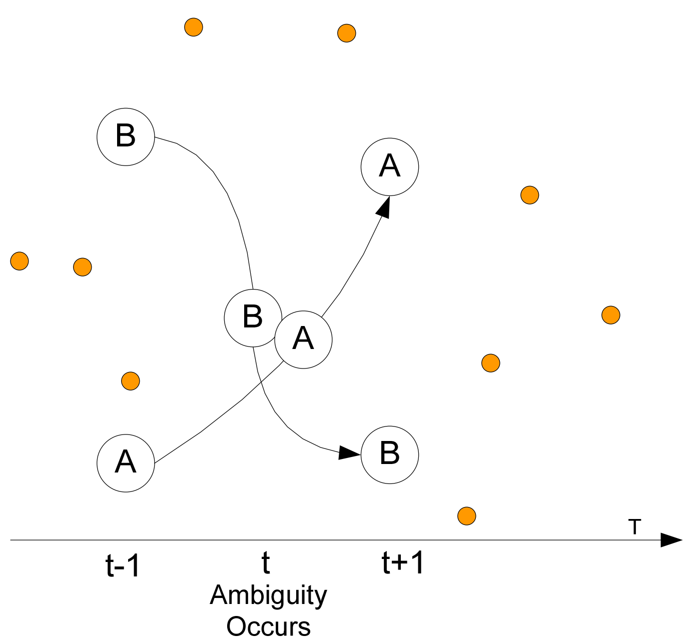

Figure 11 Second Degree Ambiguity: Spatial Correlation

Vaidya et al propose to use vision sensors to differentiate objects in such cases of ambiguity [28]. In their algorithm, vision sensors in the nearby area are activated to take pictures of both objects A and B. The pictures help the user to distinguish one object from another while the sonar sensors fail to do so effectively. The user can manually link the CNode object with the physical object in the application/HCI layer and thus resolve the ambiguity. The vision sensors are kept on as long as the ambiguity exists. When the objects move away from each other, this ambiguity automatically disappears and the vision sensors can be deactivated to save power.

Motion Information

It corresponds to the highest level in the LOD tracking process - situation assessment. Under normal circumstances, this requires sending a continuous video stream of the sensing field to the control center. Although video data are relatively easy to get, they take more power and bandwidth to transport. An alternative is to perform motion analysis on objects inside the network and send only the results to the user. Details of such a process are presented in 4.2.3. However, a motion analysis algorithm normally cannot generate accurate results when there is occlusion from view of the vision sensor. This situation, shown in the following Figure 12, is classified as third degree ambiguity. It is different from class 1 or 2 ambiguities because at level 3 there is no higher level to resort to. And since the object is occluded in the camera's view, the user receives no help from looking at the picture.

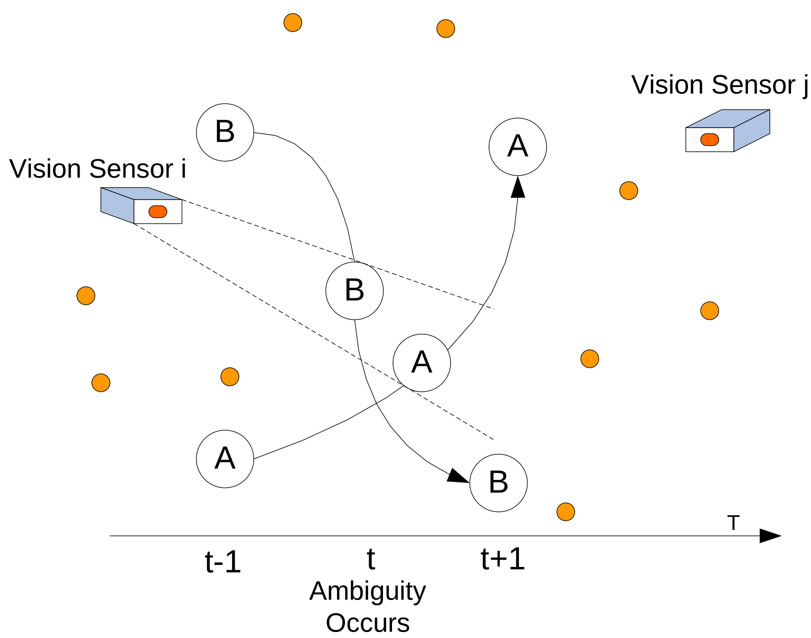

Figure 12 Third Degree Ambiguity: Visual Occlusion

This type of ambiguity can be resolved or alleviated by activating extra vision sensors that provide pictures of the objects from other angles. While this will incur higher power consumption, it is necessary when the user's interest in the object's motion is high.

In the class diagram in Figure 8, there is a *void Draw()* method. The draw method has a *switch-case* structure that checks the current tracking level. If the level is set to 1, then the method will draw a solid sphere, positioned at the center of the cell it is in; if the level is set to 2, the method will draw a 3D symbol with accurate position and rotation and a picture of the object if available; if the level is set to 3, the method will draw an appropriate 3D model that was original designated by the user during the entity discovery stage. The results are presented to the user through the HCI layer.

4.1.5 Application/HCI Layer

The HCI layer directly interfaces with the user. It has two main tasks: presenting LOD tracking results to the user and relaying the user's command to change the LOD in the tracking process.

Presenting the tracking results to the user is essentially the process of rendering both static objects and dynamic objects in the virtual sensing environment. For this purpose the HCI layer requires a graphics Application Programming Interface (API) that will efficiently render both geometries and textures of objects in the sensing field. There are currently several 3D APIs available. The major considerations for the LOD application framework are performance, platform support, and ease of support. In the tracking process, the user may also select a specific object (static or dynamic) and query the object's properties. Hence the HCI layer needs to incorporate that capability as well.

For the second task, the HCI layer needs a parser that translates the user's LOD commands into network messages, and relay them to the sensor network. This can be either a command line parser or a graphical user interface. In chapter 3, we discussed the possibility of per object LOD switching. This requires the HCI layer to allow the user to select a specific object and issue LOD commands regarding that object. A common way to do this is to use the graphics API's built-in *picking and selection* capability.

This kind of object level interaction between the user and the software objects is not available in conventional backing approaches based on video streaming. It is essential for advanced operations such as querying object history data, identifying previous path, enumerating object properties, detecting interaction and social network between objects.

4.2 Dataflow

4.2.1 From Sensor Site to Control Center

Data coming from the sensor site are network messages that contain information regarding events in the sensing field. There are three major types of messages with respect to object *i*: entrance (E_i), exit (X_i), and object movement (M_i). An entrance occurs every time IR line-breaking/motion sensors on the boundary of the sensing field are triggered. A unique ID is immediately assigned to the object, which is used in subsequent tracking and exit events. An object is considered to exit the sensing field once it crosses the boundary from inside the sensing field. These two types of messages are simple, containing only Boolean event indicators together with the object's ID.

Object tracking related messages are more complicated. There are four kinds of tracking related messages based on the contents of the message.

<u>Initial Image Data</u>

After an entrance message E_i is received, the control center normally sends out a request for a picture of the newly discovered object. The sensor network's response is a message containing initial an image of the object taken by the vision sensors. As mentioned in Chapter 3, the image can be a segmented foreground image of the object, which reduces the amount of pixels being transmitted.

Once the initial image of an object is received, the user at the control center can determine the type of object, whether it is a human being, a non-human animal, or an automobile. Understanding the object type at this early stage is helpful for determining $L_i(t)$. For instance, if the object is a non-human animal that is of no significance to the mission of the application, the user simply locks $L_i(t)$ to the lowest level without any further attention during its presence in the sensing field.

Location Information

During each iteration of the object tracking loop, the sensor network reports the object's location (level 1 or 2) to the control center. When the tracking is set at level 1, the message contains only the cell $\{C_x, C_y\}$ coordinates. The sensing field is separated into many cells by line-breaking sensors. Given the cell coordinates, an approximate location of the object can be estimated. When the tracking is set at level 2, the message contains more accurate position information $\{x_i(t), y_i(t), z_i(t)\}$ provided by the tracking layer.

Video Stream

Under extreme circumstances, the user might require continuous video streaming from the sensor network to monitor the object's activities. The sensor network will need to response with real-time video streams. Since this is very expensive, it is highly desirable to use an alternative method which still allows the user to understand the events happening in the sensing field. Section 4.2.3 outlines some potential in-network processing and optimization techniques that can greatly reduce the actual amount of data being transferred.

4.2.2 From Control Center to Sensor Site

The user sends three types of messages to the sensor network: a request for an initial image upon discovery of new entity, a request for tracking results, or a command o the sensor network to switch the level of detail. These messages are encapsulated in predefined network message types and sent to the sensor network without further user intervention.

4.3 Communication and Computation Optimization

The basic idea is to reduce the amount of data being transferred by increasing on-site processing. As mentioned in Chapter 2, data communication is much more expensive than data processing. Hence, local data processing is an effective and critical way to minimize power consumption in a multi-hop sensor network. In 4.2, we analyzed the typical data flow for each level. It is clear that at level 1 and 2, the data being transferred from the sensor site to the control center is already very dense after the fusion stage. On the other hand, level 3 requires a large amount of image data transfer for each frame, and has great potential for compression and optimization.

The first natural thought is to compress the image/video stream to be transmitted from the sensor site to the control center. However, streaming compressed video is still too expensive in terms of both power consumption and network bandwidth. According to Akyildiz, the energy cost of transmitting 1 KB a distance of 100 meters is approximately the same as that for executing 3 million instructions by a 100 million instructions per second (MIPS)/W processor [2]. And the distance from sensor site to control center is normally much larger than 100 meters, making the alternative of in-network processing even more favorable.

The ideal solution is to eliminate actual video streaming altogether, and only transfer information pertaining to object movement. Another advantage of in-network object movement processing is to facilitate object level interaction and query during the tracking process in real-time. In pure video streaming, the user is not able to select an object and query its properties. Although it is possible for the user to perform post-tracking image analysis, the user may have missed some important information during the tracking process.

Such a scheme requires three steps. First, motion analysis must be performed on the objects in the sensing field; second, motion information must be extracted and sent to the control center, and third, the scene must be reconstructed using the received motion information. Consider tracking a human being. The motion information that needs to be extracted includes position and rotation of the body, as well as position of the limbs. Since this three-step operation need to be performed on each object, hence it is better suited for low occupancy wide-area tracking applications.

What is not included in the motion analysis and reconstruction process is small scale body movements such as facial expressions and hand gestures. This is because capturing small scale body movements is computationally expensive, hence not power-efficient for per frame processing. If small scale body movement is indeed required, it is relatively easy to acquire image data directly.

4.3.1 Motion Analysis and Pose Tracking

Computer vision-based human motion analysis can be broken down into four processes: initialization, tracking, pose estimation, and recognition. Before a system is ready to process data it needs to be initialized; e.g., an appropriate model of the object must be established. Since it is important to select an appropriate model of the object from the beginning, the user needs to be involved in the loop. The user requests a picture of the object when it enters the sensing field, and chooses the right model (human being, non-human animal, automobile, etc) from a library of predefined models. Next the object's motion is tracked. This implies a way of segmenting the object from the background and finding correspondences between segments in consecutive frames. A foreground extraction algorithm [Appendix A] can serve this purpose in addition to getting an initial image of the object. The pose of the object's body often needs to be estimated since this may be the desired output of the system (e.g., to control skeletal animated model in a virtual environment in the control center), or it may be processed further by the recognition process. Some higher level knowledge, e.g., a human model, is typically used in pose estimation. The final process analyzes the pose or other parameters in order to recognize the actions performed by the object [36].

For the above mentioned four steps to work properly a number of assumptions are made. These assumptions can be categorized into object movements, environmental setting, or object. Fortunately, the assumptions are either automatically satisfied because of the nature of the LOD based applications, or can be satisfied with certain level of processing or human intervention, as illustrated in the third column in Table 4.

Table 4 Assumptions by Motion Capture Systems

Category	Assumption	Notes
Object movements	1. The object remains inside the work space.	Object outside the sensing field is automatically neglected.
	2. None or constant camera motion	Vision sensors are stationary after deployment
	3. Low occlusion rate	Wide area, low occupancy
Environmental setting	5. Constant lighting	Yes.
	6. Static background	Yes.
	7. Known camera parameters	Yes.
Object	8. Known start pose	Determined from initial object image.
	9. Known object	Determined from initial object image.

Tracking and pose estimation is a well-established research field which may be addressed from various viewpoints [36]. Generally, there are several steps First, nearly every tracking algorithm that deals with human motion capture starts with the figure–ground problem, i.e., separating the human figure from the rest of the image. Second, the images are transformed into another representation (box, silhouette, bones) to reduce the amount of information or to suit a particular algorithm. The object is tracked from frame to frame. Menier et al. use a skeletal articulated model to approximate and fix the segmented images or the human figure [44]. These steps can be accelerated using configurable hardware [49, 50].

4.3.2 Quaternion Based Data Transfer

After the object's pose is estimated, the sensor network needs to send the results back to the control center. Since human object are of high interest to the tracking application, and a human object's body pose is very complex, we will consider the case for human objects only.

There are mainly two ways to represent the results of pose estimation. The first one is indexing into an established motion database, such as the CMU MoBo database [41], the UMD database for human identification at a distance [42], or the full-body gesture database developed at Korea University [43]. However, this approach requires the database to be deployed in the sensor site, which normally does not have the computational facilities to support such a database.

Another approach is to store the pose information in a highly compact data structure, and pass it back to the control center. The human body consists of bones and muscles. The position of bones, or more specifically, of joints uniquely determine the pose of the human body. Similarly, in a 3D model, beside the mesh 3d-data representing the character of the human object, a skeletal animation system consists of a series of hierarchical transformations which represent bones. Like real bones in the human body they influence the shape of the skin. Only the bone data needs to be stored for every frame of the animation.

Usually, the bone data is represented by quaternions or a transformation matrix. This eliminates the need to store vertex positions for all the vertices for every frame of the animation, as in vertex based animation. We can derive intermediate frames by using spherical interpolation. Quaternion algebra is one possible way to represent three dimensional orientations, or other rotational quantities, associated with solid 3D objects. Quaternions have 4 dimensions (each quaternion consists of 4 scalar numbers), one real dimension and 3 imaginary dimensions. Each of these imaginary dimensions has a unit value of the square root of -1, but they are different square roots of -1 all mutually perpendicular to each other, known as i,j and k. So a quaternion can be represented as follows:

$$a + i * b + j * c + k * d \qquad \text{Equation 4-5}$$

The main practical application of this algebra is to represent 3D rotations. It is quite difficult to give a physical meaning to a quaternion, and many people find this similarity to axis-angle as the most intuitive way to think about it, others may just prefer to think of quaternions as an interesting mathematical system which has the same properties as 3D rotations. The quaternion in terms of axis-angle is:

$$q = \cos(\frac{a}{2}) + i * [x * \sin(\frac{a}{2})] + j * [y * \sin(\frac{a}{2})] + k * [z * \sin(\frac{a}{2})] \qquad \text{Equation 5-4}$$

where:

- a = angle of rotation.
- x, y, z = vector representing axis of rotation.

So it is closely related to the axis angle representation of rotations.

4.3.3 Skeletal Animation and Scene Recreation

Skeletal animation is a technique in computer animation, particularly in the animation of vertebrates, in which a character is represented in two parts: a surface representation used to draw the character (called the skin) and a hierarchical set of bones used for animation only (called the skeleton).

This technique is involves by constructing a series of bones. Each bone has a three dimensional transformation (which includes its position, scale and orientation), and an optional parent bone. The bones therefore form a hierarchy. The full transform of a child node is the product of its parent transform and its own transform. So moving a thigh-bone will move the lower leg too. As the character is animated, the bones change their transformation over time, under the influence of some animation controller. Using the quarternions sent back by the sensor network, we can reconstruct a skeleton of a moving human object in the following sequence.

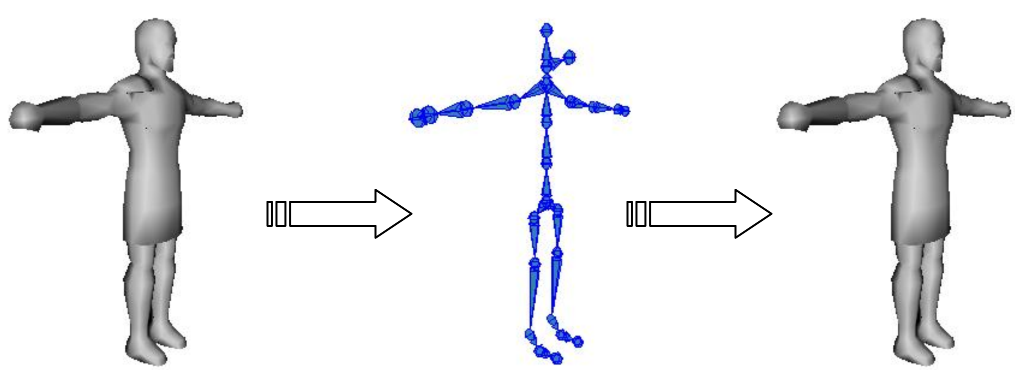

Figure 13 Motion Reconstruction

Each bone in the skeleton is associated with some portion of the character's visual representation, including muscles and skin. By transforming the muscles and skin along with the bones that they are attached to, we can fully reconstruct the entire human body as seen in the above figure.

CHAPTER 5. VIRTUAL LOD SENSING SYSTEM TEST-BED

To verify the effectiveness of the proposed architecture, a test-bed has been built to run various scenarios of tracking applications. The experimental setup includes two major components, the SenRoam software emulator, which runs on a remote host and emulates the function and behavior of a physical wireless sensor network, and the SenTrack that runs in the control center and interacts with the user directly. These two hosts communicate with each other through a wireless network.

The virtual sensing system test-bed allows the user to emulate actual object tracking process in the physical world. The reasons setting up a software emulator system instead of using hardware sensors are multiple. First, it is much easier to monitor the states of sensor nodes and calculate important metrics such as on/off time, power consumption, bandwidth usage, etc. Second, it allows experiments with different deployments. The time and effort needed to deploy a virtual sensor network is negligible compared with a physical sensor network. And since the sensors deployed are software agents, they are infinitely reusable. Third, the virtual setup is extremely low cost. The current price for a typical mote sensor is around $100 and there is significant additional cost to deploy and maintain a sensor network with a large number of sensor nodes. In the software emulation environment, we can emulate a very large geographical area with a massive network of virtual sensors without incurring any substantial hardware costs.

The test-bed itself is a distributed software system that is intrinsically event driven. Development of the software test-bed has gone through a typical Waterfall life cycle [45]. The remainder of this chapter will present the development of the test-bed system and its subsystems in accordance with the waterfall development process. Integration and testing of the test-bed system is provided in section 5.7.

5.1 Test-bed System: Requirements Specification

From the highest level, the test-bed shall provide three major functions: tracking in a virtual sensing environment, delivering the tracking results to the user, and interfacing with the user. These three functions naturally map to the three major components of the sensing system architecture. Detailed requirements for each component are listed below.

5.1.1 Tracking Objects in the Virtual Sensing Environment

The sensing environment shall emulate an environment with events that resemble the real-world operation of the sensor network. The following elements are considered crucial in setting up such an environment:

- Natural environment

This dissertation research is aimed to provide a solution to a class of real-world problems. In these real-world problems, the WSNs have a similar working environment, a wide area wilderness space with low occupancy. Hence the test-bed should be able to create a fairly large 3D space corresponding to a physical world of at least one square mile. The 3D space shall have:

  - Terrain: a three dimensional terrain where all the activities happen
  - Buildings: enhance virtual reality and geometry complexity
  - Plants: enhance virtual reality and geometry complexity
  - Physics: Newton’s Three Laws, gravity.

- Sensor network and deployment

A virtual network of wireless sensors shall be created and maintained. Although it is not necessary to model the communication protocol of the network, it is important to model the operational states of each sensor for energy consumption analysis.

- Object movement

An object will be able to freely enter the virtual sensing field, move within the field, and exit the field. The movement can be controlled by the user, controlled by a script, or randomized by the computer. The object will have six degree of freedom in its movement. It will be able to move on the surface of the terrain and on the surfaces of buildings, stairs, etc, consistent with the Newton's Three Laws and gravity.

- Interaction between objects

Different objects will be able to co-exist in the sensing field. They will be able to move independently of each other. If two or more objects get too close to each other, inter-object collision will be detected..

- Interaction between an object and the surrounding environment

An object will be able to interact with its surrounding environment. There will be constant collision detection between an object and its surrounding environment.

- Sensor management and configuration capability

The test-bed will provide capability for the LOD layer to relay commands to the sensor network, and change the operational states of sensors from one level to another. The sensor management and configuration module will have precise control of each individual sensor node in real-time.

5.1.2 Sending Tracking Results to the User

- Wireless communication

The sensor site and control center will be separated (no shared memory). The only way for them to communicate is to send network messages through a wireless communication link.

- Limited available bandwidth

If broad-band connection is used between the sensor site and the control center, a bandwidth limitation needs to be imposed on the connection to simulate the real-world long range communication link.

5.1.3 Viewing Results in the Integrated Control Center

- Natural environment

This component of the test-bed will reproduce the environmental setup used in the Virtual Sensing Environment. The control center will use the same map that defines the terrain size, shape, static objects, and deployment of sensors.

- Viewing object movement

The user will be able to view the movement of User Controlled Character (UCC) in the virtual sensing environment in real time.

- Viewing interaction between objects

The user will be able to view any interaction between objects in the virtual sensing environment in real time.

- Viewing interaction between an object and the surrounding environment

The user will be able to view any interaction between an object and its surrounding environment in the virtual sensing environment in real time.

- Interaction with objects

The user will be able to select an object and query object information, including object ID, current tracking level, position, orientation, and velocity, and to see a picture of the object (if any).

- Issuing LOD commands

This component will provide the user with controls to change the level of detail throughout a tracking process.

5.1.4 Scalability, Customizability, and Interoperability

The test-bed will be designed with the flexibility to allow experiments with different scales, regarding the size of the simulated 3D world, the shape of the terrain, the size of the sensing field, the number of sensor nodes, the number of UCCs allowed in the field, as well as the number of static objects (buildings, plantations, etc) defined in the sensing field.

The test-bed will not be designed as a single-purpose platform for LOD proof of concept only. It will be flexible enough to be modified for other types of research in wireless sensor networks. It will allow other researchers to customize certain modules in the program, such as the power consumption of the sensor nodes, the inter-sensor communication protocols, and the sensor management schemes. The customizability will allow other researchers to experiment with different tracking algorithms, power management schemes, deployment strategies, etc.

The test-bed will allow interoperability with other tools and software programs. Towards that goal, the test-bed will employ an easy-to-understand and easy-to-use data format for its input and output files, so that results can be easily exchanged between the test-bed and other research tools.

5.2 Architectural Design of the Test-bed

The requirements for building a test-bed have been taking into consideration into the overall design and the design of each subsystem. The test-bed consists of two parts connected through a wireless network. The computer running SenRoam is a server that provides tracking results, while the computer running SenTrack is the client in the control center. SenTrack requests tracking results from the SenRoam server as shown in the following Figure 14.

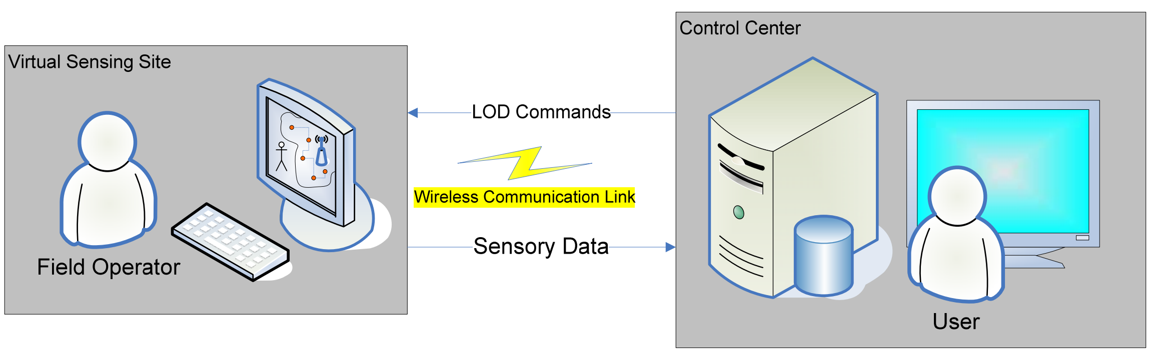

Figure 14 Virtual Sensing Site vs. Control center

From the above diagram we see that the operator of the virtual sensing site runs the SenRoam program, and navigates the UCC through the virtual sensing environment. As the UCC walks through the sensing field, sensors are continuously triggered and events are reported to the control center via a wireless communication link. The computer running SenTrack in the control center receives these network messages, display the tracking results on the screen frame by frame. At the same time, the underlying LOD layer computes $L_i(t)$ for the user, and allows the user to switch to a different level of details. If the user chooses to change the LOD to $L_i(t+1)$, a command is sent to the SenRoam program wirelessly. Any future movement or action of the object i will cause the sensors to react only at level $L_i(t+1)$.

The user in the control center can opt to fully automate the LOD adjusting process, leaving the decision up to $S_i(t)$ completely. For instance, the user can define a rule so that if an object gets within 150 units of distance of any ROIs, the sensor network switches to the highest level of operation. With this rule, the LOD layer will automatically send an LOD command to the SenRoam host when an object gets within the predefined range of any ROIs.

Both the virtual sensing sites and the control center use the same modules to read map files, which contain information from the sensor network's deployment to the setup of the virtual 3D world. Using external map files allows other researchers to create and carry out their own experiments with different deployment schemes, etc.

Sections 5.3 through 5.6 below describe the construction of the test-bed system and its subsystems.

5.3 SenRoam: the Virtual Sensor Network Emulator

5.3.1 Architecture

The SenRoam emulator is a virtual 3D world that allows the UCC to roam inside the sensing field. This virtual world evolves by itself with an internal clock. In the meantime, user commands and network messages are processed at each step. The program consists of several major components as shown in Figure 15.

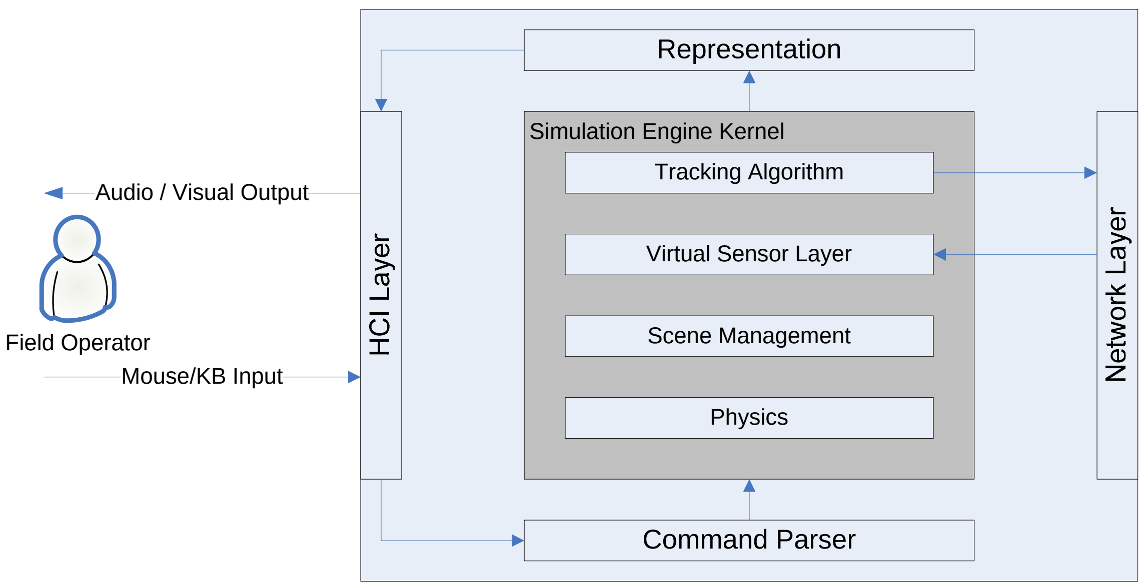

Figure 15 SenRoam Architecture

<u>HCI Layer</u>

The HCI layer is designed to directly interact with the user. During each loop, it scans the keyboard and mouse to detect events. Keyboard or mouse events are sent to a command parser to be translated into commands to control the UCC. The HCI layer is also responsible for rendering the 3D world to the user. The 3D world is rendered using the OpenGL [46, 47] graphics library. Among the available graphics APIs, OpenGL provides the best combination of platform support, speed, functionality, and ease of use. Table 5 compares these graphics APIs: OpenGL, Direct3D and Java3D.

Table 5 Graphics API Comparison

	Direct3D	OpenGL	Java 3D
PC Driver Support	High	Medium	Low
Speed	Medium	High	Low
Ease of Coding	Medium	High	Medium
Multi-Platform support	Low	High	High

Command Parser

The command parser is an event handler that mainly consists of a set of switch-case statements. It takes the user's keyboard/mouse inputs and translates them into commands to control the movement of the UCC. Keyboard events correspond to movements of the UCC; while mouse event correspond to rotation and orientation of the camera and the UCC.

Simulation Engine Kernel

The simulation engine kernel is the core of the SenRoam program. Inside the kernel are four major sub-components: physics engine, scene management, virtual sensor layer, and tracking algorithm. The tracking algorithm is responsible for finding the position of the UCC and sending the results to the network layer.

The virtual sensor layer maintains a grid of virtual sensors, which are modeled in 5.2.2. The virtual sensors are activated/deactivated according to the position of the target. Deployment of the sensors is generated by a tool developed to create experiments. The tool, normally referred to as the level editor, is integrated in the SenTrack program. Once the target crosses the boundary of the sensing field, the tracking loop starts. The target's movement in the sensing field is constantly monitored by virtual line-breaking sensors, and by more advanced sensors such as virtual sonar sensors and virtual vision sensors depending on which level of detail the sensor network is at.

The scene management layer organizes the virtual 3D world, including terrain, buildings, plants, virtual sensors, and the UCC. Static objects such as buildings and plants are stored in an octree data structure. An octree is a tree data structure in which each internal node has up to eight children. It partitions the three-dimensional space by recursively subdividing it into eight octants. It is intensively used in the physics engine for collision detection, and used by the representation layer for hierarchical view frustum culling.

The physics engine computes the UCC's next position based on its movement velocity, while consistent with Newton's Laws and gravity. Before the UCC can be moved to a projected location, the physics engine performs collision detection of the UCC against the 3D world. The above-mentioned octree data structure makes this process efficient because collision detection is only performed on the nodes that are in the same sub-space as the UCC. When a collision is detected, there are a few possible scenarios: the UCC can climb up the obstacle if it is not too high, the UCC can be bounced back to its last position if its moving direction is perpendicular to the surface of the obstacle, or the UCC can slide along the surface of the obstacle to another position.

Representation

This layer prepares the data to be rendered by the HCI layer. Using the octree data structure, one can recursively traverse the entire tree and mark the nodes that are inside the user's view frustum and hence need to be rendered. Any node that is outside of the view frustum is not further explored, because none of its children can be inside the view.

Network Layer

The network layer is responsible for sending and receiving messages. As discussed previously, messages received by the SenRoam program are requests for sensory data from the control center, as well as LOD commands; while messages sent by the SenRoam program are sensory data pertaining to events. The network layer is implemented using the RakNet library [48].

With these functional modules, SenRoam creates a virtual 3D world that allows the user to control the virtual character, navigate in the sensing field, and interact with objects in the world (terrain, buildings, plants, etc). The portion of the 3D world that is inside the view frustum of the first-person camera is rendered onto the screen as shown in Figure 16.

Figure 16 Virtual 3D World of Sensing Environment

The user can move freely in the scene with six degrees of freedom. To increase the reality of the emulation, the simulation engine allows the user to climb stairs or jump onto surfaces, under the influence of earth gravity. Such flexibility allows the user to realistically simulate various types of human actions and movements in the physical world.

5.3.2 Virtual Sensor Nodes

A physical sensor is modeled as an entity with the following attributes: location, ID, sensing range, sensor type, and total consumed energy as shown in the following class definition. The class also provides a set of public member functions for an external class to access the protected member variables.

```
class CSensorNode
{
public:
    CSensorNode(void);
    ~CSensorNode(void);

protected:
    int m_nGridX;
    int m_nGridY;
    int m_nSensorID;
    int m_nSensorRange;
    int m_nSensorType;
    int m_nWorkingLevel;
    int m_nTotalEnergy;

public:
    //member funcitons
    …
};
```

The attributes that are directly related to experimental results are *m_nWorkingLevel* and *m_nTotalEnergy*. The *m_nWorkingLevel* indicates which level the sensor currently works at, and is associates with a specific instantaneous power level of the sensor. The *m_nTotalEnergy* records total energy that has been consumed by the sensor at time t after k iterations. It is defined as:

$$E_{iTotalEnergy} = \sum_{j=0}^{k} P(m\_nWorkingLevel_j) \times T \qquad \text{Equation 5-1}$$

$P(m\_nWorkingLevel_j)$ is a mapping from the working level to a predefined power level. In first-order approximation, $P(m\_nWorkingLevel_j)$ only consists of dynamic working current while static current is ignored.

SenRoam maintains a two-dimensional grid of virtual sensor nodes. Each sensor is responsible for covering the space inside its sensing range defined as *m_nSensorRang*.

During each iteration of the simulation loop, the underlying simulation engine broadcasts the target's position to the sensors nearby. The sensors receiving the broadcast are a subset of the entire sensing grid such that $S_{Broadcast} = \{X \mid Dist(X, Pos_{Target}) \leq X.m\_iSensorRange\}$. These sensors are considered active while others are inactive. Inactive sensors' power consumption is ignored in first-order approximation.

5.3.3 Workflow

The process of the experiment is essentially the user controlling the 3D character to interact with the virtual natural environment and the virtual sensor network. The interaction process lasts from the moment the character enters the sensing field until it leaves the field. During each iteration, SenRoam performs a series of operations as illustrated in the following diagram.

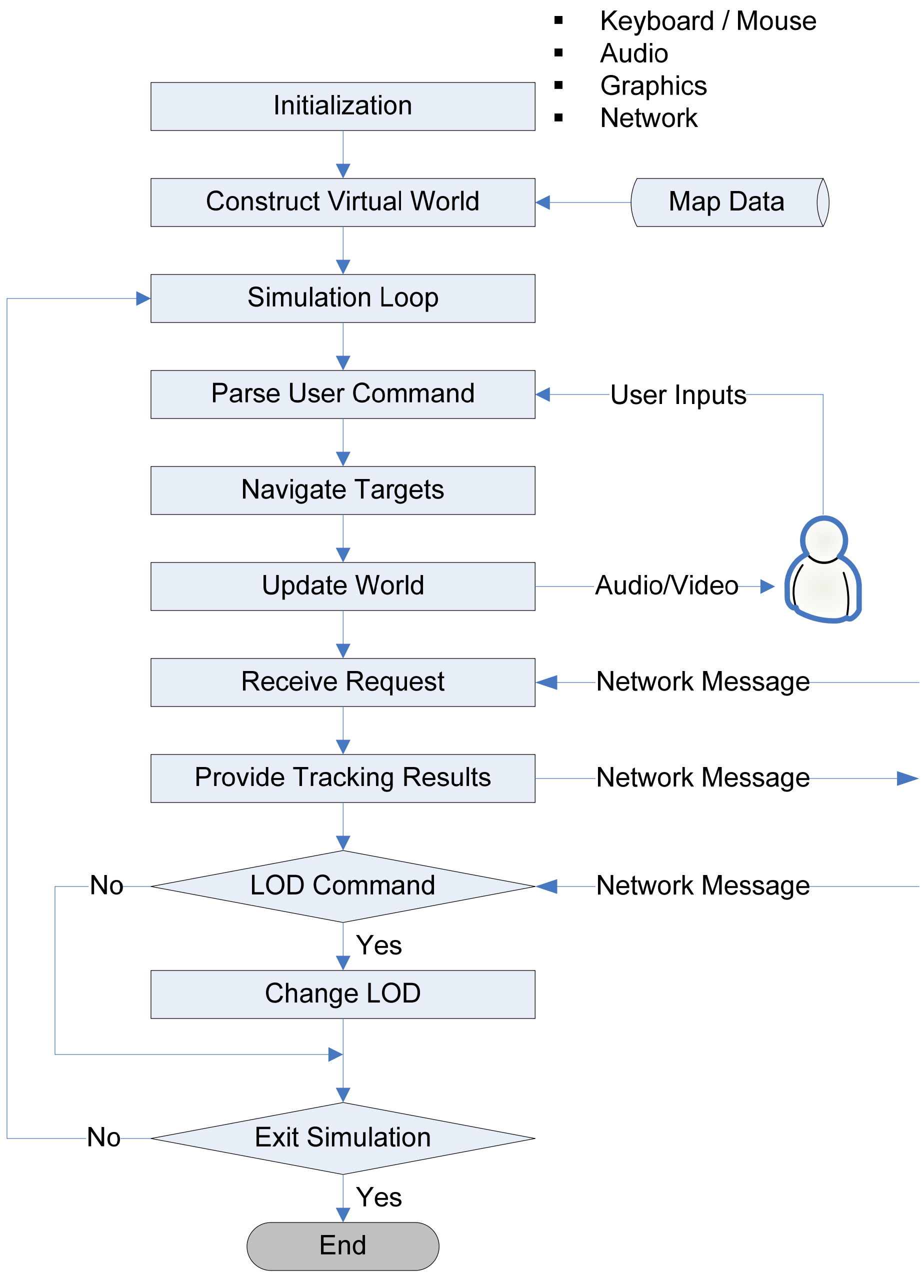

Figure 17 SenRoam Work Flow

When the program starts, it initializes the computer, including the keyboard/mouse event handling routine, the audio, the video and the network layer. Then the program loads the map data for the experiment. The map includes information regarding the virtual world, such as terrain, plants, and buildings present in the sensing field. It also includes layout of the sensor network. After this, the program enters the main simulation loop.

Inside the loop, SenRoam not only interacts with the operator through keyboard/mouse controls, but also with the remote computer running SenTrack through the wireless network.

5.4 SenTrack: the Integrated Control Center

From the vantage point of object tracking, SenTrack is the counterpart of SenRoam. The user moves the UCC in the virtual sensing field in SenRoam, while tracking results are presented in SenTrack. SenTrack is an integrated environment that not only allows the user to view tracking results, but also allows the user to set up experiments, deploy a virtual sensor network, issue LOD commands, and interact with any object in the virtual 3D world through a Microsoft Foundation Class Library (MFC) based multi-viewport interface. .

5.4.1 Architecture

SenTrack is a 3D third-person environment that provides the user with a bird's eye view of real-time events happening in the virtual sensing field simulated by SenRoam. SenTrack also provides the user with controls to change the LOD on the fly. SenTrack has a simpler architecture than SenRoam as shown in Figure 18. The major difference is that SenTrack does not need a simulation engine kernel to glue all the components together.

SenTrack does not have an internal clock to drive the evolution of the world either. In fact, SenTrack uses a pure event-driven architecture that responds to two types of events: incoming network messages that contain tracking results and user control commands that are intended to change the level of detail in subsequent tracking processes.

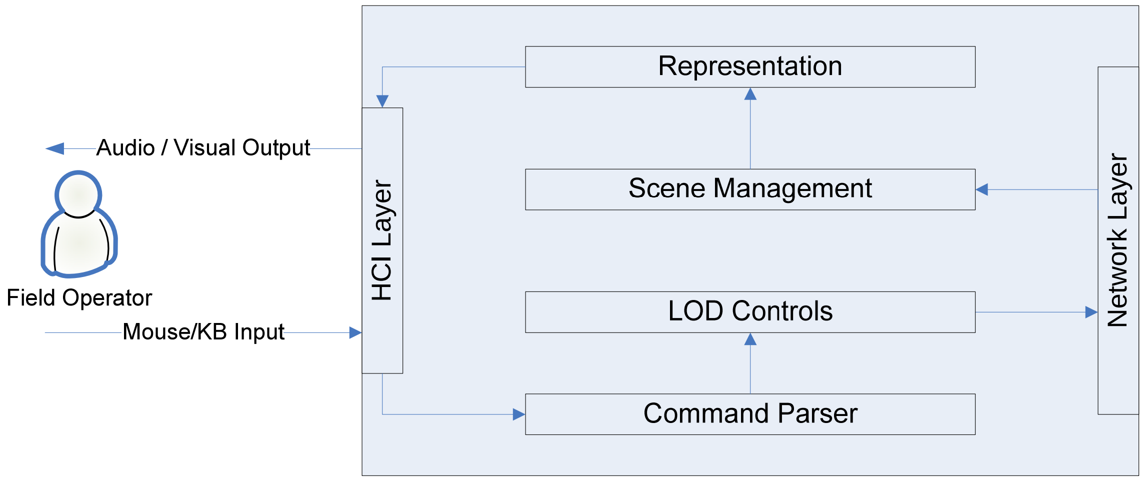

Figure 18 SenTrack Architecture

5.4.2 Main Work Area

SenTrack provides the user with four view ports (top, front, left, and perspective) to interact with the virtual sensing environment and objects within it. The view ports can be resized to provide better viewing results from a specific view.

The left side of SenTrack is a multi-purpose control panel, which consists of three parts: an object list view, an object property table, and network control dialog. The user can select an object from the list view, or by clicking on the object in any of the four view ports. The selected object will be highlighted with a white bounding box as shown in Figure 19. In the meantime, the selected object's attributes, including object ID, object type, position, orientation, etc are automatically displayed in the property window.

The terrain in Figure 19 is covered by a predefined grid of sensors. The red grid indicates the cells divided by line-breaking/motion sensors in the field. A cell is highlighted in red, indicating that an object is present but that accurate position information is not yet available for it (level 1 tracking).

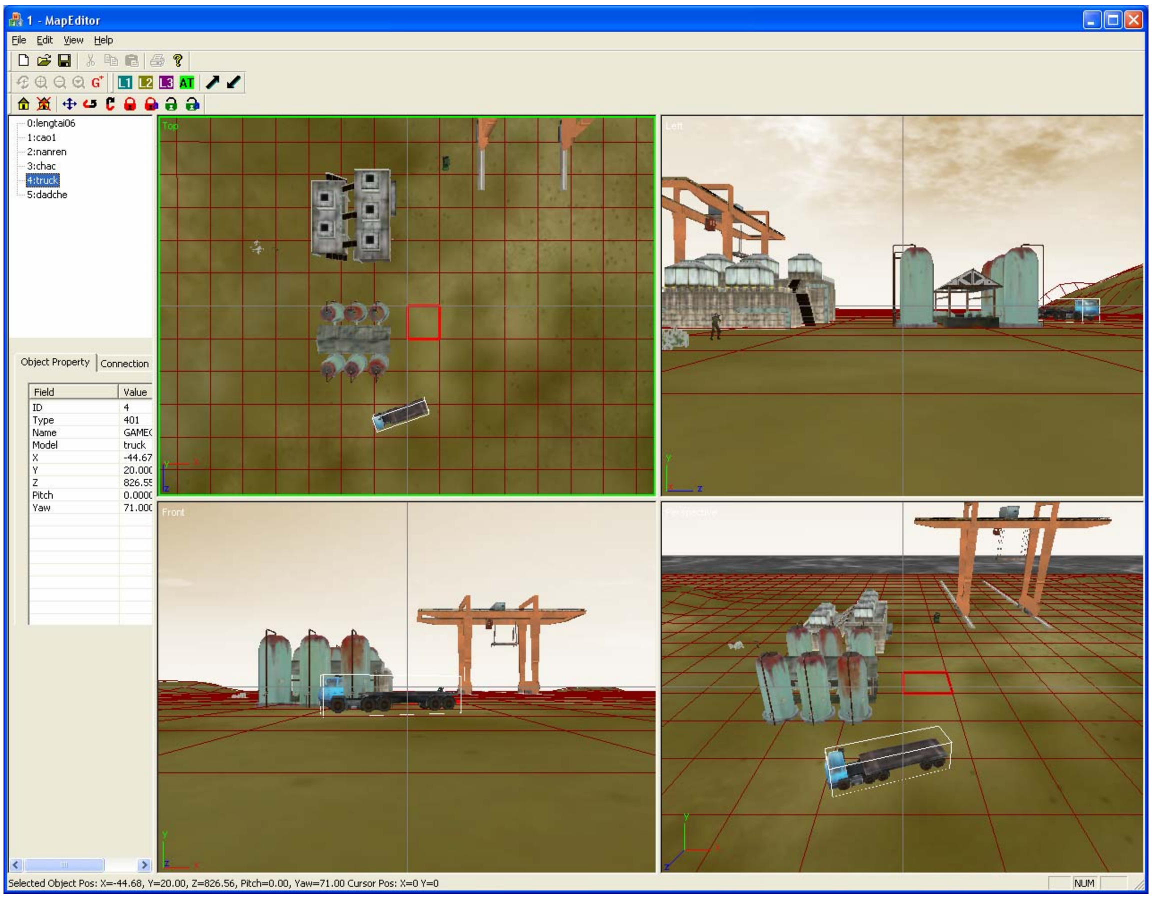

Figure 19 SenTrack Integrated Control Interface

The network control dialog, which is not visible in the above screenshot, is used to establish a connection between SenTrack and SenRoam. The connection is established when the user inputs the IP address of the server machine that runs the SenRoam program and initiates a connection request.

5.4.3 Workflow

The SenTrack program mainly consists of an event-driven loop in which the user interacts with the object in the virtual sensing field. The interaction process lasts from the moment the character enters the sensing field until it leaves the field.

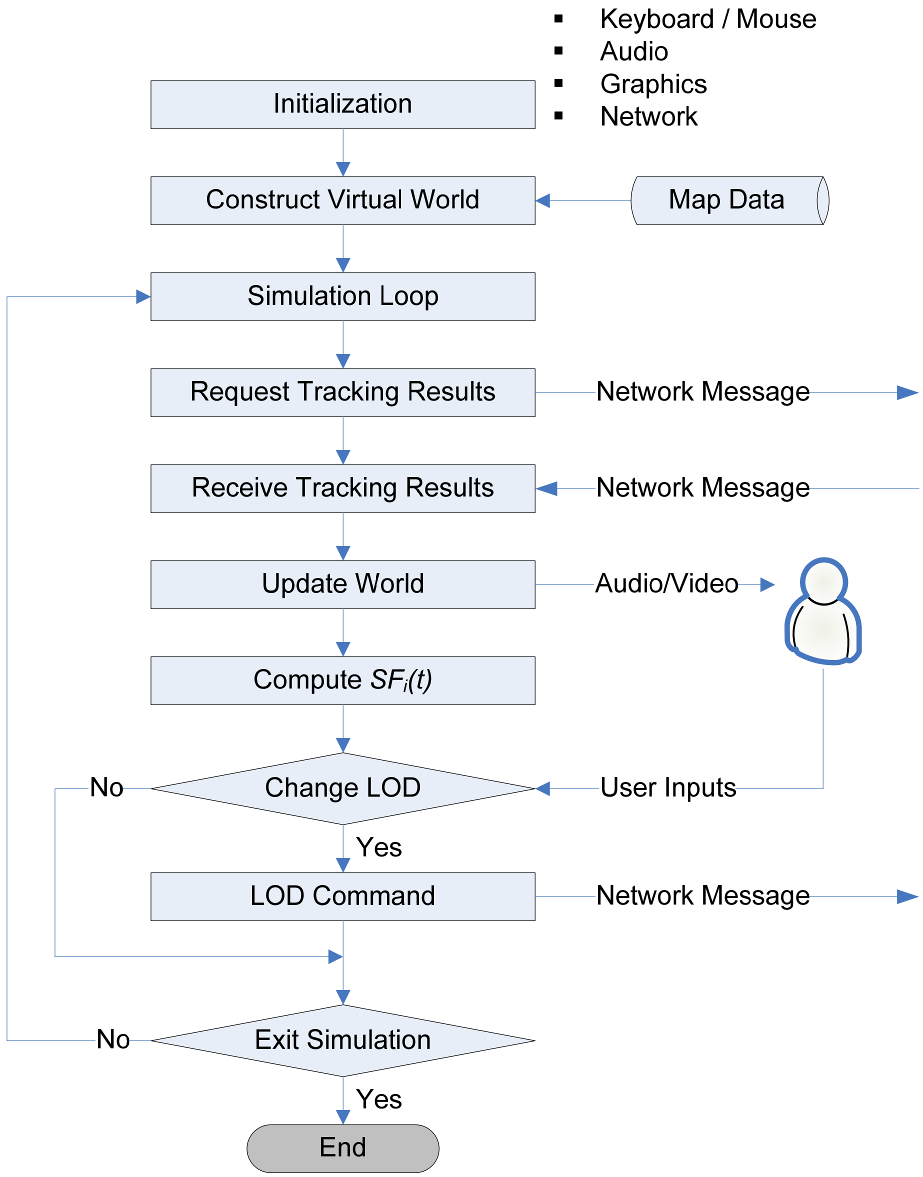

Figure 20 SenTrack Work Flow

During each iteration, SenTrack requests updates from SenRoam and presents the results to the user. The user can choose a different LOD, query objects in the virtual world, or observe the tracking scenario from various angles and view ports. The human operator is tightly integrated into the loop. The user's commands to change the LOD are translated into predefined network messages and sent to SenRoam. The later will incorporate the new LOD into its tracking process, and in the meantime change the power consumption level of the entire sensor network.

5.5 Communication

5.5.1 Message Definition

There are two types of packets being transmitted on the network: the sensor data packets and user command packets. Sensor data packets are variable length structures depending on the content. A sensor data packet always contains a header, which defines what kind of data is contained in the packet, and the actual chunk of data. Command packets are simpler, fixed length structures. A command packet contains predefined commands that users will use to perform an operation, such as requesting the initial image of a newly discovered entity, changing the LOD of sensors across the network, etc. Along with the command, there are some auxiliary variables in the packet, such as target ID, which identifies the target users desire to operate upon. The following code segment shows the definition of the various types of network messages that are used between the client and the server.

```
#include "networktypes.h"
#include "packetenumerations.h"

struct ClientMsgStruct_StatusRequest {
    unsigned char typeID; //ID_USER_PACKET_CLIENT_STATUS_REQUEST
    char msg[32];
};

struct ServerMsgStruct_Status {
    unsigned char typeID; //ID_USER_PACKET_SERVER_STATUS
    float x;
    float y;
    float z;
    float pitch;
    float yaw;
};

struct ClientMsgStruct_ViewImageRequest {
    unsigned char typeID; //
    ID_USER_PACKET_CLIENT_VIEWIMAGE_REQUEST
};

struct ServerMsgStruct_ViewImageResult {
    unsigned char typeID; //        ID_USER_PACKET_SERVER_VIEWIMAGE
    int width;
    int height;
    unsigned char imgdata[36864];
};

struct ClientMsgStruct_SetLODRequest {
    unsigned char typeID; //
    ID_USER_PACKET_CLIENT_SETLOD_REQUEST
    int lod;
};
```

5.5.2 UDP Message Passing

Communication between the sensor network and the LOD layer is done by sending and receiving the above mentioned data packets between the modeling/visualization layer and the sensor management layer. In our experimental setup, we have created a UDP socket on the server that runs the LOD modeling layer and visualization layer. The socket then listens to a specific port. As soon as a new data packet comes into the port, the LOD modeling layer captures it, processes it, and sends it to the tracking layer. Communication from users to the sensor management module is performed in a similar way by writing command packets to a specific port. On the other side of the network, the listening socket of the sensor management component receives the command and executes it.

5.5.3 The RakNet Network Library

RakNet is a cross-platform, open-source, object-oriented networking library for C++ created by RakkarSoft. It uses UDP for network communications, and supports Client/Server and P2P systems. RakNet provides a comprehensive set of methods to facilitate connections, communication and management.

Server Side

Use the Start member function of the RakServerInterface object to open the port and start listening for incoming connections. The user can specify the port number and the maximum allowed number of clients to be connected to the server at any one time (up to 65535) through this function. If the user needs to ban an IP address, he can use the AddToBanList member function to ban someone. Call the Receive member function to fill a Packet structure with received data. Check the first entry ([0]) of the data member of the Packet structure against the server-specific values in the PacketEnumerations header file to find out what kind of data was received. When replying to the client, use the Send member function.

Client Side

Use the Connect member function of the RakClientInterface object to connect to the server. The user can specify the server address and port number of the remote server through this function. Call the Receive member function to fill a Packet structure with received data. Check the first entry ([0]) of the data member of the Packet structure against the server-specific values in the PacketEnumerations header file to find out what kind of data was received. When the user needs to reply to the server, he can use the Send member function.

5.6 Saving Results

Experimental results need to be saved for several purposes: data analysis, scene replay, and export to third party tools. The main goal of the virtual LOD system test-bed is to prove the effectiveness in reducing power consumption. It is critical for the test-bed to precisely record experimental data, and perform power consumption analysis afterwards. The information needed for power analysis includes the status of each sensor at each time instance.

Saving experimental data into the file at each time instance will also allow the user to replay the entire tracking process at sampled moments. Assuming that the simulation program can run at one hundred frames per second (FPS) on a mainstream desktop PC, the data is recorded at every ten milliseconds. This refresh rate is sufficient for tracking human beings in the sensing field, given that the speed of the fastest human being is around ten meters per second. Hence within the sampling period, the target will only move ten centimeters, roughly equal to four inches. If higher resolution is required, the simulation program can be executed on parallel hardware to achieve higher frame rate and shorter sampling interval.

It is also essential for the test-bed to provide a way for its results to be further analyzed or processed by third party research tools. A data file in text format is an ideal choice for such purpose. A disk file is created every time an experiment is executed with the file name set as current time. The file is used to store position of the UCC and status of the sensors the tracking process. During each iteration of the simulation loop, the engine saves the UCC's 3D coordinates, orientation, and status of each sensor to the file using the following format:

```
@time instance
UCC_x UCC_y UCC_z UCC_pitch UCC_yaw
S[1][1]    S[1][2]    S[1][3]    S[1][4]    …
S[2][1]    S[2][2]    …
…
                      S[i][j]

S[n][1]    S[n][2]    …
```

In the meantime, total energy consumption by each sensor S[i][j] is computed according to Equation 5-1. At the end of the simulation, the time averaged power is saved to another file, as shown in APPENDIX D: SAMPLE RAW DATA.

5.7 Integration and Testing

The software test-bed was implemented in C++ with OpenGL 3D API and RakNet network library. The development work is done in the Microsoft Visual Studio 2005 platform. The test-bed consists of the SenRoam and SenTrack programs, and a communication channel connecting them. The test-bed itself is tested before full-scale experiments are carried out. The tests performed include unit testing, integration, functional and system testing. The test cases have been designed based on the systematic testing approach in [51,52]. Details of the test cases and test execution can be founded in APPENDIX B: TEST-BED TEST CASES. While here in this section, only the testing results are given in the following table.

Table 6 Test-bed Integration and Testing

Test Level	Sub Level	Features to be tested	Reporting
Unit Testing	SenRoam	Rendering	Passed
		Scene management	
		Character control	
		Tracking objects	
		Physics	
		Interpreting LOD commands	
		Exporting experiment data	
	SenTrack	Level editing	Passed
		Rendering	
		Scene management	
		Receiving tracking results	
		Viewing tracking results	

		Issuing LOD commands	
	Communication	Port creation	Passed
		Sending/receiving messages	
		Message parsing	
Integration Testing	SenTrack to SenRoam	Exporting level to SenRoam	Passed
		Connecting to SenRoam	
		Requesting tracking results from SenRoam	
		Sending LOD commands	
	SenRoam to SenTrack	Importing level edited by SenTrack	
		Accepting and establishing connection with SenTrack	
		Sending tracking results to SenTrack	
		Receiving and interpreting LOD commands	
System Testing	Functional	Level editing	Passed
		Level loading	
		Sensor deployment design	
		Navigating UCC in SenRoam	
		Sending tracking results to SenTrack	
		Viewing tracking results in SenTrack	
		Issuing LOD command from SenTrack	
		Interpreting LOD command in SenRoam	

		Switching sensor tracking level in SenRoam	
		Saving experiment data to text file	
		Replicating experiments from saved results	
	User interface	Navigation,	
		Picking and selection	
		HUD	
	Usability	Integrated experiment setup and execution	
	Performance	Frame per second	
		Communication latency	
	Compatibility	OS: Windows	
		API: OpenGL, RakNet	
		Peripherals: keyboard, mouse, network interface card, display	
		Exporting saved results to third party tools (Excel, access)	

CHAPTER 6. EXPERIMENTAL RESULTS AND ANALYSIS

The research reported in this dissertation does not attempt to go into the detail regarding quantitative performance comparison against other architectures. The difficulty here lies in acquiring and presenting such results in a comparable way. There appear to be no performance results in scientific publications on the topic of wireless sensor networks. This is due to the lack of a suite of benchmarking applications, with clearly delimited rules and assumptions [3].

Hence, the analysis conducted is based on comparison of performance differences between two sets of experiments. The first set of experiments involves tracking applications with an LOD function. The same tracking scenario is replicated by recording and replaying the paths of the moving targets, but with all LOD function turned off. By doing this, we can objectively measure the performance differences with and without LOD while all other settings remain unchanged, including the number of sensor nodes, the actual deployment, the routing scheme, the communication bandwidth, etc.

It is important to point out that in such control experiments, selection of an actual deployment, routing scheme, tracking algorithm, or even the sensors' physical specifications do not affect the comparative analysis of the LOD's effectiveness. In Equation 2-3, we have defined total energy consumption as the product of unit data cost and data volume, that is $\tilde{E}_{Total} = (C_{CPB} + C_{iTPB} + C_{eTPB}) \times \tilde{M}$. Parameters such as tracking algorithm, sensor management scheme, and sensor specifications only contribute to C_{CPB}, C_{iTPB}, C_{eTPB}. These parameters are independent of $\tilde{M}$, hence their actual values only have scalar effect on $\tilde{E}_{Total}$ in the control experiments. The actual tracking selected for the experiments is based on the one proposed by Vaidya et. al [26].

Results of the experiments include the status of every sensor (on, off, level 1, 2 or 3), and the position of the UCC being tracked for every time instance *t*. These results are saved into disk files to facilitate off-line analysis. Average power consumption is also saved into the same file. From the status of the sensors and the average power consumption, we are able to estimate power consumption of the entire network, and calculate the amount of data that has been transferred across the network (sink node to control center only). We can also conduct a qualitative analysis of attention economics and response time based on the status of the sensors. The remainder of this chapter will cover these aspects based on some initial experiment results.

6.1 Experiment Design

In order to fully test the effectiveness of the LOD, the experiments are designed in two equivalence classes: single and multiple object tracking. The first class of tracking is to test the global LOD adjusting capability, and the second class will use per object and per cell LOD adjusting through the tracking process. Since SenRoam only allows one user to control one character at a time, the other characters have to be controlled by the computer. In the second class of experiments, the characters are referred to as computer controlled characters (CCC) as compared to UCC. Movement of a CCC can either be randomly generated with a certain set of constraints, including maximum speed, maximum acceleration etc. A CCC's movement can also be controlled by a script.

6.2 Experiment Execution

The control experiments are always executed twice, the first round with the LOD turned on (such that the sensors' working level can be manually changed by the user or automatically changed by the rule stack). The second round replicates exactly the same movement of the UCC, this time with the LOD function turned off. When the LOD is off, tracking is performed purely by the underlying algorithm. Sensors are turned on or off by the algorithm. Exact replication between two rounds are made possible by saving the UCC's status (position, rotation, velocity, etc) into an experiment file; while in the second, a script reads in the saved UCC status and automatically leads the UCC to travel on the same path step by step.

While the experiment runs, two operators are needed. One works with SenRoam and controls the UCC's movement inside the virtual 3D world. The simulation engine takes a snapshot of the UCC and the sensors' status and automatically saves them into an experiment file at each iteration. The other operator works with SenTrack. Besides viewing tracking results, this operator is responsible for changing the LOD from time to time to simulate a real-world tracking process with LOD. Changing the LOD can be fully automated if the user has already defined a rule stack to compute $S_i(t)$ and perform level switching.

The following code segment shows a simplified subroutine that computes $S_i(t)$ based on normalized linear distance between the UCC and ROIs. The user can compute $S_i(t)$ using any other way meaningful to him.

```
#define MAXNUMROIS 128
#define TERRAINDIMENSION 8192
#define THRESHOLD_L1 0.1
#define THRESHOLD_L2 0.2
#define THRESHOLD_L3 0.8

float m_fSigFactor[MAXNUMROIS];
CROI m_cROIs[MAXNUMROIS];
float tmpDist=0;

for(int j=0; j<m_nROIs; j++){
   tmpDist = Distance(m_cROIs[j].GetPosition(),pUCC-
>GetPosition());
   m_fSigFactor[j] = 1- tmpDist/TERRAINDIMENSION;
}

for(int j=0; j<m_nROIs; j++){
   if(m_fSigFactor[j]< THRESHOLD_L1){ //Very insinificant
         // Broadcast LOD command to sensors
   }
   else if(m_fSigFactor[j]> THRESHOLD_L3){ //Significant
         // Broadcast LOD command to sensors
   }
   else{ //Medium significance
         // Broadcast LOD command to sensors
   }
}
```

Levels can also be switched based on temporal significance factors. However in the experiments, no TOIs are defined. As a result, the same events that happen at different times will have the same $S_i(t)$. The following section (6.3) shows the experimental data recorded in experiments with the accompanying average power analysis.

6.3 Average Power Consumption

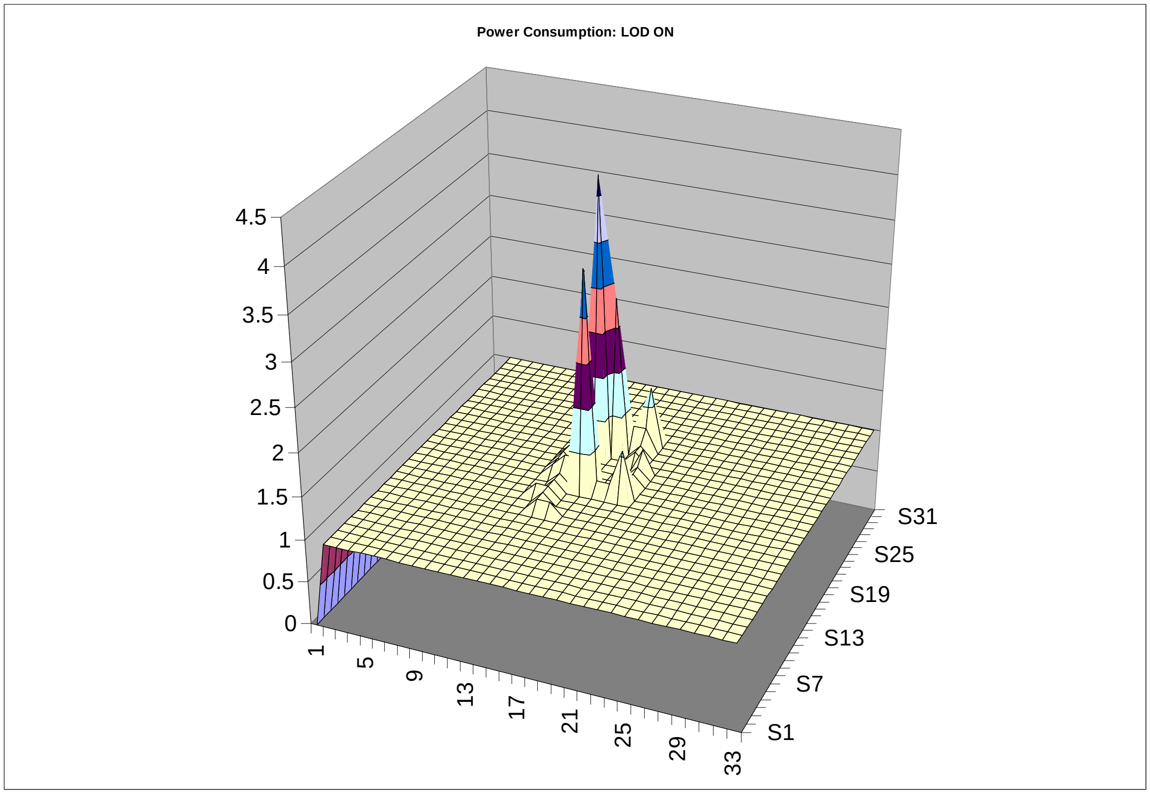

Figure 21 Normalized Level of Detail in Tracking Process – Manual LOD ON

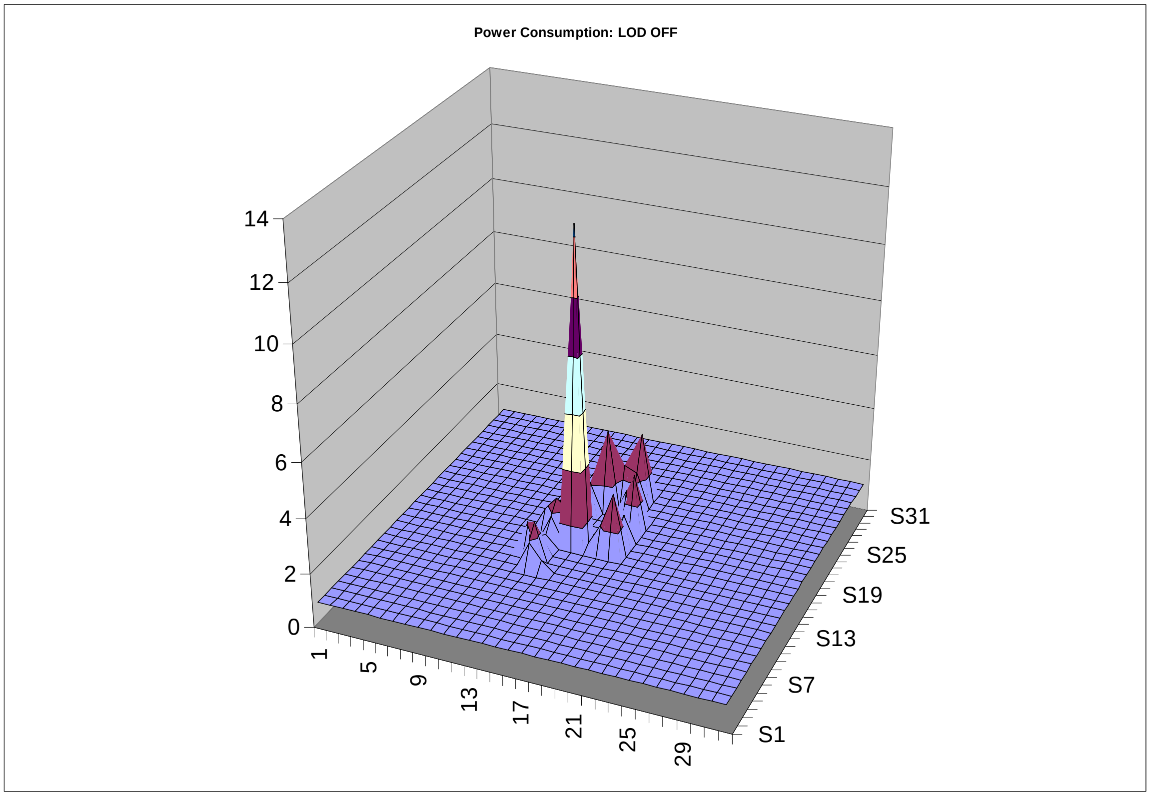

Figure 22 Normalized Level of Detail in Tracking Process – LOD OFF

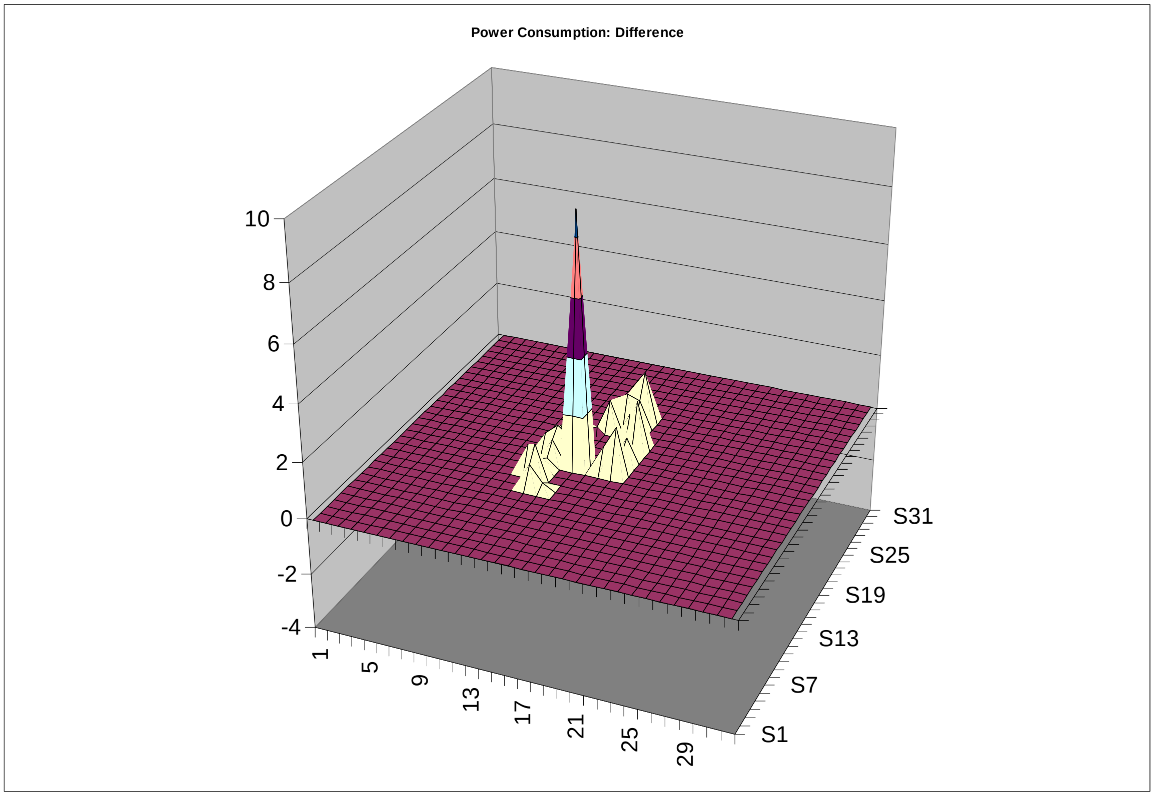

Figure 23 Normalized Level of Detail in Tracking Process – Difference

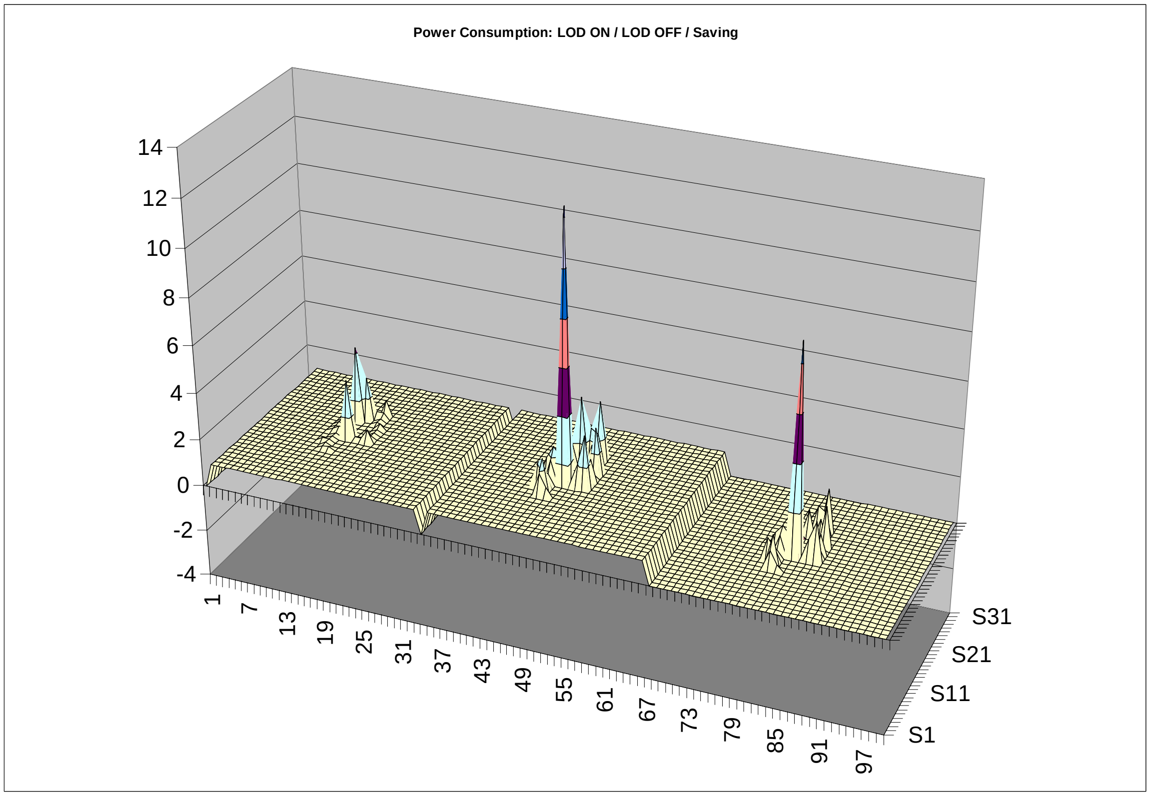

Figure 24 LOD ON vs. LOD OFF vs. Saving

The above figures show the results of control experiments. With LOD enabled, there are very few moments when the user need more data than provided at the lowest level of detail.

6.4 Attention Economics and Response Time

Although the test-bed does not provide the capability to model attention economics and response time, the experimental results can be used for qualitative analysis for these two aspects. From 6.3, we can conclude that an LOD enabled sensing system can substantially reduce unnecessary data collection and communication compared to conventional sensing architectures. Such reduction in data volume allows the user to focus on the objects around the ROIs that interest the user most. In other words, the user now gets fewer irrelevant messages from the sensor network. This directly results in improved attention economics and cut down response time, especially so when there is an outburst of incoming information.

.

CHAPTER 7. CONCLUSION

7.1 Concluding Remarks

This research work provides a framework that helps realize the goal of ubiquitous computing and ubiquitous information access anytime, anywhere, at the desired level of detail.

A virtual sensing system test-bed has created to execute control experiments of the proposed LOD-based tracking process. Experimental results prove that LOD results in a tremendous energy saving as compared to the current sensing paradigm.

The test-bed can be further used as a testing environment for research work in deployment, routing, tracking algorithms, etc.

7.2 Future work

In addition to this first order simulation, we can add second order behavior to the sensor network (such as node failure, leakage current, circuit start-up current, etc) to achieve more accurate results. Additional optimization techniques can also be used to lower the power consumption of the sensor network even further, such as configurable hardware for image processing.

APPENDIX A: FEATURETRACKER PROGRAM

1. Storing background

```
p_BackgroundYUVImage = new BYTE[nWidth * nHeight];
p_BgFgDifference = new BYTE[nWidth * nHeight];
for (int i=0; i<nHeight; ++i) {
    for (int j=0; j<nWidth; ++j) {
            index = 3*(i*nWidth+j);
            Y = floor(RGB2YUV_Y(
            lpData[index+2],lpData[index+1],lpData[index]) + 0.5);
            p_BackgroundYUVImage[i*nWidth+j]=Y;
    }
}
```

2. Extracting background

```
int index; BYTE Y;
for (int i=0; i<nHeight; ++i) {
    for (int j=0; j<nWidth; ++j) {
            index = 3*(i*nWidth+j);
            Y = floor(RGB2YUV_Y(
            lpData[index+2],lpData[index+1],lpData[index]) + 0.5);
            if(p_BackgroundYUVImage){
                    if(abs(Y-p_BackgroundYUVImage[i*nWidth+j])<=
                    bBGCancellationThreshold)
                    {
                            lpData[index+2]=0;
                            lpData[index+1]=0;
                            lpData[index+0]=0;

                            p_BgFgDifference[i*nWidth+j] =1;
                    }
                    else{
                            p_BgFgDifference[i*nWidth+j]=0;
                    }
            }
    }
}
```

APPENDIX B: TEST-BED TEST CASES

Table 7 Terrain Test Case 1:

Purpose	Input data	Expected results
Works with small terrain	1024x1024	Load success

Table 8 Terrain Test Case 2:

Purpose	Input data	Expected results
Works with large terrain	32768*32768	Load success

Table 9 Scene Management Test Case 1:

Purpose	Input data		Expected results
Works with low geometry complexity	static object	1	Load success
	UCC	1	

Table 10 Scene Management Test Case 2:

Purpose	Input data		Expected results
Works with low geometry complexity	static object	100	Load success
	UCC	2	

Table 11 UCC Movement Test Case

Purpose	Input data		Expected results
UCC movement	Forward	W	Correct movement
	Backward	S	
	Left	A	
	Right	D	
	Accelerate	Shift	

Table 12 UCC Rotation Test Case

Purpose	Input data		Expected results
UCC rotation	Pitch	Mouse	Correct rotation
	Yaw	Mouse	

Table 13 UCC-Environment Collision Detection

Purpose	Input data	Expected results
Collision detection between object and environment	N/A	Object movement stalled

Table 14 UCC-UCC Collision Detection

Purpose	Input data	Expected results
Collision detection between objects	N/A	Object movement stalled

Table 15 Physics

Purpose	Input data	Expected results
Newton’s Law and gravity	N/A	Correct movement under influence of Newton’s Law and gravity

Table 16 Sensor Status

Purpose	Input data	Expected results
Polling and saving sensor status	N/A	Save sensor status into file

Table 17 Communication

<table>
<tr><th>Purpose</th><th colspan="2">Input data</th><th>Expected results</th></tr>
<tr><td rowspan="3">Functionality of RakNet network library</td><td>Initialization</td><td>Open port command</td><td rowspan="3">Correct operations</td></tr>
<tr><td>Transfer</td><td>Send/Recv command</td></tr>
<tr><td>Termination</td><td>Close port command</td></tr>
</table>

Table 18 Object Selection

Purpose	Input data	Expected results
User selecting objects in SenTrack program	Mouse click	Object selected and highlighted

Table 19 Query Object Status

Purpose	Input data	Expected results
User querying objects in SenTrack program	Mouse click	Object properties displayed

Table 20 Issuing LOD Command

Purpose	Input data	Expected results
User issuing LOD command in SenTrack program	LOD command	LOD command to be delivered to sensor management & configuration module

Table 21 Executing LOD Command

Purpose	Input data	Expected results
Sensor management & configuration module executing LOD command in SenRoam program	LOD command	Sensor tracking levels changed

APPENDIX C: QUARTERNION BASED TRANSFORMATION

```
for(int x = 0; x < m_usNumJoints; x++)
{
	//Transformation matrix
	CMatrix4X4 matTmp;
	//Current joint
	SMs3dJoint * pJoint = &m_pJoints[x];
	//Current frame
	unsigned int uiFrame = 0;

	//if there are no keyframes, don't do any transformations
	if(pJoint->m_usNumRotFrames == 0
		&& pJoint->m_TransKeyFrames == 0)
	{
		pJoint->m_matFinal = pJoint->m_matAbs;
		continue;
	}
	//Calculate the current frame
	//Translation
	while(uiFrame < pJoint->m_usNumTransFrames &&
		pJoint->m_TransKeyFrames[uiFrame].m_fTime < fTime)
		uiFrame++;
	pJoint->m_usCurTransFrame = uiFrame;

	float fTranslation[3];
	float fDeltaT = 1;
	float fInterp = 0;

	SMs3dKeyFrame * pkCur = &pJoint->m_TransKeyFrames[uiFrame];
	SMs3dKeyFrame * pkPrev = &pJoint->m_TransKeyFrames[uiFrame-1];

	fDeltaT = pkCur->m_fTime - pkPrev->m_fTime;
	fInterp = (fTime - pkPrev->m_fTime) / fDeltaT;

	//Interpolate between the translations
	fTranslation[0] = pkPrev->m_fParam[0] +
				(pkCur->m_fParam[0] - pkPrev->m_fParam[0]) * fInterp;
	fTranslation[1] = pkPrev->m_fParam[1] +
				(pkCur->m_fParam[1] - pkPrev->m_fParam[1]) * fInterp;
	fTranslation[2] = pkPrev->m_fParam[2] +
				(pkCur->m_fParam[2] - pkPrev->m_fParam[2]) * fInterp;
```

```
		//Calculate the current rotation
		uiFrame = 0;
		while(uiFrame < pJoint->m_usNumRotFrames &&
			pJoint->m_RotKeyFrames[uiFrame].m_fTime < fTime){
			uiFrame++;
		}

		SMs3dKeyFrame * pkCur = &pJoint->m_RotKeyFrames[uiFrame];
		SMs3dKeyFrame * pkPrev = &pJoint->m_RotKeyFrames[uiFrame-1];

		fDeltaT = pkCur->m_fTime - pkPrev->m_fTime;
		fInterp = (fTime - pkPrev->m_fTime) / fDeltaT;

		//Create a rotation quaternion for each frame
		CQuaternion qCur;
		CQuaternion qPrev;
		qCur.FromEulers(pkCur->m_fParam);
		qPrev.FromEulers(pkPrev->m_fParam);
		//SLERP between the two frames
		CQuaternion qFinal = qCur.Slerp(qPrev, qCur, fInterp);
		//Convert the quaternion to a rotation matrix
		matTmp = qFinal.ToMatrix4();

		//Set the translation part of the matrix
		matTmp.SetTranslation(fTranslation);

		//Calculate the joints final transformation
		CMatrix4X4 matFinal = pJoint->m_matLocal * matTmp;

		if(pJoint->m_sParent == -1)
			pJoint->m_matFinal = matFinal;
		//otherwise the final matrix is the parents final matrix * the new matrix
		else
			pJoint->m_matFinal = m_pJoints[pJoint-
>m_sParent].m_matFinal * matFinal;

	}
```

APPENDIX D: SAMPLE RAW DATA

//Timestamp: 21:34:33, 4-4-2007.

LOD ON:

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1.14	1	1	1.14	1.14	1.14	1.14	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1.14	1.14	1.14	1.14	1.14	1.28	1	1	1.14	1.14	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1.14	1	1	1	1	1.28	1	1	1	4.23	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	3.53	1	1	1	1	1	2.83	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1.14	1	1	1	1	1	1	1.28	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1.14	1	1	1	1	1.14	1.14	1.28	1.7	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1.56	1.14	1.14	1.14	1.28	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

LOD OFF:

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	2.11	1	1	1.86	1.2	1.2	1.47	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1.94	2.45	1.7	1.48	1.94	2.38	1	1	1.48	1.58	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1.39	1	1	1	1	2.02	1	1	1	2.11	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	12.44	1	1	1	1	1	3.91	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1.23	1	1	1	1	1	1	2.48	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	2.05	1	1	1	1	2.11	1.47	2.3	3.55	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	3.27	1.7	2	1.22	3.03	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Saving

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0.97	0	0	0.72	0.06	0.06	0.33	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0.8	1.31	0.56	0.34	0.8	1.09	0	0	0.34	0.44	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0.25	0	0	0	0	0.73	0	0	0	-2.13	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	8.91	0	0	0	0	0	1.08	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09	0	0	0	0	0	0	1.2	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.91	0	0	0	0	0.97	0.33	1.02	1.84	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.7	0.56	0.86	0.08	1.75	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

REFERENCES

1. B. I. Koerner, “Intel’s Tiny Hope for the Future,” *Wired Magazine*, Issue 11.12, Dec. 2003.
2. I.F. Akyildiz et al., “A Survey on Wireless Sensor Networks,” *Computer Networks*, vol. 38, Issue 4, pp. 393-422, March 2002.
3. H. Karl and A. Willig, *Protocols and Architectures for Wireless Sensor Networks*. Hoboken, NJ: Wiley, 2005.
4. J.L. Hill, System Architecture for Wireless Sensor Networks, Doctoral dissertation, University of California, Berkeley, 2003.
5. A. Chandrakasan, R. Amirtharajah, S. Cho, J. Goodman, G. Konduri, J. Kulik, W. Rabiner, and A. Wang, “Design considerations for distributed micro-sensor systems,” in *Proc. IEEE 1999 Custom Integrated Circuits Conference*, San Diego, CA, May 1999, pp. 279–286.
6. C. Karlof and D. Wagner, “Secure routing in wireless sensor networks: attacks and countermeasures,” in *Proc. IEEE 2003 International Workshop on Sensor Network Protocols and Applications*, May 2003, pp. 113-127.
7. M. Rahimi, H.Shah, G.S. Sukhatme, J. Heideman, and D. Estrin, “Studying the feasibility of energy harvesting in a mobile sensor network,” in *Proc. ICRA '03 IEEE International Conference on Robotics and Automation*, Sept 2003, pp 19-24.
8. A. Mainwaring, “Wireless sensor networks for habitat monitoring,” in *Proc. 1st ACM international workshop on Wireless sensor networks and applications, Georgia*, 2002, pp. 88-97.
9. M.A.M. Vieira, et al, “Survey on wireless sensor network devices,” in *Proc. ETFA '03 IEEE Conference on Emerging Technologies and Factory Automation*, Sept. 2003, vol.1, pp 537- 544.
10. J.M. Rabaey, M.J. Ammer, J.L. da Silva Jr, D. Patel, and S. Roundy, “PicoRadio Supports Ad Hoc Ultra-low Power Wireless Networking,” *IEEE Computer*, 33(7), pp. 42-48, July 2000.
11. K. Sohrabi, J. Gao, V. Ailawadhi, and G.J. Pottie, “Protocols for Self-Organization of a Wireless Sensor Network,” *IEEE Personal Communications*, vol. 7, Oct. 2000.
12. V. Srinivasan, P. Nuggehalli, and R. Rao, “Design of optimal energy aware protocols for wireless sensor networks,” *Vehicular Technology Conference (VTC'01). IEEE VTS 53rd* , May 2001, vol. 4, pp. 6-9.
13. J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan, “Building Efficient Wireless Sensor Networks with Low-Level Naming,” in *Proc. Symposium on Operating Systems Principles*, Oct. 2001, pp. 146-159.
14. A. Howard, M.J. Mataric, and G.S. Sukhatme, “An Incremental Self-Deployment Algorithm for Mobile Sensor Networks,” *Autonomous Robots, Special Issue on Intelligent Embedde d Systems*, 13(2), pp. 113-126, 2002.

15. S. Meguerdichian, F. Koushanfar, M. Potkonjak, and M.B. Srivastava, “Coverage problems in wireless ad-hoc sensor networks,” *Infocom*, pp. 1380-1387, April 2001.
16. NB Priyantha, H Balakrishnan, E Demaine, and S Teller, “Mobile-Assisted Localization in Wireless Sensor Networks,” in *Proc. IEEE Infocom*, March 2005, vol. 1, pp. 172-183.
17. C. Savarese, J. Rabaey, and J. Beutel, “Locationing in Distributed Ad-Hoc Wireless Sensor Networks,” in *Proc. IEEE ICASSP*, Salt Lake City, UT, May 2001, pp. 2037–2040.
18. R. L. Moses, D. Krishnamurthy, and R. M. Patterson, “A selflocalization method for wireless sensor networks,” *Eurasip Journal on Applied Signal Processing, special Issue on Sensor Networks*, March 2003,
19. R. Nagpal, H. Shrobe, and J. Bachrach, “Organizing a Global Coordinate System from Local Information on an Ad Hoc Sensor Network,” in *Proc. 2nd IPSN*, Palo Alto, CA, 2003.
20. Y. Shang, W. Ruml, Y. Zhang, and M. Fromherz, “Localization from mere connectivity,” in *Proc. 4th ACM MOBIHOC*, Annapolis, Maryland, 2003, pp. 201–212.
21. N. Priyantha, H. Balakrishnan, E. Demaine, and S. Teller, “Anchor-Free Distributed Localization in Sensor Networks,” MIT Laboratory for Computer Science, Tech. Rep. 892, April 2003.
22. D. Rakhmatov and S. Vrudhula, “Energy Management for Battery-Powered Embedded Systems,” *ACM Transactions on Embedded Computing systems*, vol. 2, pp. 277-324, August 2003.
23. N. Ramanathan and M. Yarvis, “A Stream-Oriented Power Management Protocol for Low Duty Cycle Sensor Network Applications,” *Emnets-II*, May 2005, pp. 53-62.
24. G. Hoblos, M. Staroswiecki, and A. Aitouche, “Optimal design of fault tolerant sensor networks,” *IEEE International Conference on Control Applications*, Anchorage, AK, Sep. 2000, pp. 467–472.
25. G. Gupta and M. Younis, “Fault-Tolerant Clustering of Wireless Sensor Networks,” *Wireless Communications and Networking*, vol.3, pp. 1579-1584, March 2003.
26. D. Luebke, M. Reddy, J.D. Cohen, A. Varshney, B. Watson and R. Huebner, *Level of Detail for 3D Graphics: Application and Theory*. San Francisco, CA: Morgan Kaufmann, 2002.
27. J. C. Xia, J. El-Sana, and A. Varshney, “Adaptive real-time level-of-detail based rendering for polygonalmodels”, *IEEE Transactions on Visualization and Computer Graphics*, 3(2), pp. 171 - 183, 1997.
28. D. Vaidya, J. Peng, L. Yang, and J. W. Rozenblit, “A Framework for Sensor Management in Wireless and Heterogeneous Sensor Network,” in *Proc. 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, 2005, pp. 155-162.
29. L. Yang, C. Feng, J. W. Rozenblit, and H. Qiao, “Adaptive tracking in distributed wireless sensor networks,” in *Proc. 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems*, March 2006.

30. M. Bhardwaj, T. Garnett, and A.P. Chandrakasan, “Upper bounds on the lifetime of sensor networks,” *2001 IEEE International Conference on Communications*, 2001, vol. 3, pp. 785-790.
31. D. Smith and S. Singh, “Approaches to Multisensor Data Fusion in Target Tracking: A Survey,” *IEEE Transactions on Knowledge and Data Engineering*, 18(12), pp. 1696-1710, Dec. 2006.
32. S. Goel and T. Imielinski, “Prediction-based Monitoring in Sensor Networks: Taking Lessons from MPEG,” *ACM Comp. Commun. Rev.*, 31(5), pp. 82-98, Oct. 2001.
33. J. Peng, L. Yang, J.W. Rozenblit and M. Faisal, “Level-of-detail Based Modeling of Object Movement in Tracking and Surveillance System,” in *Proc. 2005 Winter Simulation Conference*, Orlando, FL, USA, Dec. 2005, pp. 461-466.
34. F. Momen, et al. “Three layer architecture for continuous planning and execution,” in *Proc. 2001 Army Research Laboratories Symposium*, College Park, MD, 2001.
35. P. Mittrapiyanuruk, G. N. DeSouza and A. C. Kak, “Calculating the 3D-Pose of Rigid Objects Using Active Appearance Models,” in *Proc. International Conference in Robotics and Automation*, 2004, vol. 5, pp. 5147-5152.
36. T.B. Moelsund, “A Survey of Computer Vision-Based Human Motion Capture,” *Computer Vision and Image Understanding*. 81(3), pp. 231-268, March 2001.
37. B. Atwood, 2004, http://www-bsac.eecs.berkeley.edu/archive/users/warneke-brett/SmartDust/index.html
38. L. Hong, et al., “Multirate Interacting Multiple Model Filtering for Target Tracking,” in *Proc. SPIE*, Jul. 1995, vol. 2561, pp. 275-286.
39. L. Hong, et al., “A Multiresolutional Approach to Target Tracking,” *IEEE Aerospace and Electronics Conference*, May 1993, pp. 388-392.
40. L. Hong, et al., “Real-time Optimal Multiresolutional Sensor/Data Fusion,” *IEEE Robotics and Automation, International Conference*, Sep. 1993, pp. 117-122.
41. R. Gross and J. Shi. “The CMU motion of body (MoBo) database,” Technical Report CMU-RI-TR-01-18, Robotics Institute, Carnegie Mellon University, 2001.
42. T. Chalidabhongse, V. Kruger, and R. Chellappa. “The UMD database for human identification at a distance,” Technical report, University of Maryland, 2001.
43. B.W. Hwang, S. Kim, and S.W. Lee, “A Full-Body Gesture Database for Automatic Gesture Recognition,” *7th International Conference on Automatic Face and Gesture Recognition*, April 2006, pp. 243-248.
44. C. Ménier, E. Boyer, and B. Raffin, “3D Skeleton-Based Body Pose Recovery,” in *Proc. 3rd International Symposium on 3D Data Processing, Visualization and Transmission*, Chapel Hill, June 2006, pp. 389-396.

45. S. McConnell, *Rapid Development: Taming Wild Software Schedules*. Redmond, WA: Microsoft Press. 1996.
46. M. Segal and K. Akeley, “The OpenGL Graphics System: A Specification,” Silicon Graphics, Inc, 2004.
47. M. Woo, J. Neider, T. Davis, and D. Shreiner, *OpenGL Programming Guide: The Official Guide to Learning OpenGL*. Version 1.4, Fourth Edition, Reading, Mass: Addison-Wesley Professional, 2003.
48. 2006, http://www.rakkarsoft.com/raknet/manual/, Rakkarsoft, L.L.C.
49. K. Appiah, A. Hunter, and T. Kluge, “GW4: An FPGA-driven image segmentation algorithm,” in *Proc. 5th World Scientific and Engineering Academy and Society International Conference on Signal, Speech and Image Processing*, 17-19 August, Corfu Island, Greece, 2005.
50. A.E. Miguel, M.R. Selene, T.H. Cesar, and K. Nasser, “FPGA-based architecture for motion recovering in real-time”, *in Proc. SPIE*, March 2002, vol. 4666, pp. 116-123.
51. G. J. Myers, *The Art of Software Testing*. New York: Wiley, 1979.
52. R. Patton, *Software Testing*. Indianapolis, IN: Sams Pub, 2005.