Graduate School ETD Form 9
(Revised 12/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Aboli Manas Phadke

Entitled
DESIGNING AND EXPERIMENTING WITH e-DTS 3.0

For the degree of Master of Science

Is approved by the final examining committee:

Dr. Rajeev Raje Chair	
Dr. Mihran Tuceryan	
Dr. Yao Liang	

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): Dr. Rajeev Raje

Approved by: Dr. Shiaofen Fang 11/14/2013
Head of the Graduate Program Date

DESIGNING AND EXPERIMENTING WITH e-DTS 3.0

A Thesis

Submitted to the Faculty

of

Purdue University

by

Aboli Manas Phadke

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

December 2013

Purdue University

Indianapolis, Indiana

ACKNOWLEDGEMENTS

I would like to thank my thesis advisor Dr. Rajeev Raje for trusting me with an opportunity to take up this research activity under his able guidance. I have always respected and will continue to respect his tremendous patience, motivation and his dedication. I would like to thank Dr. Mihran Tuceryan for graciously sharing his knowledge and suggestions. I would also like to thank Dr. Yao Liang for agreeing to be a part of my examining committee.

My friends and colleagues in Software Engineering and Distributed Systems lab have always guided and helped me throughout the journey. I would like to extend special thanks to Ryan Rybarczyk, who went the extra mile to help me during the research work. I would also like to thank Lahiru Gallege, Dimuthu Gamage and Sonali Sharma for sharing their research experiences and guiding me.

My association with Indiana University Purdue University Indianapolis (IUPUI) has imparted a great deal of knowledge and has helped me to grow professionally. Hence, I would like to thank the entire Department of Computer and Information Science at IUPUI.

I cannot finish without acknowledging the incredible support my husband, Manas, has given to me. Thanks to him, my parents, my in-laws and my family for their love and support. I would not have taken up this research activity without their motivation and the faith they have in me.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Phadke, Aboli Manas, M.S., Purdue University, December 2013. Designing and experimenting with e-DTS 3.0. Major Professor: Dr. Rajeev Raje.

With the advances in embedded technology and the omnipresence of smartphones, tracking systems do not need to be confined to a specific tracking environment. By introducing mobile devices into a tracking system, we can leverage their mobility and the availability of multiple sensors such as camera, Wi-Fi, Bluetooth and Inertial sensors. This thesis proposes to improve the existing tracking systems, enhanced Distributed Tracking System (e-DTS 2.0) [19] and enhanced Distributed Object Tracking System (eDOTS) [26], in the form of e-DTS 3.0 and provides an empirical analysis of these improvements. The enhancements proposed are to introduce Android-based mobile devices into the tracking system, to use multiple sensors on the mobile devices such as the camera, the Wi-Fi and Bluetooth sensors and inertial sensors and to utilize possible resources that may be available in the environment to make the tracking opportunistic. This thesis empirically validates the proposed enhancements through the experiments carried out on a prototype of e-DTS 3.0.

CHAPTER 1. INTRODUCTION

1.1 Introduction

Object detection and indoor tracking or positioning is the basis of many applications in surveillance and activity recognition such as pedestrian tracking, gait analysis, locating patients and instruments in hospitals, locating workers and tools on a shop-floor. Most object tracking and positioning systems need additional infrastructure such as cameras and wireless access points (WAPs or APs) to be deployed. This research work focuses on Opportunistic Tracking which is a type of tracking where the object of interest can be detected and tracked with minimal infrastructure installation or changes; in other words, by taking advantage of the available infrastructure and any other opportunity that may become available during tracking.

1.1.1 Need for indoor tracking systems

The Global Positioning System (GPS) makes the positioning information available to anyone, anywhere and free of charge after a small initial investment in the GPS receiver. However the GPS needs a clear view of the sky (i.e. availability of satellites) which may not be possible in an indoor environment. Also, a wide variety of physical barriers and, potential interference sources, such as thick walls and ceiling, metallic objects, microwave ovens and other wireless transmitters/receivers, make it difficult for the GPS to pinpoint its location accurately.

Before 2000, public GPS signals were intentionally degraded for national security reasons. This is called Selective Availability (SA). The error with the standard GPS receivers without Selective Availability is typically three to five meters and may be up to ten meters [45] in the worst case which is not sufficient for an indoor positioning application such as asset tracking. Hence, for object tracking in an indoor environment, we cannot rely on GPS and need an indoor positioning system (IPS) that can work well in an indoor environment.

1.1.2 Need for multiple sensors and mobile devices in tracking

As indicated earlier, an opportunistic IPS must take advantage of any possible sensors in an environment. Indoor environments are characterized by occlusions, people and objects moving in non-deterministic fashion. For example, Wi-Fi signal may be weak inside a room rendering Wi-Fi-based tracking impossible. In such a scenario, another tracking modality e.g. vision-based tracking may help to overcome this problem. Hence, only one class of sensors such as Wi-Fi or vision may not be able to track the position of a moving object accurately. Majority of the tracking systems that exist today are standalone systems (e.g. [1], [3], [5], [7]) and use a single tracking device (specialized head-mounts or generic tracking devices such as a mobile/desktop application, GPSs). Also, most of these systems employ only a single sensor modality for tracking. If the environment is unable to support that tracking modality (e.g., poor lighting conditions for vision-based tracking or absence of Wi-Fi access points (APs) and associated connectivity) then the tracking is rendered useless. Having multiple sensor modalities can increase the fault tolerance of the tracking system, help to complement various modalities, and thus reduce error associated with the tracking results. This observation demands that the design of any next-generation IPS must support sensors of different modalities (e.g., vision, Wi-Fi, and Bluetooth) and cooperate by fusing their observations about the location of an object, to provide accurate tracking result.

In the recent past, various mobile devices (e.g., cell phones) are becoming omnipresent in indoor setting and are getting more sophisticated in terms of their capabilities and computational power. In addition standard capabilities, such as placing and receiving calls and messages, current mobile devices have many sensors such as camera, inertial sensors (accelerometer, gyroscope, magnetometer, etc.), Bluetooth, and Wi-Fi. According to a study conducted by Pew Internet in January 2013 [13], 45% of Americans own a smart-phone and number of mobile devices is soon expected to exceed world's population. These figures show that the smart-phones are nearly ubiquitous. Cell phones, due to their omnipresence and newer capabilities, present themselves as powerful sources of additional modalities in indoor tracking. Cell phones, hence, can augment the traditional sensors (e.g., webcams) found in an existing IPS (e.g., e-DTS 2.0 [19]) and thus, possibly improve the accuracy of tracking and extend the scope of an IPS.

This thesis aims to precisely address these two requirements of an IPS – namely, the incorporation of mobile devices (e.g., Android-based cell phones) and different modalities (vision, Wi-Fi, sensors on smart phones) – and create a next generation of the IPS called the e-DTS 3.0. The design of such an IPS must consider various underlying research issues such as heterogeneity of sensors or trackers, semi-automatic discovery of sensors, mobility of sensors, sensor data fusion, scalability and fault-tolerance. These issues are tackled in this thesis and proposed solutions are described in later chapters.

1.2 Objectives of the thesis

The specific objectives of this thesis are as follows:

1. To incorporate mobile devices and their associated sensors in the design of existing tracking systems (e-DTS 2.0 [19] and eDOTS [26]) to improve their scope and accuracy.
2. To empirically validate the proposed tracking system.

1.3 Problem statement discussion

The 'enhanced Distributed Tracking System' (e-DTS 2.0) proposed in [19] is a distributed tracking system composed of vision-based tracking services running on desktop machines. Vision-based tracking in e-DTS 2.0 [19] is enhanced by Wi-Fi tracking in 'enhanced Distributed Object Tracking System' (eDOTS) [26]. In our research, we propose to introduce mobile devices in the distributed tracking system. With the addition of mobile devices, tracking can be carried out at any place where a mobile device can be carried, thus, improving the physical scope of operation. We propose to further the attempt of indoor tracking by introducing Bluetooth radio present in an Android-based mobile device. Although Bluetooth-based tracking does not exhibit a promising accuracy, it can be helpful when no other tracking modality can be used or it may complement Wi-Fi-based tracking when vision-based tracking is not possible due to poor lighting conditions or occlusions. This improves the fault tolerance of the tracking system compared to that in [19]. User should be able to use at least one tracking modality without any additional infrastructure and hence, this tracking can be called as Opportunistic Tracking which takes advantage of the infrastructure that is readily available. To maintain the information about tracking services and to aid the semi-automatic discovery of tracking services, we propose to build a Discovery Service. To deal with the dynamicity of the environment and the physical scope of the tracking environment, we propose to build and maintain a history of tracking services with the Discovery Service. This history helps a tracking device to maintain the information about other tracking devices or nodes in e-DTS 3.0, the patterns they see and their last known positions. With the help of this history, tracking services can estimate locations of different patterns that they can detect directly or indirectly.

1.4 Contributions of the thesis

This thesis uses and improves the features of [19] and [26] to develop e-DTS 3.0. This thesis enhances e-DTS 2.0 [19] by introducing mobile devices, in addition to the stationary sensors, which enhance the scope and accuracy of tracking. Experiments conducted in this thesis demonstrate that the use of multiple sensor modalities helps to complement their error characteristics. It empirically proves that the capabilities of Android-based devices, e.g. hotspot capability, can be used to carry out Opportunistic Tracking. We address the challenges presented by the dynamic nature of the mobile devices, in terms of their position, availability and connectivity, by adding new features a) to the Discovery Service such as maintaining environmental history and b) to the tracking services such as maintaining a graph of peer tracking services.

1.5 Overall organization of the thesis

This chapter discussed the motivation behind this thesis along with the proposed solution, hypothesis and the contributions of this thesis. The rest of the thesis is organized in four chapters. Chapter 2 reviews the related work. Chapter 3 discusses the design and implementation of e-DTS 3.0. Chapter 4 lays out the details of the experiments carried out and the results of the experiments. Chapter 5 assesses this research work and suggests some future enhancements.

CHAPTER 2. RELATED WORK

2.1 Related work in tracking

There have been many indoor tracking and positioning systems implemented till date. Different tracking systems have been built for different applications (e.g. indoor and outdoor applications, an application, such as aircraft welds, demanding accuracy of 10µm [46] and systems offering accuracy of 10m such as GPSs). The types of sensors and algorithms used and accuracy of the system developed depend upon the intended application area. In addition to these, Mautz's thesis [6] outlines different parameters such as coverage area, cost, scalability, availability etc. to be considered before developing or selecting appropriate indoor tracking system. There are numerous successful attempts in devising indoor tracking systems for different applications.

Kothari et al. [5] use two complimentary approaches of Wi-Fi and dead-reckoning using sensors built-in a commercial smartphone. They use the incremental approach to test the accuracy of the tracking system in which they first use dead reckoning and Wi-Fi individually and then use both these techniques together. We use a similar incremental approach in order to gauge the performance of our proposed tracking application.

Laoudias et al. [4] use radio map which is a collection of RSS values or fingerprints built a priori. The position is estimated by comparing the previously obtained fingerprints and fingerprints obtained at a given position while tracking. A similar effort, proposed by Shin et al. [3], divides the tracking environment in fixed cells and identifies RSS from all access points (AP) in each cell a priori. Thus during the tracking process, they match current RSS with that calculated a priori and then estimate user’s current cell. In this approach the accuracy is equal to the area of the cell. Also, due to the use of RSS fingerprint comparison technique, this approach demands some pre-processing. Cricket [47] developed at MIT, recognizes the spaces where a user might be rather than estimating or calculating user’s position. This is achieved by beacons periodically transmitting text strings which announce the name assigned to their positions. Users interested in finding out the location of them do so by listening to the beacon advertisement. This approach preserves the user’s privacy, is easy to administer and deploy and has low cost. Similarly, one of the early systems RADAR proposed in [48] uses the RSS and Signal to Noise ratio to estimate the user’s position. It uses the triangulation based on the measured and theoretical signal strength information. The approach proposed in [50] shows how location systems can configure the radio-map or the map of Wi-Fi devices by using public war driving databases, Wi-Fi hotspot finders and GPSs.

Werner et al. [2] present indoor positioning in which image recognition is used for positioning. This involves building a database of environmental images. As a part of tracking, these images are compared to what the user sees in order to estimate the user’s location. This approach involves building database (online or offline) and may not be suitable for dynamic, Opportunistic Tracking since it needs a priori data collection and maintenance.

Aubeck et al. [1] use magnetometer, gyroscope, accelerometer and camera in a smartphone. This approach is very attractive for pedestrian tracking since they use step detection using camera. Complementing inertial sensors with camera based step detection algorithm helps to overcome error introduced by the inertial sensors. Shala et al. [7] also tested and used individual and fused results from Wi-Fi fingerprinting and inertial sensors complemented with step detection to track the location of a moving person on a smartphone. The application that we intend to use the tracking system for is tracking a person as well as a marker of interest (affixed on an instrument). Hence, the step detection approach may not be suitable for our intended application.

Wang et al. [36] used the Inside-Out tracking with the help of Infrared Light Emitting Diodes (LEDs) fixed on the ceiling and optical sensors fixed on a head-mount. In an Inside-Out tracking system, the patterns or the markers are fixed in the environment and the sensors move in the tracking environment to detect the markers. It is cheaper to install markers in an environment than it is to install sensors. Also, for a small movement of a head-mount sensor, there is a large change in tracking result because of which inside-out tracking proves to be more sensitive [36] and gives more accurate orientation results [9]. Since we propose to use mobile devices that users carry in their hands (similar to head-mount sensors), we expect to have the advantages that an Inside-Out tracking system offers.

Keitler et al. [37] proposed indirect tracking in order to reduce the effect of occlusions on tracking. They use stationary trackers (cameras) to track mobile trackers which in turn track the markers. To remove the errors because of the differences in orientations of stationary and mobile markers, they introduce one or more reference points in the tracking environment which are visible to both types of trackers. Since the e-DTS 3.0 involves both mobile trackers (tracking services hosted on Android-based mobile devices) and stationary trackers (tracking services from e-DTS 2.0 [19]), we will carry out experiments to test the accuracy of such a concatenation of stationary and mobile trackers.

Waechter et al. [38] devised a mobile setup of multiple sensors such as cameras capable of marker tracking and Natural Feature tracking and an Optical Odometric Sensor system consisting of optical flow sensor units which help to measure the displacement of the mobile unit (a tripod fixed with wheels) in two dimensions. This work also helped us understand the Spatial Relationship Graphs which is a way of expressing the coordinate systems (or orientations) of different components in a tracking system.

Klinker et al. [39] envision future tracking systems comprised of mobile sensors situated on a wearable or hand-held computing device and stationary sensors provided as a part of the infrastructure. Stationary sensors help to estimate the position of the mobile sensors and mobile sensors fine-tune the position estimation further. In order to improve the tracking system, Klinker et al. [39] suggest re-evaluating concepts of Distributed Systems such as adaptability, load balancing, security, adjusting Quality of Service (QoS) on demand while developing a tracking system. This paper contributes useful guidelines for Distributed Tracking Systems' developers.

Bauer et al. [40] have proposed and developed a prototype indoor and outdoor navigation system framework. This framework is based upon the concepts of reusable components. An Augmented Reality (AR) system built using this framework consists of services which have their own memory, I/O devices and network connections and can run on separate hardware components. They collaborate with each other by publishing their ‘abilities’ and ‘needs’. Bauer et al. show that a prototypical application based on this framework can be developed within few weeks and shows promising results in terms of the response time and overall performance.

In addition to navigation purposes, there have been some notable attempts in indoor positioning and tracking. For example, the Smart Home technology [28] which leverages ongoing work at the University of Missouri helps in monitoring health conditions of elderly people. This technology uses motion sensors for activity monitoring, webcam silhouette images and Kinect depth images for gait analysis, vision and acoustic sensing for fall detection, and a hydraulic bed sensor that captures pulse, respiration and restlessness. All these sensors gather useful information about daily activities of elderly persons which may help in analyzing early signs of a potential disorder.

In addition to the above mentioned research work, e-DTS 2.0 [19] and eDOTS [26] provided a basis for this research work. e-DTS 2.0 [19] is a succession of previous successful attempts at Distributed Tracking Systems [20], [21]. It proposed dynamic discovery of tracking infrastructure using multicast protocol. They also experimented with camera calibration and clock synchronization. eDOTS [26] enhanced the work done in [19] by introducing Wi-Fi-based tracking. With e-DTS 3.0, we used the work and results published in [19] for clock synchronization. Wi-Fi-based tracking used in e-DTS 3.0 is based on the work done in [26]. We also used [19] and [26] as baselines to validate our results in terms of accuracy of the tracking results. In addition to the vision-based sensors, we propose to introduce more tracking sensors (such as Wi-Fi and Bluetooth sensors) and their functional capabilities present on the Android-based devices which can help us to achieve our objective of Opportunistic Tracking in e-DTS 3.0.

2.2 Related work in dynamic discovery of mobile devices

This application demands an ad-hoc network of mobile devices over which different mobile devices acting as independent tracking devices can communicate with each other and with the Discovery Service and the Fusion Service to achieve collaborative tracking. Mobile devices discover their neighbors using tracking sensors or by communicating with the Discovery Service. As specified in IETF RFC 2501, these ad-hoc networks have following characteristics [10] –

1. Dynamic topologies
2. Constraints on Bandwidth
3. Energy-constrained operation
4. Limited physical security

With the evolution of smartphones, device manufacturers are offering more and more features with the newer versions of smartphones to address above mentioned characteristics and to let developers concentrate more on the application development. Recent Android-based devices offer two types of service discovery APIs – Network Service Discovery [11] and using Wi-Fi Direct [12] starting from API level 16 and 14 respectively. With the new devices this would certainly be an attractive alternative for the dynamic service discovery over an ad-hoc network of mobile phones. But since we would like to use android-based devices running API level 7(Éclair) and above, we are not going to avail this utility.

e-DTS 2.0 [19] uses Jini multicast request and announcement protocol to discover tracking services present within the network. Once the lookup service is discovered, the camera service registers with the lookup service and then the tracking client can get a reference to the tracking services through lookup. Jini is based on Remote Method Invocation (RMI) which Android-based devices do not support. Hence, we developed a basic centralized semi-automatic Discovery Service that creates and maintains the registry of currently active tracking services and a history of tracking services that were active in the past.

Thus to summarize the related work we studied, many of the systems use only a subset of the sensors available on a mobile device. They use the data collected a priori and infer the position of the target based on the comparison or by actually calculating the position. The tracking services deployed on mobile devices described in previous attempts are standalone systems and not distributed systems. Hence, to take advantage of a distributed system and improve the accuracy, availability and fault tolerance of the tracking system we propose e-DTS 3.0 composed of multiple Android-based mobile devices, stationary sensors, Fusion Service and Discovery Service.

CHAPTER 3. DESIGN AND IMPLEMENTATION OF e-DTS 3.0

Chapters 1 and 2 discussed the general outline of e-DTS 3.0 and described previous and related efforts in the area of tracking. In this chapter, we present technical details about the design and implementation of the system.

3.1 Rationale behind developing e-DTS 3.0

The main idea behind developing a new tracking system is to complement existing distributed tracking systems, e-DTS 2.0 developed in [19] and eDOTS developed in [26], with the introduction of mobile devices. e-DTS 2.0 [19] and its enhancement, eDOTS [23] use only vision and Wi-Fi-based tracking. Also, these tracking systems, due to the use of immobile tracking sensors such as web-cameras, remain confined to an area where the infrastructure can be installed. Our goal behind developing e-DTS 3.0 is to be able to use the tracking system where it is needed and to collaborate with the infrastructure that is available in order to improve the accuracy and/or the efficiency of tracking.

3.2 Architecture of e-DTS 3.0

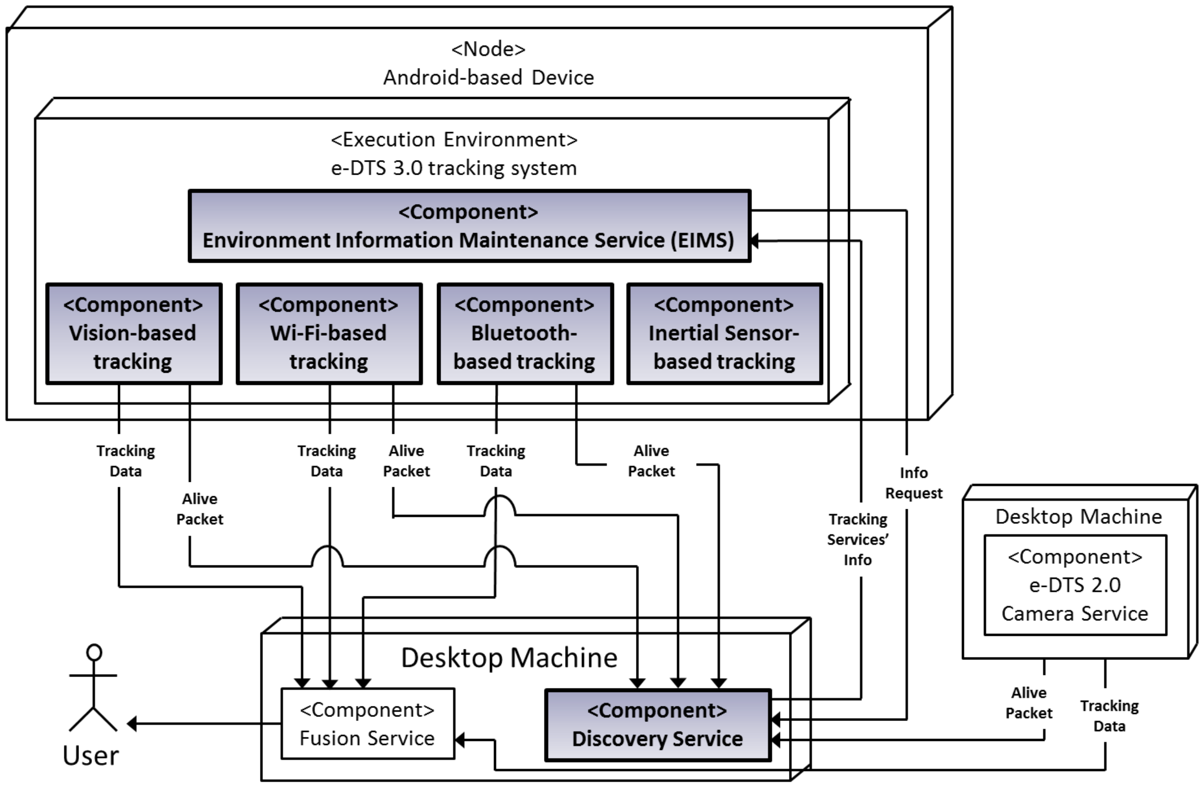


Figure 3.1 Architecture of e-DTS 3.0

Figure 3.1 above shows the architecture of e-DTS 3.0. Since e-DTS 3.0 is an enhancement of e-DTS 2.0 [19] and eDOTS [26], e-DTS 3.0 has building blocks similar to that of [19] and [26]. As shown in Figure 3.1, e-DTS 3.0 consists of Fusion Service, Discovery Service, e-DTS 2.0 camera service and Android-based mobile device(s). The components enclosed in the shaded boxes in Figure 3.1 are the components that are newly added or enhanced as part of this thesis.

The Fusion Service is responsible for fusing the tracking results from multiple tracking services. Tracking services communicate the tracking results to the Fusion Service using TCP/IP sockets. The Fusion Service may receive tracking data from tracking services continuously. This frequent communication quickly drains the battery of an Android-based device and the device may not be available for the Fusion process. The tracking system application demands the Fusion Service to have the availability and connectivity of close to 100%; hence, we run the Fusion Service on a desktop machine to eliminate the effect of limited battery power and intermittent connectivity of resource-constrained mobile devices such as Android-based smartphones.

The Discovery Service is responsible to keep track of active tracking services using a ‘Registry’. This registry is updated after every two seconds by removing the entry of tracking services that have not communicated with the Discovery Service in the past two seconds period. This period can be adjusted programmatically before the Discovery Service is run. In addition to the registry, Discovery Service maintains the ‘History’ of tracking services that may or may not be active currently but were active at some time during the history of this Discovery Service instance. In other words, the history is maintained as long as the Discovery Service is running and is purged when it is shut down. The history is updated every time a tracking service detects a new pattern and sends an ‘Alive Packet’ to the Discovery Service. To overcome the issues with mobile devices such as intermittent connectivity and limited battery life, similar to the Fusion Service, the Discovery Service is also run on the same or a different desktop machine as that of Fusion Service.

The e-DTS 2.0 camera service encapsulates the inexpensive web-cameras plugged into a desktop machine and is the elementary building block of e-DTS 2.0 [19]. If a camera service from e-DTS 2.0 [19] is running in the environment, e-DTS 3.0 takes advantage of it and is treated as a tracking service of e-DTS 3.0.

An Android-based mobile device hosts different tracking sensors such as camera, Wi-Fi and Bluetooth radio and inertial sensors such as an accelerometer and an Environment Information Maintenance service. Every sensor on an Android-based device is encapsulated as a tracking service. Whenever the tracking results are available, these services communicate the results to the Fusion Service and their aliveness to the Discovery Service shown as ‘Tracking Data’ and ‘Alive Packet’ in Figure 3.1. The Environment Information Maintenance Service (EIMS) running on an Android based device may request the information about other tracking services such as their IP addresses and the patterns they can track to the Discovery Service shown as ‘Info Request’ in Figure 3.1 and the Discovery Service responds by sending the ‘Tracking Services’ Info’. This information may further be used by a tracking device to communicate with other tracking devices to carry Opportunistic Tracking. Although there is no explicit authentication implemented for the communication between different entities, they follow certain protocols (explained in detail in sections 3.4, 3.5 and 3.6) that ensures that a malicious entity outside the tracking system cannot misuse any other entity or the resources in e-DTS 3.0.

As indicated earlier, the communication between all these entities is implemented using TCP/IP sockets since the data exchanged (in the form of Filter Data Packet and the Discovery Object explained in section 3.3.5) between various entities is very small and the overall architecture of the system does not demand a sophisticated technique for communication. In the following sections, we will discuss each component of e-DTS 3.0 in detail.

3.3 Tracking services in e-DTS 3.0

Android-based devices contain different sensor modalities such as vision, Wi-Fi, Bluetooth and Inertial sensors such as accelerometer and Gyroscope. Out of these available sensors, in e-DTS 3.0, we have used the vision sensor, i.e., camera, the Wi-Fi sensor and the Bluetooth sensor for the purpose of tracking. Out of the Inertial Sensors found on the Android-based device we have used an Accelerometer for tracking but there is further scope to improve the implementation algorithm in e-DTS 3.0.

Details of each of these tracking modalities are discussed in the following sections.

3.3.1 Vision-based tracking

Vision based tracking uses a pin-hole camera model embedded within the Android hardware. In order to visually locate and track the objects of interest, these objects are marked with distinct markers or patterns. Two patterns commonly used in vision-based tracking are shown below in Figure.3.2.

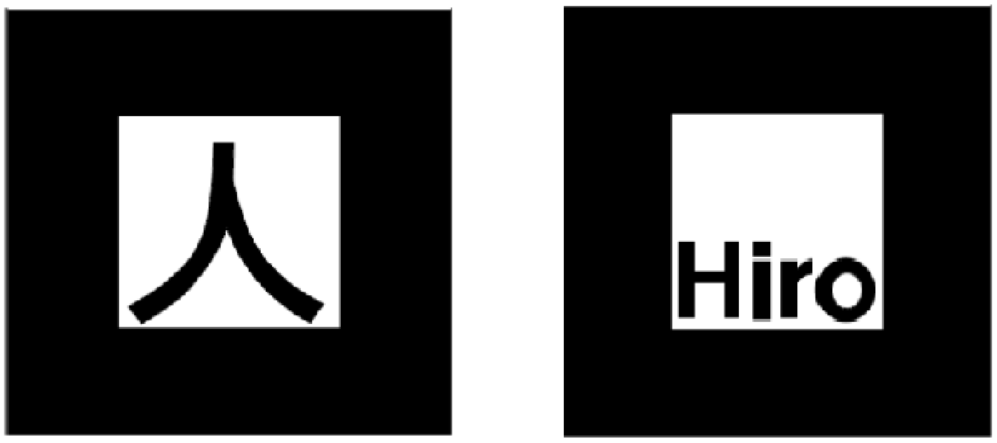


Figure 3.2 Commonly used patterns in vision-based tracking – kanji pattern and hiro pattern [44]

For detecting a pattern, we use a third-party library called AndAR [14] which is based on ARToolkit and OpenGL and provides the API necessary for Android-based Augmented Reality (AR) applications' development. Figure 3.3 shows the class diagram of AndAR library.

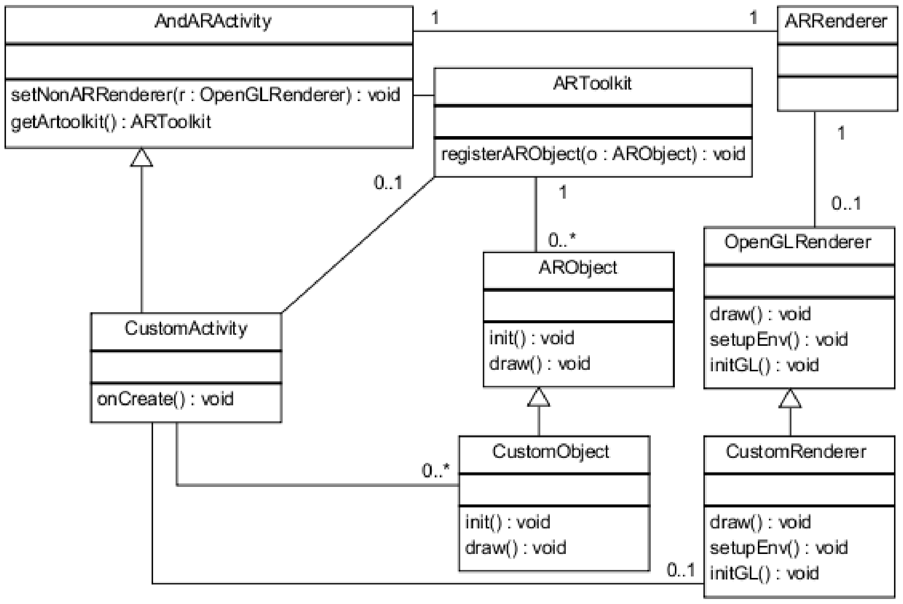


Figure 3.3 Architecture of AndAR library [29]

Every activity in the Android runtime is a single focused task that a user intends to carry out. In order to carry out multiple tracking sub-tasks, one for each tracking mode (or sensor, or service), we use a parent activity called *'TrackingActivity'* which initiates different modes of tracking. *TrackingActivity* extends *AndARActivity,* which handles the operations related to AR such as opening the camera, displaying video stream and detecting marker(s), and spawns a new thread for vision-based tracking. Following is the algorithm of the vision-based tracking thread of *TrackingActivity*.

```
Start
register an ‘ARObject’ corresponding to each pattern to be tracked
initiate myPosition as a 4 x 4 Identity Matrix
for (every ARObject){
    if (pattern_detected){
        get the transMat using AndAR library
        if (e-DTS 2.0 can give us recent myPosition){
            update myPosition
        }
        transMat = myPosition * transMat obtained from AndAR
        create the Filter Data Packet and Discovery Object
        send the Filter Data Packet to the Fusion Service
        send the Discovery Object to the Discovery Service
    }
}
End;
```

In order to detect custom markers, we have to register an ARObject for a marker by passing required arguments such as name and path of the file that stores binary representation of a marker and marker width. Once a marker is detected, the API returns the transformation matrix (the *transMat* in the above algorithm) shown in Figure 3.4, which is a 4 x 4 matrix stored in a ‘*double’* array in the column-major fashion. The first 3 x 3 sub-matrix, shown in Figure 3.4 below, represents the rotation between the camera and the pattern and the top three elements of the last column represent the position estimates of the camera on X, Y, and Z axes with respect to the marker. Other indices of the matrix (four elements in the last row) have fixed values as shown in Figure 3.4.

$$\begin{array}{ccc|c} \multicolumn{3}{c|}{\textbf{Rotation matrix}} & \textbf{Translation matrix} \\ \multicolumn{3}{c|}{\mathbf{3 \times 3}} & \mathbf{3 \times 1} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{array}$$

Figure 3.4 4 x 4 Transformation matrix

An Android-based device generally gives the transformation matrix of a pattern with respect to itself (i.e. considering its own position as the origin). To estimate the position of a pattern with respect to the world origin, which is the assumed origin of the tracking system with a fixed orientation, we multiply the transformation matrix obtained by the API (such as the one shown in Figure 3.4) with the matrix that represents the location of the Android-based camera from this assumed world origin. The matrix that represents the position of the Android-based device (*myPosition* in the algorithm) may be obtained from the e-DTS 2.0 camera service that can visually track an Android-based phone by tracking the pattern affixed to the Android-based phone. If T_1 is the position of a pattern with respect to the Android-based camera (estimated by the AndAR library in vision-based tracking service) and T_2 is the position of the Android-based camera with respect to the world origin (or referred to as the origin, henceforth), then the position of the pattern with respect to the origin, T, is computed as:

$$T = T_2 \, T_1$$

Such a transformation, which represents the position of the pattern with respect to the origin, is carried out in the vision-based tracking thread of *TrackingActivity*.

Once the tracking result is available, vision-based tracking thread sends this result to the Fusion Service. Additionally, it sends related information such as the pattern detected, the timestamp, the Android ID to the Discovery Service which maintains the registry and the history of tracking services. TCP/IP sockets are used for this communication. Vision-based tracking is carried out in a separate thread which shares the execution environment with Wi-Fi, Bluetooth and inertial sensor-based tracking.

3.3.2 Wi-Fi-based tracking

Unlike vision-based tracking, Wi-Fi–based tracking does not track a physical marker; rather, the Android-based device itself acts as a marker for Wi-Fi-based tracking. In other words, Wi-Fi-based tracking estimates the position of an Android-based device running the Wi-Fi tracking algorithm with respect to the world origin.

There are two types of Wi-Fi-based tracking techniques that have been implemented in literature [17].

1. Using Wi-Fi fingerprinting technique
2. Using Wi-Fi triangulation/trilateration technique

In fingerprinting, Received Signal Strength (RSS) from each wireless access point (WAP or AP) is gathered at pre-determined points in the environment. This is usually an offline process carried out once, unless there are significant changes in the WAP placement or the environment itself. Triangulation or trilateration is based on the geometry of circles, spheres and triangles. When it is known that a point (an Android-based tracking device running Wi-Fi trilateration algorithm) lies on the surfaces of three circles (assumed to have formed by the WAP coverage area) with radii equal to the respective distances of the device from the WAPs, we can estimate the position of the Android-based device with the help of location of the centers of the circles (WAPs) and their radii [18]. Availability of more than three APs may allow us to choose the APs based on their QoS. Similar to eDOTS [26], e-DTS 3.0 uses the standard Wi-Fi trilateration algorithm based on this premise. Following is the algorithm of Wi-Fi-based tracking.

```
Start
initiate WiFiManager
start the WiFi scan
Register a BroadcastReceiver with the scan_completed and RSSI_changed intent
when broadcast received{
    if (scanResults.size < 3){
        if (no hotspot is active){
            newDiscoveryObject.setPatternID(PATT_DUMMY)
            send newDiscoveryObject to the DS
            if (DS responds as ACK){
                turn the hotspot on
                end
            }
            else if(DS responds NAK){
                wait for the new scanResults from broadcast
```

```
                    }
                }
                else{
                    wait for new scanResults from broadcast
                }
            }
            else{
                sort the scanResults based on their RSSIin descending fashion
                assign positions to  first three access points from accessPoints.txt data file
                position = trilateration (access points list)
                create the Filter Data Packet and Discovery Object
                send the Filter Data Packet to the Fusion Service
                send the Discovery Object to the Discovery Service
            }
        }
End;
```

We use the *WifiManager* class which is the primary API provided by the Android environment that manages Wi-Fi connectivity [22]. Instead of constantly monitoring the changes in the Wi-Fi connectivity, the WifiManager lets developers register a broadcast receiver that, depending upon the registered intent action, gets executed. For example, for Wi-Fi-based tracking, we register a broadcast receiver with the application environment i.e. the *Context* which is invoked whenever the results of Wi-Fi scan are available or there is any change in the RSS. Change in the RSS indicates physical movement of the device given that the surrounding environment (particularly WAP) is immobile.

At least three APs are necessary for the trilateration algorithm because if it is known that a point lies on the intersection of two circles, then there might be two such possible positions of the point. The third AP provides additional information to narrow down the possible positions of the point to just one [18]. If the number of APs detected in a scan is less than three, then the Android-based device may consider activating hotspot on the device. With the hotspot capability of the Wi-Fi enabled device, it can act as a WAP and let other Wi-Fi enabled devices connect with itself in order to use the Wi-Fi. For the trilateration algorithm in Wi-Fi-based tracking, the hotspot-activated device acts as a WAP and increases the chances of the Wi-Fi scan initiated on other tracking devices picking up three WAPs necessary for the trilateration. Since we allow only one hotspot in the tracking system to be active at a time, the device first checks with the Discovery Service (DS in the algorithm) if there is an active hotspot in the environment before it becomes a hotspot itself. The Discovery Service maintains a local Boolean variable which indicates whether the hotspot is running or not. If there is no hotspot activated in the system, the Android-based device requests permission to the Discovery Service to allow activating the hotspot capability. Upon receiving permission from the Discovery Service, the device ceases to be a tracking device and continues to run as a hotspot. If there is an active hotspot in the system or the Discovery Service rejects the request to activate hotspot, then the device simply waits for the results of the next scan.

On the other hand, if the scan results contain three or more APs, the device sorts the list of APs in the descending order based on the RSS. By referring to the accessPoints.txt data file (please refer Appendix B), it assigns the positions to the APs and invokes the standard trilateration algorithm. The trilateration algorithm returns the position of the Android-based device in terms of X and Y coordinates. Similar to vision-based tracking, Wi-Fi-based tracking also sends available tracking result to the Fusion Service for the fusion process using a Filter Data Packet. But unlike vision-based tracking, the results of Wi-Fi-based tracking do not contain any information about rotation or orientation of the Android-based device with respect to the origin. For every new result of Wi-Fi-based tracking, an entry of the tracking device is also updated in the registry and the history at the Discovery Service.

3.3.3 Bluetooth-based tracking

Android platform offers support for the Bluetooth network stack which allows the Bluetooth-enabled devices to scan, connect and share information with other Bluetooth-enabled devices, wirelessly [30]. The operating range of Bluetooth is 1 meter to 100 meters depending upon the class of radio used [31].

Following is the algorithm used for Bluetooth-based tracking

```
Start
initiate Bluetooth adapter
scan the environment for Bluetooth enabled devices
register a BroadcastReceiver with the scan_finished intent
create a list of Bluetooth enabled devices
for (each device in the list){
        if(a tracking device found){
                assign its position by using local graph at EIMS
        }
        else{
                remove the device from the list
                continue to the next iteration of for loop
        }
        assign the RSS value read from the scan to the device
}
if(list size >= 3){
        sort the list based on RSS in descending order
        position = trilateration (Bluetooth devices' list)
        create the Filter Data Packet and Discovery Object
        send the Filter Data Packet to the Fusion Service
        send the Discovery Object to the Discovery Service
}
End;
```

Similar to *Wi-FiManager* used in Wi-Fi-based tracking, we use ‘*BluetoothAdapter*’ which represents the Bluetooth radio on the local Android-based device. Using the Android API, we can programmatically start scanning the environment in order to discover Bluetooth enabled devices. Since the discovery of remote Bluetooth-enabled devices demands a lot of resources, we abort any discovery operation that may be in progress and restart it [30]. We also register a broadcast receiver with the application environment which is invoked whenever the results of Bluetooth discovery are available. For every discovered device, whether it is paired or unpaired, the results of the above mentioned scan provide a '*BluetoothDevice'* object which contains information about the remote object such as the remote device's user friendly name, its UUID which uniquely identifies an application's Bluetooth service, its hardware address and the RSS. If the discovered device is one of the known devices in the tracking environment, i.e., if the entry of a discovered device is found in the graph maintained by EIMS, then we assign the known position to this device. If the device is not a known tracking device, then we continue with the next discovered device. After assigning the positions to all the known devices in the list, we check if the size of the list is equal to or greater than three, since three devices are necessary for the trilateration. If it is, then based on the RSS, we sort the list of Bluetooth enabled devices in descending order. We then invoke the trilateration algorithm to estimate the position of the device on which Bluetooth tracking service is running with respect to the devices that the Bluetooth scan has discovered. The position estimate obtained as a result is with respect to the origin of the world. This position is then communicated, using Filter Data Packet, to the Fusion Service and the availability of the Bluetooth tracking service is communicated, using Discovery Object, to the Discovery Service. If the Bluetooth scan does not discover three Bluetooth enabled known tracking devices, the Bluetooth tracking service returns the control to the last task that was executing before the broadcast receiver was invoked.

3.3.4 Tracking using inertial sensors

An Android-based device typically contains three broad categories of sensors viz., Motion Sensors (e.g., accelerometer and gyroscope), Environmental Sensors (e.g., barometer and photometer) and Position Sensors (e.g., orientation sensor and magnetometer) [32]. In addition to these hardware sensors, the Android-based sensor framework provides an access to software sensors which encapsulate these hardware sensors. Out of the available sensors, we use the accelerometer sensor since we intend to track only the position of the Android-based device and not the orientation. Additional sensors such as Gyroscope and Magnetometer can be introduced to further improve the position estimation by adding the orientation information.

Accelerometers use the standard coordinate system as indicated in Figure 3.5.

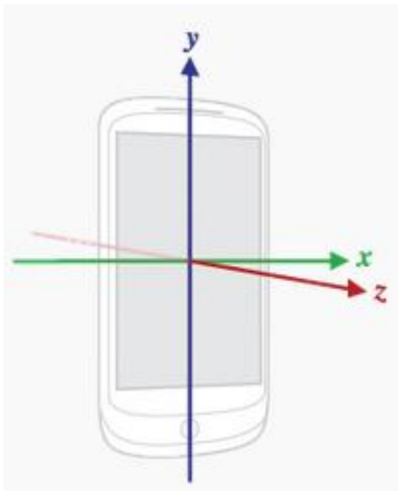


Figure 3.5 Android-based motion sensors' coordinate system [32]

An acceleration sensor measures the acceleration (the rate of change of velocity) of the device including earth's gravitational acceleration. Thus, a standstill device lying flat on the table with the screen facing upwards, shows the acceleration of +9.81 meter/second^2 which corresponds to the acceleration of the device (0 meter/second^2 minus the gravitational acceleration, which is -9.81 meter/second^2) [33].

Following algorithm for inertial sensor-based tracking shows how e-DTS 3.0 uses low-pass filter, high-pass filter and double-integrator module to get the tracking data from device acceleration.

```
Start
initiate SensorManager
get an instance of accelerometer
register a listener for sensor
when sensor listener event received{
        isolate the gravity using a low-pass filter
        remove the gravity using a high-pass filter
        double integrate the sensor reading without gravity
}
End;
```

SensorManager provides access to the sensors (hardware or software) embedded in the Android-based device. Using *SensorManager* we initiate the accelerometer in e-DTS 3.0. We then register an event listener for the sensor similar to a broadcast receiver that notifies when the sensor value has changed. A change in accelerometer reading indicates that Android-based device has moved. When the event listener presents the accelerometer reading, we further process it in order to retrieve the value of displacement.

As indicated in Android documentation for motion sensors [33], we use a low-pass filter to isolate the effect of gravity, and then a high-pass filter to remove the contribution of gravity. After we get the acceleration without the effect of gravity, we integrate the acceleration data twice with respect to time to get the displacement of the device.

Since the effect of gravity cannot be completely removed from the accelerometer reading, the results cannot be guaranteed to be correct. Also even if there is a small error in the accelerometer data, double integration amplifies the error. Hence, a study regarding better gravity filter and a better way to derive the displacement from the acceleration are required as a part of the future work.

3.3.5 Communication between tracking services and Fusion and Discovery Services

A tracking service is responsible for acquiring tracking results from the sensors and communicating them to the Fusion Service. Tracking services also communicate their availability to the Discovery Service. Following are the details of the communication between tracking services and Fusion and Discovery services.

As indicated in the algorithms discussed in sections 3.3.1 to 3.3.3, whenever a tracking service running on an Android-based device gets the position estimate of a pattern from the sensor it is encapsulating, it creates a packet to be sent to the Fusion Service, called Filter Data Packet. A tracking service inserts relevant data in Filter Data Packet such as, the pattern ID, position of the tracking service, sender's and receiver's IP addresses, the unique ID assigned to the Android-based device, the world to which the device belongs and the timestamp along with the tracking result i.e. a 4 x 4 transformation matrix. Following is the detailed discussion about these fields in a packet routed to the Fusion Service.

1. Android ID – Android ID is a unique ID assigned to an Android-based phone. ANDROID_ID is a 64-bit number (represented as a hex string) that is randomly generated on the device's first boot and should remain constant till a

factory reset is performed on the device [25]. In e-DTS 3.0, this Android ID is considered unique and helps an Android-based phone to identify its neighbors using their IDs and their physical location.

2. Pattern ID – A pattern ID is a unique ID or a name given to a pattern in order to identify and distinguish it from others. Distinguishing patterns from each other is particularly important for the Fusion Service, since fusing the results obtained by tracking two different patterns would be incorrect. Pattern IDs are maintained in the application's configuration file which is shared and maintained by the tracking services and Fusion and Discovery services. Apart from pattern IDs used for visual tracking (PATT_HIRO and PATT_KANJI), this file has patterns defined for Wi-Fi-based tracking (PATT_WIFI), Bluetooth-based tracking (PATT_BLUETOOTH) and for inertial sensor-based tracking (PATT_INERTIAL) which help to identify the Filter Data Packet generated from these respective sensors. There are two special patterns (PATT_INFO and PATT_DUMMY) that tracking services use while communicating with the Discovery Service. These patterns are discussed in the later part of this section
3. Transformation Matrix – Transformation matrix is the tracking result that is returned by the tracking algorithm. For example, the AndAR library used in visual tracking returns a 4 x 4 matrix. This transformation matrix is used in fusion process.
4. Timestamp – In order to address the issues of clock skews and inherent message delays in a distributed system, it is necessary to synchronize clocks

of all the devices and mark each packet in e-DTS 3.0 with a timestamp. The timestamp serves a dual purpose: a) it allows the receiver (e.g., the Fusion Service) to discard stale packets, and b) it assists in identifying relevant tracking results obtained from various sensors during the fusion process. In a real-time tracking system, a packet is declared as a stale packet if it was sent more than 30 milliseconds before the current time. This criterion of 30 milliseconds comes from the fact that in real-time tracking, the data is considered invalid if it is older than 30 milliseconds since, by then, the moving object or the pattern might have moved from its last tracked position, invalidating the last obtained results [20]. Since the tracking system is distributed, there is a need to synchronize the clocks of the sender (tracking service) and the receiver (filter). We use the experiments carried in [19] for the clock synchronization and use Simple Network Time Protocol client with "ntp.iupui.edu" time agent since this time agent is physically closest to the tracking environment which reduces the round-trip delay in the communication with the synchronizing time source.

5. World ID – Any large tracking environment may be divided into multiple worlds so that the devices belonging to a single world may be managed more easily. Also, position estimation of a pattern with respect to an origin far away may not be practical. Thus, each world has its own origin and the devices in a particular world estimate the position of a pattern with respect to the origin of that world. When a tracking service sends the tracking information to the

Fusion Service, it must also specify to which world the tracking service belongs. World IDs are maintained in the configuration file.

Tracking services also notify their availability to the Discovery Service. They communicate with the Discovery Service using a Discovery Object which has important data fields such as the Android ID, the current location of the device, associated timestamp, sender's IP address and the pattern ID. The Android ID and timestamp are already discussed in the context of Fusion Service earlier in this section and carry the same semantics for the Discovery Service. Hence, for the sake of brevity, we do not elaborate on these two fields again. Following is the detailed discussion of various fields in a Discovery Object routed to the Discovery Service.

1. Android ID – This field represents the unique ID of an Android-based device.
2. Timestamp – This field indicates the time at which the tracking results were captured. Depending upon the value of this timestamp, the Discovery Service removes the stale data (data that is older than a fixed amount of time) from the registry.
3. Current position of the tracking service – This is a 4 x 4 array representing the location of the tracking service. This field is particularly important when a tracking service estimates the position of a pattern with respect to the world's origin or tries to explore the tracking environment by identifying its neighbors depending upon the physical distance between the two tracking services.
4. Sender's IP Address – Sender's IP address registers the address of the tracking service or the EIMS (discussed in section 3.6) with the Discovery Service. If a tracking service wishes to communicate with its peer tracking service(s) in order

to exchange the tracking data and environment information, it may request the information about its peers from the Discovery Service and then communicate using this IP address.

5. Pattern ID – In addition to various pattern IDs representing markers used in various modalities discussed in the context of the Filter Data Packet earlier in this section, there are two special purpose pattern IDs used for communicating with the Discovery Service. PATT_INFO is used when a tracking service requests the history of the tracking environment maintained at the Discovery Service. Discovery Service provides this information to the tracking service by sending the history it maintains. This history is the list of most recent Discovery Objects (one per tracking service) it has received so far. The Discovery Service can also be designed to send the registry (which represents only active tracking services' information and not all tracking services' information) rather than history, but the goal is to share as much data as possible to make the e-DTS 3.0 opportunistic.

   PATT_DUMMY is another such special purpose pattern ID which is used by the Wi-Fi-based tracking service to notify the Discovery Service that this tracking service is going to activate a hotspot; the Discovery Service sends ACK/NAK response to the Wi-Fi-based tracking service notifying its permission for activating a hotspot. If the Discovery Service receives a Discovery Object with a pattern ID other than the two discussed above, it simply adds the data to the registry and to the history so that it can be available to any tracking service that requests it.

3.4 Fusion Service

Since the tracking system we developed contains multiple Android-based devices hosting multiple tracking services, there is a need to merge results of a specific tracking activity after considering all valid results from Android-based devices. The Fusion Service or AFilter (analogous to the Visibility Filter in [19]) is responsible for this merging task.

In a multi-sensor tracking system, such as e-DTS 3.0, the tracking data generated by heterogeneous sensors may be in different formats. For example, as indicated earlier, vision-based tracking gives a 4 x 4 transformation matrix which represents the rotation as well as the translation of the pattern with respect to the camera or the origin, whereas Wi-Fi tracking gives the position of the marker (the Android-based device itself) in terms of X and Y coordinates. Moreover, each type of sensor has inherent QoS that we need to be able to differentiate between. Thus, the Fusion Service should decide how to handle the tracking data from different sensor modalities. As outlined in [26], there are three approaches which the Fusion Service may follow. The first approach is to send all the data to the Fusion Service and let it decide the weights to be used for the data from different sensors. These weights can be based on heuristics or on the sensor ranking. The second approach is to fuse the data of each sensor modality separately, thus generating a single representative result for each sensor type. The third approach is to use a proxy service that filters out the data that does not meet the QoS criteria set by the user and forwards the qualifying data to the Fusion Service. The Fusion Service then fuses the data together without any prior knowledge about its origin. In e-DTS 3.0, we have chosen the second option where the Fusion Service fuses the data originated from the same sensor modality. The reason for choosing this option is that each of the sensors used in e-DTS 3.0 has a different accuracy. For example, the accuracy of vision-based tracking is far better than Wi-Fi-based tracking. If we fuse the data from the vision-based sensor i.e. the camera and the Wi-Fi sensor, then the accuracy of Wi-Fi sensor may worsen the overall accuracy of the fused reading. Another important reason behind choosing this fusion technique is that different sensors in e-DTS 3.0 have different response times. For example, since the Bluetooth-based tracking scans the tracking environment for Bluetooth enabled devices, it may take up to twelve seconds to present the result of the tracking [30]. Fusing the result of vision-based tracking with Bluetooth-tracking will worsen the overall response time since the fusion process will be suspended until the Bluetooth-based tracking results are available. The Pattern ID field in the Filter Data Packet helps the Fusion Service to distinguish tracking data generated from different sensors. Also, if a particular type of sensor can see two different patterns, then we fuse the data related to the same pattern. For example, an Android-based camera may see two different patterns (PATT_HIRO and PATT_KANJI) at the same time. In such a case, the result will be the position estimates of two different patterns and we do not fuse the data together.

There are two types of fusion techniques that can be carried out with e-DTS 3.0 viz., Averaging Fusion and Kalman Filter-based fusion.

3.4.1 Averaging Fusion

Averaging Fusion is a simple technique where results available from all tracking devices are averaged. This average is simple unbiased average of all the results. We could, alternately, implement a weighted average or average after excluding the outlying results. However we have borrowed the Averaging technique implementation from e-DTS 2.0 and implemented just a simple averaging technique.

3.4.2 Kalman Filter

"The Kalman Filter is a set of mathematical equations that provides an efficient computational (recursive) means to estimate the state of a process, in a way that minimizes the mean of the squared error" [34]. A Kalman Filter can be used with the systems which are linear and their noise or the error characteristics follow a Gaussian distribution. Kalman Filter cycles in two steps viz. prediction and updating. In the prediction state, it predicts the current state of the system along with the uncertainties of the state variables. Once the results of the measurement are available, it corrects or updates the state of the variables using weighted average. If the uncertainty of a state variable is more, it gets lower weight during the correction step.

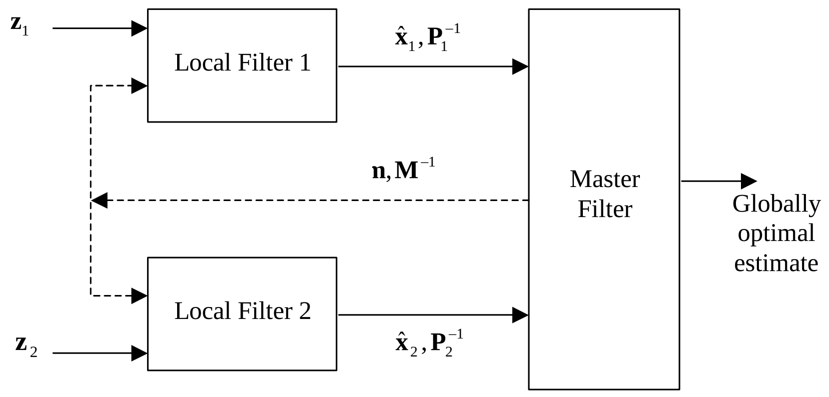

$\mathbf{z}_1$
Local Filter 1
$\hat{\mathbf{x}}_1, \mathbf{P}_1^{-1}$
$\mathbf{n}, \mathbf{M}^{-1}$
Master Filter
Globally optimal estimate
Local Filter 2
$\hat{\mathbf{x}}_2, \mathbf{P}_2^{-1}$
$\mathbf{z}_2$

Figure 3.6 Framework for the federated Kalman Filter [35]

The Kalman Filter used in e-DTS 3.0 is based on the federated Kalman Filter method that was proposed by Neal Carlson in 1990 [35] shown in Fig. 3.6. The transformation matrices received from the tracking services are forwarded to the Kalman Filter which gives the accurately fused result that represents the location of the object being tracked.

3.5 Discovery Service

The discovery service imitates the Jini-based discovery service described in [19]. It maintains a registry of all active tracking services as well as the environment history to provide opportunistic data to tracking services. The Discovery Service runs on a desktop machine (same as or different from the machine on which Fusion Service runs) at an IP address declared in the configuration file and hence can be termed as a Semi-automatic Discovery Service. Following is the algorithm of the Discovery Service.

Main Thread

```
Start
Create the 'registry'
Create the 'history'
boolean isHotspotActive = false
start a thread for refreshing registry
start a thread for handling clients' requests
End;
```

<u>*Refresh Registry Thread*</u>

```
        Start
        while (true){
          for(each DiscoveryObject in the registry){
                    if (DiscoveryObject.timestamp is older than refresh interval{
                              delete DiscoveryObject from the registry
                    }
          }
}
End;
```

<u>*Client Request Handler Thread*</u>

```
        Start
        while (true){
          accept a new connection from the tracking device
          read the DiscoveryObject sent
          if(DiscoveryObject.PatternId = PATT_INFO){
                    send history to the tracking device
          }
          if(DiscoveryObject.PatternId = PATT_DUMMY){
                    if(isHotspotActive = false){
                              send 'ACK' to the tracking device
                              isHotspotActive = true
```

```
                }
                else if(isHotspotActive = true){
                        send NAK
                }
        }
        else{
                add DiscoveryObject to the registry
                add DiscoveryObject to the history
        }
}
End;
```

Discovery Service has two threads – one for refreshing the registry in order to keep track of active tracking services and the other one for responding to clients' requests (such as updating the registry and the history and sending the environment information back to the client) depending upon the pattern ID set in the incoming pattern. The main thread starts these two threads and then they work in parallel.

Refresh Registry Thread keeps the registry of the tracking services current. It iterates in an infinite while loop and removes the entry of the tracking service that has not communicated with the Discovery Services in the past two seconds' period. This refresh interval of two seconds can be changed using the configuration file before the Discovery Service is started.

Client Request Handler Thread accepts new connections from the tracking services or the EIMS and reads the Discovery Object sent. As discussed in section 3.3.5, if the

Pattern ID in the Discovery Object is set as PATT_INFO, the Discovery Service sends the history to the tracking service. If Pattern ID is set as PATT_DUMMY, the Discovery Service responds ACK/NAK to the Wi-Fi-based tracking service. If it finds any other Pattern ID in the Discovery Object, it adds the entry of new tracking service to the registry and to the history. An entry in the registry or the history has a unique 'tracking service Android ID – pattern ID' combination. For example, if a vision-based tracking service hosted on Android-based device with Android ID 'ABCD1234' sees two patterns 'PATT_HIRO' and 'PATT_KANJI', there should be two entries 'ABCD1234PATT_HIRO' and 'ABCD1234PATT_KANJI' added to the registry and to the history. Similar to the Refresh Registry thread, Client Request Handler Thread iterates in an infinite while loop and keeps accepting connection requests and keeps responding to the requests accordingly.

3.6 Environment Information Maintenance Service (EIMS)

Every Android-based device locally maintains information about all the tracking devices and the patterns present in the tracking environment. The EIMS is responsible for requesting and maintaining the information about the tracking services hosted on other devices, such as the patterns they see and the transformation matrices, from the Discovery Service and the tracking services themselves. It also maintains local variables which indicate more information about the tracking environment such as whether the hotspot is active or not, whether e-DTS 2.0 camera service is running or not. The EIMS on a tracking device has a local graph in which a node represents another tracking device (Android-based or a desktop). A node has attributes such as Android ID, its IP Address, its position from the origin, a list of pattern IDs that this device can detect and a Boolean variable indicating whether it qualifies to be a neighbor of the device on which EIMS is running. The qualifying criteria for a device to be a neighbor is maintained in the configuration file. An example of such a criterion is if the distance between two devices is less than or equal to five meters. Following is the algorithm of the EIMS.

```
Start
boolean isHotspotActive = false
boolean isEdotsRunning = false
initialize localGraph
while (true){
  newDiscoveryObject.setPatternID(PATT_INFO)
  send newDiscoveryObject to the DS
  receive tracking history from the DS
  for(each DiscoveryObject in the history)
  {
          if(DiscoveryObject.patternId = PATT_DUMMY{
                  isHotspotActive = true
          }
          else if(DiscoveryObject.AndroidId == eDOTS_ID{
                  isEdotsRunning = true
          }
          if(localGraph contains DiscoveryObject){
                  update the patterns seen for the DiscoveryObject in localGraph
```

```
                }
                else{
                        add a node for the DiscoveryObject in localGraph
                }
                decide if the DiscoveryObject is neighbor
                communicate with the neighbor to get the info about patterns it sees
        }
        sleep for 2000 ms
}
End;
```

EIMS requests the history information to the Discovery Service using the Discovery Object discussed in section 3.3.5. It sets the pattern ID field to PATT_INFO to request the history from the Discovery Service. EIMS receives this information through a Hash Map of Discovery Objects which represents the most recent information from the tracking services that may be active or were active in the past. For every Discovery Object received through the history, if the EIMS sees an entry with pattern ID set as PATT_DUMMY, it sets a local Boolean variable which indicates that there is a hotspot activated in the system and therefore the Wi-Fi-based tracking service on this device should not attempt to activate its hotspot features. This scenario is also explained in section 3.3.2. EIMS then checks whether the Discovery Object carries the information from e-DTS 2.0 camera service by checking the Android ID field. If it does, then the EIMS sets a Boolean variable that indicates e-DTS 2.0 is also running in the environment. Now, EIMS checks if there is an entry of the device on which the tracking service is running in the local graph maintained by EIMS. This can be easily achieved since the Discovery Object contains the Android ID for the device on which tracking service is running, which can be checked with the Android ID of a graph node. If the graph already contains an entry for this tracking device, EIMS simply updates the patterns list associated with the tracking device. If not, then the EIMS adds a node for the tracking device to the local graph. The EIMS then calculates the distance between itself and the tracking device represented by the Discovery Object using the standard distance formula as:

$$distance = \sqrt{x^2 + y^2 + z^2}$$

where x, y and z are the distances between two devices on respective axes. The criterion used for deciding whether a given device can be considered a neighbor is that its distance from the device running EIMS is less than or equal to five meters (or 5000 millimeters). As mentioned before, this criterion is maintained in the configuration file. Now, if a device satisfies the criteria for the neighbor, the Boolean variable associated with the graph node is set. The EIMS contacts the neighbor in order to get the position estimates of the pattern the neighbor sees. EIMS thus enables an Android-based tracking device to present the position estimates of the patterns it cannot directly detect, through its neighbor and also adds some intelligence to e-DTS 3.0 to make it opportunistic.

This chapter has explained the functionality of various building blocks of e-DTS 3.0 and the rationale behind their development. It shows the how some of the enhancements proposed to e-DTS 2.0 [19] and eDOTS [26] are designed and implemented in e-DTS 3.0. In the next chapter we present the results obtained from the experiments carried out using a prototype of e-DTS 3.0.

CHAPTER 4. EXPERIMENTATION AND VALIDATION

In this chapter, we describe different categories of experiments carried out using a prototype of e-DTS 3.0. The experimental setup of the prototype consists of three Android-based phones with the following specifications:

1. HTC Google Nexus One running on Android 2.1 (Éclair) and later updated to Android 2.3 (Gingerbread) [15].
2. Samsung Galaxy Nexus running on Android 4.0 (Ice-cream Sandwich) later updated to Android 4.2.2 (Jelly Bean) [16].
3. LG Vortex running on Android 2.2.2 (Frozen Yogurt) [24].

Each of these Android-based devices has a pin-hole model-based camera (of varying pixel sizes), Wi-Fi and Bluetooth capability and inertial sensors such as accelerometer, gyroscope and compass. We use eclipse IDE and Android Software Development Kit (SDK) [42] for the Android application development and debugging. We run the Fusion Service and the Discovery Service on a Dell OptiPlex machine running Windows XP SP 3 with Pentium 4 CPU connected on a 10 Mbps local area network connection. Also, if e-DTS 2.0 camera service is running in the environment, it can be hosted on the same or a different machine as Fusion and Discovery Services. The communication between Tracking Service(s) and the Fusion and Discovery services is achieved using TCP/IP sockets.

We tested the prototype of e-DTS 3.0 in the laboratory setup inside one of the buildings (SL building) at the Indiana University-Purdue University Indianapolis (IUPUI) campus.

4.1 Experiments to test tracking modalities and fusion

In this set of experiments, we repeated each experiment three times and the results obtained were averaged and are discussed below. As e-DTS 3.0 is an enhancement of e-DTS 2.0 and eDOTS [26], in the following sections, the results obtained with a prototype of e-DTS 3.0 are compared to those obtained from e-DTS 2.0 [19] and eDOTS [26] to the extent feasible.

4.1.1 To empirically determine the accuracy of vision-based tracking using Averaging Fusion

To validate the accuracy of vision based tracking in e-DTS 3.0, we first tracked a pattern using the camera on an Android-based mobile device. We also measured the physical distance (on X, Y and Z axes) between the origin and the pattern using a measuring tape which we call as the Ground Truth. To calculate the error using single Android-based device, we subtracted the measured value of the ground truth from the reading obtained from e-DTS 3.0 and considered the absolute value of the error. We later introduced another camera (on another Android-based device) to track the same pattern and fed the results of the two tracking devices to the Fusion Service. We used Averaging Fusion for this experiment. To calculate the error in the fused result, we now subtracted the measured value of the ground truth from fused result and considered the absolute value of the error. We calculated the error on each of the axes and compared it with the error obtained from Averaging Fusion used in e-DTS 2.0 [19].

Table 4.1 summarizes the results obtained from this experiment.

Table 4.1 Error in vision-based tracking using Averaging Fusion

Error (mm)	Single Android Device	Fused Result using e-DTS 3.0	e-DTS 2.0 [19]
X axis	41	31.75	59.98
Y axis	72	69.66	316.44
Z axis	120	104.63	114.83

From the table we conclude that the accuracy of Averaging Fusion obtained by averaging results from two Android-based devices running visual tracking is better than that of e-DTS 2.0 [19]. This is an expected outcome since the resolution of the video stream obtained from the lowest quality Android-based camera, out of the three available devices, is much better than the web-camera [24] [43]. Also, Averaging Fusion in general gives better results than the results obtained from a single device. This is because the fusion process averages the X, Y and Z position estimates and thereby averages the error on the three axes.

We executed the experiment explained above in two different environments with different intensities of light. One of the tracking environments was brightly lit using the lights on the ceiling. The other environment was moderately lit with day-light. Comparing the results of the experiments executed in the two environments, it was observed that the accuracy of tracking was better with the naturally and moderately lit environment. We conclude that the tracking accuracy may change depending upon the lighting conditions of the tracking environment. If the light is too bright and gets reflected from shiny surfaces such as metallic objects, the overall error may increase.

Table 4.2 Response time of tracking with and without Averaging Fusion

	Without fusion	With Averaging Fusion
Response time (ms)	232	316.8

As shown in Table 4.2, the response time required for the vision-based tracking with Averaging Fusion is more than that without fusion. This is due to the fact that the fusion process introduces additional delay before presenting the results. The tracking is considered real-time tracking if the tracking process, right from pattern detection to obtaining fused result, can be achieved in 30 milliseconds [20]. e-DTS 2.0 achieves real-time tracking but e-DTS 3.0 fails to do so. The reasons behind more response time required by e-DTS 3.0 compared to [19] are-

1. Communication in e-DTS 3.0 is using TCP/IP sockets and the data travels over the Wide Area Network which can increase the number of hops required to reach the destination.
2. Although the Android-based device and the desktop machine(s) synchronize their clocks from the same time source (that was used in [19]), we cannot set the system clock on the Android-based device programmatically. Thus, if the communication with the time source for clock synchronization fails while marking the timestamp on the data packet originating from an Android-based device, the synchronization cannot be guaranteed between desktop machines

and Android-based device(s). This may introduce the error on the time recorded at the start of the pattern detection or at the end of the fusion process.

We conclude from the above results that Averaging Fusion, used in the prototype of e-DTS 3.0, gives better accuracy than the e-DTS 2.0 at the cost of increased response time.

4.1.2 To empirically determine the accuracy of vision-based tracking using Kalman Fusion

Similar to the experiment mentioned in 4.1.1, we carried out tracking using a single camera on an Android-based device and then introduced another camera on another Android-based device. Also, the ground truth was established by measuring the actual physical distance between the origin and the pattern. However in this experiment, we used Kalman Filter technique for the fusion process and compared the results with results from e-DTS 2.0 [19] also running Kalman Filter technique.

Table 4.3 summarizes the results obtained from the experiment.

Table 4.3 Error in vision-based tracking using Kalman Fusion

Error (mm)	Single Android Device	Fused Result using e-DTS 3.0	e-DTS 2.0 [19]
X axis	41	27.47	16.11
Y axis	72	35.91	150.35
Z axis	120	25.05	99.92

From Table 4.3, we observed that the X error from e-DTS 3.0 is 11.47 mm greater than the error from e-DTS 2.0; we note that the error on the Y and Z axes is significantly greater in e-DTS 2.0, 73.95 mm on the Y axis and 114.09 mm on the Z axis.

From the results of this experiment, we concluded that Kalman Fusion exhibits better accuracy compared to that of Averaging Fusion as seen in Table 4.1. The improvement in the accuracy over e-DTS 2.0 [19] can again be attributed to the better quality of Android-based camera and the improvement in the accuracy over Averaging technique is because of the recursive nature of Kalman Fusion technique which tries to reduce the error with the 'Predict-Update' cycle.

Table 4.4 Response time of the tracking with and without Kalman Fusion

	Without fusion	With Kalman Fusion
Response time (ms)	244	508

As evident from table 4.4, response time increases with Kalman Fusion compared to the response time observed to obtain results without fusion. Also, if we compare the response time of Kalman Fusion with that of Averaging Fusion, we can see that Kalman Fusion takes more time compared to the Averaging Fusion. The reason behind this observation is that Kalman Fusion is more sophisticated an algorithm compared to the Averaging algorithm and it iterates multiple times (generally equal to the number of results to be fused) for the fusion activity.

Thus, we can conclude from above results that Kalman Fusion technique gives better accuracy over Averaging Fusion and e-DTS 2.0 at the cost of increased response time.

4.1.3 To empirically determine the accuracy of Wi-Fi-based tracking

To observe the accuracy of the Wi-Fi-based tracking, we ran the Wi-Fi trilateration algorithm on a single Android-based device. The Wi-Fi-based tracking estimates the position of a device on which the Wi-Fi tracking algorithm is running. Also, in this scenario, the Fusion Service cannot fuse the results obtained from two trackers since each of them is the estimation of the position of the device itself.

The result obtained from Wi-Fi-based tracking is summarized in Table 4.5.

Table 4.5 Error in Wi-Fi-based tracking

Error (mm)	e-DTS 3.0	eDOTS [26]
X axis	2416.7	2700
Y axis	1550	2760.1

One significant disadvantage of Wi-Fi-based tracking is that it can track and estimate the position of the devices only on their latitude and longitude and not in the third dimension of the physical space (the Z axis).

Wi-Fi trilateration technique used in Wi-Fi based tracking uses Wi-Fi RSS, which varies greatly by the interferences such as metallic objects, walls, doors and microwave ovens. Hence, it cannot be the perfect measure of the distance between a WAP and a Wi-Fi enabled device. On the other hand, the RSS value may not change as significantly as the physical distance changes. Because of these reasons, accuracy of Wi-Fi-based tracking is not promising compared to the vision-based tracking results. It has been mentioned in [26] that the Wi-Fi accuracy should be between 1 and 3 meters; the Wi-Fi tracking results that we obtained from our experiments are within the benchmark mentioned above. We got better accuracy using e-DTS 3.0 compared to eDOTS [26]. This was contrary to our expectation since the Wi-Fi capability of a laptop is typically assumed to be better than that of a mobile phone due to greater possible antenna size and more battery power. Our hypothesis behind this observation is that the improved tracking accuracy stems from the usage of the API provided by the Android OS to scan the environment and pick up APs. On the other hand, eDOTS [26] uses a third party library called 'Placelab' to pick up three APs. This library essentially places calls in the order: Java then C++ and then Native OS – which adds the overhead associated.

Table 4.6 shows the response time required for Wi-Fi-based tracking using e-DTS 3.0.

Table 4.6 Response time for Wi-Fi-based tracking using e-DTS 3.0

	Wi-Fi-based tracking
Response time (ms)	712

In Wi-Fi trilateration, the Android-based device first initiates a scan to pick up the RSS values and the IDs of the WAPs it can detect. This scan introduces a significant delay before the tracking results are available. Hence, the response time involved in Wi-Fi-based tracking is more compared to the vision-based tracking (Table 4.2 and Table 4.4)

4.1.4 Wi-Fi and vision-based tracking

As explained in Section 3.4, the Fusion Service in e-DTS 3.0 fuses the data together if it originates from the same sensor modality and represents the position estimate of the same pattern. However, to study the error characteristics when the readings from multiple sensors modalities are combined, we relaxed the criterion of fusing the results based on the Pattern ID. In a situation where a pattern is fixed on a mobile device and the mobile device, through Wi-Fi-based tracking, estimates its own position, we can fuse the tracking data obtained from vision and Wi-Fi-based tracking, since both the modalities estimate the position of the same object of interest.

Tables 4.7, 4.8, 4.9 and 4.10 summarize the accuracy and response time observed while fusing the data from vision-based tracking and Wi-Fi-based tracking using averaging and the Kalman Fusion technique.

Table 4.7 Error in Wi-Fi and vision-based tracking with Averaging Fusion

Error (mm)	Wi-Fi only	Vision-based only	Averaging Fusion (Wi-Fi and vision-based)
X axis	82	33.3	37.8
Y axis	840	22.5	439.2
Z axis	2457	382.9	608.7

Table 4.8 Response time for Wi-Fi and vision-based tracking with Averaging Fusion

	Wi-Fi	Vision
Without fusion (ms)	551	363
With Averaging Fusion (ms)	603	

Table 4.9 Error in Wi-Fi and vision-based tracking with Kalman Fusion

Error (mm)	Wi-Fi only	Vision based only	Kalman Fusion (Wi-Fi and vision based)
X axis	82	39.1	4.4
Y axis	840	6.5	438.7
Z axis	2457.3	21.5	543.6

Table 4.10 Response time for Wi-Fi and vision-based tracking with Kalman Fusion

Response time (ms)	Wi-Fi	vision
Without fusion	469	375
With Kalman Fusion	1557	

Above result shows that both the fusion processes used cause the error to decrease when Wi-Fi-based tracking and visual tracking are used together. The error largely present due to the Wi-Fi-based tracking is balanced at the cost of accuracy of vision-based tracking and the overall response time. The response time increases when compared to the response time required for only vision-based tracking or the Wi-Fi-based tracking. This is because of the additional time invested in the fusion process.

4.1.5 To assess the use of Bluetooth-based tracking

We did a preliminary experiment of the position estimation using Bluetooth-based tracking. We used the trilateration algorithm similar to the one used in Wi-Fi-based tracking. The relationship between the Wi-Fi RSS and the distance between the AP and the Wi-Fi radio has previously been used in works such as [26]. Establishing such a relationship between Bluetooth RSS and the distance is important since the distance calculated using this relationship decides the radius of the coverage area of the three Bluetooth enabled devices. In order to establish this relationship between a Bluetooth radio and a receiver, we measured the Bluetooth RSS and actual distance in two different environments in the presence of multiple interference sources such as Wi-Fi routers and other Bluetooth enabled devices. The reason behind introducing interference sources was to emulate the actual tracking environment. We used IGOR Pro data analysis software [41] to establish the relationship between the Bluetooth RSSI and the actual physical distance using curve fitting. The reasons behind choosing this particular software were its simplicity and free availability.

Based on this result set and using IGOR Pro data analysis software, we formulated the relationship between the actual distance and RSS that can be given as:

$$\text{Distance} = Y_0 + A * e^{(-\text{invTau}.x)}$$

where $Y_0 = 21.644$, $A = 14.36$, $invTau = 0.029696$ and x = Bluetooth RSS. If in a tracking environment, an Android-based device can detect three other Bluetooth enabled devices whose positions are known (or the Android-based tracking devices), then using the trilateration algorithm similar to the one used in Wi-Fi-based tracking, the position of the Android-based device can be estimated. Table 4.11 shows the results obtained from the position estimation using Bluetooth trilateration

Table 4.11 Error in Bluetooth-based tracking

Error (mm)	e-DTS 3.0
X axis	1100
Y axis	830

Although Bluetooth-based tracking does not show a promising accuracy compared to vision-based tracking, it provides an additional modality of tracking which improves the availability of the tracking result when there are no markers for vision-based tracking and APs to run Wi-Fi-based tracking. Similar to Wi-Fi-based tracking, Bluetooth-based tracking can estimate the position only on X and Y axes.

The response time required for Bluetooth-based tracking is approximately 12 seconds. This is because the Bluetooth scan may take up to 12 seconds to finish and present the results of the scan [30]. Thus the response time of Bluetooth-based tracking is very high compared to vision-based tracking and Wi-Fi-based tracking.

4.2 Experiments to evaluate different tracking configurations

4.2.1 Configuration 1: Identifying neighbors

In order to make the tracking system opportunistic, Android-based devices should not only take advantage of the sensors available but also use any other information that can aid the tracking process. Using the most recently known positions of all Android-based devices in the tracking, the EIMS running on a device marks its neighbors based on the physical distance between the two devices. This activity of identifying neighbors is carried out every time the EIMS requests history information from the Discovery Service. Also, since the Android-based device that is identifying its neighbors may be moving through the tracking environment, the topology of the system (in terms of a device and its neighboring devices) may change frequently depending upon the movement of the device running this algorithm as well as upon the movement of other devices in the tracking environment.

For this experiment, we assumed that the Android-based devices are the neighbors of each other if the physical distance between them is five meters (5000 millimeters). Apart from the criterion used in this work, the decision to mark an Android-based device as a neighbor can depend on a) the world to which an Android-based device belongs or b) the Wi-Fi or Bluetooth RSS from the potential neighbor. Once EIMS on the Android-based device identifies its neighbors, it initiates the communication with its neighbors by sending a handshake signal. In response to this signal, the neighboring device sends the size of the list of the tracking data that it is about to send back. The EIMS then accepts the tracking data sent by its neighbor which is a list of Filter Data Packets. The neighboring device mentioned above maintains the list of ten most recent Filter Data

Packets and shares those packets if a device requests so. This size of ten is purely based on heuristics and was designed considering the total number of patterns a device may detect from different modalities and to limit the network communication involved in neighbors' data exchange.

The time required for an Android-based device to identify its neighbors and to get the tracking information from those devices was 5793.75 milliseconds. This high latency is because of a) the time required for a device to request and get the data from the Discovery Service, b) calculate the physical distance using the distance formula mentioned above, c) initiate communication with the neighboring Android-based device and d) receive the list of tracking data. Although the response time is not very good, this information obtained from the neighbors may help a device to gain more knowledge about the tracking environment it is operating in. Even if a device cannot detect any pattern, based on its neighboring devices, it may present the location estimate of a pattern of interest to the client who is using the device for tracking.

4.2.2 Configuration 2: An Android-based device used as a hotspot

The capability of an Android-based smartphone to act as a Wi-Fi hotspot can be utilized when at least three APs that are necessary for Wi-Fi tracking are not available. In this situation, the Wi-Fi tracking service on the Android-based device first checks if there is any Android-based device acting as a hotspot, already active in the tracking environment by checking the local Boolean variable maintained by EIMS. If this variable is false, indicating that no hotspot is active in the system, it sends a Discovery Object to the Discovery Service which has the Pattern ID set to PATT_DUMMY and *myPosition* set according to the device’s position. After receiving the Discovery Object, the Discovery Service sends a positive or negative acknowledgement (ACK or NAK). The ACK response can be treated as a permission and promise from the Discovery Service that it will not allow any other tracking device to become a hotspot. The NAK response signifies that there is already a tracking device working as a hotspot, so the requesting device should not make an attempt to become another hotspot. If a device cannot pick up three APs during the Wi-Fi scan and receives ACK response from the Discovery Service, then it will become the hotspot and continue to work so, as long as it is present in the tracking environment. The Discovery Service instance will not allow any other device to become a hotspot as long as the same instance of the Discovery Service is in use i.e. if the user shuts the Discovery Service down and restarts it then it loses the previous state of the history information and then may allow another device to become a hotspot. The reason for strictly allowing only one hotspot in e-DTS 3.0 is that the accuracy of the results obtained through Wi-Fi-based tracking is not as good as visual tracking. An Android-based device acting as a hotspot cannot use its Wi-Fi connectivity. Hence, it will not be able to act as a tracking device anymore. If we activate multiple hotspots on multiple Android-based devices, we will lose those many tracking devices which may have provided more accurate results through visual tracking. If a device picks up only one AP during the Wi-Fi scan then activating hotspot capability (on the same device or on any other device) may not be effective. Nevertheless it proves helpful when a scan picks up only two APs. This scenario shows how entities in e-DTS 3.0 collaborate to overcome limitations of the infrastructure (absence of an AP, in this case) and take advantage of the opportunities to continue the tracking process.

4.2.3 Configuration 3: Coordinate system handoff

If a tracking environment is large enough, it may be divided into smaller spaces or “worlds”, so as to manage the tracking devices more efficiently and to make more sense from the tracking data they provide. For example, as a pattern moves away from its original world into a new world, it may be detected by the tracking service(s) in the new world. Such tracking services from the new world then give the position estimation of the pattern with respect to the origin of the new world the pattern now belongs to. This is important since position estimation of the pattern with respect to its original world may not be meaningful if the two worlds are far away from each other. For the sake of discussion, let us consider room A to be a world. Estimation of the position of a pattern with respect to room A may not make much sense if it moves to another room B. But the position estimation of a pattern with respect to room B may be more meaningful.

Nevertheless, in order to fuse the readings and compare the readings from different tracking services belonging to different worlds, these readings must be transformed to a common world with a known origin. Also, the handoff for the position estimation of a pattern moving across the worlds should happen seamlessly.

Such a scenario was simulated in this experiment where the tracking space (in our laboratory) was divided into two worlds with approximately equal dimensions and with known origins, as shown in Figure 4.1 below. Two Android-based devices knew their positions with respect to the origins of the worlds they belonged to. A pattern was slowly moved from World # 1 to World # 2 and then back to World # 1.

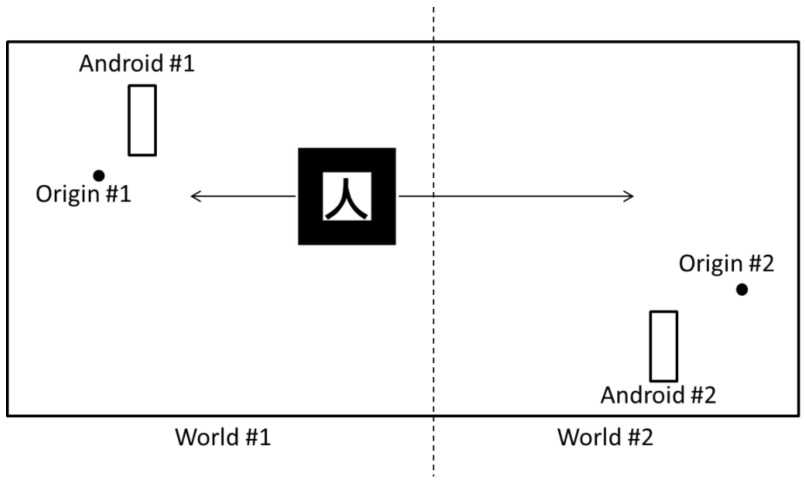


Figure 4.1 Experimental setup in lab (SL 116)

In this experiment, World # 1 was assumed to be the reference world i.e. the marker position estimated by the tracking service running on Android # 2 was converted to the position estimate with respect to the Origin #1. This conversion takes place at the Fusion Service just before it fused the results together. To achieve this transformation of results from World # 2 to World # 1, the Fusion Service applies the transformation matrix of Origin # 2 with respect to Origin # 1 which is the position (translation and rotation) of the coordinate system at Origin # 2 with respect to the coordinate system at Origin # 1. Suppose $T_{O1\text{-}O2}$ is the transformation (translation and the rotation) of coordinate system at Origin # 2 with respect to the coordinate system at Origin # 1. $T_{O2\text{-}P}$ is the position estimate of the pattern with respect to Origin # 2 as estimated by Android # 2 in World # 2. To calculate the position of the pattern in World # 2 with respect to Origin # 1, say $T_{O1\text{-}P}$, we use the transformation as

$$T_{O1\text{-}P} = T_{O1\text{-}O2} * T_{O2\text{-}P}$$

Following results shown in figures 4.2 and 4.3 pictorially represent the trace of the movement of a pattern between World # 1 and World # 2 based on the readings from the

Fusion Service. X and Y axes in the graph indicate the distance of the pattern on X and Z axes (in millimeters) respectively from Origin # 1.

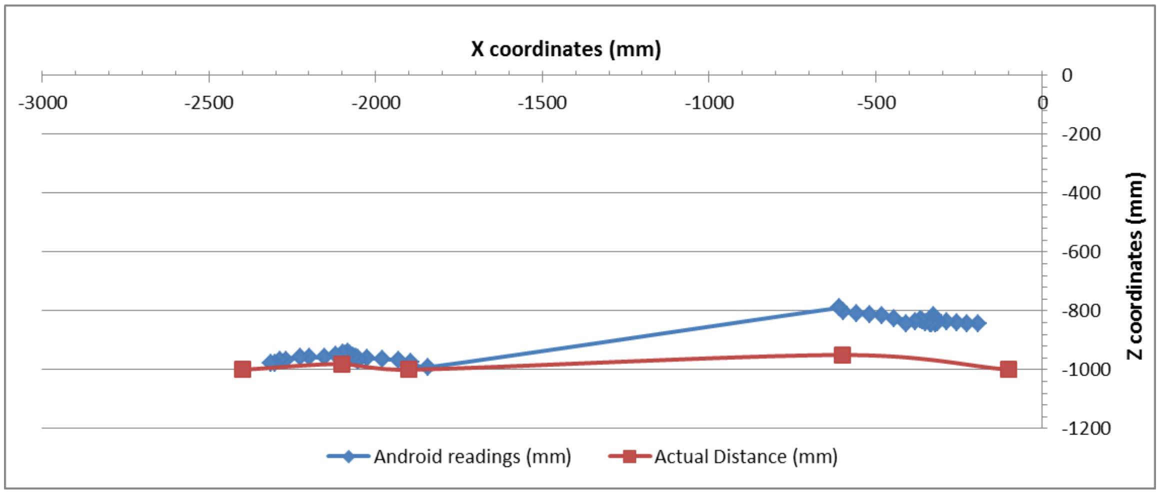


Figure 4.2 Coordinate system handoff from World 1 to World 2

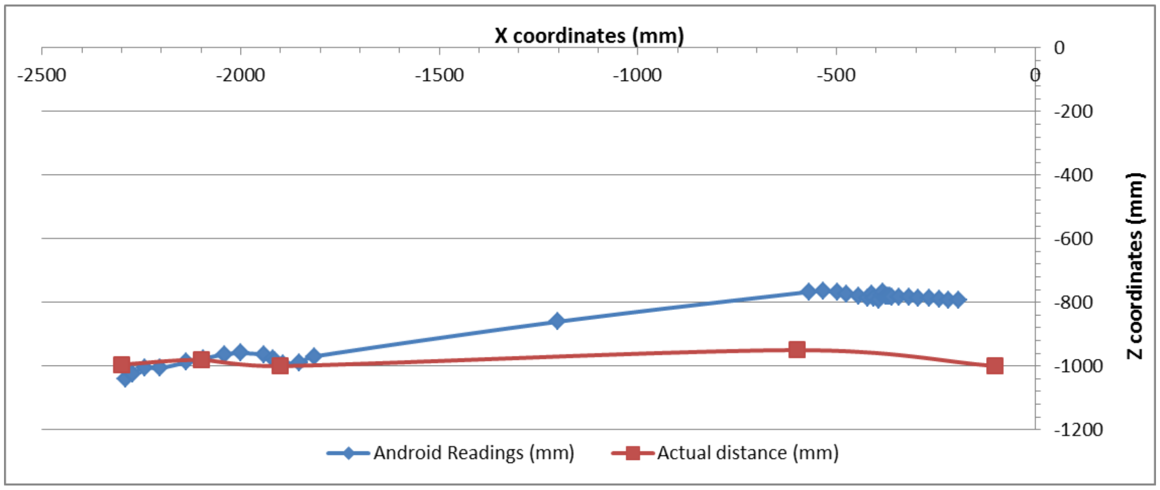


Figure 4.3 Coordinate system handoff from World 2 to World 1

As seen in figures 4.2 and 4.3, the pattern is visible to Android-based devices when it enters into the world in which Android-based devices are situated. Although Android # 2 estimates the position of the pattern with respect to the origin of its world i.e., Origin # 2, the Fusion Service transforms the results from World # 2 to World # 1 correctly. The slight disagreement between the actual pattern movement and the android readings in World # 2 (near Y axis of the graph) is because of the error introduced by the Android-based device in the tracking. The HTC Google Nexus smartphone was operating in World # 2 which does not exhibit the tracking accuracy as good as Samsung Galaxy Nexus smartphone. The area on the border of World # 1 and World # 2 was not in the view of either of the Android-based devices. Hence, there were no readings recorded when the pattern was moving through that area. Nevertheless, the coordinate handoff happened seamlessly since there was no sudden change or ‘jump’ in the readings because of the transformation applied.

4.2.4 Configuration 4: Using e-DTS 2.0 [19] and e-DTS 3.0 together

The e-DTS 3.0 can take advantage of the tracking data obtained from the e-DTS 2.0 [19] if it is active in the tracking environment. As mentioned in Section 3.3.1, in order to estimate the position of a pattern with respect to the world origin, an Android-based device needs to know the position of itself with respect to the world origin. In other words, we need to track the Android-based device and communicate its position to the device. For this, we attached a pattern used in vision-based tracking onto an Android-based device so that e-DTS 2.0 [19] can track the device using vision based tracking. e-DTS 2.0 [19], thus, acts as a tracking service in e-DTS 3.0. Whenever tracking data from e-DTS 2.0 [19] is available, it sends this data (the transformation matrix) to the Discovery Service and the Fusion Service in e-DTS 3.0.

Before applying the transformation to estimate the marker's position (as mentioned in Section 3.3.1) an Android-based device communicates with the Discovery Service to fetch the tracking data that may have been submitted by e-DTS 2.0. The Android-based device uses this position in the transformation to calculate the pattern's position with respect to its world origin.

Table 4.12 shows the results obtained from this experiment

Table 4.12 Error when using e-DTS 2.0 [19] and e-DTS 3.0 together

	X Axis	Y Axis	Z Axis
Error (mm)	590.9	343.8	3061.9

Since the tracking results are obtained from the composition of two tracking services (camera service in [19] and vision-based tracking in e-DTS 3.0), the error introduced by both the tracking services adds up. In order to reduce this error, we can calibrate the Android-based camera to find out its Quality of Service parameters similar to e-DTS 2.0 [19]. We can also use a vision-based tracking service from e-DTS 3.0 to replace the e-DTS 2.0 tracking service since the accuracy of Android-based visual tracking is observed to be better than the e-DTS 2.0 camera service (from 4.1.1 and 4.1.2).

This scenario is particularly useful if the tracking results are required to be estimated with respect to the world origin. It also shows that e-DTS 2.0 and e-DTS 3.0 can collaborate in the tracking process.

4.2.5 Study of scalability and fault tolerance

To study the scalability of e-DTS 3.0, we gradually increased the number of patterns that a prototype of e-DTS 3.0 was tracking. We started with a single pattern and ended the experiment with three patterns. Figure 4.4 shows that the response time increases as we increase number of patterns present in the tracking environment.

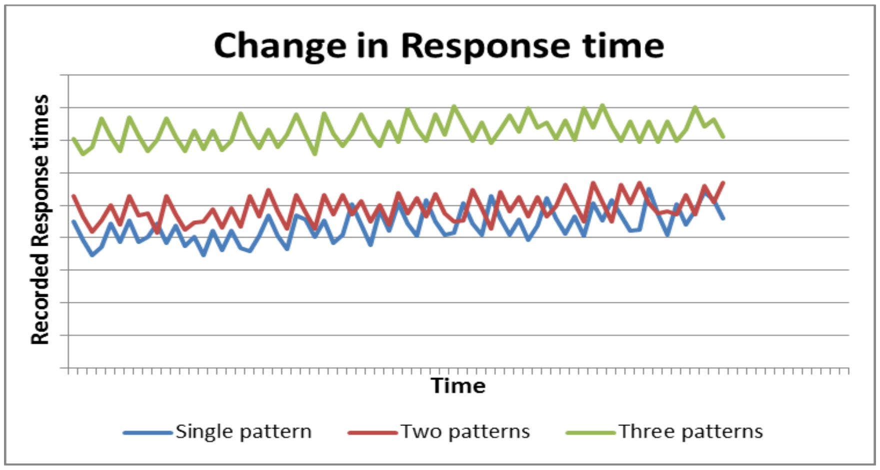


Figure 4.4 Scalability based on number of patterns

This observation is due to the fact that the vision-based tracking in e-DTS 3.0 tracks one pattern after the other. Hence, if more patterns are present in the tracking environment, overall response time required for the tracking will increase.

We also used a prototype of e-DTS 3.0 for tracking people and assets (such as chairs) in our laboratory (SL 116). Multiple patterns were affixed on backpacks, chairs and on the back of a volunteer (the author of the thesis) working in the room. This tracking experiment was run for five hours using two Android-based devices one after the other. The patterns were randomly moving in the room. Some patterns that were fixed on the chairs did not move at all since the chairs were stationary. These patterns were not detected since they were not in the viewing area of the tracking device. All the patterns that were in the viewing area of the Android-based devices' cameras were successfully detected and tracked. Figure 4.4 shows the loci of different patterns used in this tracking experiment. X and Y axes in the graphs show the positions of the patterns (in millimeters) on X and Z axes of the tracking environment. Depending upon the time for which a pattern was visible to the Android-based device, we recorded different number of readings for different patterns. In the graph that shows movement of Hiro.patt using Nexus phone, we recorded 627 readings over a period of five hours. For the graph showing the locus of AndAR.patt detected by Nexus phone, we recorded 1562 readings over a period of five hours. For the graph showing the movement of AndAR.patt detected by the Samsung phone, we recorded 23 readings over five hours and for the graph that shows the movement of Kanji.patt detected by the Samsung phone, we recorded 10 readings over five hours.

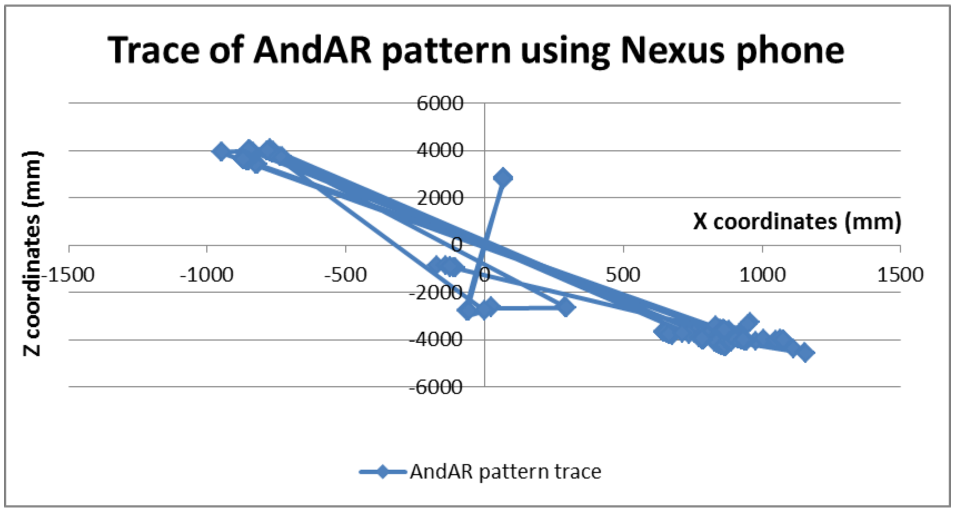


Figure 4.5 Loci of the patterns used in scalability experiment

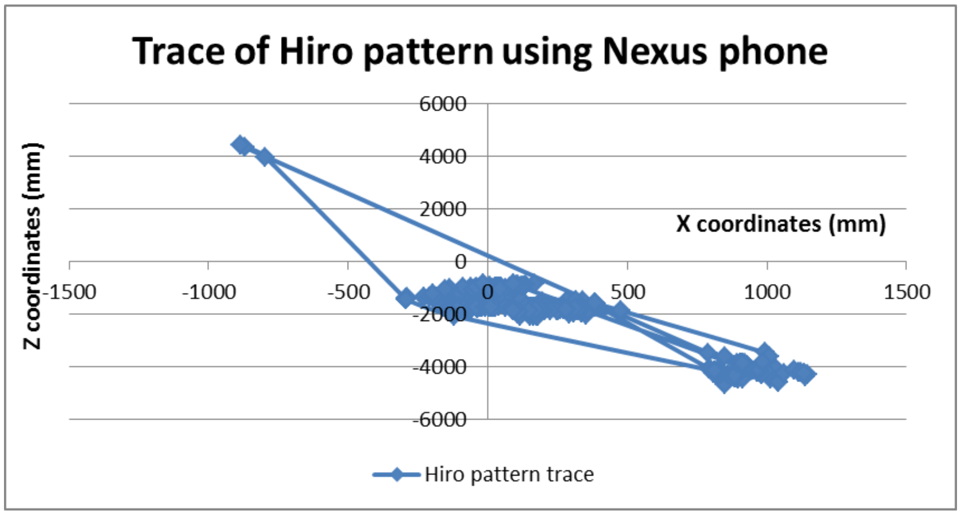


Figure 4.5 (continued) Loci of the patterns used in scalability experiment

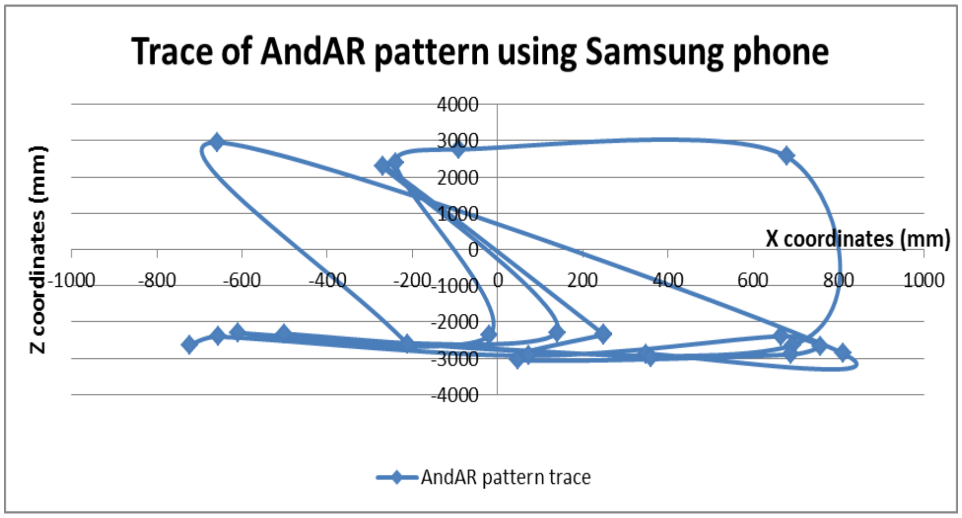


Figure 4.5 (continued) Loci of the patterns used in scalability experiment

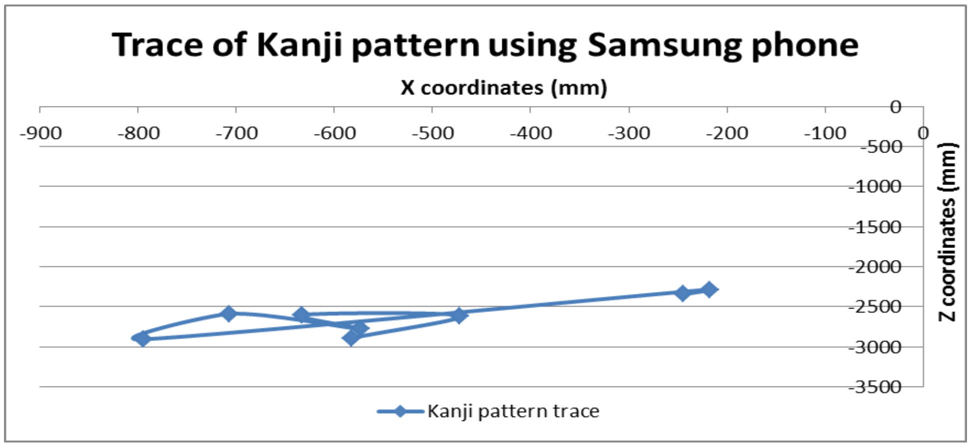


Figure 4.5 (continued) Loci of the patterns used in scalability experiment

To study the fault tolerance of e-DTS 3.0, we studied how each entity behaves in a faulty situation. Following are the points that we noted from the fault-tolerance study. To make Discovery Service and the Fusion Service fault tolerant, we ensured that the Discovery Service and Fusion Service can handle the exceptions arising from the socket communication. For example, even if a tracking service shuts off abruptly, Fusion and Discovery Services can handle the exception and can continue to run. If a tracking service does not communicate the tracking results to the Discovery Service in every two seconds period, the Discovery Service removes its entry from the registry. Hence, from Discovery Service's perspective a tracking service is dead even if it is inactive. The rationale behind this is, if a tracking service cannot produce the tracking results that may be useful to the other tracking services or to the client, then there is no need to keep a track of whether the tracking system is really dead or just inactive. As a result, the Discovery Service and the Fusion Service do not have any effect on them even if a tracking system is inoperative or shuts down.

As explained in section 3.4, the Fusion Service fuses the readings if it has received more than one reading. In addition to this criterion, we also use the criterion that the time difference between the readings to be fused should be less than or equal to 600 milliseconds. As demonstrated in experiments discussed in sections 4.1.1 and 4.1.2, the response time may be as much as approximately 500 milliseconds for the vision-based tracking. Hence, based on those observations and heuristic study, we designed this time bound of 600ms. This ensures that the Fusion Service does not fuse the stale readings together.

During the tracking experiment, we abruptly terminated the Fusion and Discovery Services in the middle of their execution to study how the tracking services handle the fault. In the absence of Discovery and Fusion Services, the tracking services can continue to track the object but individual tracking services cannot reliably initiate a communication with their peers. This is because the IP addresses at which peers are listening may have changed since they last updated their graph with help from the Discovery Service.

For the communication between two peer tracking services, we use socket timeout mechanism so that a tracking service can handle the failure of the other tracking service gracefully. Thus, we can claim that e-DTS 3.0 handles the exceptions arising at runtime without hampering the availability of the core tracking functionality.

In this chapter, we empirically demonstrated that e-DTS 3.0 improves the vision-based and Wi-Fi-based tracking accuracy compared to the accuracy obtained using e-DTS 2.0 [19] and eDOTS [26] respectively. It also shows that the fusion techniques implemented improve the accuracy compared to the accuracy of an individual tracking service at the cost of increased response time. e-DTS 3.0 also introduces additional modality of Bluetooth-based tracking that can be used when vision-based or Wi-Fi-based tracking cannot be carried out. While executing experiments related to vision-based tracking, it was observed that the tracking result changes significantly even for a slight change in the position or the orientation of the Android-based device. This sensitivity of the inside-out tracking helps to improve the accuracy of vision-based tracking. e-DTS 3.0 takes advantage of many possible opportunities in the tracking environment, such as the hotspot capability of Android-based devices, e-DTS 2.0 [19] tracking service(s) and peer tracking services running in the tracking environment, to demonstrate Opportunistic Tracking. It also shows an ability to tackle the handoff across multiple worlds seamlessly.

CHAPTER 5. CONCLUSION AND FUTURE WORK

5.1 Conclusion

This thesis has introduced mobile devices in the existing tracking systems (e-DTS 2.0 [19] and e-DOTS [26]) which mainly use stationary sensors. Through various experiments, we have shown that additional sensor modalities, available on mobile devices, help to improve tracking accuracy and enable the tracking system to carry out tracking even in the absence of a fixed set of stationary sensors. Since e-DTS 3.0 uses commercially available smartphones, this tracking system can be used anywhere without installing any additional infrastructure.

This thesis has developed the Discovery Service that maintains information about the tracking environment in the form of a registry of active services and history of the tracking services. History helps the tracking services to infer and maintain more information about their peers. This information is locally maintained at the tracking services in form of graph where an edge represents the neighbor of the tracking service. Experimental scenarios show that this history and graph maintenance make e-DTS 3.0 opportunistic.

5.2 Future work

Future work to enhance this tracking system includes:

1. Studying effects of camera calibration in tracking using Android-based devices' camera.
2. Detecting and using orientation of the marker and the Android-based tracking device.
3. Introducing Multilateration and Wi-Fi Simultaneous Localization and Mapping (WiFiSLAM) in Wi-Fi-based tracking.
4. Comprehensive study of inertial sensor fusion techniques.
5. Augmenting the current tracking system by allowing the user to interact with it.
6. Automatic discovery of entities in e-DTS 3.0 such as Fusion Service, Discovery Service and Tracking Services.

LIST OF REFERENCES

LIST OF REFERENCES

[1] Aubeck, F., Isert, C. Gusenbauer, D., Camera based step detection on mobile phones, International Conference on Indoor Positioning and Indoor Navigation (IPIN), Vol. 1, No. 7, Pages 21-23, Sept. 2011, DOI: 10.1109/IPIN.2011.6071910

[2] Werner, M., Kessel, M., Marouane, C., Indoor positioning using smartphone camera, International Conference on Indoor Positioning and Indoor Navigation (IPIN), Vol. 1, No. 6, Pages 21-23, Sept. 2011, DOI: 10.1109/IPIN.2011.6071954

[3] Shin, B., Lee, K.; Choi, S., Kim, J., Lee, W., Kim, H., Indoor WiFi positioning system for Android-based smartphone, International Conference on Information and Communication Technology Convergence (ICTC), Vol. 319, No. 320, Pages 17-19, Nov. 2010, DOI: 10.1109/ICTC.2010.5674691

[4] Laoudias, C., Constantinou, G., Constantinides, M., Nicolaou, S., Zeinalipour-Yazti, D., Panayiotou, C.G., The Airplace Indoor Positioning Platform for Android Smartphones, IEEE 13th International Conference on Mobile Data Management (MDM), Vol. 312, No. 315, Pages 23-26, July 2012, DOI: 10.1109/MDM.2012.68

[5] Kothari, N., Kannan, B., Dias, M., Robust indoor localization on a commercial smart-phone, tech. report CMU-RI-TR-11-27, Robotics Institute, Carnegie Mellon University, August, 2011

[6] Mautz, R., Indoor Positioning Technologies, Habilitation Thesis, Department of Civil, Environmental and Geomatic Engineering, ETH Zurich, 2012

[7] Shala, U.; Rodriguez, A., Indoor Positioning using Sensor-fusion in Android Devices, Master’s Thesis, Department of Computer Science Embedded Systems, Kristianstad University, 2011

[8] Vera, R., Ochoa, S., Aldunate, R., EDIPS: an Easy to Deploy Indoor Positioning System to support loosely coupled mobile work, Personal Ubiquitous Computing, Vol. 15, No. 4, Pages 365—376, April 2011, DOI: 10.1007/s00779-010-0357-x

[9] Klein, G., Visual Tracking for Augmented Reality, PhD Thesis, Department of Engineering, University of Cambridge, 2006

[10] IETF Mobile Ad hoc Networking (MANET) RFC 2501, http://www.ietf.org/rfc/rfc2501.txt, accessed March 2013

[11] Using Network Service Discovery, Android Developers Documentation, http://developer.android.com/training/connect-devices-wirelessly/nsd.html, accessed March 2013

[12] Connecting with Wi-Fi Direct, Android Developers Documentation, http://developer.android.com/training/connect-devices-wirelessly/wifi-direct.html, accessed March 2013

[13] Highlights of the Pew Internet Project's research related to mobile technology, http://pewinternet.org/Commentary/2012/February/Pew-Internet-Mobile.aspx, accessed March 2013

[14] AndAR - Android Augmented Reality library homepage, https://code.google.com/p/andar/, accessed August 2012

[15] HTC Google Nexus One, specifications, http://www.gsmarena.com/htc_google_nexus_one-3069.php, accessed August 2012

[16] Samsung Galaxy Nexus I9250, specifications, http://www.gsmarena.com/samsung_galaxy_nexus_i9250-4219.php, accessed August 2012

[17] Henniges, R., Current approches of Wifi Positioning, Service-centric Networking research group seminar, TU Berlin, 2012

[18] Trilateration , Wikipedia article, http://en.wikipedia.org/wiki/Trilateration, accessed March 2013

[19] Rybarczyk, R., e-DTS 2.0: A Next-Generation of a Distributed Tracking System, M.S thesis submitted to Indiana University Purdue University Indianapolis, Department of Computer and Information Science, Dec. 2010

[20] Joshi, G. G., Raje, R. R., Tuceryan, M., Designing and Experimenting with a Distributed Tracking System, 14th IEEE International Conference on Parallel and Distributed Systems (ICPADS '08), Pages 64-71, Dec. 2008

[21] Talavdekar, N., e-DTS Enhanced Distributed Tracking System, M.S thesis submitted to Indiana University Purdue University Indianapolis, Department of Computer and Information Science, Dec. 2009

[22] WifiManager, Android Developers Documentation, http://developer.android.com/reference/android/net/wifi/WifiManager.html, accessed March 2013

[23] What Mobile Will Look Like In the Future – The Facts, http://www.engagemobile.com/what-mobile-will-look-like-in-the-future-the-facts/, accessed March 2013

[24] LG Vortex VS660, specifications, http://www.gsmarena.com/lg_vortex_vs660-3630.php, accessed July 2013

[25] Settings.Secure class overview, Android Developers Documentation, http://developer.android.com/reference/android/provider/Settings.Secure.html#ANDROID_ID, accessed March 2013

[26] Rybarczyk, R., Raje, R. R., Tuceryan, M., A Novel Approach To Indoor Tracking In A Multi-Sensor Environment, Department of Computer and Information Science, Indiana University Purdue University Indianapolis, Indiana USA, 2011

[27] Indoor Positioning Systems: We Know Where You Are, http://www.smart-buildings.com/uploads/1/1/4/3/11439474/2013febindoor.pdf, accessed June 2013

[28] Home sensors enable seniors to live independently, National Science Foundation website, http://www.nsf.gov/news/special_reports/science_nation/eldertech.jsp?WT.mc_id=USNSF_51, accessed August 2013

[29] How To Build Applications Based On AndAR, AndAR library, https://code.google.com/p/andar/wiki/HowToBuildApplicationsBasedOnAndAR, accessed July 2012

[30] Bluetooth, Android Developers Documentation, http://developer.android.com/guide/topics/connectivity/bluetooth.html, accessed March 2013

[31] Bluetooth vs. Wi-Fi (IEEE 802.11), Wikipedia article, http://en.wikipedia.org/wiki/Bluetooth#Bluetooth_vs._Wi-Fi_.28IEEE_802.11.29, accessed March 2013

[32] Sensors Overview, Android Developers Documentation, http://developer.android.com/guide/topics/sensors/sensors_overview.html, accessed March 2013

[33] Motion Sensors, Android Developers Documentation, http://developer.android.com/guide/topics/sensors/sensors_motion.html, accessed March 2013

[34] Welch, G., Bishop, G., An Introduction to the Kalman Filter, http://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf, Department of Computer Science, University of North Carolina at Chapel Hill, accessed March 2013

[35] Carlson, N. A., Federated square root filter for decentralized parallel Processors, IEEE Transactions on Aerospace and Electronic Systems, Vol. 26 , No. 3, Pages 517 - 525, May 1990, DOI 10.1109/7.106130

[36] Wang, J., Azuma, R., Bishop, G., Chi, V., Eyles, J., Fuchs, H., Tracking a head-mounted display in a room-sized environment with head-mounted cameras, SPIE Proceedings, Vol. 1290, Pages 47-57, April 1990

[37] Keitler, P., Schlegel, M., Klinker, G., Indirect Tracking to Reduce Occlusion Problems, Proceedings of the 4th International Symposium on Advances in Visual Computing, Part II. Pages 224 - 235, 2008, DOI 10.1007/978-3-540-89646-3_22

[38] Waechter, C., Huber, M., Keitler, P., Schlegel, M., Klinker, G., Pustka, D., A Multi-Sensor Platform for Wide-area Tracking, 9th IEEE International Symposium on Mixed and Augmented Reality (ISMAR), Pages 275 - 276, Oct. 2010, DOI 10.1109/ISMAR.2010.5643604

[39] Klinker, G., Reicher, T., Brügge, B., Distributed User Tracking Concepts for Augmented Reality Applications, Proceedings of ISAR, Pages 37 – 44, 2000

[40] Bauer, M., Bruegge, B., Klinker, G., MacWilliams, A., Reicher, T., Riß, S., Sandor, C., Wagner, M., Design of a Component–Based Augmented Reality Framework, Proceedings of IEEE and ACM International Symposium on Augmented Reality, Pages 45 – 54, Oct. 2001, DOI 10.1109/ISAR.2001.970514

[41] WaveMetrics IGOR Pro 6, http://www.wavemetrics.com/index.html, accessed July 2013

[42] Android Software Development Kit (SDK), http://developer.android.com/sdk/index.html, accessed August 2012

[43] Logitech Communicate Deluxe Web camera specifications, http://reviews.cnet.com/webcams/logitech-quickcam-communicate-deluxe/4507-6502_7-32617298.html, accessed July 2013

[44] Kato, H., Billinghurst, M., Marker Tracking and HMD Calibration for a video based Augmented Reality Conferencing System, Proceedings of the 2nd International Workshop on Augmented Reality (IWAR 99), 1999

[45] GPS Accuracy Levels, http://www.oc.nps.edu/oc2902w/gps/gpsacc.html, accessed September 2013

[46] Machine Vision for Robot Guidance, http://www.robotics.org/content-detail.cfm/Industrial-Robotics-Featured-Articles/Machine-Vision-for-Robot-Guidance/content_id/3476, accessed August 2013

[47] Priyantha, N. B., Providing precise indoor location information to mobile devices, M.S thesis submitted to Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, Jan 2001

[49] Bahl, P., Padmanabhan,, V.N., RADAR: an in-building RF-based user location and tracking system,. Proceedings of Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2000), Vol. 2, Pages 775 - 784, Mar 2000, DOI 10.1109/INFCOM.2000.832252

[50] LaMarca, A., Hightower, J., Smith, I., Consolvo, S., Self-Mapping in 802.11 Location Systems, Proceedings of 7th International Conference on Ubiquitous Computing (UbiComp 2005), Pages 87-104, Sept. 2005, DOI 10.1007/11551201_6

APPENDICES

Appendix A Class diagrams of e-DTS 3.0

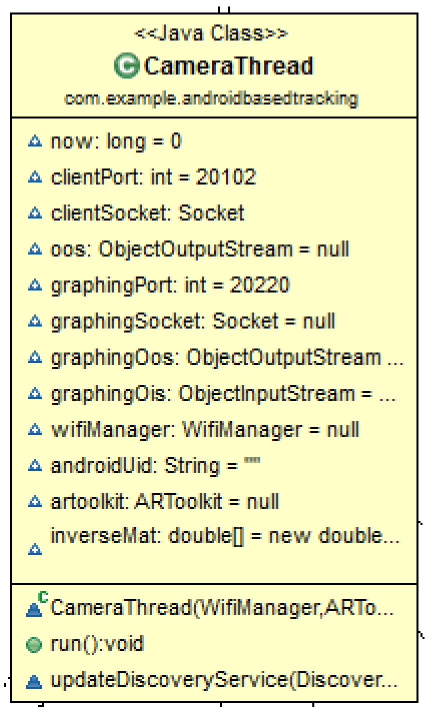
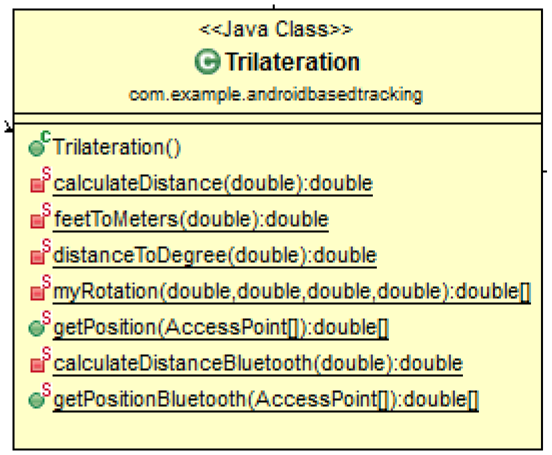
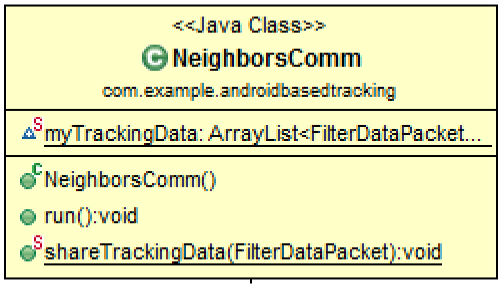

<<Java Class>>
CameraThread
com.example.androidbasedtracking
now: long = 0
clientPort: int = 20102
clientSocket: Socket
oos: ObjectOutputStream = null
graphingPort: int = 20220
graphingSocket: Socket = null
graphingOos: ObjectOutputStream ...
graphingOis: ObjectInputStream = ...
wifiManager: WifiManager = null
androidUid: String = ""
artoolkit: ARToolkit = null
inverseMat: double[] = new double...
CameraThread(WifiManager,ARTo...
run():void
updateDiscoveryService(Discover...

Figure A 1 Camera thread



<<Java Class>>
Trilateration
com.example.androidbasedtracking
Trilateration()
calculateDistance(double):double
feetToMeters(double):double
distanceToDegree(double):double
myRotation(double,double,double,double):double[]
getPosition(AccessPoint[]):double[]
calculateDistanceBluetooth(double):double
getPositionBluetooth(AccessPoint[]):double[]

Figure A 2 Trilateration class



<<Java Class>>
NeighborsComm
com.example.androidbasedtracking
myTrackingData: ArrayList<FilterDataPacket...
NeighborsComm()
run():void
shareTrackingData(FilterDataPacket):void

Figure A 3 NeighborsComm class

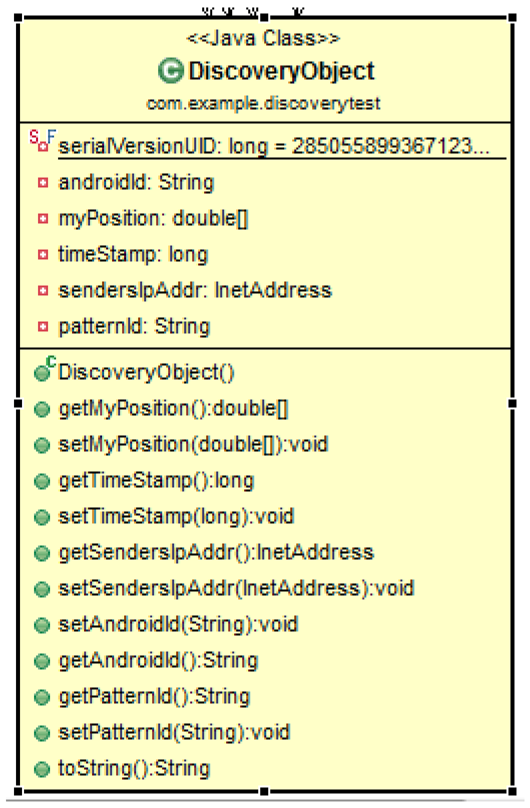
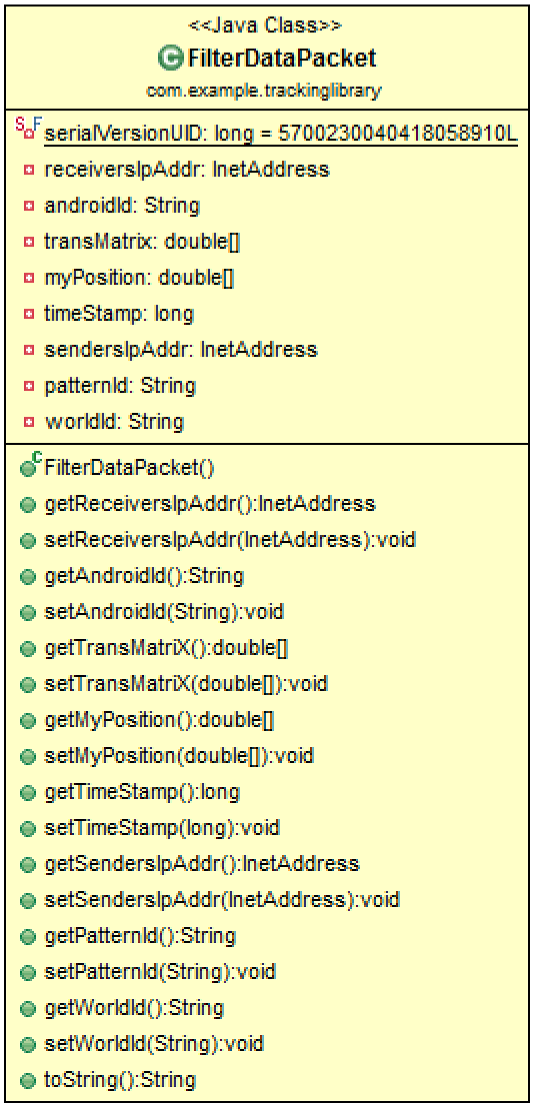


Figure A 4 DiscoveryObject class



<<Java Class>>
FilterDataPacket
com.example.trackinglibrary
serialVersionUID: long = 5700230040418058910L
receiversIpAddr: InetAddress
androidId: String
transMatrix: double[]
myPosition: double[]
timeStamp: long
sendersIpAddr: InetAddress
patternId: String
worldId: String
FilterDataPacket()
getReceiversIpAddr():InetAddress
setReceiversIpAddr(InetAddress):void
getAndroidId():String
setAndroidId(String):void
getTransMatriX():double[]
setTransMatriX(double[]):void
getMyPosition():double[]
setMyPosition(double[]):void
getTimeStamp():long
setTimeStamp(long):void
getSendersIpAddr():InetAddress
setSendersIpAddr(InetAddress):void
getPatternId():String
setPatternId(String):void
getWorldId():String
setWorldId(String):void
toString():String

Figure A 5 FilterDataPacket class

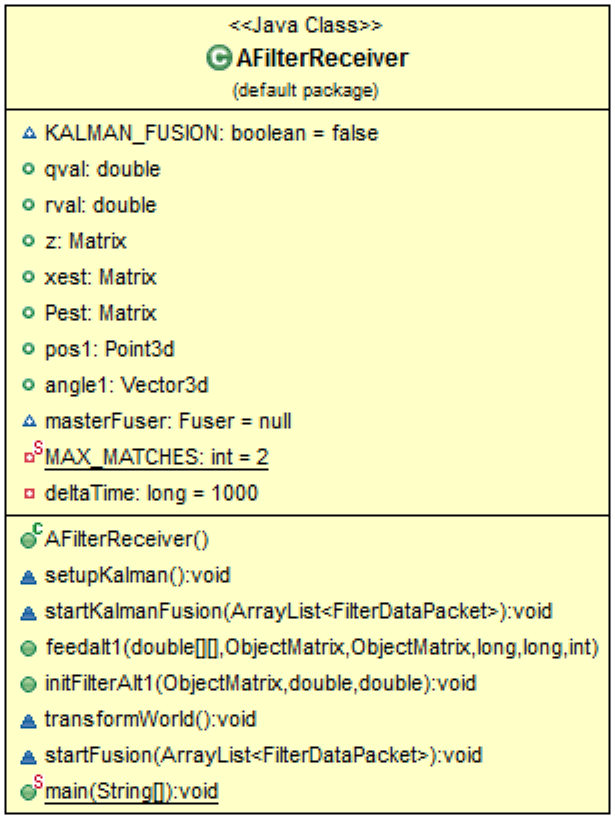

<<Java Class>>
AFilterReceiver
(default package)
KALMAN_FUSION: boolean = false
qval: double
rval: double
z: Matrix
xest: Matrix
Pest: Matrix
pos1: Point3d
angle1: Vector3d
masterFuser: Fuser = null
MAX_MATCHES: int = 2
deltaTime: long = 1000
AFilterReceiver()
setupKalman():void
startKalmanFusion(ArrayList<FilterDataPacket>):void
feedalt1(double[][],ObjectMatrix,ObjectMatrix,long,long,int)
initFilterAlt1(ObjectMatrix,double,double):void
transformWorld():void
startFusion(ArrayList<FilterDataPacket>):void
main(String[]):void

Figure A 6 AFilterReceiver class

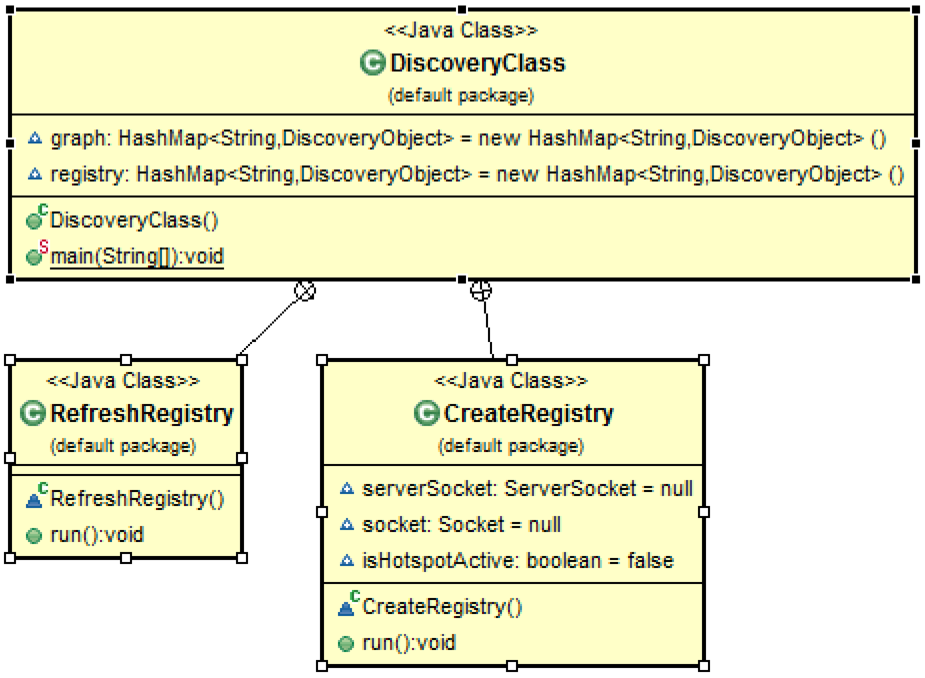
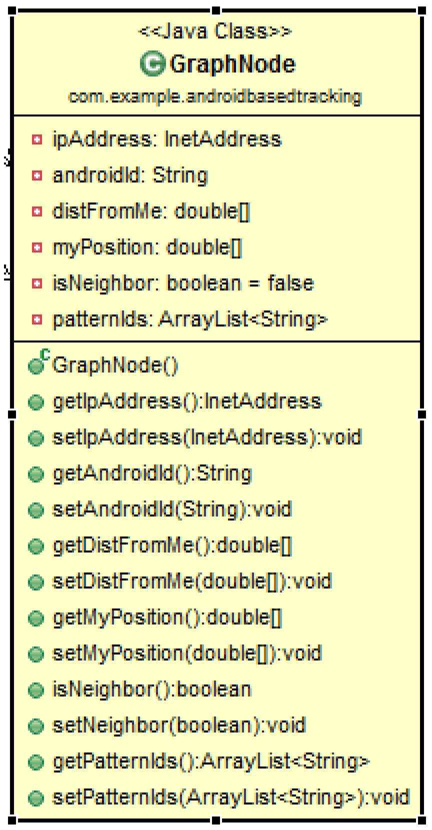
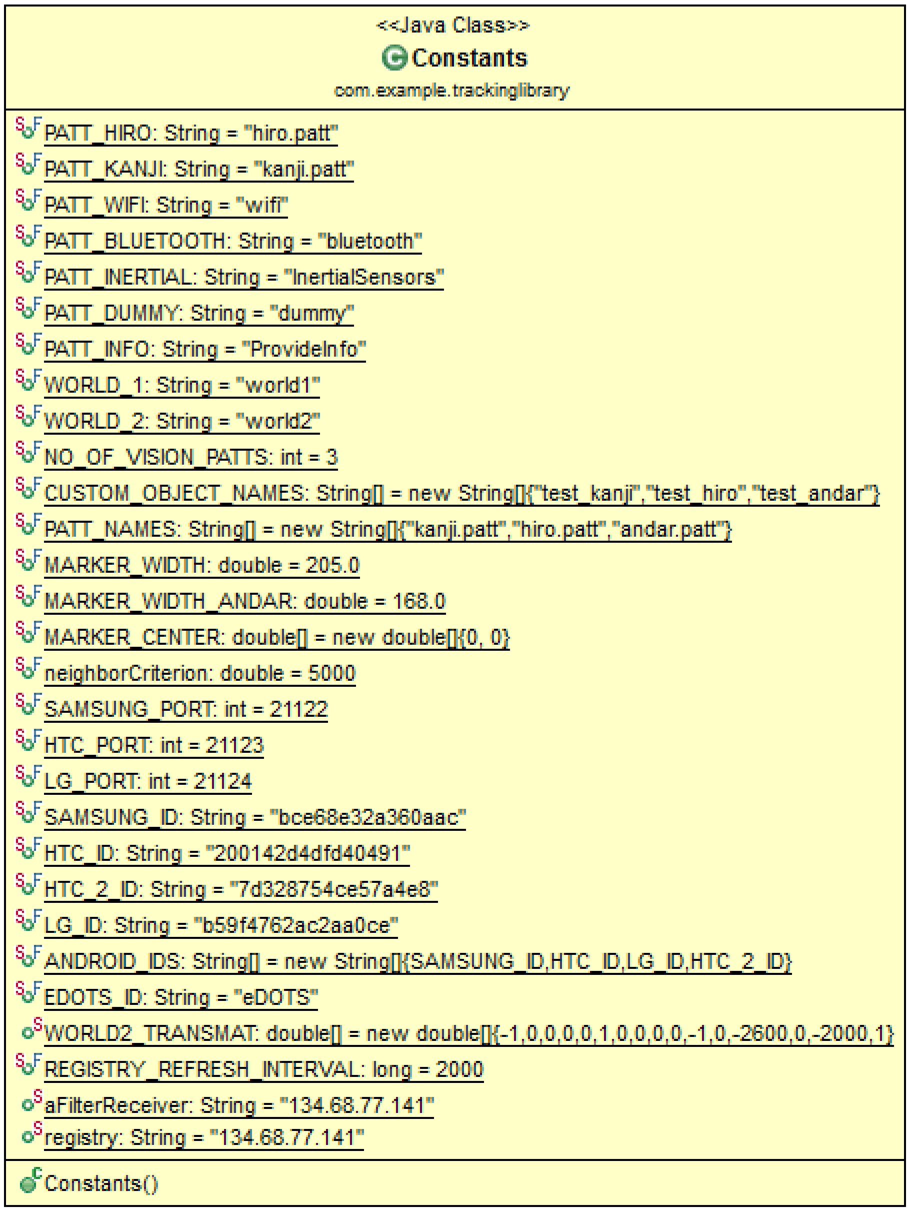


Figure A 7 DiscoveryClass and associated threads



<<Java Class>>
GraphNode
com.example.androidbasedtracking
ipAddress: InetAddress
androidId: String
distFromMe: double[]
myPosition: double[]
isNeighbor: boolean = false
patternIds: ArrayList<String>
GraphNode()
getIpAddress():InetAddress
setIpAddress(InetAddress):void
getAndroidId():String
setAndroidId(String):void
getDistFromMe():double[]
setDistFromMe(double[]):void
getMyPosition():double[]
setMyPosition(double[]):void
isNeighbor():boolean
setNeighbor(boolean):void
getPatternIds():ArrayList<String>
setPatternIds(ArrayList<String>):void

Figure A 8 GraphNode class

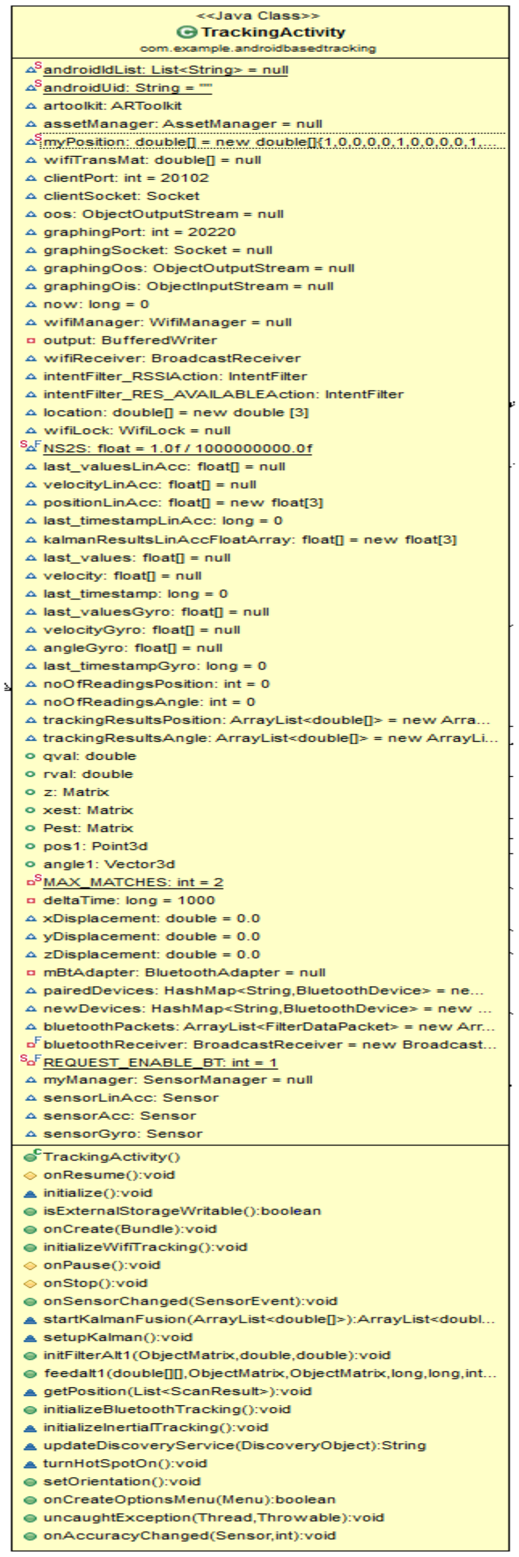


```
<<Java Class>>
Constants
com.example.trackinglibrary

PATT_HIRO: String = "hiro.patt"
PATT_KANJI: String = "kanji.patt"
PATT_WIFI: String = "wifi"
PATT_BLUETOOTH: String = "bluetooth"
PATT_INERTIAL: String = "InertialSensors"
PATT_DUMMY: String = "dummy"
PATT_INFO: String = "ProvideInfo"
WORLD_1: String = "world1"
WORLD_2: String = "world2"
NO_OF_VISION_PATTS: int = 3
CUSTOM_OBJECT_NAMES: String[] = new String[]{"test_kanji","test_hiro","test_andar"}
PATT_NAMES: String[] = new String[]{"kanji.patt","hiro.patt","andar.patt"}
MARKER_WIDTH: double = 205.0
MARKER_WIDTH_ANDAR: double = 168.0
MARKER_CENTER: double[] = new double[]{0, 0}
neighborCriterion: double = 5000
SAMSUNG_PORT: int = 21122
HTC_PORT: int = 21123
LG_PORT: int = 21124
SAMSUNG_ID: String = "bce68e32a360aac"
HTC_ID: String = "200142d4dfd40491"
HTC_2_ID: String = "7d328754ce57a4e8"
LG_ID: String = "b59f4762ac2aa0ce"
ANDROID_IDS: String[] = new String[]{SAMSUNG_ID,HTC_ID,LG_ID,HTC_2_ID}
EDOTS_ID: String = "eDOTS"
WORLD2_TRANSMAT: double[] = new double[]{-1,0,0,0,0,1,0,0,0,0,-1,0,-2600,0,-2000,1}
REGISTRY_REFRESH_INTERVAL: long = 2000
aFilterReceiver: String = "134.68.77.141"
registry: String = "134.68.77.141"

Constants()
```

Figure A 9 Constants or configuration file



```
<<Java Class>>
TrackingActivity
com.example.androidbasedtracking

androidIdList: List<String> = null
androidUid: String = ""
artoolkit: ARToolkit
assetManager: AssetManager = null
myPosition: double[] = new double[]{1,0,0,0,0,1,0,0,0,0,1,...
wifiTransMat: double[] = null
clientPort: int = 20102
clientSocket: Socket
oos: ObjectOutputStream = null
graphingPort: int = 20220
graphingSocket: Socket = null
graphingOos: ObjectOutputStream = null
graphingOis: ObjectInputStream = null
now: long = 0
wifiManager: WifiManager = null
output: BufferedWriter
wifiReceiver: BroadcastReceiver
intentFilter_RSSIAction: IntentFilter
intentFilter_RES_AVAILABLEAction: IntentFilter
location: double[] = new double [3]
wifiLock: WifiLock = null
NS2S: float = 1.0f / 1000000000.0f
last_valuesLinAcc: float[] = null
velocityLinAcc: float[] = null
positionLinAcc: float[] = new float[3]
last_timestampLinAcc: long = 0
kalmanResultsLinAccFloatArray: float[] = new float[3]
last_values: float[] = null
velocity: float[] = null
last_timestamp: long = 0
last_valuesGyro: float[] = null
velocityGyro: float[] = null
angleGyro: float[] = null
last_timestampGyro: long = 0
noOfReadingsPosition: int = 0
noOfReadingsAngle: int = 0
trackingResultsPosition: ArrayList<double[]> = new Arra...
trackingResultsAngle: ArrayList<double[]> = new ArrayLi...
qval: double
rval: double
z: Matrix
xest: Matrix
Pest: Matrix
pos1: Point3d
angle1: Vector3d
MAX_MATCHES: int = 2
deltaTime: long = 1000
xDisplacement: double = 0.0
yDisplacement: double = 0.0
zDisplacement: double = 0.0
mBtAdapter: BluetoothAdapter = null
pairedDevices: HashMap<String,BluetoothDevice> = ne...
newDevices: HashMap<String,BluetoothDevice> = new ...
bluetoothPackets: ArrayList<FilterDataPacket> = new Arr...
bluetoothReceiver: BroadcastReceiver = new Broadcast...
REQUEST_ENABLE_BT: int = 1
myManager: SensorManager = null
sensorLinAcc: Sensor
sensorAcc: Sensor
sensorGyro: Sensor

TrackingActivity()
onResume():void
initialize():void
isExternalStorageWritable():boolean
onCreate(Bundle):void
initializeWifiTracking():void
onPause():void
onStop():void
onSensorChanged(SensorEvent):void
startKalmanFusion(ArrayList<double[]>):ArrayList<doubl...
setupKalman():void
initFilterAlt1(ObjectMatrix,double,double):void
feedalt1(double[][],ObjectMatrix,ObjectMatrix,long,long,int...
getPosition(List<ScanResult>):void
initializeBluetoothTracking():void
initializeInertialTracking():void
updateDiscoveryService(DiscoveryObject):String
turnHotSpotOn():void
setOrientation():void
onCreateOptionsMenu(Menu):boolean
uncaughtException(Thread,Throwable):void
onAccuracyChanged(Sensor,int):void
```

Figure A 10 TrackingActivity class

Appendix B Wi-Fi-based tracking data file (accessPoints.txt)

Format: “BSSID of AP”, “floor-plan image”, “X”, “Z”, “Y”, “path of floor-plan image”

```
6c:f3:7f:b0:5a:00,0-0,4.5,7.5,-2,N/A
6c:f3:7f:b0:5a:c0,0-0,25.5,7.5,-2,N/A
6c:f3:7f:b0:56:c0,0-0,47,7.5,-2,N/A
6c:f3:7f:b0:5a:40,0-0,52,-2,-2,N/A
6c:f3:7f:b0:5b:20,0-0,37,-2,-2,N/A
00:25:9c:4c:c9:5e,1-1,0.5,-4,1.5,N/A
6c:f3:7f:b0:58:e0,1-1,8.02,9.6,3,N/A
6c:f3:7f:b0:5a:80,1-1,6.6,-3.3,3,N/A
6c:f3:7f:4f:32:60,1-1,25.5,7.5,3,N/A
02:1a:11:f3:cf:6d,4-1,-2,2,1,N/A
6c:f3:7f:b0:59:f0,1-1,47,7.5,3,N/A
6c:f3:7f:b0:59:e4,1-1,47,7.5,3,N/A
6c:f3:7f:b0:52:e0,1-1,37,-4.5,3,N/A
6c:f3:7f:b0:58:e0,2-1,4.5,7.5,8,N/A
6c:f3:7f:4f:33:00,2-1,4.5,-3,8,N/A
6c:f3:7f:b0:54:20,2-1,24.5,-3,8,N/A
6c:f3:7f:b0:54:40,2-1,47,7.5,8,N/A
6c:f3:7f:b0:54:80,2-1,4.5,7.5,8,N/A
6c:f3:7f:b0:53:60,3-1,6,-2,13,N/A
6c:f3:7f:4f:33:40,3-1,5,6,13,N/A
```

```
6c:f3:7f:b0:53:e0,3-1,25.5,6.5,13,N/A

6c:f3:7f:b0:53:a0,3-1,47,5.00,13,N/A

5c:da:d4:54:37:72,4-1,-2,2,1,N/A

00:20:a6:54:cb:71,0-0,-1.8,1.8,5,N/A
```