2010

Sensor Augmented Large Interactive Surfaces

Prasad Ramanahally Siddalinga
Iowa State University

Sensor augmented large interactive surfaces

by

Prasad Ramanahally Siddalinga

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Program of Study Committee:
Daji Qiao, Major Professor
Stephen Gilbert
Govindarasu Manimaran
Chris Harding

Iowa State University

Ames, Iowa

2010

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to take this opportunity to express my gratitude to everyone who helped me in this endeavor. I would like to express my sincere gratitude to Dr. Daji Qiao and Dr. Stephen Gilbert who guided me in this research work. Their patience, guidance, support and words of inspiration throughout the research work were instrumental in producing this work.

I would like to thank Dr. Manimaran who has been a source of encouragement right since the beginning of my graduate program. I would also like to thank Dr. Chris Harding for his valuable inputs in writing this thesis.

I would like to thank my colleagues Mike Oren, Britta Mennecke, Jay Roltgen, Eric Marsh for their inputs and making work fun. A special thanks to my friend Morgan Thomas who patiently read my work and gave valuable suggestions. I would like to thank my friends for their support and making my stay in Iowa enjoyable.

I would like to thank my parents R.P.Paramashivaiah and K.M.Kathyayini, my sister Divya for their constant support, encouragement and inspiration.

ABSTRACT

Large interactive surfaces enable effective multi-user collaboration, but a majority of the current multi-touch systems are not truly multi-user. In this work we present a novel sensor-based approach for both user identification around a touch table and integration of unique gestures above the table. The work proposes the criteria for a successful and robust user identification system. The Cricket sensor based user identification system is integrated with an open source gesture recognition system "Sparsh-UI" to enable rapid multi-touch application development. Finally we evaluate the Cricket-based algorithm with contemporary multi-user, multi-touch systems and describe the various interaction affordances provided by the Cricket based user identification system.

CHAPTER 1. OVERVIEW

1.1. Introduction

Multi-touch technology research has been of great interest in the recent past and is revolutionizing human-computer interaction. Advances in multi-touch research have increased the availability of a wide variety of multi-touch devices with different underlying technologies and configurations.

Some of the commonly used technologies include FTIR, Diffused Illumination, Electrode-based detection and Infrared bezel-based displays. The following is a brief description of these:

FTIR (Frustrated Total Internal Reflection): Frustrated Total Internal Reflection is one way of detecting multiple touches [11] [14]. The mechanism is similar to fiber optics. A strong infrared (IR) source is placed along the edges of the Plexiglass; Usually an array of IR LEDs is employed for this purpose; Some of the light that enters the edge of the Plexiglass reflects back and forth between the inside planes of the Plexiglass due to total internal reflection (TIR). The light that does not undergo TIR leaves the Plexiglass near the edge.

The Plexiglass is now flooded with IR light. By touching the surface with human finger, the reflection is disturbed due to change in boundary conditions, and some of the light is scattered. Hence the term Frustrated Total Internal Reflection. This scattered light when captured by an IR sensitive camera appears as a "blob". The blobs correspond to the position of the touch and are processed using thresholding algorithms to determine the position of the touch.

Diffused Illumination: In this technique we flood the surface of the Plexiglass/ Acrylic with diffuse IR light, illuminating the fingertips of the user [8]. The reason to use diffuse IR light is to ensure a uniform light intensity at all points on the Plexiglass. Objects closer to the surface of the Plexiglass will reflect more light, so by very carefully filtering and thresholding the camera image, the fingers touching the surface of the Plexiglass can be isolated, again as blobs. The downside to this process is that any object

close to the surface will show up, including the user's palms, shirtsleeves, or bracelets. With careful tweaking of the filtering and thresholding algorithms, these effects can be minimized.

Electrode based multi-touch devices: DiamondTouch [7] works by transmitting signals through antennas in the table. These signals are capacitively coupled through the users and chairs to receivers, which identify the parts of the table each user is touching.

Infrared bezel based multi-touch displays: In these Multitouch devices there is a bezel in front of a display (such as a large flat screen panel) which contains a set of IR transmitters and receivers on opposite ends, thus forming a grid of infrared rays [17]. Upon bringing the finger close to the surface of the screen, the path of the light is obstructed and its location can be determined by using the information from the receptor on the x and y axes.

A major advantage of the multi-touch technology is that it allows for a very rich and effective co-located collaboration experience which was previously impossible with the traditional mouse and keyboard interface. [10] and [19] discuss the effectiveness of multi-touch interfaces for co-located and remote collaboration. Large interactive surfaces are usually put to use as multi-user devices as they provide simultaneous accessibility to multiple users. However, for an interactive surface to be truly multi-user- in addition to supporting multi-touch - it has to support the ability to differentiate and identify the users in the environment. Most commercially available multi-touch interactive surfaces are not capable of identifying the users, while some have limited ability such as identifying the presence of a user on a particular side.

The collaboration experience can be greatly enhanced if the multi-touch architecture (Hardware and Software) is made aware of the users. User identification is crucial for multi-touch applications that require authentication. In this work the criteria that need to be considered while designing a user identification system are established and novel method of identifying users on interactive surfaces

called “Cricket based user identification system” is presented. The various features and the enhanced interaction affordances enabled by this system are explained. Finally the performance of the algorithm used in this system is evaluated and compare the Cricket based user Identification system against contemporary user identification systems.

1.2. Related Work

Table-top interfaces provide new avenues for co-located collaboration but pose new challenges as well. Though much work is concentrated on new gestural interfaces and improved interaction techniques, there have been very few recent works in the area of user identification for multi-touch interfaces.

DiamondTouch [7] (now sold by Circle Twelve Inc) was one of the first multi-user multi-touch interfaces, which provided user identification. Diamond touch has a set of antennas that are embedded in the table top, with receivers in chairs. It works by transmitting signals through these antennae; user identification is achieved by capacitive coupling of these signals through users to receivers located in the chairs. Thus it knows “who” and “where” a particular user is touching. However, a major drawback of such a system is that the users are associated to the chairs that they are using, i.e. the association is between the chairs and the touches rather than the user himself. Thus, individual hands (left vs. right) of each user cannot be distinguished. Further, with such a solution we are forced to use a specific type of hardware and cannot be deployed on existing multi-touch interfaces.

Dohse et al [9] present a very basic camera-based hand-tracking user identification system, which identifies users based on the side of the table from which the hands appear. However, this approach has the same limitation as DiamondTouch; if the user switches sides of the table, the system cannot recognize the user as identical. Also, if there are multiple users on the same side of the table, then this approach does not work.

Schmidt et al in [23] present a vision based approach for identifying users in a multi-touch environment by identifying distinct hand geometry using an overhead camera. In this system the users can switch from an un-restricted input mode (without identification) to identification mode by laying their hands flat on the surface. The contours of the hand are then analyzed to achieve user identification. A major drawback that may be observed in this system is that to afford identification, the user has to perform a special gesture. Further the users are not continuously tracked, limiting the number of applications that can use such architecture. Further it might be quite inconvenient to the user if one has to spread the hand often while using a multi-touch interface.

Christoph etal in [2] present “Infractables” a multi-user Multitouch system capable of user identification, however this system requires the users to use an Infrared stylus based input device instead of their hands.

Franks et al in [1] discuss using a sensor-based approach to detect the proximity of users to the multi-touch table. The approach discussed consists of a Multitouch table where each side of the table has an array of infrared reflection sensors. These sensors scan a limited area around that side of the table and report a rough distance value, which is influenced by the reflective properties of the surface of the object in the front. By mapping each sensor to a specific location around the table, a proximity map can be calculated which shows which area is occupied. Thus the sides of the table or regions of the table, which are occupied by users can be determined.

Though this model provides an approach where the system is housed within the Multitouch table hardware and allows for proximity detection, it has several drawbacks. It cannot differentiate among the users present, and although the paper suggests that complicated models, which allow specific shape models to be associated with a particular user, can be developed, it does not discuss the details of it. Further, such a model would heavily depend on the orientation of the user and may not work with all

orientations. Also the system cannot detect if a user reaches over to another part of the table while standing in the same position. It merely associates a side to the user.

Schöning et al [13] present a mobile phone user identification and authentication mechanism for multi-touch walls. The proposed approach requires that the Multitouch system is FTIR based. The user touches a region of the FTIR Multitouch wall with a phone, which is equipped with accelerometer to detect movements. Upon detecting the movement, the phone generates a flash of light, which can be detected by the camera in the FTIR wall. This flash can be differentiated from the one generated by touches as it generates a much brighter blob in the visual range than the one generated by touch which is mostly in the infrared range. The user ID is then transmitted via Bluetooth. The flash of light and the user id reception (via Bluetooth) must happen in a very small window of time to allow associating that region to a particular user. If there is a collision (with a light flash and User Id reception from another phone) the whole process has to be repeated again. The work also describes to provide an authentication mechanism by using a challenge response mechanism by exchanging messages over Bluetooth.

The proposed approach provides limited user identification and secure authentication mechanism for users on a Multitouch screen, but the procedure is cumbersome and works only with a particular type of hardware, i.e. FTIR. Also, it is vulnerable to denial of access attacks as a malicious user can easily cause collisions thus requiring a legitimate user to repeat the process again and again.

1.3. The Need For User Identification In a Multi-touch Environment

User identification in multi-touch interfaces is needed for several reasons, which we will discuss in this section.

1. **Provide user-centric experience**

Large multi-touch interfaces can serve as an excellent platform for co-located collaboration. Therefore, applications developed on these surfaces are sometimes developed to be more user-centric than traditional desktop applications. For an application to be truly multi-user, it needs to be aware of the users. For example, since the user interface can be accustomed to change dynamically with the number of users present in the multi-touch environment, the user interface could further adapt based on the position or orientation of each individual user. One could also apply user-specific settings in a multi-user environment. A plethora of customizations and UI enhancements are possible by detecting the proximity of users to the multi-touch interface. Thus, in order for the interface to recognize individual users and provide an engaging experience, it is necessary to recognize which user is interacting with the application at a given time.

2. **User authentication**

In a multi-user interface, users interact in a shared space. For certain Multitouch applications such as Geographical Information systems (GIS) which are used for crisis management, tactical control of unmanned vehicles, or something as simple collaborative photo tagging applications, it might be necessary to have several users working with different access rights. For example, some have permission to only view the data while others will have the permission to modify the data or certain users can be restricted to specific regions of the interactive surface. Thus, in addition to recognizing the position of the touches, the system needs to be aware of the user generating those touches to grant the appropriate privileges. As a result, it becomes necessary to identify and authenticate the user in order to provide access to a particular region or data.

3. **Enhanced interaction affordances**

Systems which provide a means of user identification can provide enhanced interaction based on detection of user presence or proximity to the multi-touch interface. Proximity detection of users opens

up new modes of interaction possibilities such as orientation of widgets / layout based on the number of users or position of users. If the system is capable of detecting the "hover" in addition to touch, new gestures can be developed. Differentiating the left hand gestures from right hand gestures can also form the basis of new gestures. Thus, user identification facilitates the development of new gestures, which require multiple user participation, and provide the basis for the concept of object or region ownership.

4. Research tool

Finally, accurate user identification for multi-touch surfaces could prove to be an indispensable tool in the field of CSCW (computer supported cooperative work) by providing data related to distribution of touches of each user in various scenarios. This would encourage research efforts into how users create and use personal and shared workspaces on interactive surfaces. The study of this data can be vital in defining the next generation of user interface to support collaborative work.

CHAPTER 2. EXISTING COMPONENTS USED BY CRICKET BASED USER IDENTIFICATION

2.1. Sparsh-UI

In this section, a Multitouch gesture recognition system "Sparsh UI" is described. This Multitouch gesture recognition forms the foundation for our Cricket Based User Identification software architecture. Sparsh UI was developed by our group at Virtual Reality applications center [21]. In particular my contribution to this project consisted of the architecture, C++ version of the gesture server, FTIR, Bezel, Stantum device drivers, TUIO [26] adapter for Sparsh-UI and the "Flick gesture" algorithm was my individual contribution to this project. It is capable of supporting a wide variety of Multitouch hardware and provides a rich gesture library. We enhance this Multitouch gesture recognition framework by adding User identification system using the Cricket systems. Sparsh UI was selected for this purpose due to its following features:

1. Ability to support a variety of hardware

Sparsh UI supports a wide variety of hardware such as FTIR-based, diffuse-illumination based, bezel or capacitive screen. It decouples the hardware details from the gesture framework by specifying a standard protocol for communicating with the hardware.

Most multi-touch devices generate additional information from a touch, in addition to the (x, y) coordinate values on the screen. The state of a touch point can be defined using three states defined below:

- Point Birth: Information related to the creation of the touch point when the finger comes in contact with the screen.
- Point Move: Information related to the motion of the touch point (referred to as Point Move).
- Point Death: Information related to release of the finger from the touch screen (Point Death).

- An identification number for each touch point generated when a finger comes in contact with the touch screen (Point ID).

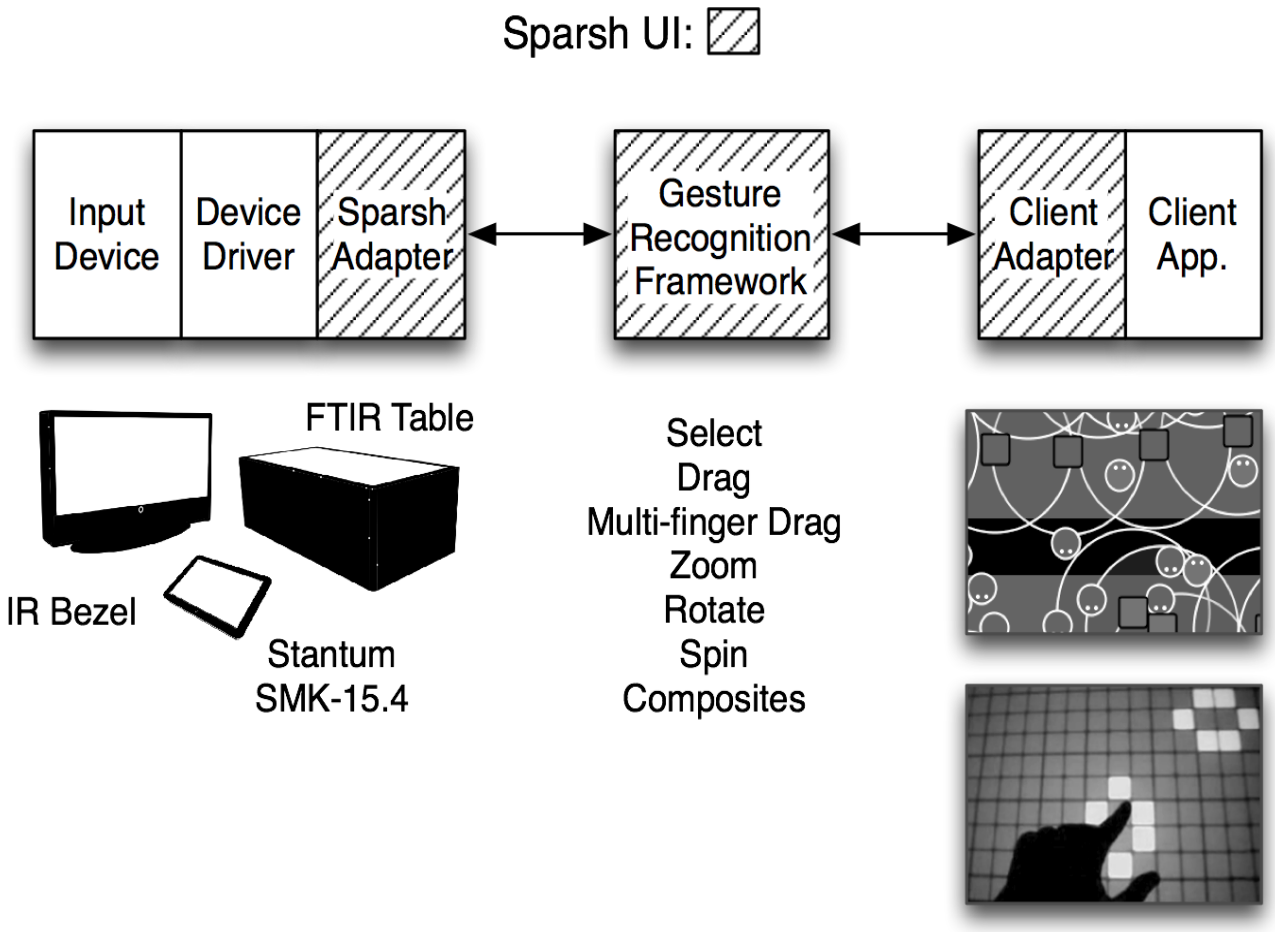

Figure 1: The Sparsh UI Architecture: The Sparsh Adapter standardizes touch events from varied hardware, sends the events over TCP to the Gesture Recognition Framework, which then sends appropriate events to the software client via the Client Adapter.

Thus, each touch point can be distinctly identified by the x-y coordinates, the state of the touch point (Point Birth, Point Move and Point Death) and the ID of the touch point (Point ID). Sparsh UI specifies a

format in which a device should send touch data to it and provides a driver adapter that can standardize the input from the driver, which would be compatible with Sparsh UI (Figure 1).

The driver adapter uses the data structure with the parameters mentioned above. As of now we have Sparsh compatible drivers for a 60" FTIR-based touch table created at Iowa State University[25], a Stantum SMK-15.4" multi touch tablet, and a 42" infrared-based bezel display from IR Touch. An adapter for the Dell Latitude XT is in progress.

Communication between the device driver and the Sparsh UI gesture recognition framework takes place over sockets using TCP protocol. This allows the driver and driver adapter to be written in the language of choice.

2. Gesture recognition

Most contemporary open source multi-touch software libraries provide only the ability to recognize the touch points and pass the touch coordinates directly to the application, leaving the application to do the gesture processing.

Multi-touch is made intuitive by means of gestures; it is vital for a multi-touch library to provide gesture recognition support. The following considerations should be addressed while providing gesture support to multi-touch applications:

- Flexibility to specify the supported gestures at an application level and UI component level.
- Support for providing touches point coordinates if the application does need to do custom gesture recognition.
- Ease of adding new gestures to the framework.

The usage of various gestures can be specific to the application. For example, our image manipulation application Picture App makes use of the Drag, Zoom, and Rotate gestures, but another application

called “Table Tower Defense” just makes use of touch gestures. One needs to process the touch point data to recognize all possible gestures for the former and just send out the touch coordinate data for the latter. Hence it is inefficient to analyze raw touch data for various gestures unless the application needs it. Similarly, not all the UI components would require all the gestures. For instance, a button would allow only “Select”, while a window title bar would allow “Drag” and other gestures that indicate minimize, maximize, etc. To incorporate this flexibility, we use the concept of Group ID to identify the various UI components on the screen. On each point birth, the application is queried for a Group ID and the allowed gestures corresponding to the point location. Different point birth sequences can be associated to the same Group ID (analogous to multiple fingers on the same UI component). If no gesture recognition is required for a given touch coordinate, it can return a null value indicating that there is no need for gesture processing. The Sparsh UI gesture recognition framework processes the incoming touch point coordinate data, recognizes the associated gestures, and sends back the associated gesture event for the Group ID. At the application end, it can be easily identified as to which component has been acted upon and the gesture event can be handled appropriately.

Applications can register for receiving the raw touch points if they need to process special custom gestures. The modular design of Sparsh UI makes it easy to add new gestures to the gesture recognition framework. The intuitiveness of a multi-touch interface is achieved through the use of gestures which are highly intuitive in the application context in which they are used. The usefulness of a gesture library would in part depend upon the number of intuitive gestures that a library can support.

Sparsh UI currently supports the following gestures. More gestures are being added as a part of continuous improvement of the framework.

Select gesture: Simply placing a finger on the multi-touch device performs this gesture. This gesture is normally used for selection purposes. It can also be used creatively as in our Table Tower Defense game.

The touch coordinates are passed to the application whenever the gesture framework detects this gesture.

Spin gesture: This gesture (Figure 2) is performed by placing two fingers on the multi-touch device that creates an invisible axis, somewhat similar to Jeff Han's two-handed hold-and-tilt gesture [14].

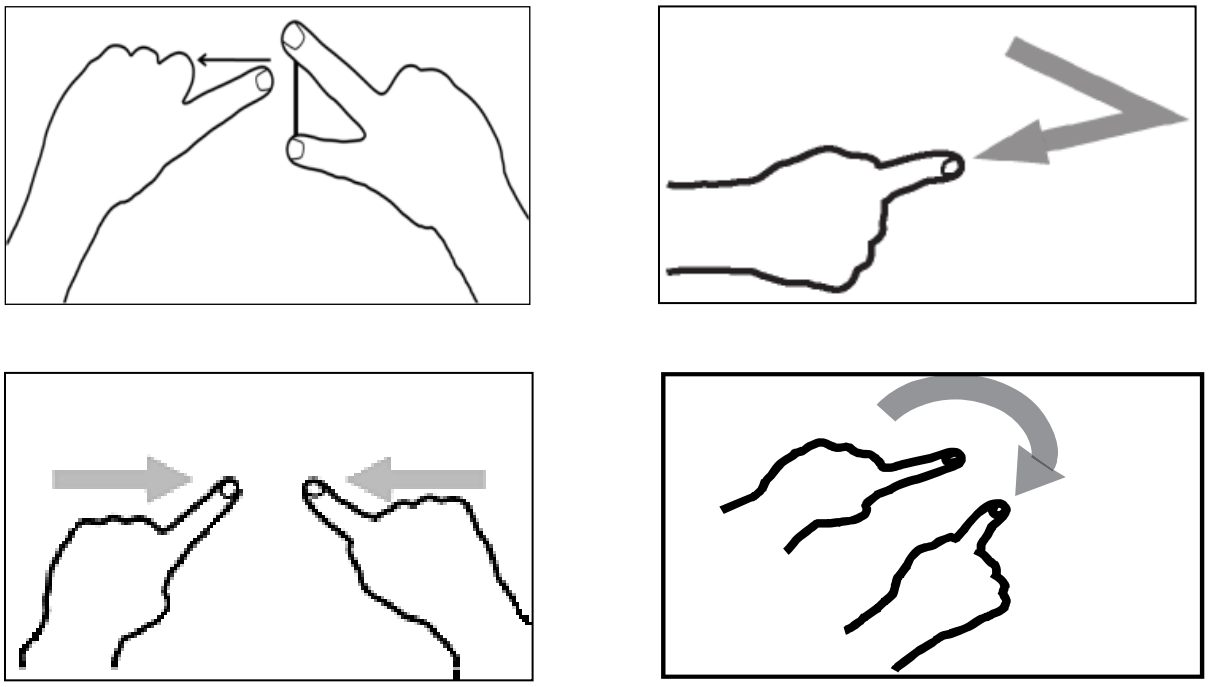

Figure 2: From top left, Spin Gesture, One-finger Drag Gesture, Zoom Gesture, Rotate Gesture.

In a 3D CAD-like application, once the axis has been established by one hand, the user is able to spin the 3D view point by dragging a third finger perpendicular to the axis created by the first two fingers. It can be used for any chosen axis of rotation. This gesture can be used to manipulate views in any 2D or 3D environment.

Multi-Finger drag gesture: The multi-finger drag gesture is a generic drag gesture which detects the drag (or swipe) when one or more fingers are moved across the touch screen. If applications need not

differentiate between the number of fingers that are used to do the drag operation, they can register for the multi-finger drag gesture.

In all the drag gesture implementations, the gesture recognition framework generates drag events with the parameters ΔX and ΔY, the amount of offset from the initial position (initial position of the centroid if it's not single touch).

If an application needs to distinguish between the number of fingers that resulted in the drag gesture, it can register for one or more of the following gestures:

One-finger drag: The user performs this gesture by placing a finger on the device and dragging it across the surface (Figure 2). This gesture can be used for moving graphic elements on the screen. It can also be used for panning a view (e.g., panning a map).

Two-finger drag: This is similar to a one-finger drag gesture except that two fingers are used instead of one. The two fingers may be held close to one another or apart.

Three- finger drag: In this case three fingers are used to perform the drag or swipe operation. This gesture is being considered to manipulate 3D objects in a multi-touch environment where both 3D and 2D objects are present. Similarly, Sparsh UI offers gestures for four-or five-finger gestures.

Rotate gesture: This gesture is performed by placing two fingers, either from the same hand or different hands; on the multi-touch device and rotating them clockwise or counter-clockwise (Figure 2). The gesture framework generates an event with the parameters consisting of the angle of rotation and the coordinates of center about which rotation occurs.

Zoom gesture: This gesture (Figure 2) is performed by placing two fingers on the multi-touch device and dragging them away or towards each other along a line. The gesture is typically used for zooming in and

out of maps, or more generally, scaling and resizing screen elements. When a zoom gesture occurs, an event consisting of the scale factor is generated.

3. Support of different platforms

A generic framework should be capable of operation across popular operating systems (Windows, Linux, and MacOS X). Sparsh UI exists in both Java, which is supported by most popular operating systems, and the C++ version is written using the Boost library, which makes it cross platform compatible.

4. Support for different programming languages

Since Sparsh UI uses socket-based communication to communicate with the multi-touch application, this allows one to write Sparsh-based applications in the language of choice. The system currently has client adapters for Java / C++ which abstract the communication protocol over sockets between the client application and the gesture framework.

5. Support of wide interface scale

Since collaboration across multi-touch devices often use disparate multi-touch devices of varying dimensions and configurations, it is important that gesture processing is not affected by the varying resolutions of different devices. This is achieved by using relative values for touch coordinates instead of absolute coordinates. However, it is the responsibility of application developers to ensure the usability and ergonomics of applications across devices of varying dimensions.

6. Support of collaboration

Sparsh UI provides a platform to develop collaborative multi-touch applications where collaboration is achieved at application level, e.g., by using TCP sockets to have two instances of the same application on different systems exchange data. In future versions we plan to support collaboration in Sparsh-UI so that various gesture events can be exchanged across networked multi-touch devices. But nevertheless there will always be application data, which needs to be exchanged at application level.

7. Open source Community support

Since Sparsh UI is distributed free of cost under the LGPL license, there is an active community support. This helps in the continuous development of the software to support more hardware and addition of new gestures.

2.2. Cricket Sensors

In this section, the cricket sensors which are used in our user identification architecture are described. The Cricket indoor location system was developed by MIT [22] and is currently available commercially from Crossbow technologies [6]. The crickets were originally intended for usage in building wide deployment for the purposes of location depended applications and user tracking. The design of cricket based systems was driven by the following considerations:

1) Decentralized administration: As Crickets were designed for building-wide deployment, it would be impractical if a centralized control and management system was used. In the cricket-based system, there are fixed entities which announce their location, and the receiver hardware can compute its location in space using an inference algorithm. Hence there is no need for a central entity to keep track of individual component in the system.
2) Scalability: Decentralized approach, distributed communication protocol, and a passive listener allow greater scalability thus allowing a large number of users to be tracked.
3) Ease of Deployment: Since the cricket transmitters do not need any backbone infrastructure, it is easier to deploy. Their small size allows them to be easily deployed in constrained spaces.
4) Cost: The cricket devices are built using commercially available off-the-shelf inexpensive components so that wide scale deployments can be realized in a cost efficient manner.
5) Granularity / Accuracy: The cricket system is capable of providing a reasonably high degree of accuracy of the order of 4-5 centimeters. It further allows demarcating boundaries between

regions corresponding to different beacons. Further, it provides the ability to create virtual boundaries by assigning different space identifiers to beacons.

6) Network heterogeneity: This is mainly to enable heterogeneous networked devices to obtain their location irrespective of the type of their communication technology. For example, different devices (which are using cricket service) can use Ethernet, WLAN, IR or any other mode of communication in a single space, but still get location information from cricket and benefit from location based services.

7) User Privacy: In a cricket-based system, since each user tracks himself without a central tracking system, privacy of the user is safeguarded. Cricket is more of a “location support system” rather than a location tracking system.

2.2.1. Cricket Hardware Details:

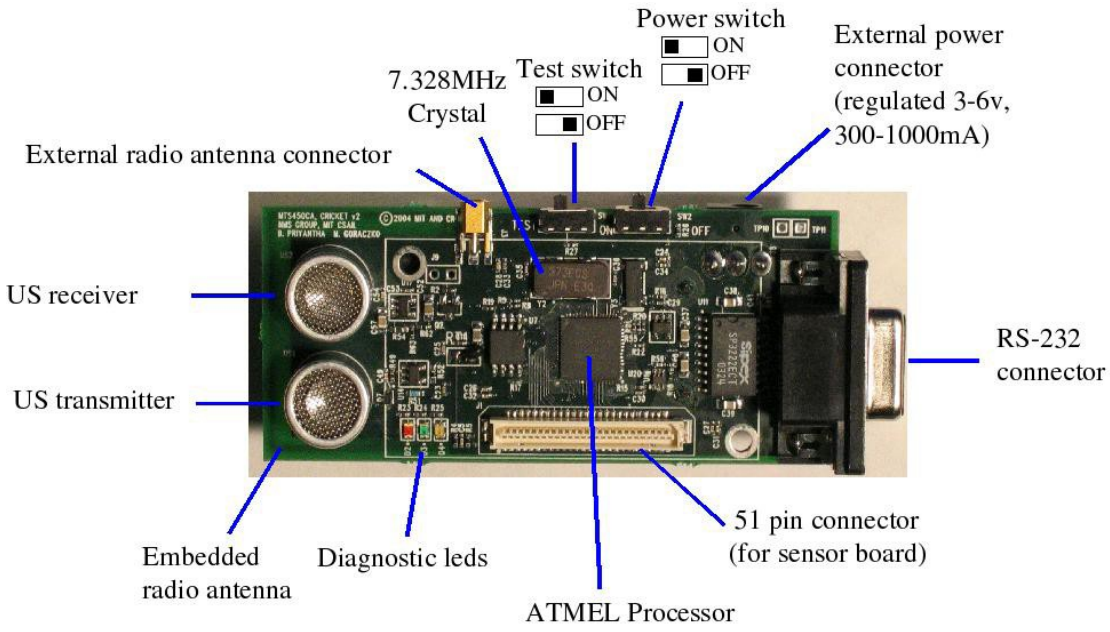

Figure 3: Cricket v2 Hardware details

Figure 3 (from Cricket manual [5]) shows the cricket hardware with all the important hardware components marked. The main components of the cricket [18] are as described below:

1) Microcontroller : The Cricket sensor has a 8 bit , 7.4 MHz ATMEL 128L microcontroller with 8 kB of RAM and 128kBytes of flash ROM for program memory and 4 kB of EEPROM as Read only memory. It needs an operating voltage of about 3 V and draws a current of 8 mA and 8 uA during the active and sleep modes respectively.
2) RF Transceiver: The Cricket sensor has a CC1000 RF transceiver which operates at 433 MHz. The operational data rate is 19.2 kbps.
3) Ultrasonic (US) transmitter / Receiver: The ultrasonic modules use a 40 kHz piezoelectric open-air ultrasonic transmitter / sensor. The transmitter generates ultrasonic pulses of duration 125 us. The transmitter uses a voltage multiplier to generate a 12 V supply from the 2 V supply

voltage while the receiver uses a two sage amplifier to get a variable gain between 70dB and 78 dB.

4) Expansion connector: The cricket node includes a 51 pin connector that is compatible with MICA mote sensor interface. This allows connecting MICA sensor boards for enhanced sensing or Mote processor boards for increased processing power.
5) RS-232 interface: The RS-232 interface is used to connect Crickets to host devices / PC for programming or data collection purposes.
6) Temperature Sensor: As the velocity of sound in air depends on the ambient temperature, it is necessary to compensate for these variations. An on-board pre-calibrated temperature sensor is used for this purpose.
7) Diagnostic LEDs- These LEDs can be programmed to provide various status indications of the cricket sensor board (like successful reception / collision indication etc).
8) Power source: The crickets can be powered either by two AA batteries (with battery holders on the back) or by a DC source using the external power connector.

Each Cricket has a unique 8 –byte ID, which uniquely identifies every Cricket node.

2.2.2. Cricket Firmware:

The crickets run the TinyOS 1.1.6 [20] distributed operating system for sensor motes. The TinyOS platform has an inbuilt RF stack which is used by the cricket for RF packet transmission and reception. Figure 4(from [20]) shows the TinyOS RF message format.

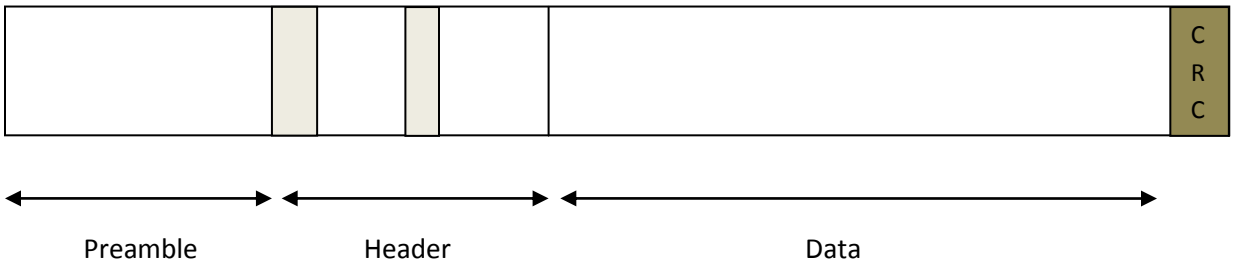

Figure 4: Tiny OS message format

The RF message consists of a preamble, which is transmitted prior to the actual message transmission. The Preamble allows time for the receiver to wake up from sleep state and also to sync its clock with that of the incoming signal.

The preamble is followed by the message header, which consists of

a) Source address (2 bytes) – The address of the transmitter

b) Destination Address (2 bytes) – the address of the receiver

c) Message length (1 byte) – The length of the message

d) Group ID (1 byte) – group id of the messages.

The header is followed by the message content. The message can be up to 256 bytes in length but is limited to 29 bytes in the firmware. The message also has a CRC appended at the end of the message to check for the integrity of the message contents.

2.2.3. Beacon Firmware and Message Architecture:

Cricket Operating Modes

Each cricket sensor consists of ultrasonic and RF radios which are used to determine the location. The cricket sensor can be deployed in one of the following two roles:

1) Beacons: These are cricket sensors that are mounted at fixed locations. The cricket beacons have fixed space IDs (a string that represents the space in which the beacon is present) and fixed position co-ordinates. The beacons transmit their space–IDs and position co-ordinates periodically over the radio frequency channel. This RF transmission is followed by a concurrent ultrasonic transmission.

   The transmission occurs every 800 ms and this is a configurable parameter, which can be varied according to deployment requirements. A value that is too small would increase the probability of RF / US collisions. Similarly, a high value of this value would result in infrequent distance measurements, which may provide reduced accuracy if we are tracking moving objects.

2) Listeners: These are cricket sensors that are mounted on stationary or mobile objects that need to be tracked. The listener listens to the RF and the ultrasonic transmission from the beacon and correlates them to each other and using the difference in propagation times of the ultrasonic and RF pulses calculate the relative distance to a specific beacon. If the distance of listener to three or more beacons is known, then the location of the listener in co-ordinate space can be inferred using known triangulation techniques.

Cricket Deployment Modes

Smith et al. in [24] describes two modes of deployment name Active Mobile architecture and passive Mobile architecture.

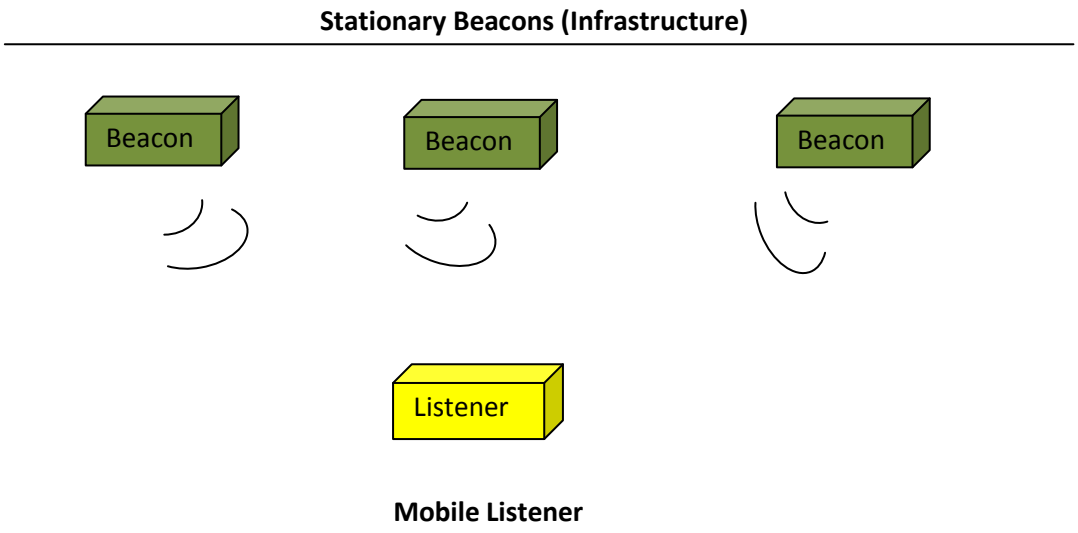

Figure 5: Configuration 1, Passive Mobile Architecture

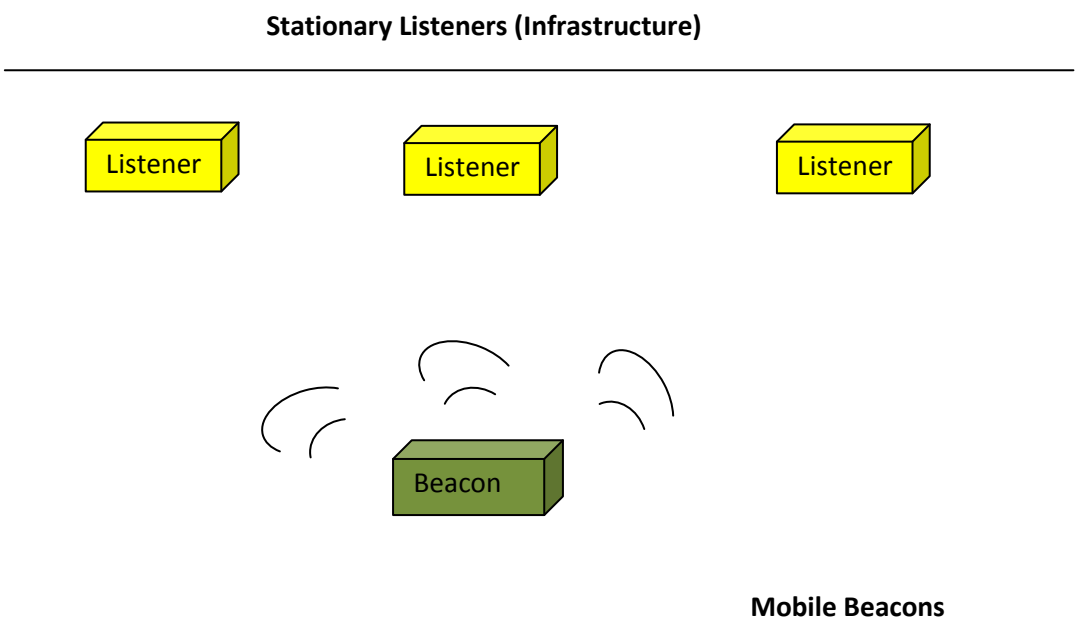

Figure 6: Configuration 2, Active Mobile Architecture

Figure 6 shows an active mobile architecture with passive nodes (Listeners) mounted on a surface such as ceiling and we have mobile transmitters which periodically broadcast RF and US pulses.

The receivers mounted at known locations use these broadcasts to compute the distances to these mobile nodes. The distances can be passed on to a central co-ordinator, which can aggregate the distance measurements from all the listeners and compute the location of the mobile node.

2.2.4. Comparison of Active and Mobile architectures:

In [24] an in-depth analysis of both active and passive mobile architecture is provided and the following conclusions are drawn:

1) Active mobile architecture outperforms passive mobile architecture as long as there are fewer mobile nodes. As the number of transmitting mobile nodes increase, the number of collisions increase, resulting in less accurate measurements or delayed distance measurements due to increase in the number of collisions.

2) Passive mobile architecture scales with the increase in the number of mobile nodes. Since the mobile nodes do not transmit periodically and the number of beacons is fixed, there will be less contention over the wireless medium. For ultrasonic transmissions, it is important to have a line of sight transmission for more accurate distance measurements. Experiments show that it is easier to block ultrasonic impulses by placing an obstacle (such as hand) in front of the transmitter than compared to placing an obstacle in front of the receiver.

CHAPTER 3. CRICKET BASED USER IDENTIFICATION SYSTEM

In this chapter the design criteria for user identification architecture are established and then we describe the hardware and software architecture of the Cricket based user identification system.

3.1. Design Criteria for user identification architecture

In order to serve as a reliable and scalable tool for user identification, robust user-identification architecture should satisfy the following design criteria.

1. **Support for a wide variety of multi-touch hardware**

With recent advances in multi-touch technology, several hardware options such as FTIR, Diffuse illumination, IR based solutions have become readily available. Furthermore, these hardware options can be used in either a horizontal configuration such as a multi-touch table, or in a vertical multi-touch wall. A robust user identification mechanism should be able to support multiple hardware architectures and configurations so that it provides the user with the flexibility to choose a hardware solution that best fits their needs.

2. **Support for user mobility**

Due to the size of the device and the nature of interactive surfaces, there might be a need for the user to switch sides (in a multi-touch table) or move across the multi-touch wall. A robust user identification mechanism should be able to maintain the same user association even if the user switches sides or moves to a different part of the interface.

3. **Support for User Authentication**

Large multi-touch surfaces allow multiple users to interact the same time. Certain applications might need to limit certain users to a certain region or limit the data that certain users can manipulate. In such a scenario, it becomes necessary to provide some means of authentication so that data or territorial

integrity can be protected. Thus, a robust user identification system should provide a means for authenticating users.

4. Support for presence / proximity detection

For certain applications it might be necessary to detect the arrival of a user within the vicinity of the multi-touch interface or to detect the continued presence of the user near the interface. Providing such abilities would allow interesting applications to be developed on a multi-touch interface. For example, in a user-aware photo application, the user's photos can be displayed or hidden depending on the user's presence or absence. Hence proximity / presence detection is a desirable feature of a robust user identification system.

5. Continuous user tracking

The multi-touch user-identification system should be capable of detecting the user for each touch he or she makes. Such capability is required for most applications. Discontinuous tracking or the need to switch to a "tracking" mode, though useful in some applications, would limit the scope of user-centric applications that can be developed. Continuous user tracking is also vital for user authentication otherwise security will be compromised.

6. Software support

Good user identification architecture should provide a pluggable software interface, which may be integrated into a wide variety of multi-touch software platforms. As such, the framework should be platform-independent and support frameworks written in a variety of programming languages. A gesture support system, which would recognize gestures in addition to user identification, would be desirable for rapid Multitouch application development.

The systems described in Chapter 2 do not address all the criteria described above. In our work, effort is made to address the shortcomings faced in those systems. In summary, the cricket sensor-based approach described in this work is motivated by the following factors:

- Provide a reliable and robust user identification mechanism for large interactive surfaces irrespective of the underlying Multitouch hardware.
- Provide added benefits of User Presence / Proximity detection.
- Provide enhanced interaction affordances by enabling new gestures that use z-axis information in addition to user identification
- Provide a basis for user authentication mechanisms to be developed based on the user identification architecture

3.2. Infrastructure Details

Figure 7 schematically depicts the deployment of cricket nodes in their various configurations for user identification on a Multitouch-table. The cricket infrastructure that is deployed for user identification consists of the following elements:

(1) Beacons: The infrastructure consists of four beacons deployed on the ceiling. The beacons are encoded with distinct space IDs so that individual distance measurements from them can be identified. The beacons are powered using a dc source.

The beacons are mounted on the 80-20 mount, which is attached to the ceiling. The beacons are attached to the mount using a Velcro tape and are mounted such that they do not form a rectangle. This is done so because mounting the cricket on the four corners of a rectangle leads to a set of degenerate equations. The 3D co-ordinates of the beacons are manually measured (instead of using the auto-localization algorithm provided by cricket) so that accurate measurements of listener positions can be

achieved. One of the corners of the mount is used as the origin and the positions of the beacon are calculated according to that. Table shows the co-ordinate positions of the beacons used in the setup:

Beacon Id	Co-ordinate position (units in cm)
Beacon 1	66.0 , 2.0
Beacon 2	280.0 , 2.0
Beacon 3	252.0 , 212.5
Beacon 4	49.0, 182.5

Table 1: Coordinate positions of various beacons above the Multitouch table.

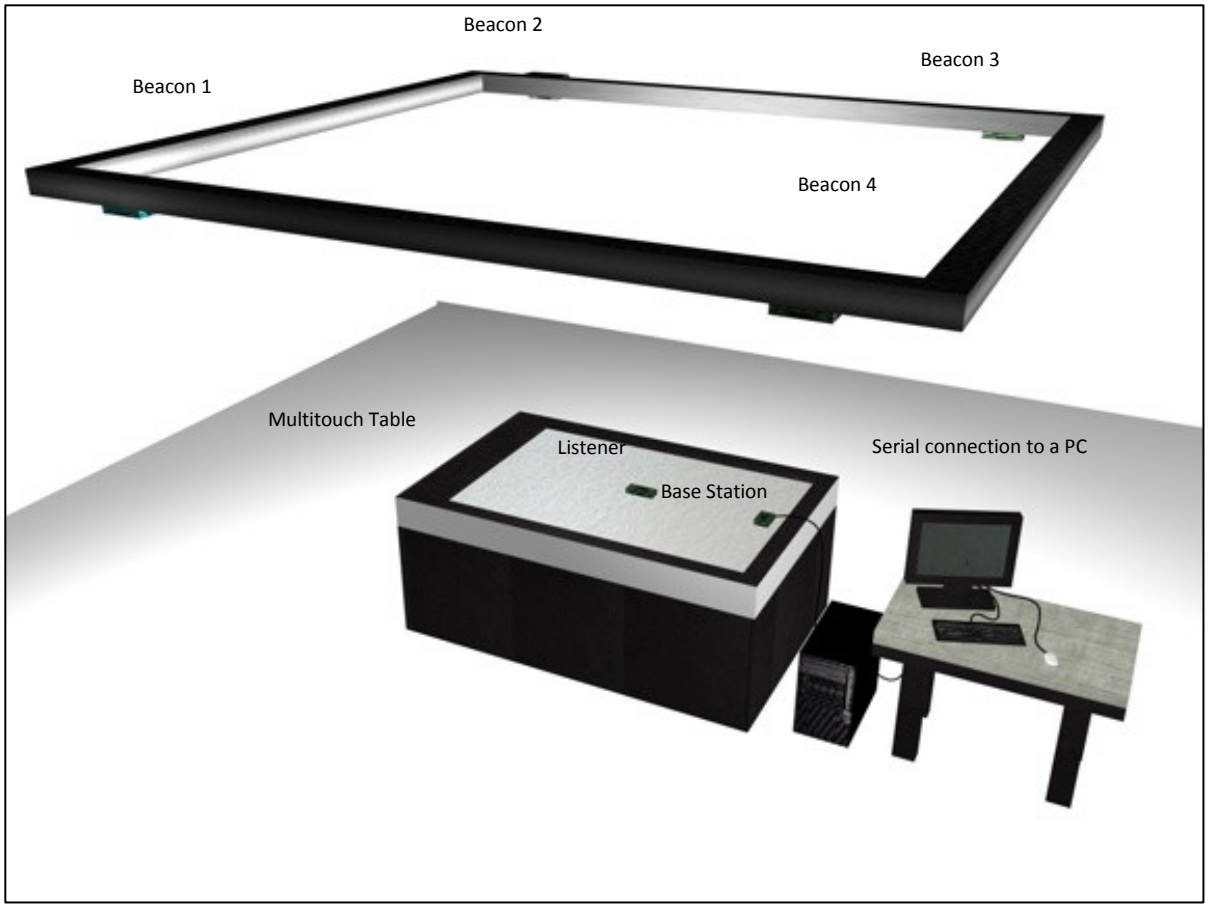

Figure 7: Cricket based Infrastructure for User Identification on a Multitouch Table

The beacons are configured to send the transmissions at regular intervals of 800 milliseconds. Also the arrangement is such that there is an obstacle-free path from the beacons to the multi-touch table's touchable surface.

Listener:

The Listener is strapped on to the hand using a Velcro strap as shown in Figure 8. The Listeners listens to the transmissions from beacon, compute the distance to the beacon, and transmit the distance measurement and the beacon's space id over the RF link to the base station. The Listener aggregates the distance measurements to the beacons, and then transmits the aggregate data in order to reduce the number of packets transmitted, which minimizes the chances of collisions.

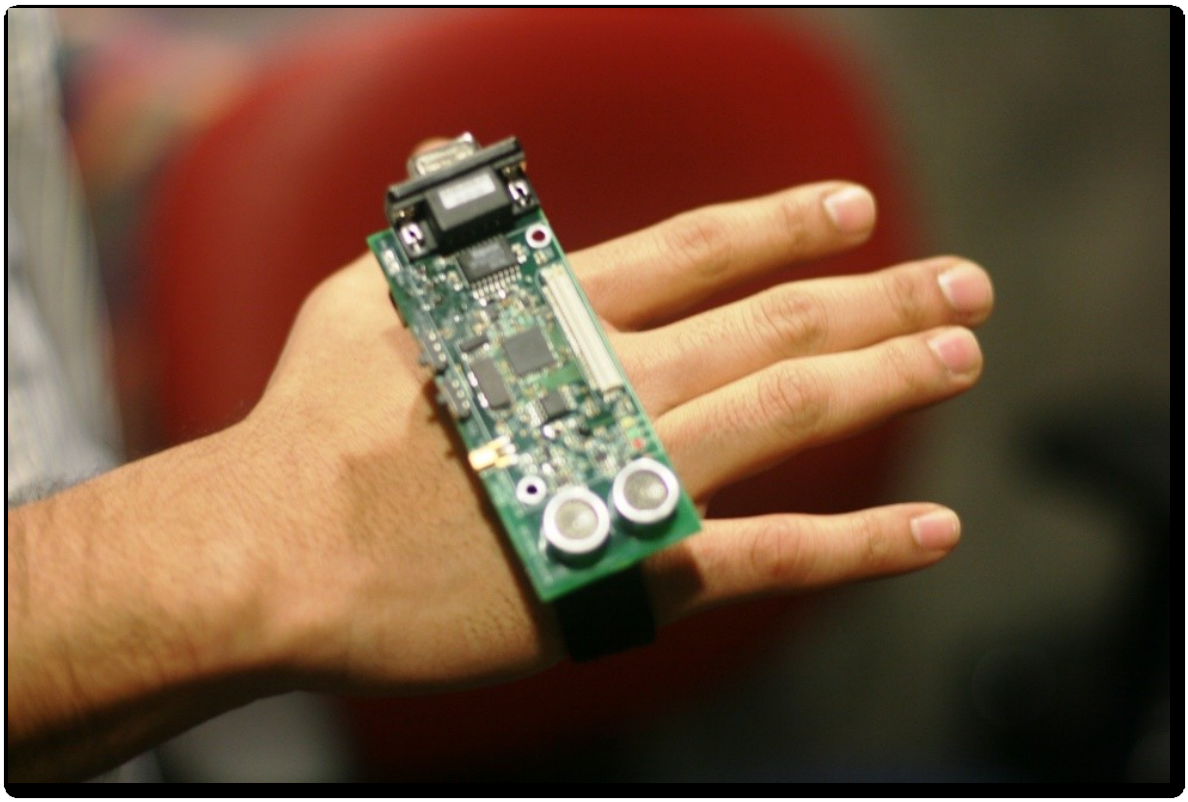

Figure 8: A Cricket Listener mounted on user's hand using Velcro strap.

Base Station: The base station is a cricket node, which listens to transmissions from all the Listeners and transmits them to a PC over a serial interface. Thus the base station acts as a bridge between the crickets and the computer.

PC: The PC runs the cricket-based user identification software and Sparsh UI – the Multitouch gesture recognition system. The software (explained in later sections) associates each touch with a user based on Listener locations and supplies the gesture and user ID information to the application.

3.3. Cricket Based User Identification (CrUId) Software Architecture

In this chapter, we describe the software architecture of "Cricket based User Identification" system. The system essentially consists of three main parts to it:

1) Cricket-related modules : Responsible for extracting the information from cricket sensors and processing the extracted information to obtain User id information
2) Sparsh related modules: Responsible for extracting information from Multitouch device and processing gesture information. (1) and (2) communicate with each other for associating touch with a user.
3) Multitouch application(s): Communicate with Sparsh for gesture information and user id information.

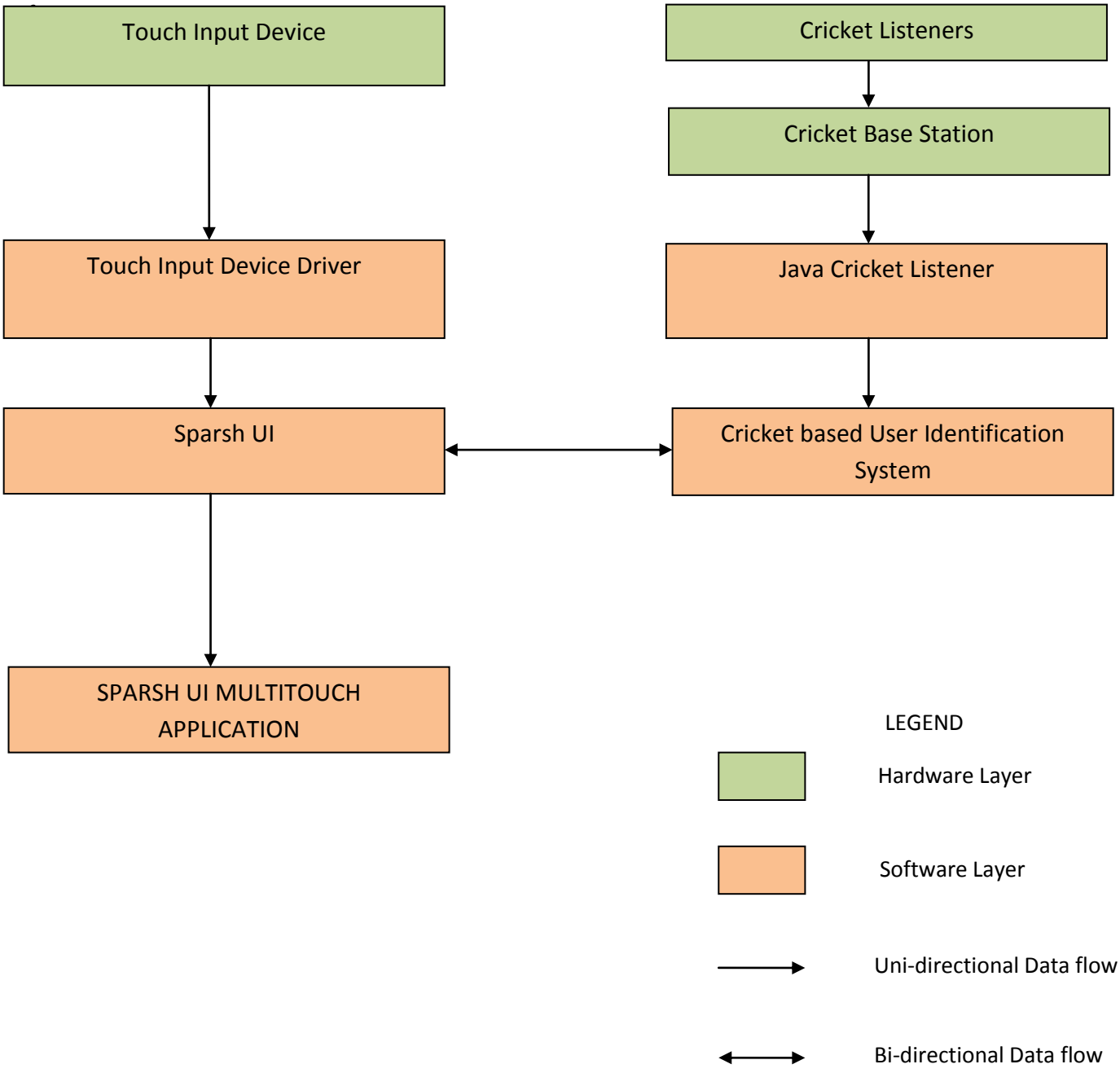

Figure 9: CrUId Software Architecture

The software architecture for our user identification architecture is guided by the following criteria:

- Hardware agnostic: The software architecture is modularized such that it would work with any hardware as long it follows the specified protocol.
- Platform independent: The application is written in Java so that it can be used in any operating system platform.
- Easy plug ability with multi-touch gesture recognition engine.

In order for a user identification mechanism to be useful to write multi-touch applications, it needs to be integrated with a multi-touch gesture recognition engine. As described earlier, we chose SparshUI [21][25] - multi-touch gesture recognition engine, which is platform independent and hardware agnostic. (Refer to chapter 5 for details on Sparsh UI).

Figure 9 shows the schematic of the software architecture of Cricket based User Identification (CrUId) system. The cricket software consists of a Java cricket Listener which listens over the serial interface, and responsible for receiving the distance measurements of Listener to each beacon. It also computes the 3-D co-ordinates of each Listener by employing triangulation techniques.

The Java Cricket Listener is available as a part of open source distribution. This is modified to track multiple Listeners and send the co-ordinate information over sockets to the CrUId module.

CrUId Module

The CrUId module is the central component of the user identification software. It performs the following main functions:

- Implements the algorithm to associate the user based on the touch point and Listener locations.

- Implements the necessary logic to detect the entry and exit of users from the multi-touch system.
- Establishes communication with the multi-touch gesture recognition engine SparshUI.

The CrUId module gets the touch point (x, y) co-ordinate at each touch point birth (touch point down – the event related to creation of the touch point when the finger comes in contact with the screen) as input from SparshUI. Since the same user would continue to be associated with that touch point until touch point death (touch point up – the event related to release of finger from the touch screen) occurs, it is sufficient only to associate the user with the touch point generated when the user touches the multi-touch interface.

User id estimation algorithm

Estimating the user ID given the user locations from the touch table and the Listener positions is not a trivial problem.

Naïve approach for User Id estimation: The naïve approach of associating user id with a touch point is a simple two step procedure. In this algorithm, the distance readings from listener are always assumed to be stable and also it assumes that the distance measurements are current at any given instant. The algorithm consists of computing the distance of each listener to the touch point, and then the listener with the smallest distance to touch point co-ordinate is associated with the touch point.

This naïve approach performs poorly because of the slow update rate of listeners and owing to the fact that the spatial resolution of Cricket sensors is around 5 cm. The multi-touch environment poses the following challenges to the cricket system for a continuous user- ID association with the touch points:

```
WHILE ( LatestListenerCo-ordinate[i] exists )
        Transform the co-ordinates from cricket co-ordinate space to table co-ordinate space
         Distance  =   TouchPoint ~ LatestListenerCo-ordinates[i]
         DisplacementFactor = MeanPosition[i] ~ ListenerCo-ordinates[i]
        If( Distance <  STABLE_TPLP_THRESHOLD )                          /* case 1 */
        stableReadingfound = true;
          if(distance < closestListenerDistance)
          {
           closestListenerDistance = distance;
           matchingUserID = currentListenerID;
          }
         Endif
        ELSE IF (distance < MEDIUM_TPLP_THRESHOLD)
        IF ( displacementFactor < DISPLACEMENT_THRESHOLD1)               /* case 2 */
        stableReadingfound = true;
         IF ( distance <  closestListenerDistance )
           closestListenerDistance = distance;
           matchingUserID = current listener id;
          end IF
        END IF
        ELSE IF (displacementFactor < DISPLACEMENT_THRESHOLD2)
                add listener id to unstable listeners list
         IF
        END IF
        ELSE IF (distance < UNSTABLE_TPLP_THRESHOLD)
          IF(displacementFactor > DISPLACEMENT_THRESHOLD1 && displacementFactor <
        DISPLACEMENT_THRESHOLD2)
                unstableListeners.add((Integer)Key);
           END IF
        END IF

        IF ( stablereading not found and unstablelisteners list is not empty ) .   /* case 3 */
        closestExPtDist = 10000;
        FOR each unstable listener location
         Draw a line form mean of listener locations to the current  listener location
          Extend the line by EX_DISTANCE
           Compute perpendicular distance( pDist) of the point to line
            IF pDist < closestExPtDist
              closestExPtDist = dist
              userId   = currentUnstableListenerID
           END IF
        END FOR
          IF ( closestExPt < MEDIUM_TPLP_THRESHOLD )
                  matchinguser Id = userId
          END IF
        END IF
End While
```

Figure 10: Pseudo code of Linear Extrapolation based User id Estimation Algorithm

1. The location tracking via the Listeners is discrete; the coordinate position of the Listener can be computed only when the distance measurements to all the beacons are obtained. Since hand movements are fast and often over very small distances, we may not have the latest Listener location measurements all the time.

2. When the distance between the users' hands is of the order of the resolution offered by the cricket system (i.e. around 4 cm), then the naïve approach for user ID association would not work well. Our experiments showed that the wrong user was associated with touches more than 50 % of the time when the users' hands are close by to each other.

Figure 10 provides the pseudo code of the user ID estimation algorithm. The algorithm maintains a list of current Listener locations, and a list of the mean of the last five positions for each Listener. We define "Displacement Factor" for a given Listener position reporting as the measure of the degree of movement of the listener (user's hand) in the recent past. It is the displacement from the mean of last five positions of the listener to its current position.

We also define two classes of thresholds: one class serves to compare the present lag between the touch point co-ordinate and the listener position namely STABLE_TPLP_THRESHOLD, MEDIUM_TPLP_THRESHOLD and UNSTABLE_TPLP_THRESHOLD. The second class relates to Displacement Thresholds namely DISPLACEMENT_TH_LOWER and DISPLACEMENT_TH_UPPER which represent the lower and the upper thresholds of Displacement factor respectively. The algorithm approaches the problem of associating the Listener with a particular user by evaluating the following cases:

1. The Listeners have stable readings; this is the best case where there is minimal lag between the Listener and the touch point. For this to happen, the distance (D_{TpL}) between touch point and the Listener location should be less than the stable threshold

2. The Listeners have moderately stable readings. This means that D_{TpL} is less than the medium threshold and the displacement of the current Listener location from its mean (termed as Displacement factor) is less than lower displacement threshold. This double check eliminates readings which have significant lag and fast hand movements.

3. The third case essentially takes care of the scenario in which a significant lag between the current hand position and the mean of previous positions occurs. To arrive at the best match, we use an approximation based on Linear Extrapolation. The idea is to find the Listener location that best fits the touch point given the path the hand has traversed in the recent past. To do this we draw a straight line from the mean location of the past five positions of the Listener and current location. To account for the lag we extrapolate this straight line by a certain fixed length, which is derived empirically. This is repeated for all the Listeners, which have unstable readings. The path with which the touch point proves to be the best fit gives the Listener that is best match to the touch point. For higher accuracies, a quadratic curve fitting approach may be employed, but we have found that the liner extrapolation method suffices our needs. If we do not arrive at a User Id measurement after all these steps, we wait for 500ms so as to allow for additional Listener reading(s) to arrive and then repeat the process again.

CHAPTER 4. EVALUATION OF CRICKET BASED USER IDENTIFICAITON SYSTEM

4.1. Enhanced Interaction Affordances Enabled by the Cricket System

The cricket system opens up new modes of interaction with the multi-touch interface. In this section we discuss three example applications and some of the new interaction affordances enabled by our system.

1. **User awareness**

The Cricket system can determine the entry or exit of the user from the vicinity of the multi-touch interface. The CrUId system continuously monitors for any new listener IDs being transmitted to the base station. If a new listener is detected, a check is made to see if it lies within a certain boundary. For our setup, we report a "User Entry Event" if the position reported is in a bounding box twice the size of the multi-touch interface. Similarly if the user is outside this bounding box beyond a period of two minutes the CrUId system reports a User Exit Event. The applications can be made user-aware. To demonstrate this feature we developed a Photo application (shown in Figure 11) that is user-aware. If user "A " is in the vicinity of the system that particular user's are displayed on the interface, now if user "B" arrives near the system his / her photos are displayed as well. Individual users' photos are distinguished by differently colored borders. If a user leaves the vicinity of the system for duration of more than a predetermined value, his / her photos are removed from the display. Thus the application is made user aware.

2. **Object ownership**

The cricket system provides a mechanism to authenticate the user as each Listener transmits a distinct ID. We define two modes of object ownership that is enabled by our system.

3. **Fixed ownership**

Here the UI widgets are associated with a particular user throughout their lifetime. Examples of this include where exclusive ownership of widgets is needed all the time. Figure 12 shows the Paint

application where there is a shared canvas and individual palettes for users. Each user can choose their particular color and brush size. While the users can paint anywhere on the canvas, only the owner of the palette can change the color or the brush size.

Figure 11: User aware multi-touch Photo Application

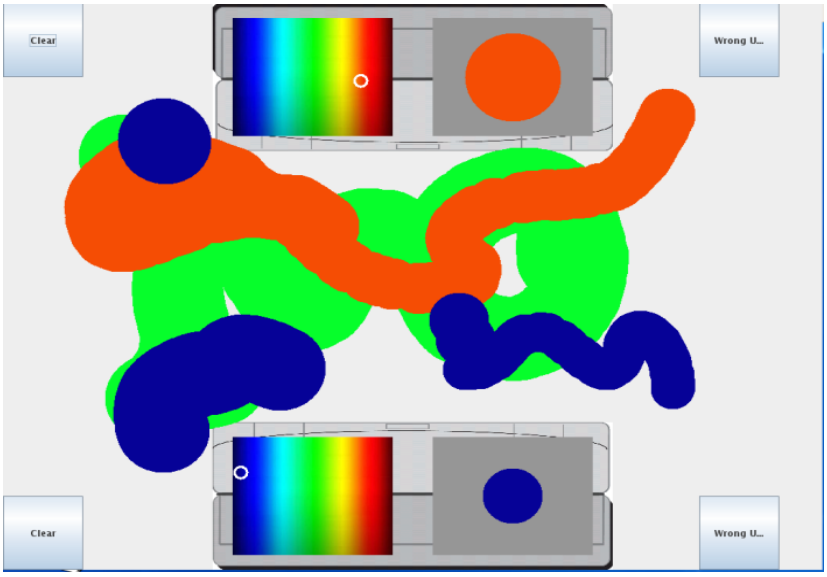

Figure 12: User aware multi-touch Paint Application

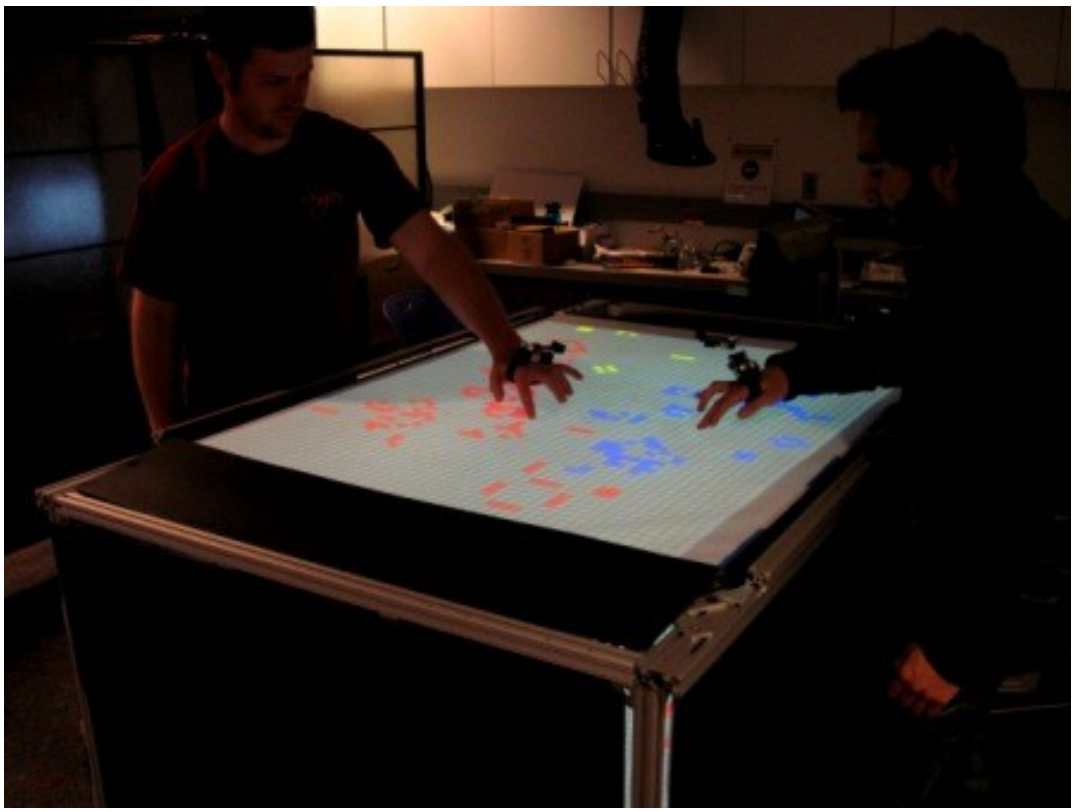

Figure 13: User Identity enhanced Conway's game of life.

4. Dynamic ownership

Since the cricket system can detect if multiple users are interacting with a widget, it is possible to transfer the ownership of UI objects from one user to another. For example, in our photo application, initially each photo is associated with a particular user and only that user has permission to interact with the object. But the user can transfer the ownership of the photo to another user using a special gesture (explained later in this section). The transfer can be temporary or permanent depending on the application needs. Thus we have dynamic ownership of UI objects.

5. Region / workspace ownership

This refers to the ownership of the workspace on the multi-touch interface. With the cricket system it is possible to physically or virtually demarcate the available workspace on the multi-touch

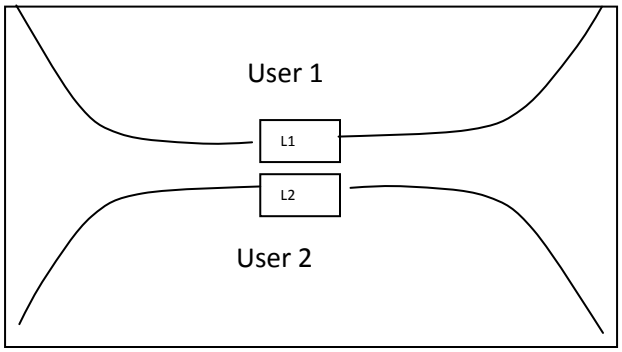

Figure 14: Workspace ownership using Crickets

interface. To provide a sense of physical demarcation, users can place their respective Cricket Listeners on the surface (as shown In Figure 14). All the touches that occur in the region "User 1 space" can be associated be with user 1 and the touches that occur in User space 2 can be associated with user 2.

A virtual demarcation is also possible by detecting the number of users interacting with the system and configuring the user space based on the number of users present. Thus cricket allows defining the regions associated with various users either "physically" or "virtually".

6. Gestures above the table

Since the cricket system is capable of tracking the users in a 3-D space, we can use the z-axis information of the user's hand to compute 3-D gestures.

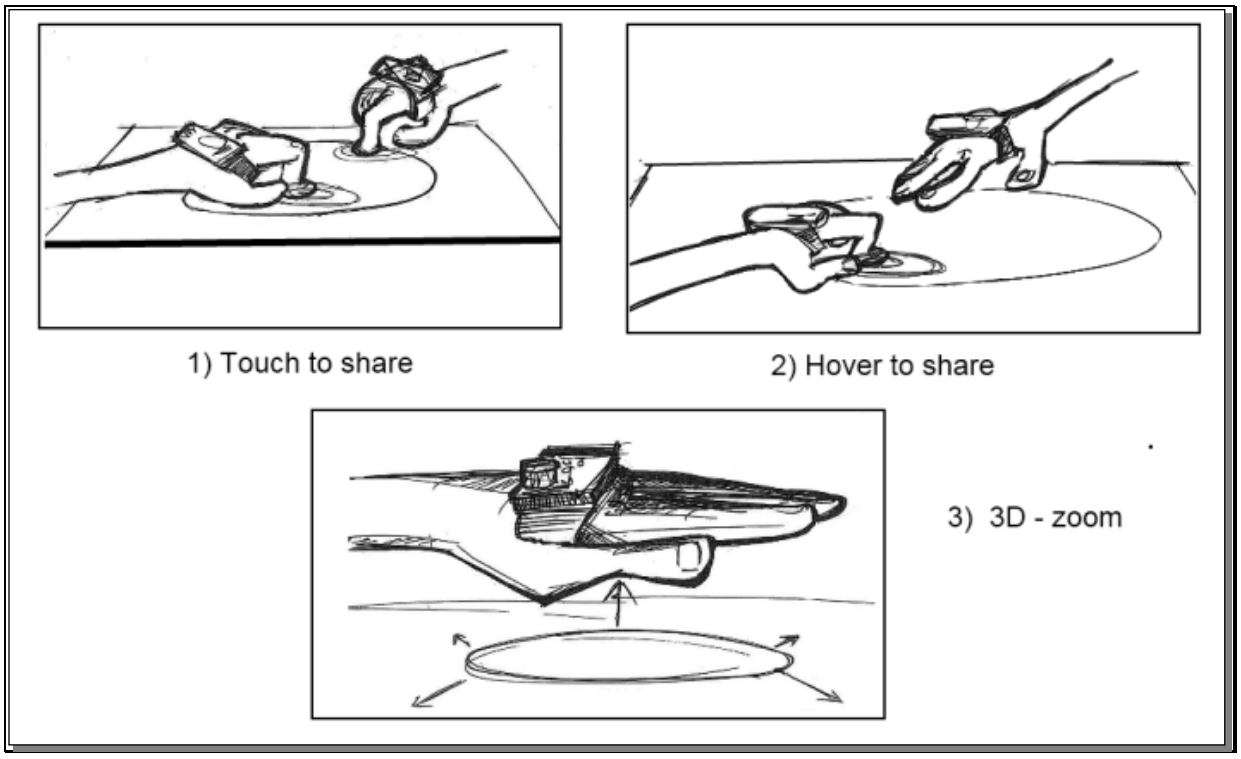

Figure 15: Some of the special gestures enabled by Cricket system

The schematic representation of the special gestures supported by the cricket system is shown in Figure 15 and is explained below.

a. **3-D zoom:** This gesture is performed by placing the hand still above the surface of the table for three seconds and then moving the hand vertically away from or towards the table to zoom in or out respectively. The challenge in implementing this gesture is to avoid false positives for a hover. To minimize false positives we use a two step approach to confirm a hover. The first condition necessitates that the height of the listener (hand) relative to the table should be above a particular threshold. The value of this threshold is fixed at 15 cm and is derived from empirical observations. The second condition is that the hand should be held in this position for at least 3 seconds. These two conditions minimize the false alarms. The paint application uses the 3-D zoom gesture to alter

the size of the paint brush. While painting on the multi-touch surface, users can vertically move their hand away or towards the surface to increase or decrease the size of paint brush size. Figure 16 shows a user changing the brush size by vertically moving one hand away from table while painting with the other hand. Such gestures would prove to be useful if the multi-touch interface is large.

Figure 16: 3D-zoom gesture with Cricket

b. **Hover to share gesture**: This gesture is used to transfer the ownership of an object from one user to another. To transfer ownership, the user seeking access or ownership places his hand on the object first and then the current owner of the object hovers his hand on that object. A time limit can be set on the duration of this "Virtual Handshake" for the ownership transfer to occur. A non-zero value for this limit can be used to avoid any false positives. The photo application uses the virtual handshake gesture to grant ownership of the photo to another user. Figure 17 shows a user transfer the ownership of a photo to another user using this gesture.

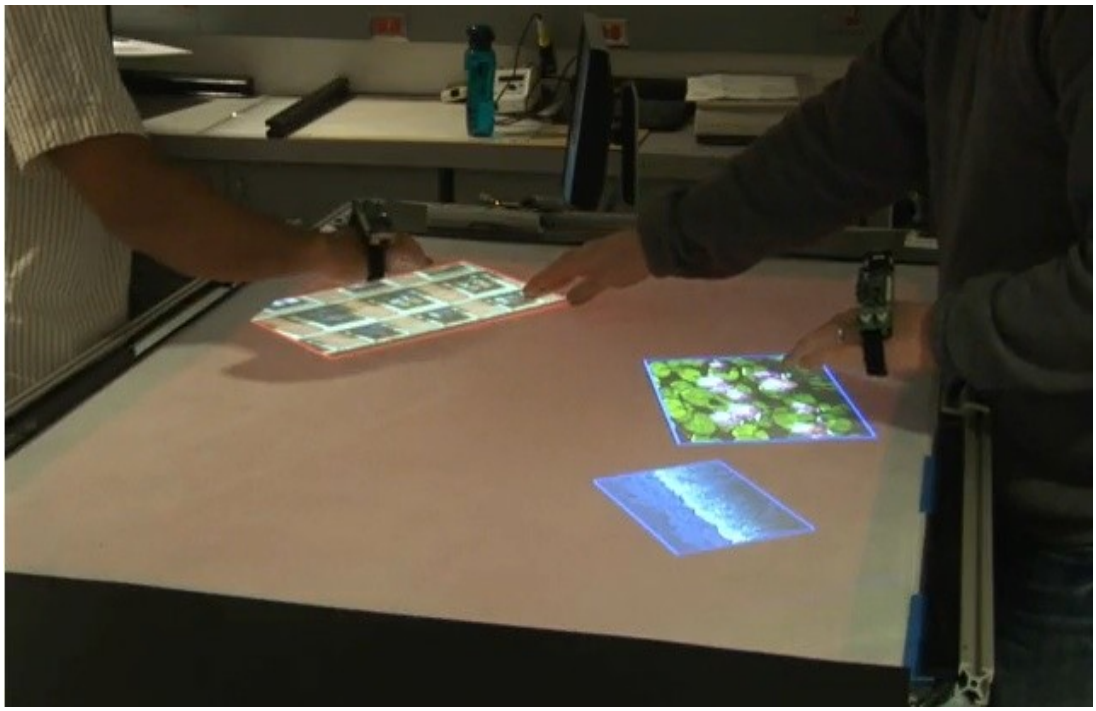

Figure 17: Ownership transfer using "Hover to share" gesture in a photo app

c. **Touch to share gesture:** Touch to share gesture is similar to the "hover to share gesture" except that the owner of the object needs to touch his hand above the object while the other user is touching it. A similar gesture is described in [16]. The time duration for which the owner needs to hover for transferring the ownership can be configured depending on the application. Figure 18 shows a user transfer the ownership of a photo to another user using this gesture.

Figure 18: Ownership transfer using "Touch to share" gesture in a photo app

4.2. Evaluation of Linear Extrapolation Based Algorithm

In this section, we evaluate our algorithm for identifying users with the Cricket location sensors. We also provide a feature-by-feature comparison of our system with other user identification systems based on the criteria we established earlier in the paper.

To evaluate the accuracy of our algorithm for identifying users, we chose to utilize the Paint application described above. For the study, we selected twenty-two participants, all college students between the ages of 18 and 24. The participants were divided into pairs, and each pair completed two fifteen-minute long sessions of painting on the multi-touch table. For one session, the system employed the naive algorithm discussed above, and for the other session, the system employed our linear extrapolation algorithm. The researchers placed no restrictions on the movement around the table of the participants, and the participants were free to draw anywhere on the multi-touch table.

To evaluate the accuracy of the algorithm, we added a logging mechanism to the application, which would record the number of touch birth events throughout the duration of the session. Users were asked to record any mismatch in color (i.e. a wrong user was associated to the touch) by pressing a button next to the color palette, as seen in Figure 6. By this approach, we were able to gather how many miss-identifications were made by the system. Users were encouraged to choose distinct colors so that easy identification of mismatches is possible. A logging mechanism was added to the application, which would record the number touch birth (touch down) events throughout the duration of the session. Users were asked to record any mismatch in color (i.e. a wrong user was associated to the touch) by pressing a button next to the color palette, as seen in Figure 12.

The results of this evaluation are shown in Figure 19. The x-axis represents the number of touch point births (touch point down) throughout the duration of the session. The y-axis represents the number of erroneous associations of the wrong user for a touch point during a particular session. As the number of touch points increases, the rate at which the number of errors in the naive algorithm increases is significantly higher than that of our linear extrapolation based algorithm. Figure 20 compares the growth rate of the error counts of the two using trend lines. We can see that the error count increases exponentially with the number of touches increase for the Naïve approach while it increases linearly for the linear extrapolation approach. If we approximate the exponential curve with a linear equation one can see that the linear extrapolation approach outperforms the naïve approach by around 8 times.

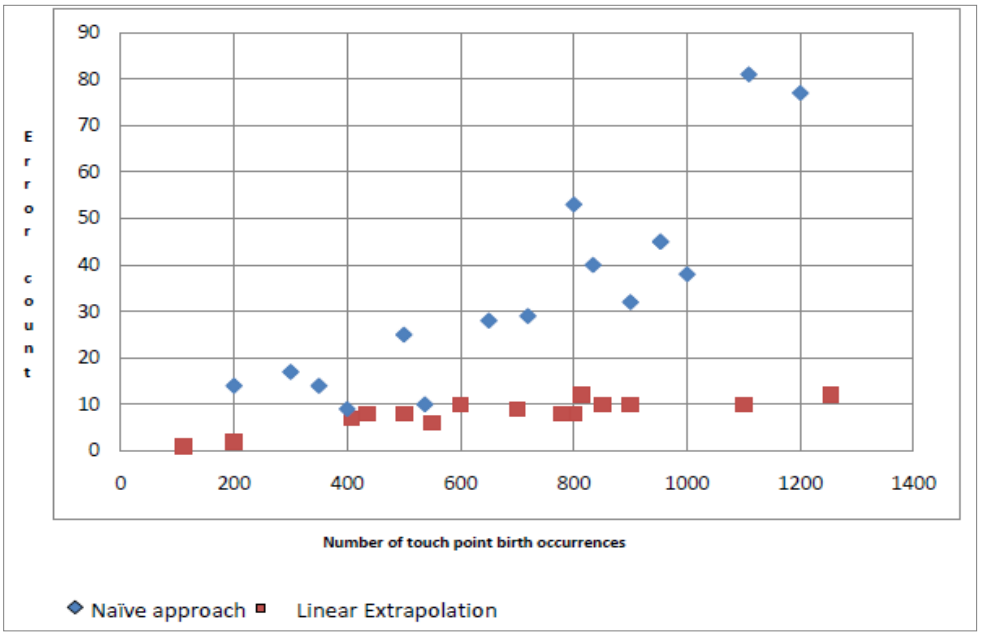

Figure 19: Comparison of Linear extrapolation algorithm and Naïve approach

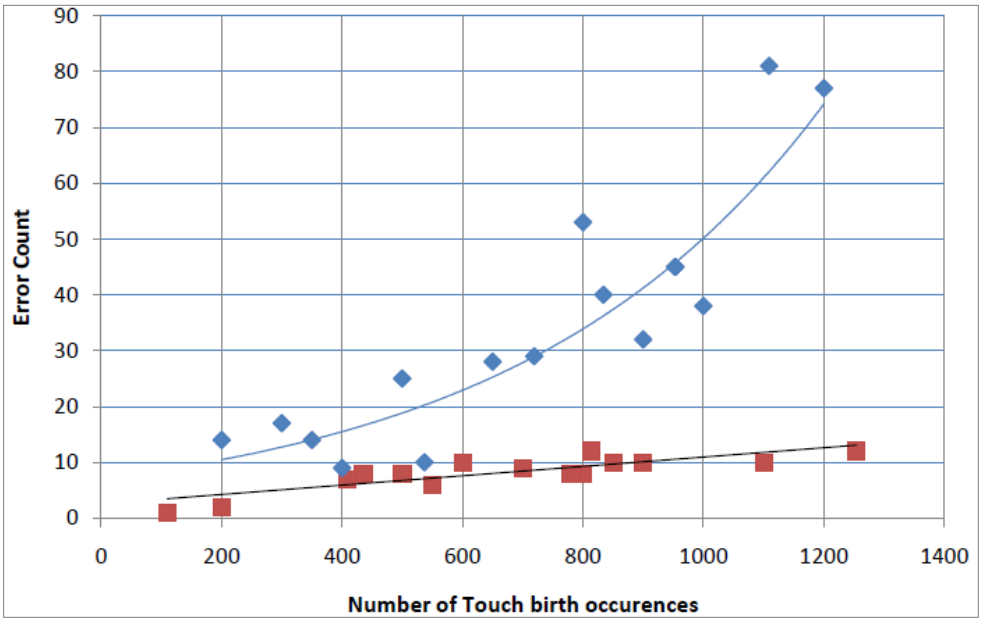

Figure 20: Comparison of error count growth rate of the two approaches

User Identification System → / Features ↓	Cricket Based System	Diamond Touch	Show your hands (vision based approach)	Proximity sensors based method	Mobile phone based method
Multiple hardware support	Yes	No Requires specific hardware	Limited, Cannot easily support multi-touch walls	Table type hardware, does not support walls	Only FTIR based
User mobility support	Yes	No. Users are tied to the chair they are using	Yes	Yes. Only user presence detection. does not identify users	Yes
Presence and proximity detection	Yes	No	No	Yes	No
Continuous tracking	Yes	Yes	No	Yes	No
Special gestures support	Yes	Yes, Limited support. No Hover or 3d-zoom detection.	Yes, Limited support. No 3d-zoom	No	No
Cross platform software support	Yes	information not available	information not available	Information not available	Information not available
Extensibility to other immersive environments like caves	Yes	No	No	No	No

Table 2: Comparison of various user identification architectures

Table 2 shows a comparison of various user identification systems based on the criteria that we discussed earlier. As can be seen, the cricket system provides a richer experience than the other systems.

CHAPTER 5. CONCLUSIONS AND FUTURE WORK

In this work, we presented a robust sensor-based user identification system that can be used with any large multi-touch interactive surface. Our approach is easy to deploy and can be used with other immersive environments as well, such as a four-walled VR cave system. In this case, Cricket sensors could easily be deployed above the users to track hand or user positions.

Additionally, we discussed the various enhanced interaction methodologies and affordances provided by this system. Our user-identification based system has additional and unique advantages over other user-identification systems in that it can detect and use novel gestural interaction techniques on top of multi-touch tables. Finally, we evaluated the performance of our user identification algorithm by comparing its accuracy against a simpler algorithm, and provided a comparison of some of the features of our system compared to other well-known systems, which provide user identification.

A limitation of our current system is the resolution of the Cricket-based location system, which provides a precision of 4-5 cm. Though this precision is acceptable when working with large multi-touch surfaces, adapting our system to smaller multi-touch devices such as tablet PCs might present difficulties for the system as it attempts to resolve individual users. However, this limitation is mitigated by the fact that applications involving the participation of multiple users can be expected to take place on multi-touch systems of sufficient size to accommodate the physical profiles of the multiple users. Future work has already been undertaken to expand our system to be compatible with multi-touch hardware and software platforms. As described in the paper, the system is already compatible with the popular Sparsh-UI gesture recognition framework. We have also written a device driver for our user identification system, which makes it compatible AQUA-G universal gesture recognition framework [12]. AQUA-G is a gesture recognition framework, which supports multiple simultaneous input devices, is platform and language-independent, and provides support for gesture recognition not limited to multi-

touch gestures. Making our system compatible with AQUA-G will make our system accessible to a wider development community.

Other future work includes upgrading the Cricket location sensors to smaller and less expensive sensors. Upgraded sensors could be used with the same user-identification algorithm and Sparsh-UI integration as described in this paper, but could serve as an effective upgrade to increase the precision of the user-identification system. Sensor network research is a very active research area and new hardware platforms become available quite rapidly. Wireless sensor cryptographic protocols can be used to develop authentication mechanisms to allow secure user authentication. Finally, we are currently working on expanding the use of Cricket-based systems to other virtual environments such as four and six-sided VR caves, where it can be used to research exciting new gesture-based interactive applications.

CHAPTER 6. BIBLIOGRAPHY

1. B. Walther-Franks, L. Schwarten, J. Teichert, M. Krause, and M. Herrlich, "User detection for a multi-touch table via proximity sensors," in Proceedings of IEEE Tabletops and Interactive Surfaces, Amsterdam, The Netherlands, 2008.
2. Christoph Ganser , Adrian Steinemann , Andreas Kunz, InfrActables: Multi-User Tracking System for Interactive Surfaces, Proceedings of the IEEE conference on Virtual Reality, p.253-256, March 25-29, 2006
3. Cricket based user identification informational video: http://www.youtube.com/watch?v=YUHAmp1Y5cs
4. Cricket based user identification technical video: http://www.youtube.com/watch?v=mDefMhoWpeU
5. Cricket Manual : http://nms.lcs.mit.edu/projects/cricket/v2man.pdf
6. Crossbow : http://www.xbow.com
7. Dietz, P. and Leigh, D. DiamondTouch: a multi-user touch technology. In UIST '01 (2001), 219–226.
8. Diffused Illumination http://nuigroup.com/wiki/Diffused Illumination Plans/
9. Dohse, K.C.; Dohse, T.; Still, J.D.; Parkhurst, D.J., "Enhancing Multi-user Interaction with Multi-touch Tabletop Displays Using Hand Tracking," Advances in Computer-Human Interaction, 2008 First International Conference on, vol., no., pp.297-302, 10-15 Feb. 2008
10. Goldberg, F. Halasz, W. Janssen, D. Lee, et al. Liveboard:a large interactive display supporting group meetings,presentations, and remote collaboration. Proceedings of the SIGCHI conference on Human factors in computing systems,pages 599–607, 1992)
11. Han, J. Y. 2005. Low-Cost Multi-Touch Sensing Through Frustrated Total Internal Reflection. In Proceedings of the 18th Annual ACM Symposium on User interface Software and Technology (Seattle, WA, USA, October 23 - 26,2005). UIST '05. ACM, New York, NY, 115-118
12. J. Roltgen, "AQUA: a universal gesture recognition framework," Iowa State University, 2010. In Press.
13. J. Schoning, M. Rohs, and A. Kr□uger. Using mobile phones to spontaneously authenticate and interact with multi-touch surfaces. In PPD'08. Workshop on design-ing multi-touch interaction techniques for coupled public and private displays, May 2008.
14. J. Y. Han. Low-cost multi-touch sensing through frustrated total internal reflection. In UIST, pages 115–118, 2005.
15. Jeff Han: Unveiling the genius of multi-touch interface design http://www.ted.com/index.php/talks/jeff_han demos his breakthrough touchscreen.html
16. Morris, M. R., Huang, A., Paepcke, A., and Winograd,T. Cooperative gestures: multi-user gestural interactions for co-located groupware. In CHI '06(2006), 1201–1210.
17. Motamedi, N. 2008. Hd touch: multi-touch and object sensing on a high definition lcd tv. In CHI '08 Extended Abstracts on Human Factors in Computing Systems (Florence, Italy, April 05 - 10, 2008). CHI '08. ACM, New York, NY, 3069-3074. DOI= http://doi.acm.org/10.1145/1358628.1358809
18. N. B. Priyantha. The Cricket Indoor Location System. PhD thesis, MIT, May 2005.
19. Oren, M. & Gilbert, S. (2009). ConvoCons: Encouraging Affinity on Multitouch Interfaces. Proceedings of Human-Computer Interaction International 2009. San Diego, CA.
20. P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, "TinyOS: An Operating System for Sensor Networks," Ambient Intelligence, 2005, pp. 115-148.

21. Prasad Ramanahally, Stephen Gilbert, Thomas Niedzielski, Desiree Velazquez, and Cole Anagnost. Sparsh UI: A Multi-Touch Framework for Collaboration and Modular Gesture Recognition. ASME Conf.Proc.2009,137(2009)

22. Priyantha, N. B., Chakraborty, A., and Balakrishnan, H. 2000. The Cricket location-support system. In Proceedings of the 6th Annual international Conference on Mobile Computing and Networking (Boston, Massachusetts, United States, August 06 - 11, 2000). MobiCom '00.

23. Schmidt, Dominik and Gellersen, Hans (2009) Show Your Hands: A Vision-Based Approach to User Identification for Interactive Surfaces. In: Interactive Tabletops and Surfaces, 23-25 Nov 2009, Banff, Canada.

24. Smith, A., Balakrishnan, H., Goraczko, M., and Priyantha, N. 2004. Tracking moving devices with the cricket location system. In Proceedings of the 2nd international Conference on Mobile Systems, Applications, and Services (Boston, MA, USA, June 06 - 09, 2004). MobiSys '04. ACM, New York, NY, 190-202. DOI= http://doi.acm.org/10.1145/990064.990088

25. SparshUI: http://code.google.com/p/sparsh-ui/

26. TUIO : http://www.tuio.org/