Increasing Situational Awareness using Smartphones

Sanjay K. Boddhu[a], Robert L. Williams[b], Edward Wasser[c], Niranjan Kode[d]
[a]Qbase, sboddhu@4qbase.com;
[b]Tec^Edge Innovation and Collaboration Ctr. and Air Force Research Lab,
Robert.Williams@wpafb.af.mil;
[c]Qbase, ewasser@4qbase.com;
[d]Qbase, nkode@4qbase.com

ABSTRACT

In recent years, the United States Armed Services and various law enforcement agencies have shown increasing interest in evaluating the feasibility of using smartphones and hand-held devices as part of the standard gear for its personnel, who are actively engaged on battlefield or in crime-prone areas. The primary motive driving analysis efforts to employ smartphone-based technologies is the prospect of the increased "Situational Awareness" achievable thru a digitally connected network of armed personnel. Personnel would be equipped with customized smart applications that use the device's sensors (GPS, camera, compass, etc...) to sense the hostile environments as well as enabling them to perform collaborative tasks to effectively complete a given mission. In this vein, as part of the Summer At The Edge (SATE) program, a group of student interns under the guidance of mentors from Qbase and AFRL, have employed smartphones and built three smart applications to tackle three real-world scenarios: PinPoint, IStream, and Cooperative GPS. This paper provides implementation details for these prototype applications, along with the supporting visualization and sensor cloud platforms and discusses results obtained from field testing of the same. Further, the paper concludes by providing the implications of the present work and insights into future work.

Keywords: smartphones, situational awareness, GPS, video streaming, sound localization, persistent surveillance and real-time, collaborative cloud architecture

1. INTRODUCTION

Ideally, complete situational awareness is possible with multiple sensors of different modalities capturing the activity in an area or tracking a target in the context accurately and then reliably communicating the exploited information back to the personnel in field in near real time. However, practical issues like network infrastructure and sensor deployment logistics posed challenging problems in achieving effective situational awareness. For a half decade, the Tec^Edge Innovation and Collaboration Center (ICC) has created a stimulating environment for government agency, university, and industry teams to explore challenging problems in national defense and public safety related to situational awareness and situational understanding through the development of effective sensor data acquisition , integration, and dissemination platforms.

There have been several collaborative research projects such as Sensor Aided Vigilance (SAVIG), Persistent Sensor Storage Architecture (PSSA), and Centralized Operational Web Promoting Animal Traceability and Health (COWPATH). These have all been conducted at the ICC facility and have been successful in creating several software and hardware frameworks including the Open Layered Sensor Test bed (OpenLST) and the PSSA Could Architecture. These frameworks were adopted and enhanced under programs such as Summer at the Edge (SATE) and Year at the Edge (YATE). The OpenLST is a dissemination platform developed to provide visualization of various in-field and experimental sensors developed at the facility[10]. The PSSA's Could Architecture[9], as shown in Figure 1, next, is a backend net-centric architecture developed to support live ingestion of various sensor data with appropriate visualization modules in real-time, with capabilities to detect and add new sensors to the OpenLST platforms (shown in Figure 2) seamlessly. In addition, multiple OpenLST visualization platforms (Command center versions and Mobile versions [PocketLST[10]]) were equipped with PSSA modules to simultaneously access the live data and real-time exploitation data from deployed sensors in the field.

Ground/Air Multisensor Interoperability, Integration, and Networking for Persistent ISR III,
edited by Tien Pham, Proc. of SPIE Vol. 8389, 83891J · © 2012 SPIE
CCC code: 0277-786X/12/$18 · doi: 10.1117/12.920035

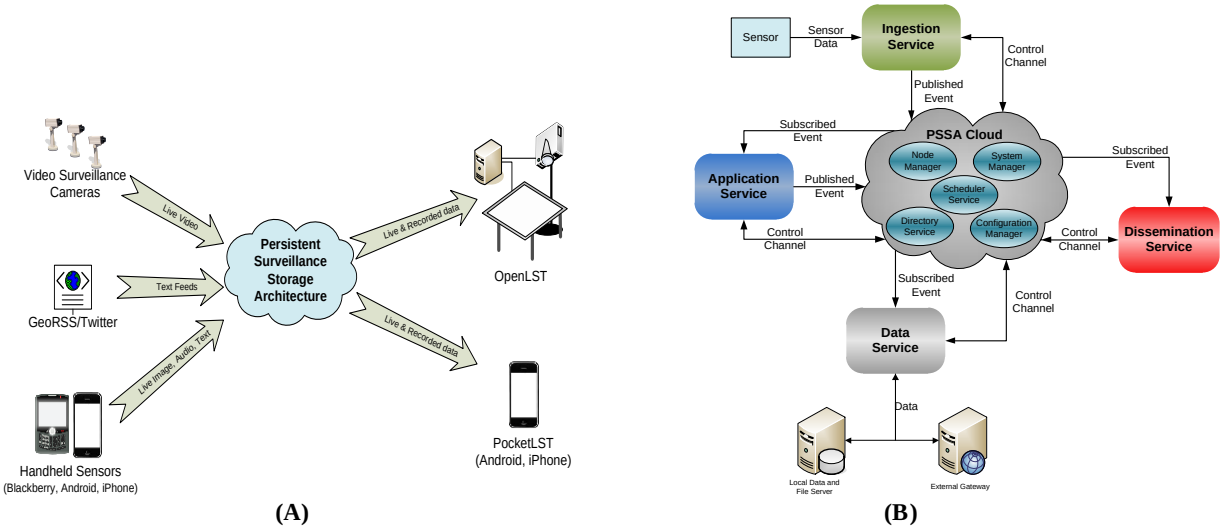
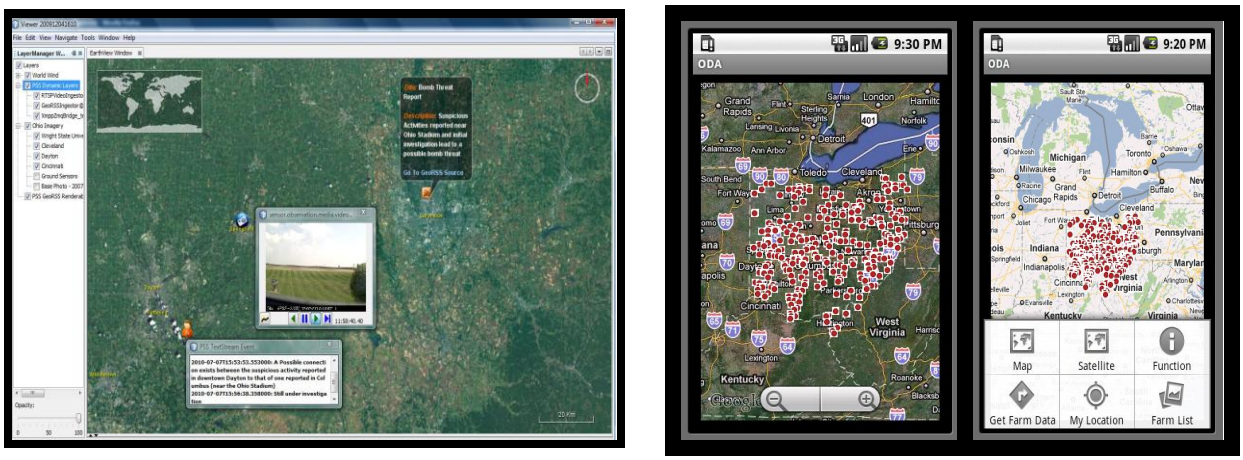


Figure 1. Figure 1(A) provides a conceptual view of Persistent Surveillance and Storage Cloud Architecture (PSSA Cloud). PSSA is a "Plug & Play" event collaboration cloud architecture that allows seamless integration of new sensors and its related services thru event normalization concept. Figure 1(B) provides an internal services view of the PSSA, where every service can be deployed and made available dynamical to other services through a dynamic registration process. Every ingestion service is responsible to register a new senor that becomes active in the field and publishes its data streams (events) to all interested subscribed services including dissemination, application, and storage services. These PSSA-enabled services are distributed across various computing platforms (desktop, server or mobile) appropriately as per the mission requirements. PSSA cloud is easy to deploy in a wired or wireless network environment with minimal core services[9] (shown in the cloud symbol in 1(B)).

In brief, one can perceive the PSSA cloud architecture as an event-based architecture which consists mainly of three concepts mainly publisher, subscriber and event. Each publisher publishes events into the cloud, which are only accessible by the registered subscribers. The architecture provides core services to seamlessly integrate sensors and dissemination platforms[9].

In last two years, the advances in smartphone-related technologies has motivated the collaborative teams to undertake research projects that were aimed at evaluating two interesting concepts in enhancing the situational awareness;(1) to employ smartphone's display capabilities to effectively communicate information from command center to the personnel in the field and, (2) to employ smartphone's existing sensors or further enhance them with external sensors to build human augmented sensors in the field (I.E. enabling the personnel/soldier/first responder as a sensor, communicating the data back to the command center). In previous efforts such as project COWPATH, developed PSSA-enabled smartphone real-time collaborative and dissemination applications have been field tested and the real-time operational integration of these smartphone sensors with the PSSA Could architecture and OpenLST (enabled by collaborative applications) demonstrated that the achievability of above two concepts has indeed enhanced the overall success of a mission in the context. Continuing to demonstrate the effectiveness of these two concepts, research teams have employed smartphones and built three smart applications to tackle three real-world scenarios: (1) PinPoint™ – a collaborative smart application that uses a network of three smartphones and their audio sensors to perform sound localization; (2) IStream – a smart application that uses the camera sensor on the smartphone to stream the camera feed to a localized video server for archival and real-time streaming to Layered Sensing Visualization platforms (like OpenLST command center); and (3) Cooperative GPS – a collaborative smart application that uses the GPS sensor on the smartphone along with external sensor hardware to accurately determine a more precise GPS location. The technical details related to concepts and development challenges of these three applications have been described in sections 2 through 4 in this paper. Further, the paper concludes with discussions of the test results of the applications and provides the implications of the present work and insights into the future work being conducted by the collaborative research teams at ICC facility.



(A) **(B)**

Figure 2. Figure 2(A) provides a snapshot of server/desktop version of OpenLST platform in action, displaying different sensor data (video, images, GeoRSS, chat, twitter, satellite imagery) feeds. PSSA cloud enabled OpenLST can provide a command center view of the area under surveillance (I.E. as a dissemination service) and also provide collaboration capabilities to communicate with other OpenLSTs in the PSSA cloud and act as a human augmented sensor. These two functionalities have been extended to the PocketLST application, snapshot shown in figure 2(B), which has shown to serve as a means to better communicate the surveillance information to the personnel carrying them in the field and also to employ them as sensor platforms being navigated and verified in real-time by humans (the concept of personnel as sensors).

2. PINPOINT

2.1 Overview

Sound localization is a widely-used technique with countless real world applications and particularly in "search and rescue" operations. However, the specialized and expensive hardware restricts its deployment in only urban or controlled environments. The goal of Project Pinpoint is to make this technology inexpensive, accessible, and portable, so that it is available to personnel navigating in an unfamiliar environment. Pinpoint is an Android® application that listens for a sound and determines the location of its source. The sound localization methodology requires four smartphones enabled with PSSA modules (ingestion publisher) and are registered in the cloud. When a sound is emitted, the phones gather the relevant data and publish to the PSSA cloud. A Pinpoint PSSA application service is developed and deployed in the cloud (as shown in Figure 3, next) which uses an algorithm to determine the location of the sound and to publish the information back to the cloud. This process provides the data to the registered subscribed modules, the phones (dissemination subscriber) in this case, to display the location on a map.

2.2 Algorithm

Our sound localization methodology employs a Time and Difference of Arrival (TDOA) algorithm to calculate the location of a sound. In an ideal scenario, sound waves travel at a constant rate and hit different locations at different times. By strategically placing receivers at varying distances from the source, the data can be used to reverse engineer the location of the sound. The algorithm requires four smartphones in separate locations, each with the ability to (1) listen for and record sounds, (2) determine its GPS location, and (3) transmit the relevant data through an ad-hoc network. Each phone transmits only two pieces of data—the time it hears the sound and its own location. In order to use this algorithm, three assumptions were made; firstly, the speed of sound is assumed to be both known and constant. Second, the phones are placed at different "rings" as represented in Figure 4; and cannot be the same distance from the sound source. Third, we set the point (0, 0) of our coordinate system to an arbitrary constant location.

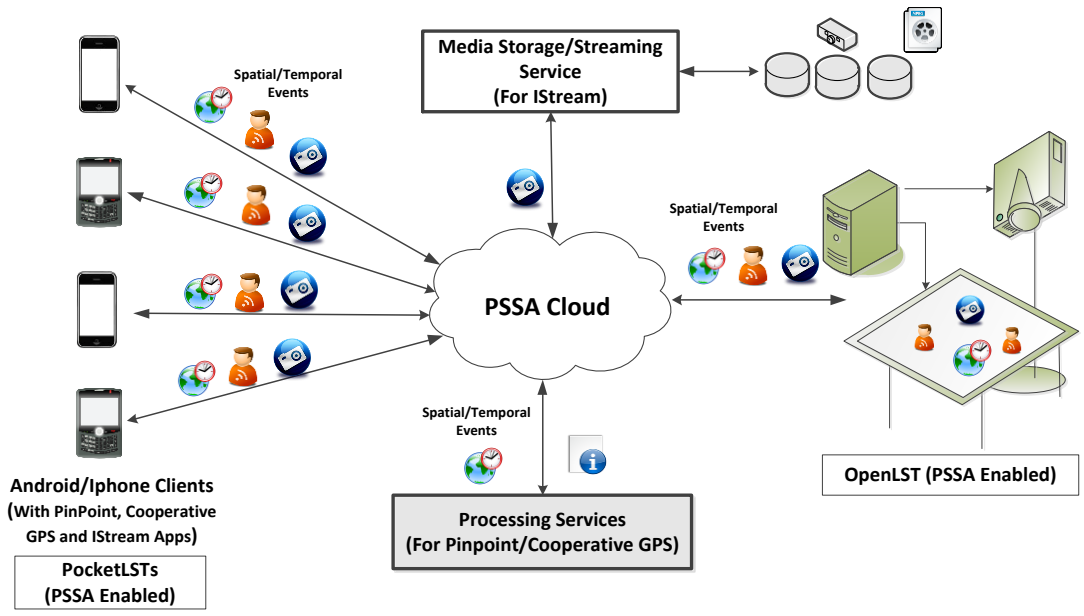


Figure 3. A conceptual view of a deployed PSSA cloud architecture along with processing services such as PinPoint, Cooperative GPS, and Media streaming services, subscribing and publishing spatio-temporal events like chat, information and media, from smartphone sensors, installed with PocketLST applications (i.e., PinPoint, Cooperative GPS and IStream apps). The PSSA enabled OpenLST and smartphone applications can register as dissemination and ingestion service simultaneously and collaborate among all the involved modules in the cloud by communicating live data and also data generated by the services in near real-time fashion, thus enhancing situational awareness and understanding of a given mission.

If the phones are located at A_0 (x_0, y_0), A_1 (x_1, y_1), A_2 (x_2, y_2), and A_3 (x_3, y_3), all of which are known (taken from the GPS receiver on each phone) and assuming the source of the sound is at an unknown location (a, b). The distance d_0 (from the closest Android to the source) is also unknown. The variables d_1, d_2, and d_3 represent the distances from each respective phone to the ring containing A_0. These can be calculated using the difference in times that the phones hear the sound:

$$\Delta T_{01} = T_1 - T_0$$

$$\Delta T_{02} = T_2 - T_0$$

$$\Delta T_{03} = T_3 - T_0$$

Then, using the constant speed of sound S, we can use the formula *rate* x *time* = *distance* to obtain the distances:

$$d_1 = \Delta T_{01} \text{ x } S$$

$$d_2 = \Delta T_{02} \text{ x } S$$

$$d_3 = \Delta T_{03} \text{ x } S$$

Finally, with all of the information, we can use distance formula to come up with a system of four equations and three unknown variables:

$$\sqrt{(x_0 - a)^2 + (y_0 - b)^2} = d_0$$

$$\sqrt{(x_1 - a)^2 + (y_1 - b)^2} = d_0 + d_1$$

$$\sqrt{(x_2 - a)^2 + (y_2 - b)^2} = d_0 + d_2$$

$$\sqrt{(x_3 - a)^2 + (y_3 - b)^2} = d_0 + d_3$$

At first glance, it seems that the algorithm can be implemented using only three phones (three equations and three unknowns). However, because of the nature of exponents and square roots, this would result in two very different—yet technically "correct"—solutions. By using four phones, and thereby four equations, we arrive at a single solution. The actual implementation of this algorithm was taken from the SoundSourceLocalizer Java class[1], which implemented the method described by Walworth and Mahajan[2].

2.3 Sound Recording & Processing

One of the major challenges of this project was recording audio in near real-time while continuously analyzing it. To accomplish this, the Android AudioRecord class is used because it automatically stops recording after a certain amount of data, making it easy to control the length of our sound samples. Next, an AudioRecord object can be configured using different sample rates, encodings, and formats, not all of which are supported by every android device. As a result, the AudioRecord is configured with a default set of options; if these are not recognized by the hardware, the application looks through all configurations to find one that is supported. The last issue was the potential for buffer under/overflows. In order to get around this, the record buffer is configured twice the minimum size; the solution was not ideal and still resulted in occasional buffer overflows, but the hack was found sufficient for this application. As it stands now, by creating a buffer double the minimum size, the phones record in roughly tenth of a second intervals and then process the audio in between. Further, in order to detect sound events, the application compares the amplitude of sound samples using the code from an open source application called Tricorder[3]; this method passes in an array from the Recorder object and calculates the amplitude by summing up the data and adjusting it to a power scale. If the amplitude of the current buffer is a user-set percentage greater than the previous buffer's amplitude, the current buffer registers as an event. This threshold defaults to 40% (meaning that a sound buffer needs to be 40% louder than the previous one to be registered), but the percentage can be adjusted via the user interface.

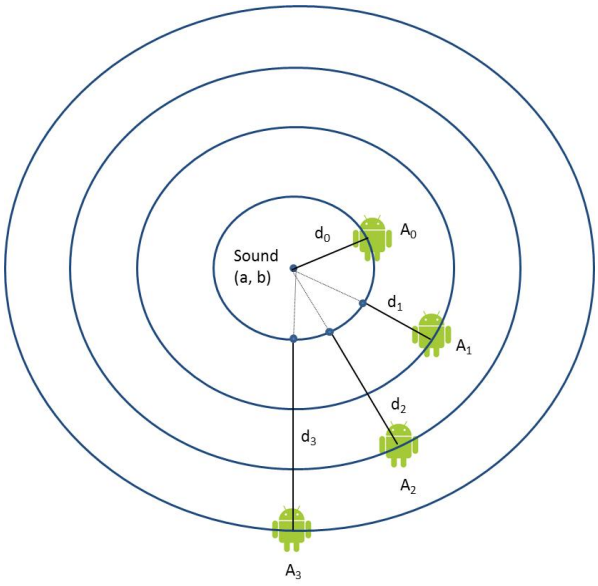


Figure 4. Conceptual representation of Time and Difference of Arrival (TDOA) Algorithm.

Once an event has been registered, the phone collects its GPS data and the time stamp on the audio to transmit to the PSSA Pinpoint application service to calculate the location. Our initial method of time stamping is to record the system time at the beginning of one sound buffer and send it if a sound event is registered. This method presents a number of problems; most notably, it does not give an exact time for when the sound occurred—only the time that the sound buffer started recording. Therefore, if a sound occurs at the end of a buffer, the data can have up to a tenth

of a second of error. We attempted to work around this fact by finding the largest single sample value in the sound buffer and time stamping the moment it occurs, but this method proved to be wildly inconsistent. One possible solution would be to shorten the length of the sound buffers, but the AudioRecord class limits the minimum size of its buffers. Ultimately though, in order to obtain the best possible data, it may be necessary to treat audio as a true stream and analyze the peaks accordingly and this method was working consistently.

Further, the TDOA algorithm expects a coordinate system where abscissa is equal to the ordinate at every point, but one degree of latitude does not equal one degree of longitude. Thus, the GPS coordinates do not correlate directly to the Cartesian coordinates that the algorithm expects. Thus, the conversion of the GPS coordinates to Universal Transverse Mercator (UTM) coordinates should happen before performing any calculations mentioned earlier. This solved the problem that GPS coordinates posed, but its functionality was also dependent on being close enough to the equator.

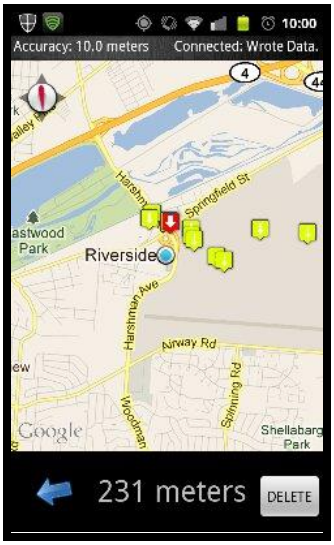


Figure 5. A screenshot of the Pinpoint Application running in the PocketLST framework. The square icons are OverlayItems containing calculated sound locations. The downward arrow embedded square icon is the current "focused" sound. The circle dot is the location of the phone.

2.4 Displaying Localization Data

Lastly, the Pinpoint application uses PocketLST framework to provide interactive layers for personnel in the field to use it in their mission. The first call to the application initializes the Recorder and LocationManager (used to track GPS) and displays the map. It also displays the accuracy of the GPS on the top-left corner and the network status on the top-right corner. The menu button provides an option to change the threshold of the recorder, as previously mentioned.

The activity uses an itemizedOverlay (a map object containing an ArrayList) to hold the calculated locations. A separate thread runs in the background, waiting for the events from the Pinpoint PSSA service to send its calculated GPS coordinates back to the phones. When the phone receives a message, the interrupted thread uses the GPS location to create an OverlayItem and places it in the itemizedOverlay. For our purposes, the OverlayItem is an object that takes a GeoPoint (the GPS coordinates) and an icon to display. It then passes the updated itemizedOverlay back to the displayed map. In Figure 5 (above), each icon is the visual representation of the sound's OverlayItem. The user can interact with the icons on the map by focusing on or deleting sounds. The user can focus on a sound by tapping the sound's icon. The bottom portion of the screen displays the relative distance and direction

from the phone to the focused sound. The directional arrow rotates in real time, based on the angle to the sound location and the phone's bearing. The "DELETE" button removes the currently focused OverlayItem from the map and sets the focus to the first sound in the itemizedOverlay. The same localization data is available to any dissemination platform like OpenLST, which can be employed to alert personnel in the field with additional information (from other sensors) that can aid in better understanding the situation.

3. COOPERATIVE GPS

3.1 Overview

The goal of Cooperative GPS is to develop a smartphone-based prototype for the Cooperative GPS Navigation algorithm from Quebe et. al[5]. This project was focused around Android-based mobile devices, but also required external hardware to complete the algorithm's data requirements. This application is intended to aid the personnel on mission with inadequate satellite coverage for accurate GPS triangulation that can be possible with single GPS unit. From technical perspective, the project consists of five major steps. First was to evaluate the requirements for the Cooperative GPS algorithm. Second was to assemble the required Arduino hardware used to collect the GPS data

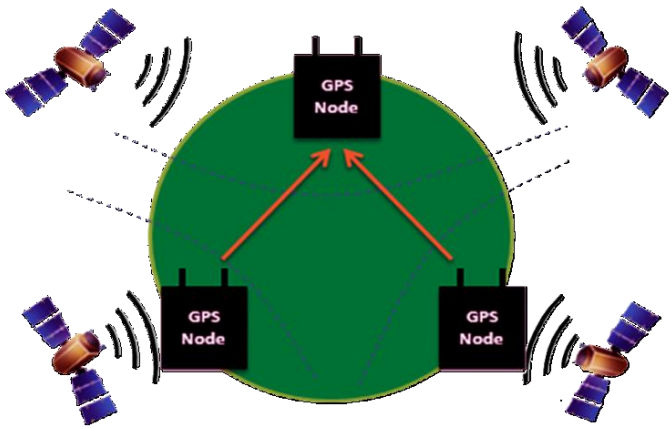


Figure 6. A conceptual representation of Cooperative GPS Algorithm.

for the algorithm. Third was to develop Arduino code to parse out the necessary data from the GPS receiver. Fourth was to develop the Cooperative application service in the PSSA cloud to compute the algorithm. Fifth is to develop and PSSA enabled Android application to collect and publish the results to the PSSA cloud and also display the results back to the personnel using the application, as shown back in Figure 3.

3.2 Algorithm Requirements

The Cooperative GPS Navigation algorithm combines GPS information from multiple agents, or nodes, to generate more accurate location estimates than any lone agent could. Cooperative GPS gives two main advantages over a normal GPS implementation: (1) the location estimates from the algorithm are more accurate than those from individual agents, (2) location estimates can be generated even when the individual agents do not have enough satellites in view to estimate their own location[5], as shown in Figure 6 (above). The algorithm specifically combines measurements for pseudo range, which is the estimated distance from receiver to satellite, satellite position, satellite clock bias, and relative position of agents to the central node in repeated matrix operations until the calculated corrections are within the a specified tolerance. There are some specific requirements regarding the algorithm: at least four different satellites need to be available between all agents, the relative position of each agent to the central node must be known, sufficient communications need to occur between the three agents to allow for fast data transport.

3.3 GPS Node Hardware

For this project's prototype, three nodes were created to act as the individual agents in the algorithm. Each node was made up of two main parts, as shown in Figure 7 (below): (1) an Android-based device, and (2) an Arduino Uno microcontroller with GPS receiver and Bluetooth modem. An external GPS receiver was required in place of the internal GPS found in most Android devices. This was implemented because the internal receiver does not provide the low-level resolution required for the algorithm.

For the external GPS receiver, the EM-406A was chosen for its ability to provide the required measurements using the SiRF binary protocol[6]. For the Arduino, the GPS receiver operates using pins 2 and 3 while the Bluetooth modem operates on pins 0 and 1, which allows for different data rates between the two serial ports. The Arduino Uno only has one serial port built in on pins 0 and 1, so the NewSoftSerial library[7] was used to create a software-based serial port on pins 2 and 3. The GPS receiver is attached using a GPS shield that provides the header for the receiver along with a reset button, on/off switch, and the ability to operate using pins 0 and 1 or 2 and 3 with the flip of a switch. The Arduino Uno microcontroller module has a 16 MHz clock speed, 32KB of flash memory, 6 analog inputs, and 14 digital pins, of which we only use four. The Arduino hardware and Android device are paired with Bluetooth as the first step in setup of the prototype.

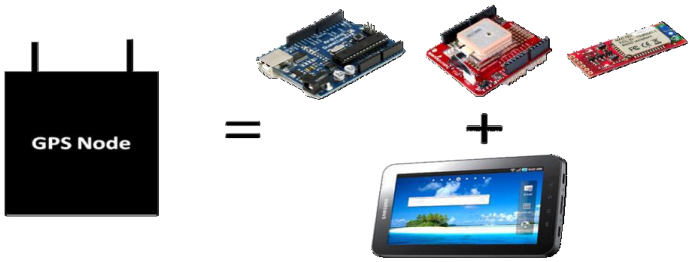


Figure 7: Prototype GPS node, composed of an Android device, Arduino Uno microcontroller, GPS shield, EM-406A GPS receiver, and BlueSMiRF Bluetooth modem.

3.4 Data Collection and Processing

The GPS receiver is required to operate in SiRF binary mode to receive high resolution GPS data required by the cooperative GPS algorithm. Once in SiRF mode, the receiver was set to send SiRF messages 28 and 30. Message 28 provides measurements for pseudo range, while message 30 provides satellite position in Earth-Centered, Earth-Fixed (ECEF) xyz-coordinates[6]. Both messages also provide a satellite's pseudo random number (PRN) and timestamp, allowing for easy tracking of data for specific satellites. These measurements and associated metadata are parsed out of their respective messages and converted from binary to their hexadecimal representation on the Arduino using print functions built into the C-based programming language. Once in hexadecimal representation, the message data is sent to the Android device using the Arduino's attached Bluetooth modem. Once the measurements are on the Android device, another conversion from hexadecimal to decimal is required. The values for pseudo range and satellite ECEF coordinates are double-precision floating point numbers following the IEEE-754 format. For floating point values from the Arduino, a byte re-ordering is required before the values are converted to decimal. After testing some sample values from our receiver, we found that if the bytes come in the order B0, B1, B2, B3, B4, B5, B6, B7 then the appropriate byte ordering is B4, B5, B6, B7, B0, B1, B2, B3. This ordering was concluded to be correct as it was the only ordering found to give realistic results in a consistent manner. Once the byte-ordering was determined, conversion to decimal was easy using the built-in methods of Java in Android.

Each android smartphone is enabled with PSSA modules (ingestion publisher) and are registered in the PSSA cloud. Each android smartphone publishes the collected data to the cloud and a "Cooperative GPS" application service is developed and deployed to handle the processing these published data, as shown in Figure 3. The application service publishes the computed data back to the smartphones (dissemination subscriber) module. These computations in the application service occur at predetermined time interval on the collected data using the Cooperative GPS algorithm.

A Java implementation of the Cooperative GPS algorithm was provided during the project[5] and placed within the application service in the cloud. The provided implementation of the Cooperative GPS algorithm only accounts for the central node and two other neighboring nodes, though the algorithm is expandable to more. The many matrix operations performed during the algorithm are done using the Efficient Java Matrix Library (EJML)[8]. The new location estimates from the algorithm are in ECEF coordinates and require a conversion to latitude and longitude before the estimates can be published back to the Android devices. This conversion was implemented by converting MATLAB code into a Java function, which proved to be simple as the conversion requires only simple math operations.

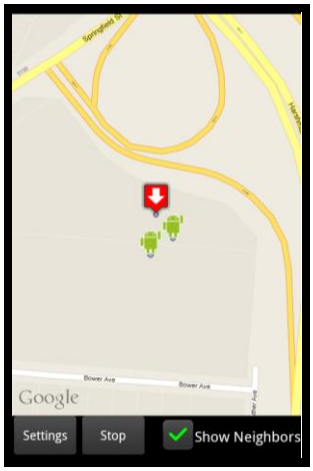
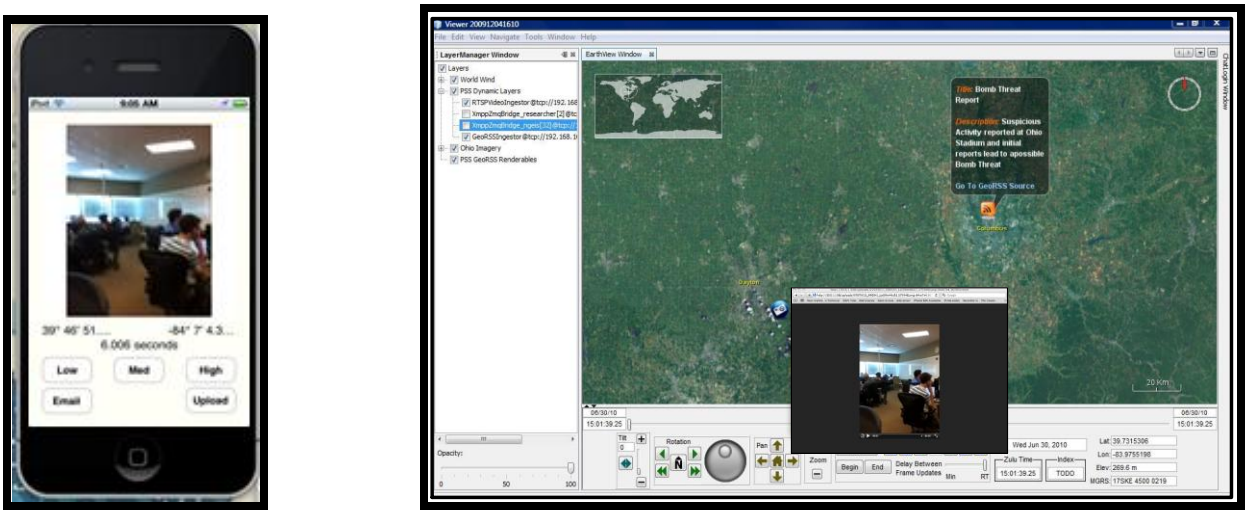


Figure 8: Main interface of Cooperative GPS Android application.

3.5 Application Interface

The main interface of the Cooperative GPS Android application is a PocketLST framework map with buttons for settings menu and start/stop of data collection, as shown in Figure 8, above. For this application UI, map overlays are used for each result point, with unique images for specific types like most recent result or neighboring node results. The Android device's most recent location estimation is displayed using a downward arrow embedded in a square, the other nodes' location estimations can be displayed as an Android marker if the check box is checked. The 15 most recent results will also be displayed on the map as a visual history. Any overlay can be tapped on to get more specific information about that point, such as time of processing and the latitude and longitude estimations.



(A) **(B)**

Figure 9. Figure 9(A) shows the IStream application interface displaying the live video, GPS location, and time stamp. The interface also provides user with ability to choose the quality of the video being published to the PSSA cloud. Figure 9(B) shows the PSSA enabled OpenLST, acting like a command center view, showing the live video feed from the IStream being shown at the GPS location on the map, along with other sensor feeds published in the PSSA cloud.

4. ISTREAM

4.1 Overview

The goal of the IStream project is to utilize the camera on the Iphone device to capture the live "first person point of view" video in the field and share it in real-time among authenticated personnel in the field and with the command center for a given mission. This iPhone application would provide enhanced dynamic "eyes in field" capabilities for the personnel, which would not be possible with static fixed cameras in the field with limited field of view and also with limited zoom levels. The IStream application employs the PSSA cloud's media storage and streaming service to scale the live video streaming capacity seamlessly among multiple personnel carrying IStream enabled Iphones in the field, as shown in Figure 3.

4.2 Methodology

The main device hardware that the IStream project will use is the iPhone's 5-megapixel rear-facing camera. Developers are able to access this camera and create a "preview," allowing the user to see everything in the camera's Field of view. Although recording video files on the device is straight forward, extracting frames from the video for live upload to an endpoint, which is publishing to PSSA cloud as events, is a major hurdle. The Global Positioning System (GPS) receiver is another important hardware element utilized by IStream.

This is able to communicate with at least three GPS satellites to accurately determine the device's location. IStream application will access this feature to create dynamic GPS tagged video frames that would facilitate effective information to the command centers using PSSA enabled dissemination platforms like OpenLST.

4.3 Implementation

The IStream application is developed with Objective-C programming language written in XCode. This language is an object-oriented language derived from C that is used for Apple Mac OSX/iOS application development. Within this language, developers are able to use existing frameworks containing pre-defined methods for frame extraction and encoding.

Apple allows the iPhone camera to be accessed through two different ways: (1) as an instance of the UIImagePicker class, and (2) through the AVFoundation framework. The UIImagePicker class allows a very basic way to display the camera's view onto the screen and save video to the device. UIImagePicker does not relinquish much control of the camera and only allows basic capture and saving. Use of the AVFoundation framework allows many features of the capture to be controlled by the developer, which is optimal for iStream. The video capture is set-up using AVFoundation to initialize camera recording and display a preview of the camera view. From this video stream, an individual frame is extracted using the Core Graphics framework. After the frame is extracted from the video, an encoding process on the device is preformed. Apple provides methods to convert a frame (image) into jpeg or png format. These formats are functional, allowing the frames to be streamed live. Despite the functionality of this method, encoding is a slow process requiring approximately one fourth of a second to encode. This slow process only allowed for streaming of a maximum of four frames per second but often times only three frames per second was achievable due to other factors. A video of four frames per second (fps) was acceptable but not desirable.

Other methods of encoding that would provide a frame rate of at least 10 fps were required. I264Encoder, a framework developed by FoxIt Solutions, was chosen to the project to allow the frames to be encoded in H264 format instead of jpeg or png format. The conversion process is quick enough to achieve a maximum of thirty frames per second, with provision to select different levels of compression qualities. H264 format, also known as mpeg4AVC, is a raw data format that is supported by the PSSA cloud's media storage and streaming service[9-10]. The IStream application employs PSSA enabled registered modules for ingestion and dissemination. Thus, after the conversion process, the application publishes each encoded video frame along with the device's GPS coordinates and system time to the PSSA cloud. The media storage and streaming service in the PSSA cloud stores the live video frames for the future retrieval and as well as publishes the frames to all the subscribed modules like OpenLST and also to other IStream applications, as shown in Figure 3.

4.4 Recording & Streaming Interface

The IStream currently provides simple functionality without any overlay on the PocketLST for usability. The user is presented with one screen containing buttons to select quality of video compression. Here the lower quality of video compression increases the frame rate of publishing to the cloud and vice-versa. The preview of the streaming video is captured on the screen along with the GPS coordinates and time stamp as shown in figure 9(A). The live video stream from the IStream published into the PSSA cloud can be streamed in real-time to the command center using PSSA enabled OpenLST visualization platform as shown in figure 9(B).

5. DISCUSSION

The field tests for the three prototype applications were ran separately and had been observed that the implementations integrated with PSSA cloud architecture and OpenLST visualization platform indeed met the theoretical expectations as mentioned earlier. But, these tests provided the insights into the flexibility of the current implementations that will be discussed in the context of related current and future work at ICC in this section.

The Pinpoint application's field tests involved a stationary sound source and four phones. These tests were performed in different configurations, with each configuration being generated with varying combinations of radius and position of the phones (shown in Figure 3). The sound localization points generated by the application were correct in the direction in all configurations but there was an 8% of error in distance for 10% of the configurations (verified with interface shown in Figure 4).

With a goal to employ Pinpoint application in battlefields, crime-prone areas and for search-and-rescue missions, even slightest error can have disastrous consequences. Though, one can argue that the effect of error can subdued with other sensors in the field, it was important to investigate the error in the application in complete isolation. After investigation, it was found that the inaccuracy in the coordinates from phone's GPS is the main reason for the error and providing an accurate or corrected GPS for every phone involved reduced the error to 2%. Since, the Cooperative GPS application's testing was successful and yielded in accurate functional results; the Pinpoint application can be supplemented with the Cooperative GPS module to produce more accurate results. Further, there are many possible future continuations and improvements that can be made to Pinpoint application.

The application will need to implement a system for detecting and adjusting for echoes dynamically, allowing it to be used in urban areas more effectively, which is presently being controlled by constant sound threshold setting. However, there are even more potential branches of exploration for Pinpoint.

Projects centered on Pinpoint application, that are currently being worked on is to use the phones to detect and locate ultrasonic sounds and also is to set off a constant, repeating sound and using the phones to detect the echoes, therefore aiding in creating a virtual mapping of the area. Although the IStream application was successfully tested to be streaming live video feed from the field to the command center in real-time, the application lacks a playback module that can take advantage of the archive streaming capabilities of the PSSA cloud. This playback mechanism is available for OpenLST visualization platform at command center, but it would definitely benefit the personnel in the field to have access to the archive past videos of a given area under surveillance to make effective localized decisions appropriately. Currently, the IStream application is appended with this PSSA enabled playback module and also the Cooperative GPS module to gather more accurate information about a given area during a mission.

Finally, in this paper, the feasibility of developing smartphone applications to increase the situational understanding and situational awareness has been demonstrated. Further, a suitable event-based sensor fusion real-time collaborative cloud architecture, PSSA has been introduced that can interpret the smartphone applications as advanced human augmented sensor modules and integrate them in real-time with command center and among themselves, providing a collaborative capability thru its ingestion and dissemination modules in OpenLST and PocketLST frameworks.

Our current and future work dealing with challenges in national defense and public safety fields are centered on the concept intuitive and intelligent sensor fusion technologies to increase situational awareness for a given mission. Finally, the success of these three prototype applications described in this paper, have collectively supported the prospective of employing smartphones to increase “Situational Awareness” by conceptually forming digital “Eyes and Ears” of the armed personnel in hostile environments.

REFERENCES

[1] CSC 2011: Audio Forensics v1.0 (2011) *Class SoundSourceLocalizer* (version 1.0). Available at http://webhome.cs.uvic.ca/~jmurdoch/edu/CSI/resources_students/docs/SoundSourceLocalizer.html (accessed June, 2011).

[2] A. Mahajan and M. Walworth, "3D Position Sensing Using the Difference in the Time-of- Flights from a Wave Source to Various Receivers", IEEE Trans. on Robotics and Automation, Vol. 17, No. 1, February 2001, pp. 91-94.

[3] Moonblink (2011) *Tricorder* (Version 5.12). Available at http://code.google.com/p/moonblink/downloads/detail?name=Tricorder-5.11.apk&can=2&q= (accessed July 2011).

[4] Baranov, Sergey (2011) *ClockSync* (Version 1.1.3). Available at http://amip.tools-for.net/wiki/android/clocksync (accessed July 2011).

[5] Quebe S, DeVilbiss S, Campbell J, Taylor C., "Cooperative GPS Navigation," Position Location and Navigation Symposium (PLANS), 2010 IEEE/ION, 834 – 837 (2010)

[6] SiRF Technology, Inc. *SiRF Binary Protocol Reference Manual*. Revision 2.4. November 2008. Accessed June 2011.

[7] Hart, Mikal, (2010) *NewSoftSerial* . Available at http://arduiniana.org/libraries/newsoftserial/ (Accessed July 2011)

[8] (2011) *Efficient Java Matrix Library* (version 0.17). available at http://code.google.com/p/efficient-java-matrix-library/ (Accessed July 2011)

[9] Rob Williams, Sanjay Kumar Boddhu, Ed Wasser. "*Adapting persistent surveillance storage innovations in for Homeland security*" for SPIE's Defense, Security and Sensing Conference 2011.

[10] Rob Williams, Sanjay Kumar Boddhu, Ed Wasser. "Smartphone innovations for persistent surveillance" for SPIE's Defense, Security and Sensing Conference 2011.