

# Computer Vision Information Systems for Remote Monitoring Systems

M. Farrell, B. Walthall., A. Vaswani, G. Thompson, M. DeVore

**Abstract—** The system designed for this research created a rapidly deployable remote monitoring system that employs a new video encoding algorithm for low latency surveillance applications. The system requires minimal bandwidth and also uses Stargates. In order to display multiple simultaneous, high-resolution video streams, a web-based user interface with alert mechanisms was developed. Through the use of a motion detection algorithm, video streams retrieved by camera nodes are sent across a central server and then to a java applet which is embedded in the user interface. The java applet displays the videos streams for each respective camera. The overall system allows for a single user to monitor a large geographic area.

## I. PROBLEM DEFINITION

WITH new advances in scalable computing and wireless sensor networks, it has become possible to design low-latency surveillance applications that consume minimal bandwidth and use low-end processors. Sparked by the desire to reduce human involvement in high risk areas, systems that are automated and can be controlled remotely are becoming more attractive to many different industries. The military is one group that is showing particular interest in these systems. For this particular study, user requirements were gathered from the clientele, which consisted of the National Reconnaissance Office and Rossetex. The system's goal was to capture image streams through the use of web cameras and then extract information that was relevant and necessary for surveillance purposes.

The Hawkeye system created for this study has the ability to collect information from multiple imaging processors at a single client in order to present an accurate display of the environment. All communications done from the camera

Manuscript received April 13, 2007. This work was supported in part by the National Reconnaissance Office and Rossetex.

M. Farrell is a graduate student in the Computer Science Department at the University of Virginia, Charlottesville, VA 22903 USA

B. Walthall is an undergraduate student in the Systems and Information Engineering Department at the University of Virginia, Charlottesville, VA 22903 USA.

A. Vaswani is an undergraduate student in the Systems and Information Engineering Department at the University of Virginia, Charlottesville, VA 22903.

G. Thompson is an undergraduate student in the Systems and Information Engineering Department at the University of Virginia, Charlottesville, VA 22903.

M. DeVore is a professor in the Systems and Information Engineering Department at the University of Virginia, Charlottesville, VA 22903.

node back to the network were done over a wireless link.

Since the system required an efficient schema that used cutting-edge technological components, the source devices within the Hawkeye system consisted of devices with limited computational, battery, and storage resources. Due to these limiting constraints, the algorithms created for video encoding and motion detection do not use expensive floating point calculations or highly-iterative processes. Another design feature is the rapid re-configurability of the Hawkeye system. In order to support rapid changes in the surveillance application, the camera node software is able to change its surveillance parameters without the need for any recompilation or binary code dissemination. This flexibility allows camera nodes to dynamically switch surveillance groups (by reporting to a different server), change detection rates to save battery power, or alter image sizes to reduce bandwidth. Additionally, the configuration parsing API allows for the facilitated addition of new parameters to the system.

Although no constraint on bandwidth was placed, the system was designed around minimal consumption by modifying detection rate and image size in order to add to the efficiency of the design. In addition to this, the system supports JPEG compression [1]. This ensures that the images sent through the network are first compressed as JPEG data segments and then those resulting segments are sent. In addition to this, the camera nodes only send the changed regions or changed-sub-regions with respect to an initial reference image whenever a change is detected. If changes are infrequent and small, we suspect that this model will greatly reduce the bandwidth used by the system.

## II. SYSTEM DESIGN

The overall system consists of five core components, the camera node (hawkeye), the central server (hawkeye\_server), the client interface library (libhawkeye\_client), the web interface, and the java applet. Each of these components inter-relate to provide a wireless remote monitoring system.

### A. Hawkeye, the camera node component

The camera nodes used for detection of undesired intruders consist of Logitech web cameras connected to Stargates (components with low-end processors). This component is responsible for monitoring significant changes in the environment and was implemented in the C

programming language using the video4linux API [2]. Upon initialization, the camera sends a reference image to the central server. This reference image is used as a basis of comparison for initial detection. Reference images are constantly changing depending upon permanent changes in the environment that are noted by the system monitor. After camera initialization, it begins its periodic detection scheme to check for changes. In between change detections, the camera goes to low power sleep to conserve resources. To allow the camera devices to go into a low-power sleep whenever they are not running code, communication from camera nodes to the outside world is entirely unidirectional. In other words, no connections will be allowed into the camera devices to query for information. This allows the cameras to form their own sleep schedules completely independent of the rest of the system. Information about the cameras will still be available to the rest of the system through a *central server*

### B. Hawkeye\_server, the central server

As important events occur in the system, the server broadcasts these events and the associated image stream to an end-user client connected to the system. More specifically, the server broadcasts to end-user clients whenever a camera joins the system, expires, or when it has detected changes. One important characteristic of the server is that it maintains a list of active camera nodes that have checked into the server during its lifetime and relays this information to clients even during the time when the camera node's themselves may be in sleep mode or powered down altogether. The server also keeps track of its connections inside a list of client objects which store properties about each client. Among these properties are the client's id, network address, and whether they are a camera node or end-user. The job of the server is to listen for incoming connections, fulfill requests from clients, and broadcast information to the connected end-user clients.

It is also useful to think of the central server as a manager for a "group" of camera nodes. By thinking this way, a global surveillance system can consist of multiple server components where each one is responsible for managing a location-dependent group of cameras. The Hawkeye server was implemented in the C programming language.

### C. libhawkeye\_client – the end-user client API

The libhawkeye\_client component allows developers to use the familiar C programming language to build applications and interfaces which will interact with the system. The API provides encapsulated routines to facilitate operations such as connecting to a server, polling for activity, and receiving images from cameras. The libhawkeye\_client component was developed in a multi-platform fashion allowing its code to work on more than one operating system.

All of the information needed by a client application is

stored in the SERVER\_INFO object which is created when the end-user connects to the server. This SERVER\_INFO object consists of a list of CAMERA\_INFO objects along with how many cameras are connected to this particular server. Within the CAMERA\_INFO object is the network address of the camera for identification purposes, and the image data for that camera. Upon connection to the server, the client library creates the information object and populates all of its fields to reflect the information that the server sends. Then, the object can be polled by the client application and updated as new information arrives from the server's broadcasts. One important feature performed inside the client library is the actual assembly of the final image from the reference image and the individual change regions. This is done simply by overlaying each change region over the original reference to form a final composed image. This resulting image is made available by the library to the client application.

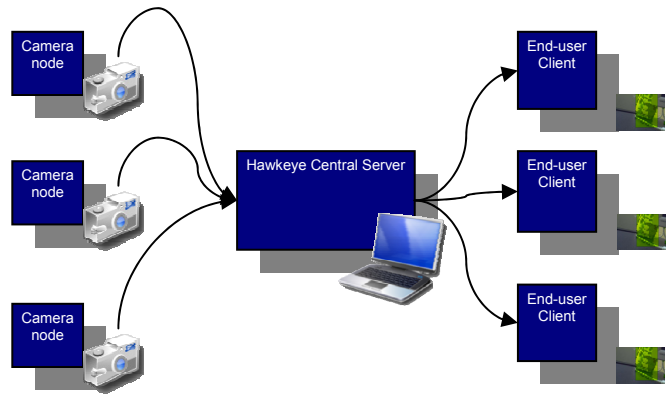


Fig. 1. The Hawkeye system. The image above represents the relationship between the camera nodes, the Hawkeye central server and the end-user client. The image information collected through the camera nodes is relayed over to the central server. Once a request to the server has been made by the client, the image streams gathered from the camera nodes representing motion detection are sent to the end-user client.

### D. Java Applet

The Java applet developed represents another client that must receive the image streams. In order for the web interface to display the streams, this component was necessary since the libhawkeye\_client could not be embedded within an html page. The java applet essentially calls the functions within the libhawkeye\_client to display the image streams in the interface by using the Java Native Interface (JNI) [4]. JNI allows for the writing of native methods and also embeds the Java virtual machine into native applications. The JNI essentially serves as the middle component between the C client and the java applet client. A

dynamic library link file is compiled for individual platforms so that it serves as the interface into the library for java.

### E. User Web Interface

The web-based interface is an application controlled by the human base controller that was built using AJAX development techniques as well as a JavaScript library for web applications called YUI-Ext, which is an extension of the Yahoo! User Interface (YUI) [5].

The interface contains information regarding maps of the building being monitored with respective camera locations and IP addresses that allows for a Java applet to display the image streams from a camera that has detected motion. Information, such as the respective camera's IP addresses' and locations, are stored in a database that resides on a University of Virginia server. Upon set-up, one can upload the building maps/blueprints of each level of the specified building. Also a list of cameras can be populated and dragged and dropped onto the appropriate location on an individual floor's map. Once a camera has been added, a form displaying the relevant information appears and the information added is stored on the database. The web interface essentially receives all of the information in the system and organizes it into helpful forms. The other significant feature of the interface is that it uses asynchronous server calls to create a user experience very similar to that of a desktop application.

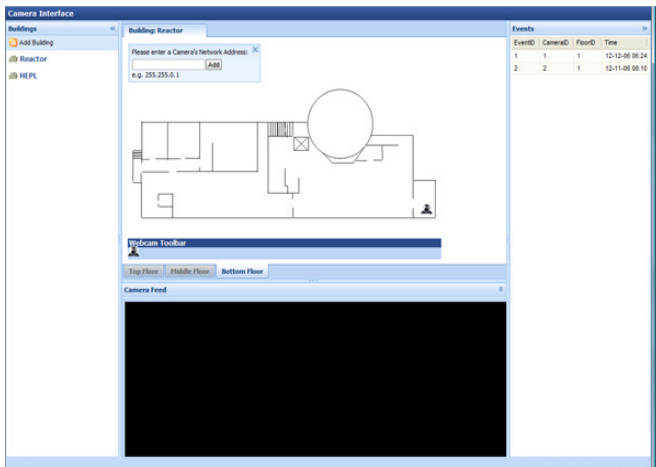


Fig. 2. The web interface. The layout of the web interface is separated into four panels. The left side panel represents the various buildings and respective floors within the buildings. The top middle panel is an overview of the floor plan of a particular level and shows relative camera locations. Below the map is the area of the java applet where the image streams of a selected image stream are displayed. In the right panel, information of cameras and events can be seen.

### III. MOTION DETECTION ALGORITHM

In order for the camera nodes to capture movements within a specified area, a motion detection algorithm had to be written. Because the Stargates, the devices controlling the cameras, have limited computational resources, the change detection algorithm had to be very robust. The algorithm quantitatively analyzes the new image against the reference image for significant changes in a single-pass detection algorithm. It does this by breaking down the image into 8x8 pixels and comparing these blocks to determine whether or not a change in the environment has occurred. After separating these images into 8x8 pixel blocks, a sum over the luminance for the block is computed:

$$L_1 = \sum_{x=1}^8 \sum_{y=1}^8 Lum_1(X \times 8 + x, Y \times 8 + y)$$

$$L_2 = \sum_{x=1}^8 \sum_{y=1}^8 Lum_2(X \times 8 + x, Y \times 8 + y)$$

where  $Lum_1$  is the luminance value for image 1,  $Lum_2$  is the luminance value for image 2 at the given pixel position, and the pair (X,Y) is the position of the current 8x8 block.

Next, the difference between the sums is compared against a local threshold value.

$$h_{(X,Y)} = \begin{cases} 1 & |(L_1 - L_2)| \geq \delta_{loc} \\ 0 & otherwise \end{cases}$$

where  $h(X,Y)$  is a binary term indicating whether a change was detected at the block.

The binary change terms are then summed over the entire set of 8x8 blocks

$$b = \sum_{X=1}^{W/8} \sum_{Y=1}^{H/8} h_{(X,Y)}$$

where  $b$  is the number of blocks where a change was detected between both pictures, and (W,H) are the dimensions of the image. It is important to note that this algorithm assumes that both images have the same dimensions.

As a last step,  $b$  is compared against a global threshold value:

$$C = \begin{cases} 1 & b \geq \delta_{glob} \\ 0 & otherwise \end{cases}$$

where C is the binary result of the entire detection algorithm.

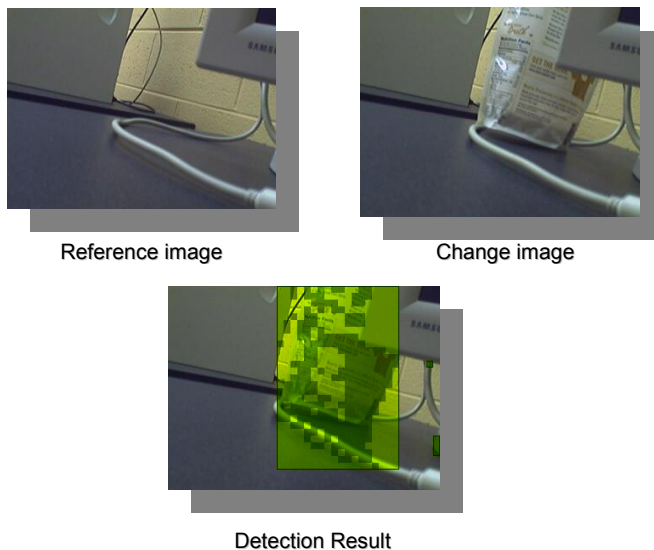


Fig. 3. The results of the change detection algorithm. The three images represent the different stages in the detection setup. The first image represents the initial image stored as a basis of comparison, the second image represents a change in the environment with an added bag, and the third image shows the result of the detection where the green area represents the changed region.

#### IV. RESULTS AND TRADEOFFS

By the end of this design study, a fully functional remote monitoring system prototype was developed. The working model is able to monitor a specified building with high precision. The camera nodes maintained the ability to send visual feeds to the central server and then to the client via the web interface and java applet. The user was able to view these feeds and see motion events.

Tradeoffs, including the quality of the image produced and sent over the network, had to be made to design the system. It was determined that the image quality used was to be just enough as to provide recognition of an object, but not to the degree of minor details. Recognition of the type of object, the color, and the relative size was deemed satisfactory. Fine details of an image would increase the amount of bandwidth used over the system. Bandwidth is a resource that is used to determine efficiency within a system and therefore was not sacrificed at the expense of minor details.

Another tradeoff made was developing the image source software in C rather than the more widespread Java. This resulted in the use of the JNI and therefore requires dynamic library links for multiple platforms to be compiled in order to use the software for various platforms, such as Windows CE, a common operating system among PDA's.

#### V. FUTURE WORK

With continuing work in this study, portable devices that could be carried by mobile would provide enhanced capability to the system. A portable device that could be

used for this function would be a PDA. PDA's possess both the size constraints and computational capabilities necessary for the mobile users. These PDA's could have the capability of containing a miniature interface that could perhaps display the image streams at a nearby location. Although they would not have the full functioning capacity as that of the user web interface, mobile users would still have a better opportunity to react and could monitor larger areas.

Another component that should be considered for future implementation and integration are Mica2 Motes, which are essentially wireless mini-computers. These powerful motes have sensor boards that can be attached to them to detect any vibrations or physical changes in the environment [6]. The sensors on the motes have far-reaching effects because they have the ability to detect seismic, magnetic, thermal, infrared, and even acoustic disturbances [7].

#### REFERENCES

- [1] "The JPEG Committee Homepage." Available: <http://www.jpeg.org/>
- [2] "Video4Linux Kernel API Reference." Available: [http://www.linuxtv.org/downloads/video4linux/API/V4L1\\_API.html](http://www.linuxtv.org/downloads/video4linux/API/V4L1_API.html).
- [3] Independent JPEG Group. Available: <http://www.ijg.org/>
- [4] "JAVA Native Interface." Available: <http://java.sun.com/j2se/1.4.2/docs/guide/jni/>.
- [5] "Yahoo! UI Library (YUI)." Available: <http://developer.yahoo.com/yui/>.
- [6] "Gateways & Network Interfaces Modules." Crossbow Technology. Available: <http://www.xbow.com/Products/productsdetails.aspx?sid=65>.
- [7] Raghavendra, C.S., Sivalingam, K., & Znati, T. "Wireless Sensor Networks." Boston : Kluwer Academic Publishers.