# A System for Video Surveillance and Monitoring *

Robert T. Collins, Alan J. Lipton, Takeo Kanade,
Hironobu Fujiyoshi, David Duggins, Yanghai Tsin,
David Tolliver, Nobuyoshi Enomoto, Osamu Hasegawa,
Peter Burt[1] and Lambert Wixson[1]

CMU-RI-TR-00-12

The Robotics Institute, Carnegie Mellon University, Pittsburgh PA
[1] The Sarnoff Corporation, Princeton, NJ

**Abstract**

Under the three-year Video Surveillance and Monitoring (VSAM) project (1997–1999), the Robotics Institute at Carnegie Mellon University (CMU) and the Sarnoff Corporation developed a system for autonomous Video Surveillance and Monitoring. The technical approach uses multiple, cooperative video sensors to provide continuous coverage of people and vehicles in a cluttered environment. This final report presents an overview of the system, and of the technical accomplishments that have been achieved.



*This work was funded by the DARPA Image Understanding under contract DAAB07-97-C-J031, and by the Office of Naval Research under grant N00014-99-1-0646.

# 1 Introduction

The thrust of CMU research under the DARPA Video Surveillance and Monitoring (VSAM) project is cooperative multi-sensor surveillance to support battlefield awareness [17]. Under our VSAM Integrated Feasibility Demonstration (IFD) contract, we have developed automated video understanding technology that enables a single human operator to monitor activities over a complex area using a distributed network of active video sensors. The goal is to automatically collect and disseminate real-time information from the battlefield to improve the situational awareness of commanders and staff. Other military and federal law enforcement applications include providing perimeter security for troops, monitoring peace treaties or refugee movements from unmanned air vehicles, providing security for embassies or airports, and staking out suspected drug or terrorist hide-outs by collecting time-stamped pictures of everyone entering and exiting the building.

Automated video surveillance is an important research area in the commercial sector as well. Technology has reached a stage where mounting cameras to capture video imagery is cheap, but finding available human resources to sit and watch that imagery is expensive. Surveillance cameras are already prevalent in commercial establishments, with camera output being recorded to tapes that are either rewritten periodically or stored in video archives. After a crime occurs – a store is robbed or a car is stolen – investigators can go back after the fact to see what happened, but of course by then it is too late. What is needed is continuous 24-hour monitoring and analysis of video surveillance data to alert security officers to a burglary in progress, or to a suspicious individual loitering in the parking lot, while options are still open for avoiding the crime.

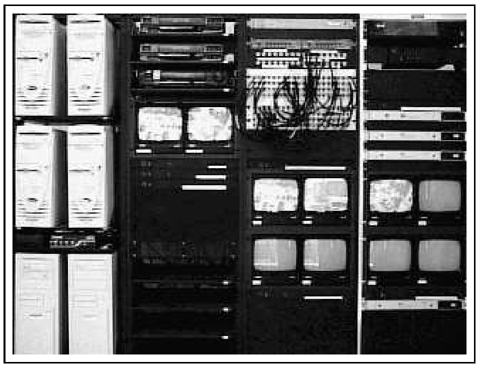
Keeping track of people, vehicles, and their interactions in an urban or battlefield environment is a difficult task. The role of VSAM video understanding technology in achieving this goal is to automatically "parse" people and vehicles from raw video, determine their geolocations, and insert them into a dynamic scene visualization. We have developed robust routines for detecting and tracking moving objects. Detected objects are classified into semantic categories such as human, human group, car, and truck using shape and color analysis, and these labels are used to improve tracking using temporal consistency constraints. Further classification of human activity, such as walking and running, has also been achieved. Geolocations of labeled entities are determined from their image coordinates using either wide-baseline stereo from two or more overlapping camera views, or intersection of viewing rays with a terrain model from monocular views. These computed locations feed into a higher level tracking module that tasks multiple sensors with variable pan, tilt and zoom to cooperatively and continuously track an object through the scene. All resulting object hypotheses from all sensors are transmitted as symbolic data packets back to a central operator control unit, where they are displayed on a graphical user interface to give a broad overview of scene activities. These technologies have been demonstrated through a series of yearly demos, using a testbed system developed on the urban campus of CMU.

This is the final report on the three-year VSAM IFD research program. The emphasis is on recent results that have not yet been published. Older work that has already appeared in print is briefly summarized, with references to the relevant technical papers. This report is organized as follows. Section 2 contains a description of the VSAM IFD testbed system, developed as a testing ground for new video surveillance research. Section 3 describes the basic video understanding algorithms that have been demonstrated, including moving object detection, tracking, classification, and simple activity recognition. Section 4 discusses the use of geospatial site models to aid video surveillance processing, including calibrating a network of sensors with respect to the model coordinate system, computation of 3D geolocation estimates, and graphical display of object hypotheses within a distributed simulation. Section 5 discusses coordination of multiple cameras to achieve cooperative object tracking. Section 6 briefly lists the milestones achieved through three VSAM demos that were performed in Pittsburgh, the first at the rural Bushy Run site, and the second and third held on the urban CMU campus, and concludes with plans for future research. The appendix contains published technical papers from the CMU VSAM research group.

# 2 VSAM Testbed System

We have built a VSAM testbed system to demonstrate how automated video understanding technology described in the following sections can be combined into a coherent surveillance system that enables a single human operator to monitor a wide area. The testbed system consists of multiple sensors distributed across the campus of CMU, tied to a control room (Figure 1a) located in the Planetary Robotics Building (PRB). The testbed consists of a central operator control unit (OCU)

(a)



(b)

Figure 1: a) Control room of the VSAM testbed system on the campus of Carnegie Mellon University. b) Close-up of the main rack.

which receives video and Ethernet data from multiple remote sensor processing units (SPUs) (see Figure 2). The OCU is responsible for integrating symbolic object trajectory information accumulated by each of the SPUs together with a 3D geometric site model, and presenting the results to the user on a map-based graphical user interface (GUI). Each logical component of the testbed system architecture is described briefly below.

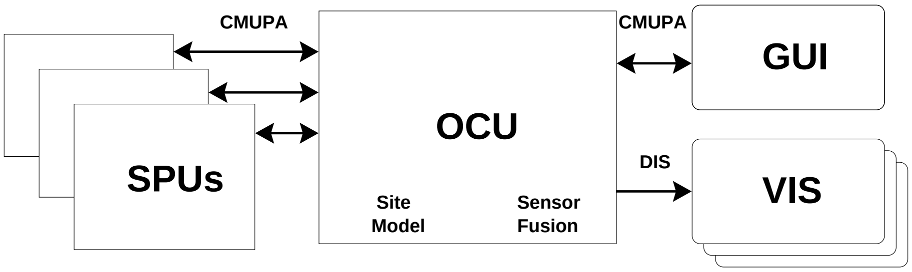



Figure 2: *Schematic overview of the VSAM testbed system.*

## 2.1 Sensor Processing Units (SPUs)

The SPU acts as an intelligent filter between a camera and the VSAM network. Its function is to analyze video imagery for the presence of significant entities or events, and to transmit that information symbolically to the OCU. This arrangement allows for many different sensor modalities to be seamlessly integrated into the system. Furthermore, performing as much video processing as possible on the SPU reduces the bandwidth requirements of the VSAM network. Full video signals do not need to be transmitted; only symbolic data extracted from video signals.

The VSAM testbed can handle a wide variety of sensor and SPU types (Figure 3). The list of IFD sensor types includes: color CCD cameras with active pan, tilt and zoom control; fixed field of view monochromatic low-light cameras; and thermal sensors. Logically, each SPU combines a camera with a local computer that processes the incoming video. However, for convenience, most video signals in the testbed system are sent via fiber optic cable to computers located in a rack in the control room (Figure 1b). The exceptions are SPU platforms that move: a van-mounted relocatable SPU; an SUO portable SPU; and an airborne SPU. Computing power for these SPUs is on-board, with results being sent to the OCU over relatively low-bandwidth wireless Ethernet links. In addition to the IFD in-house SPUs, two Focussed Research Effort (FRE) sensor packages have been integrated into the system: a Columbia-Lehigh CycloVision ParaCamera with a hemispherical field of view; and a Texas Instruments indoor surveillance system. By using a pre-specified communication protocol (see Section 2.4), these FRE systems were able to directly interface with the VSAM network. Indeed, within the logical system architecture, all SPUs are treated identically. The only difference is at the hardware level where different physical connections (e.g. cable or wireless Ethernet) may be required to connect to the OCU.

The relocatable van and airborne SPU warrant further discussion. The relocatable van SPU consists of a sensor and pan-tilt head mounted on a small tripod that can be placed on the vehicle roof when stationary. All video processing is performed on-board the vehicle, and results from object detection and tracking are assembled into symbolic data packets and transmitted back to the operator control workstation using a radio Ethernet connection. The major research issue involved in demonstrating the redeployable van unit involves how to rapidly calibrate sensor pose after redeployment, so that object detection and tracking results can be integrated into the VSAM network (via computation of geolocation) for display at the operator control console.

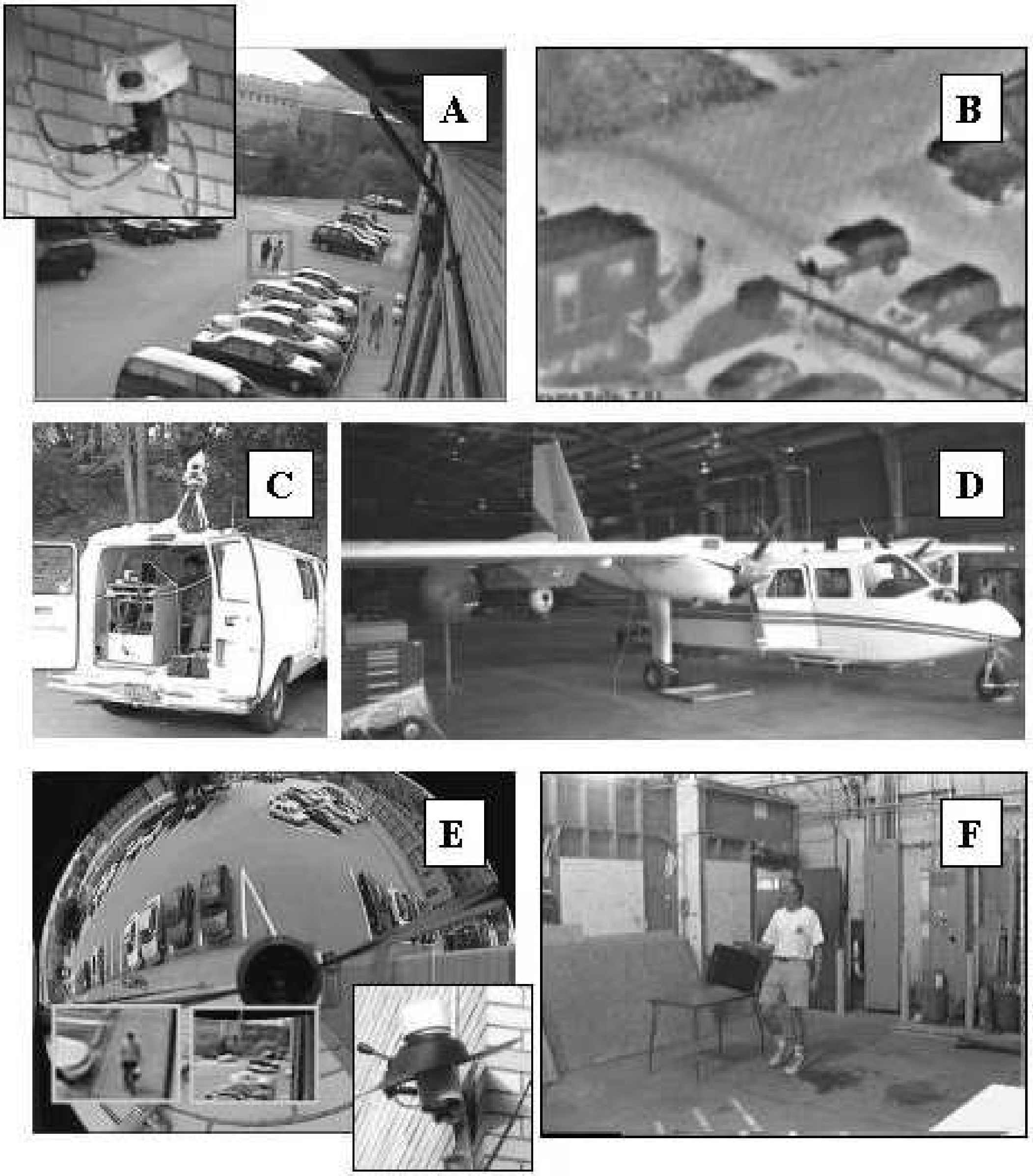



Figure 3: *Many types of sensors and SPUs have been incorporated into the VSAM IFD testbed system: a) color PTZ; b) thermal; c) relocatable van; d) airborne. In addition, two FRE sensors have been successfully integrated: e) Columbia-Lehigh omnicamera; f) Texas Instruments indoor activity monitoring system.*

The airborne sensor and computation packages are mounted on a Britten-Norman Islander twin-engine aircraft operated by the U.S. Army Night Vision and Electronic Sensors Directorate. The Islander is equipped with a FLIR Systems Ultra-3000 turret that has two degrees of freedom (pan/tilt), a Global Positioning System (GPS) for measuring position, and an Attitude Heading Reference System (AHRS) for measuring orientation. The continual self-motion of the aircraft introduces challenging video understanding issues. For this reason, video processing is performed using the Sarnoff PVT-200, a specially designed video processing engine.

## 2.2 Operator Control Unit (OCU)

Figure 4 shows the functional architecture of the VSAM OCU. It accepts video processing results from each of the SPUs and integrates the information with a site model and a database of known objects to infer activities that are of interest to the user. This data is sent to the GUI and other visualization tools as output from the system.

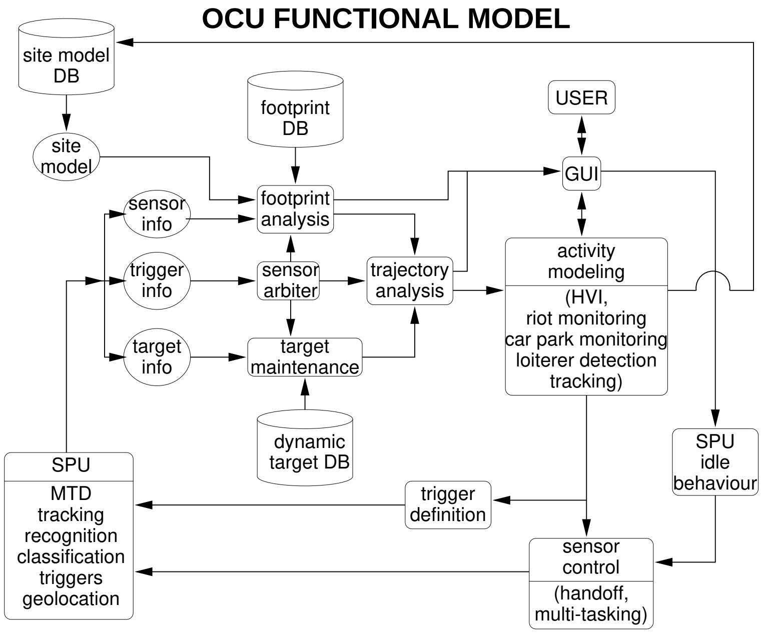


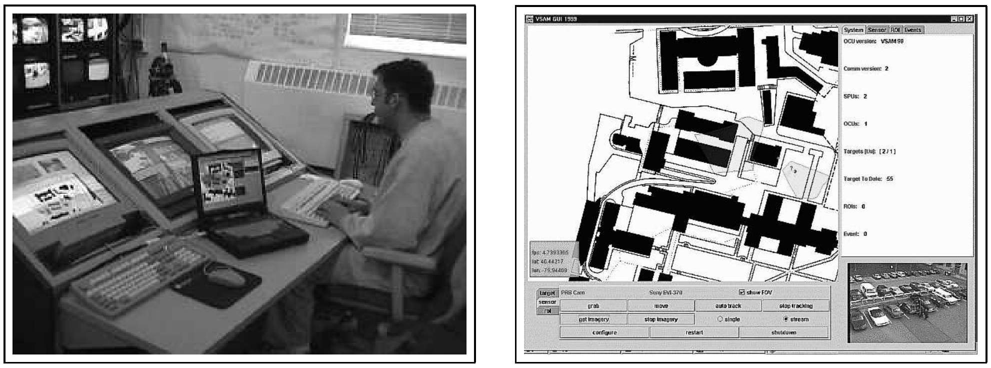

Figure 4: *Functional architecture of the VSAM OCU.*

One key piece of system functionality provided by the OCU is sensor arbitration. Care must be taken to ensure that an outdoor surveillance system does not underutilize its limited sensor assets. Sensors must be allocated to surveillance tasks in such a way that all user-specified tasks get performed, and, if enough sensors are present, multiple sensors are assigned to track important objects. At any given time, the OCU maintains a list of known objects and sensor parameters, as well as a set of "tasks" that may need attention. These tasks are explicitly indicated by the user through the GUI, and may include specific objects to be tracked, specific regions to be watched, or specific events to be detected (such as a person loitering near a particular doorway). Sensor arbitration is performed by an arbitration cost function. The arbitration function determines the cost of assigning each of the SPUs to each of the tasks. These costs are based on the priority of the tasks, the load on the SPU, and visibility of the objects from a particular sensor. The system performs a greedy optimization of the cost to determine the best combination of SPU tasking to maximize overall system performance requirements.

The OCU also contains a site model representing VSAM-relevant information about the area being monitored. The site model representation is optimized to efficiently support the following VSAM capabilities:

- object geolocation via intersection of viewing rays with the terrain.
- visibility analysis (predicting what portions of the scene are visible from what sensors) so that sensors can be efficiently tasked.
- specification of the geometric location and extent of relevant scene features. For example, we might directly task a sensor to monitor the door of a building, or to look for vehicles passing through a particular intersection.

## 2.3 Graphical User Interface (GUI)



(a) (b)

Figure 5: a) Operator console located in the control room. Also shown is a laptop-based portable operator console. b) Close-up view of the visualization node display screen.

One of the technical goals of the VSAM project is to demonstrate that a single human operator can effectively monitor a significant area of interest. Keeping track of multiple people, vehicles, and their interactions, within a complex urban environment is a difficult task. The user obviously shouldn't be looking at two dozen screens showing raw video output. That amount of sensory overload virtually guarantees that information will be ignored, and requires a prohibitive amount of transmission bandwidth. Our approach is to provide an interactive, graphical user interface (GUI) that uses VSAM technology to automatically place dynamic agents representing people and vehicles into a synthetic view of the environment (Figure 5). This approach has the benefit that visualization of scene events is no longer tied to the original resolution and viewpoint of a single video sensor. The GUI currently consists of a map of the area, overlaid with all object locations, sensor platform locations, and sensor fields of view (Figure 5b). In addition, a low-bandwidth, compressed video stream from one of the sensors can be selected for real-time display.

The GUI is also used for sensor suite tasking. Through this interface, the operator can task individual sensor units, as well as the entire testbed sensor suite, to perform surveillance operations such as generating a quick summary of all object activities in the area. The lower left corner of the control window contains a selection of controls organized as tabbed selections. This allows the user to move fluidly between different controls corresponding to the entity types Objects, Sensors, and Regions of Interest.

- Object Controls. **Track** directs the system to begin actively tracking the current object. **Stop Tracking** terminates all active tracking tasks in the system. **Trajectory** displays the trajectory of selected objects. **Error** displays geolocation error bounds on the locations and trajectories of selected objects.
- Sensor Controls. **Show FOV** displays sensor fields of view on the map, otherwise only a position marker is drawn. **Move** triggers an interaction allowing the user to control the pan and tilt angle of the sensor. **Request Imagery** requests either a continuous stream or single image from the currently selected sensor, and **Stop Imagery** terminates the current imagery stream.
- ROI controls This panel contains all the controls associated with Regions of Interest (ROIs) in the system. ROIs are tasks that focus sensor resources at specific areas in the session space. **Create** triggers the creation of a ROI, specified interactively by the user as a polygon of boundary points. The user also selects from a set of object types (e.g. human, vehicle) that will trigger events in this ROI, and from a set of event types (e.g. enter, pass through, stop in) that are considered to be trigger events in the ROI.

## 2.4 Communication

The nominal architecture for the VSAM network allows multiple OCUs to be linked together, each controlling multiple SPUs (Figure 6). Each OCU supports exactly one GUI through which all user related command and control information is passed. Data dissemination is not limited to a single user interface, however, but is also accessible through a series of visualization nodes (VIS).

There are two independent communication protocols and packet structures supported in this architecture: the Carnegie Mellon University Packet Architecture (CMUPA) and the Distributed Interactive Simulation (DIS) protocols. The CMUPA is designed to be a low bandwidth, highly flexible architecture in which relevant VSAM information can be compactly packaged without

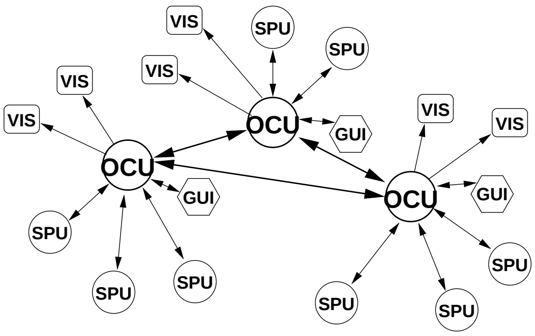



Figure 6: *A nominal architecture for expandable VSAM networks.*

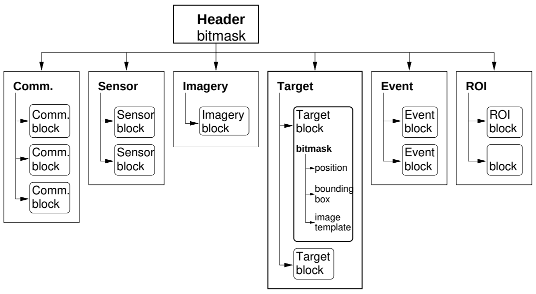



Figure 7: *CMUPA packet structure. A bitmask in the header describes which sections are present. Within each section, multiple data blocks can be present. Within each data block, bitmasks describe what information is present.*

redundant overhead. The concept of the CMUPA packet architecture is a hierarchical decomposition. There are six data sections that can be encoded into a packet: command; sensor; image; object; event; and region of interest. A short packet header section describes which of these six sections are present in the packet. Within each section it is possible to represent multiple instances of that type of data, with each instance potentially containing a different layout of information. At each level, short bitmasks are used to describe the contents of the various blocks within the packets, keeping wasted space to a minimum. All communication between SPUs, OCUs and GUIs is CMUPA compatible. The CMUPA protocol specification document is accessible from `http://www.cs.cmu.edu/~vsam`.

VIS nodes are designed to distribute the output of the VSAM network to where it is needed. They provide symbolic representations of detected activities overlaid on maps or imagery. Information flow to VIS nodes is unidirectional, originating from an OCU. All of this communication uses the DIS protocol, which is described in detail in [15]. An important benefit to keeping VIS nodes DIS compatible is that it allows us to easily interface with synthetic environment visualization tools such as ModSAF and ModStealth (Section 4.4).

## 2.5 Current Testbed Infrastructure

This section describes the VSAM testbed on the campus of Carnegie Mellon University, as of Fall 1999 (see Figure 8). The VSAM infrastructure consists of 14 cameras distributed throughout campus. All cameras are connected to the VSAM Operator Control Room in the Planetary Robotics Building (PRB): ten are connected via fiber optic lines, three on PRB are wired directly to the SPU computers, and one is a portable Small Unit Operations (SUO) unit connected via wireless Ethernet to the VSAM OCU. The work done for VSAM 99 concentrated on increasing the density of sensors in the Wean/PRB area. The overlapping fields of view (FOVs) in this area of campus enable us to conduct experiments in wide baseline stereo, object fusion, sensor cuing and sensor handoff.

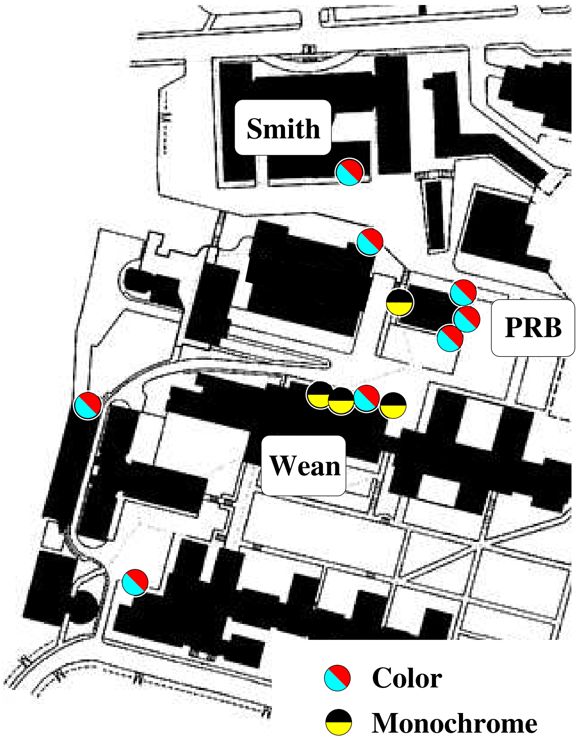



Figure 8: *Placement of color and monochrome cameras in current VSAM testbed system. Not shown are two additional cameras, a FLIR and the SUO portable system, which are moved to different places as needed.*

The backbone of the CMU campus VSAM system consists of six Sony EVI-370 color zoom cameras installed on PRB, Smith Hall, Newell-Simon Hall, Wean Hall, Roberts Hall, and Porter Hall. Five of these units are mounted on Directed Perception pan/tilt heads. The most recent camera, on Newell-Simon, is mounted on a Sagebrush Technologies pan/tilt head. This is a more rugged outdoor mount being evaluated for better performance specifications and longer term usage. Two stationary fixed-FOV color cameras are mounted on the peak of PRB, on either side of the pan/tilt/zoom color camera located there. These PRB "left" and "right" sensors were added to facilitate work on activity analysis, classification, and sensor cuing. Three stationary fixed-FOV monochrome cameras are mounted on the roof of Wean Hall in close proximity to one of the pan/tilt/zoom color cameras. These are connected to the Operator Control Room over a single multimode fiber using a video multiplexor. The monochrome cameras have a vertical resolution of 570 TV lines and perform fairly well at night with the available street lighting. A mounting bracket has also been installed next to these cameras for the temporary installation of a Raytheon NightSight thermal (FLIR) sensor. A fourth stationary fixed FOV monochrome camera is mounted on PRB pointing at the back stairwell. A SUO portable unit was built to allow further software development and research at CMU in support of the SUO program. This unit consists of the same hardware as the SPUs that were delivered to Fort Benning, Georgia in November, 1999.

The Operator Control Room in PRB houses the SPU, OCU, GUI and development workstations – nineteen computers in total. The four most recent SPUs are Pentium III 550 MHz computers. Dagwood, a single "compound SPU", is a quad Xeon 550 MHz processor computer, purchased to conduct research on classification, activity analysis, and digitization of three simultaneous video streams. Also included in this list of machines is a Silicon Graphics Origin 200, used to develop video database storage and retrieval algorithms as well as designing user interfaces for handling VSAM video data.

Two auto tracking Leica theodolites (TPS1100) are installed on the corner of PRB, and are hardwired to a data processing computer linked to the VSAM OCU. This system allows us to do real-time automatic tracking of objects to obtain ground truth for evaluating the VSAM geolocation and sensor fusion algorithms. This data can be displayed in real-time on the VSAM GUI.

An Office of Naval Research DURIP grant provided funds for two Raytheon NightSight thermal sensors, the Quad Xeon processor computer, the Origin 200, an SGI Infinite Reality Engine and the Leica theodolite surveying systems.

# 3 Video Understanding Technologies

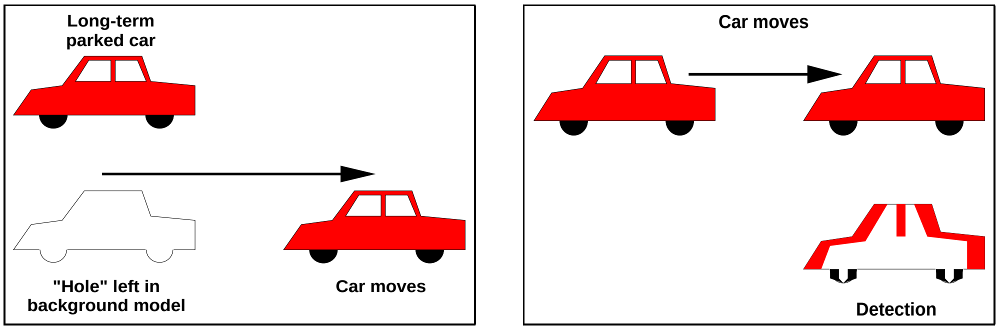
Keeping track of people, vehicles, and their interactions in a complex environment is a difficult task. The role of VSAM video understanding technology in achieving this goal is to automatically "parse" people and vehicles from raw video, determine their geolocations, and automatically insert them into a dynamic scene visualization. We have developed robust routines for detecting moving objects and tracking them through a video sequence using a combination of temporal differencing and template tracking. Detected objects are classified into semantic categories such as human, human group, car, and truck using shape and color analysis, and these labels are used to improve tracking using temporal consistency constraints. Further classification of human activity, such as walking and running, has also been achieved. Geolocations of labeled entities are determined from their image coordinates using either wide-baseline stereo from two or more overlapping camera views, or intersection of viewing rays with a terrain model from monocular views. The computed geolocations are used to provide higher-level tracking capabilities, such as tasking multiple sensors with variable pan, tilt and zoom to cooperatively track an object through the scene. Results are displayed to the user in real-time on the GUI, and are also archived in web-based object/event database.

## 3.1 Moving Object Detection

Detection of moving objects in video streams is known to be a significant, and difficult, research problem [26]. Aside from the intrinsic usefulness of being able to segment video streams into moving and background components, detecting moving blobs provides a focus of attention for recognition, classification, and activity analysis, making these later processes more efficient since only “moving” pixels need be considered.

There are three conventional approaches to moving object detection: temporal differencing [1]; background subtraction [13, 29]; and optical flow (see [3] for an excellent discussion). Temporal differencing is very adaptive to dynamic environments, but generally does a poor job of extracting all relevant feature pixels. Background subtraction provides the most complete feature data, but is extremely sensitive to dynamic scene changes due to lighting and extraneous events. Optical flow can be used to detect independently moving objects in the presence of camera motion; however, most optical flow computation methods are computationally complex, and cannot be applied to full-frame video streams in real-time without specialized hardware.

Under the VSAM program, CMU has developed and implemented three methods for moving object detection on the VSAM testbed. The first is a combination of adaptive background subtraction and three-frame differencing (Section 3.1.1). This hybrid algorithm is very fast, and surprisingly effective – indeed, it is the primary algorithm used by the majority of the SPUs in the VSAM system. In addition, two new prototype algorithms have been developed to address shortcomings of this standard approach. First, a mechanism for maintaining temporal object layers is developed to allow greater disambiguation of moving objects that stop for a while, are occluded by other objects, and that then resume motion (Section 3.1.2). One limitation that affects both this method and the standard algorithm is that they only work for static cameras, or in a ”step-and-stare” mode for pan-tilt cameras. To overcome this limitation, a second extension has been developed to allow background subtraction from a continuously panning and tilting camera (Section 3.1.3). Through clever accumulation of image evidence, this algorithm can be implemented in real-time on a conventional PC platform. A fourth approach to moving object detection from a moving airborne platform has also been developed, under a subcontract to the Sarnoff Corporation. This approach is based on image stabilization using special video processing hardware. It is described later, in Section 3.6.




(a) (b)

Figure 9: problems with standard MTD algorithms. (a) Background subtraction leaves "holes" when stationary objects move. (b) Frame differencing does not detect the entire object

### 3.1.1 A Hybrid Algorithm for Moving Object Detection

We have developed a hybrid algorithm for detecting moving objects, by combining an adaptive background subtraction technique[18] with a three-frame differencing algorithm. As discussed in [26], the major drawback of adaptive background subtraction is that it makes no allowances for stationary objects in the scene that start to move. Although these are usually detected, they leave behind "holes" where the newly exposed background imagery differs from the known background model (see Figure 9a). While the background model eventually adapts to these "holes", they generate false alarms for a short period of time. Frame differencing is not subject to this phenomenon, however, it is generally not an effective method for extracting the entire shape of a moving object (Figure 9b). To overcome these problems, we have combined the two methods. A three-frame differencing operation is performed to determine regions of legitimate motion, followed by adaptive background subtraction to extract the entire moving region.

Consider a video stream from a stationary (or stabilized) camera. Let $I_n(x)$ represent the intensity value at pixel position $x$, at time $t = n$. The three-frame differencing rule suggests that a pixel is legitimately moving if its intensity has changed significantly between both the current image and the last frame, and the current image and the next-to-last frame. That is, a pixel $x$ is moving if

$$(\ |I_n(x) - I_{n-1}(x)| > T_n(x)\ ) \text{ and } (\ |I_n(x) - I_{n-2}(x)| > T_n(x)\ )$$

where $T_n(x)$ is a threshold describing a statistically significant intensity change at pixel position $x$ (described below). The main problem with frame differencing is that pixels interior to an object with uniform intensity aren't included in the set of "moving" pixels. However, after clustering moving pixels into a connected region, interior pixels can be filled in by applying adaptive background subtraction to extract all of the "moving" pixels within the region's bounding box $R$. Let $B_n(x)$ represent the current background intensity value at pixel $x$, learned by observation over time. Then the blob $b_n$ can be filled out by taking all the pixels in $R$ that are significantly different from the background model $B_n$. That is

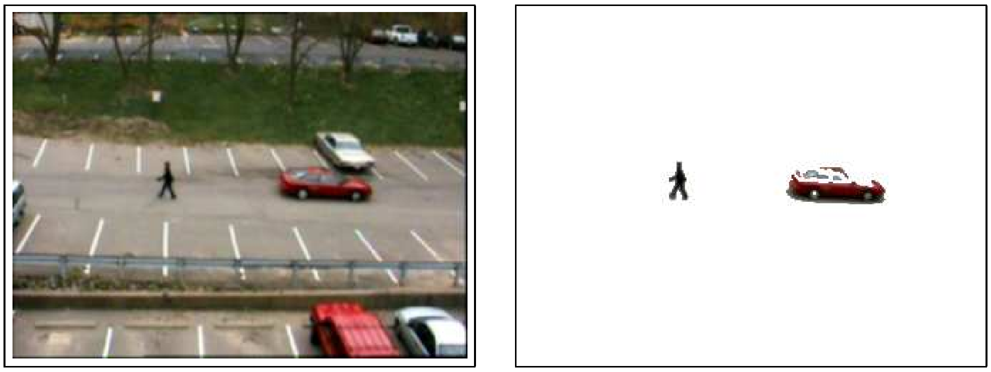
$$b_n = \{x :\ |I_n(x) - B_n(x)| > T_n(x),\ x \in R\}$$

Both the background model $B_n(x)$ and the difference threshold $T_n(x)$ are statistical properties of the pixel intensities observed from the sequence of images $\{I_k(x)\}$ for $k < n$. $B_0(x)$ is initially set to the first image, $B_0(x) = I_0(x)$, and $T_0(x)$ is initially set to some pre-determined, non-zero value. $B(x)$ and $T(x)$ are then updated over time as:

$$B_{n+1}(x) = \begin{cases} \alpha\, B_n(x) \;+\; (1-\alpha)\, I_n(x), & x \text{ is non-moving} \\ B_n(x), & x \text{ is moving} \end{cases}$$

$$T_{n+1}(x) = \begin{cases} \alpha\, T_n(x) \;+\; (1-\alpha)\, (5 \times |I_n(x) - B_n(x)|), & x \text{ is non-moving} \\ T_n(x), & x \text{ is moving} \end{cases}$$

where $\alpha$ is a time constant that specifies how fast new information supplants old observations. Note that each value is only changed for pixels that are determined to be non-moving, i.e. part of the stationary background. If each non-moving pixel position is considered as a time series, $B_n(x)$ is analogous to a local temporal average of intensity values, and $T_n(x)$ is analogous to 5 times the local temporal standard deviation of intensity, both computed using an infinite impulse response (IIR) filter. Figure 10 shows a result of this detection algorithm for one frame.



(a) (b)

Figure 10: Result of the detection algorithm. (a) Original image. (b) Detected motion regions.

#### 3.1.2 Temporal Layers for Adaptive Background Subtraction

A robust detection system should be able to recognize when objects have stopped and even disambiguate overlapping objects — functions usually not possible with traditional motion detection algorithms. An important aspect of this work derives from the observation that legitimately moving objects in a scene tend to cause much faster transitions than changes due to lighting, meteorological, and diurnal effects. This section describes a novel approach to object detection based on layered adaptive background subtraction.

### The Detection Algorithm

Layered detection is based on two processes: pixel analysis and region analysis. The purpose of pixel analysis is to determine whether a pixel is *stationary* or *transient* by observing its intensity value over time. Region analysis deals with the agglomeration of groups of pixels into moving regions and stopped regions. Figure 11 graphically depicts the process. By observing the intensity transitions of a pixel, different intensity layers, connected by transient periods, can be postulated.

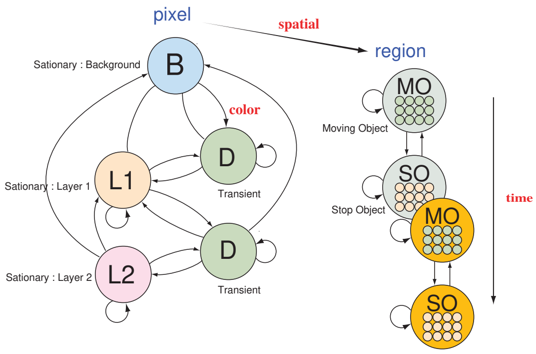



Figure 11: The concept — combining pixel statistics with region analysis to provide a layered approach to motion detection.

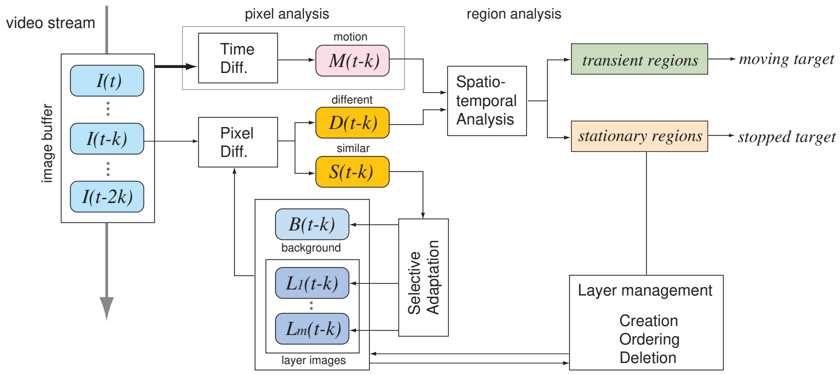



Figure 12: Architecture of the detection process. Temporal analysis is used on a per pixel basis to determine whether pixels are transient or stationary. Transient pixels are clustered into groups and assigned to spatio-temporal layers. A layer management process keeps track of the various background layers.

Figure 12 shows the architecture of the detection processes. A key element of this algorithm is that it needs to observe the behavior of a pixel for some time before determining if that pixel is undergoing a transition. It has been observed that a pixel's intensity value displays three characteristic profiles depending on what is occurring in the scene at that pixel location

- A legitimate object moving through the pixel displays a profile that exhibits a step change

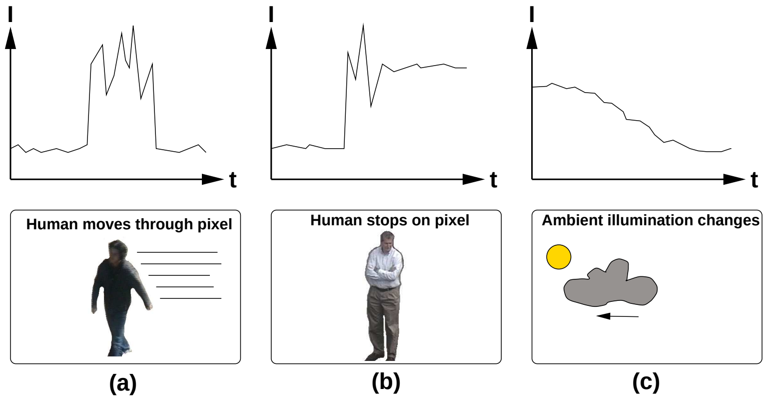



Figure 13: *Characteristic pixel intensity profiles for common events. Moving objects passing through a pixel cause an intensity profile step change, followed by a period of instability. If the object passes through the pixel (a), the intensity returns to normal. If the object stops (b), the intensity settles to a new value. Variations in ambient lighting (c) exhibit smooth intensity changes with no large steps.*

in intensity, followed by a period of instability, then another step back to the original background intensity. Figure 13(a) shows this profile.

- A legitimate object moving through the pixel and stopping displays a profile that exhibits a step change in intensity, followed by a period of instability, then it settles to a new intensity as the object stops. Figure 13(b) shows this profile.

- Changes in intensity caused by lighting or meteorological effects tend to be smooth changes that don't exhibit large steps. Figure 13(c) shows this profile.

To capture the nature of changes in pixel intensity profiles, two factors are important: the existence of a significant step change in intensity, and the intensity value to which the profile stabilizes after passing through a period of instability. To interpret the meaning of a step change (e.g. object passing through, stopping at, or leaving the pixel), we need to observe the intensity curve re-stabilizing after the step change. This introduces a time-delay into the process. In particular, current decisions are made about pixel events $k$ frames in the past. In our implementation $k$ is set to correspond to one second of video.

Let $I_t$ be some pixel's intensity at a time $t$ occurring $k$ frames in the past. Two functions are computed: a motion trigger $T$ just prior to the frame of interest $t$, and a stability measure $S$ computed over the $k$ frames from time $t$ to the present. The motion trigger is simply the maximum absolute difference between the pixel's intensity $I_t$ and its value in the previous five frames

$$T = \max \left\{ |I_t - I_{(t-j)}| \, , \, \forall j \in [1,5] \right\}$$

The stability measure is the variance of the intensity profile from time $t$ to the present:

$$S = \frac{k\sum_{j=0}^{k} I_{(t+j)}{}^2 - (\sum_{j=0}^{k} I_{(t+j)})^2}{k(k-1)}$$

At this point a transience map $M$ can be defined for each pixel, taking three possible values: background=0; transient=1 and stationary=2.

```
if ((M = stationary or background) AND (T > Threshold))
   M = transient
else {
   if ((M = transient) AND (S < Threshold)) {
      if (stabilized intensity value = background intensity)
         M = background
      else
         M = stationary
    }
}
```

Non-background pixels in the transience map $M$ are clustered into regions $R_i$ using a nearest neighbor spatial filter with clustering radius $r_c$. This process is similar to performing a connected components segmentation, however gaps up to a distance of $r_c$ pixels can be tolerated within a component. Choice of $r_c$ depends upon the scale of the objects being tracked. Each spatial region $R$ is then analyzed according to the following algorithm:

```
if (R = transient) {  %all pixels in R are labeled as transient
    R -> moving object
}
elseif (R = stationary) { %all pixels in R are labeled as stationary
    %remove all pixels already assigned to any layer
    R = R - (L(0) + L(1) + .. + L(j))
    %if anything is left, make a new layer out of it
    if (R != 0) {
        make new layer L(j+1) = R
        R -> stopped object
    }
else {  %R contains a mixture of transient and stationary pixels
    perform spatial clustering on R - (L(0) + L(1) + .. + L(j)
    for each region SR produced by that spatial clustering
        if (SR = transient) {
            SR -> moving object
```

```
        }
        if (SR = stationary) {
            make new layer L(j+1) = SR
            SR -> stopped object
        }
        if (SR = (stationary + transient)) {
            SR -> moving object
        }
    }
```

Regions that consist of stationary pixels are added as a layer over the background. A layer management process is used to determine when stopped objects resume motion or are occluded by other moving or stationary objects. Stationary layered regions and the scene background $B$ are updated by an IIR filter, as described in the last section, to accommodate slow lighting changes and noise in the imagery, as well as to compute statistically significant threshold values.

**Detection Results**

Figure 14 shows an example of the analysis that occurs at a single pixel. The video sequence contains the following activities at the pixel:

1. A vehicle drives through the pixel and stops
2. A second vehicle occludes the first and stops
3. A person, getting out of the second vehicle, occludes the pixel
4. The same person, returning to the vehicle, occludes the pixel again
5. The second car drives away
6. The first car drives away

As can be seen, each of these steps is clearly visible in the pixel's intensity profile, and the algorithm correctly identifies the layers that accumulate.

Figure 15 shows the output of the region-level layered detection algorithm. The detected regions are shown surrounded by bounding boxes — note that all three overlapping objects are independently detected. Each stopped car is depicted as a temporary background layer, and the person is determined to be a moving foreground region overlayed on them. The pixels belonging to each car and to the person are well disambiguated.

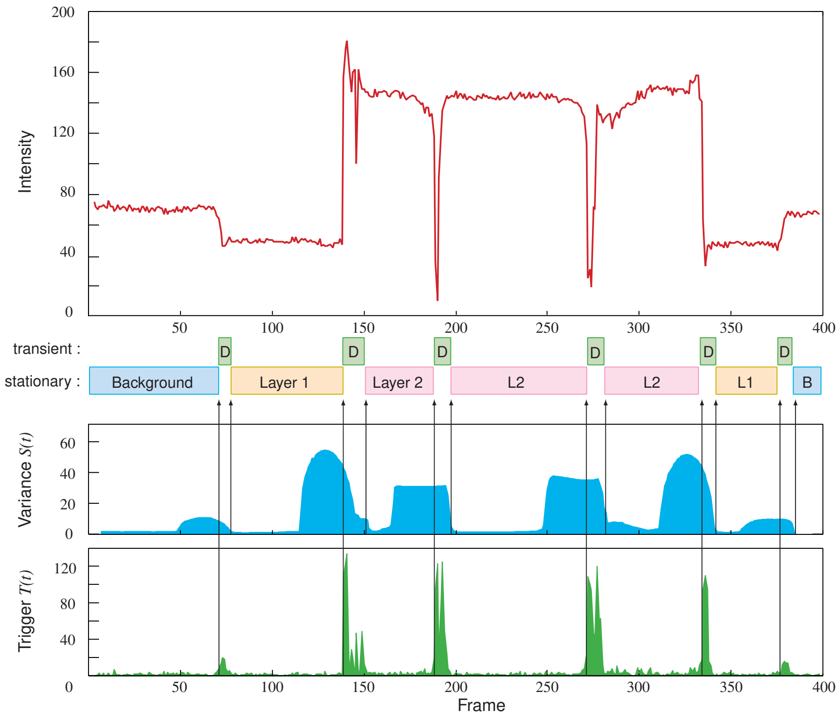



Figure 14: Example pixel analysis of the scene shown in figure 15. A car drives in and stops. Then a second car stops in front of the first. A person gets out and then returns again. The second car drives away, followed shortly by the first car.

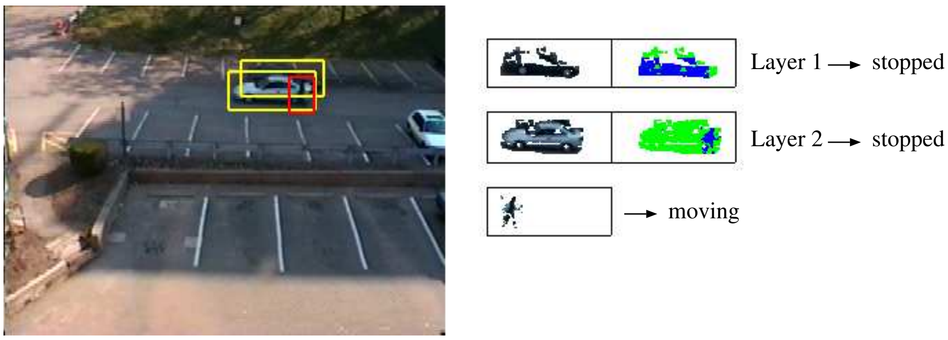



Figure 15: Detection result. Here one stopped vehicle partially occludes another, while a person in moving in the foreground. Displayed on the right are the layers corresponding to the stopped vehicles and the moving foreground person, together with bitmaps denoting which pixels are occluded in each layer.

### 3.1.3 Background Subtraction from a Continuously Panning Camera

Pan-tilt camera platforms can maximize the virtual field of view of a single camera without the loss of resolution that accompanies a wide-angle lens. They also allow for active tracking of an object of interest through the scene. However, moving object detection using background subtraction is not directly applicable to a camera that is panning and tilting, since all image pixels are moving . It is well known that camera pan/tilt is approximately described as a pure camera rotation, where apparent motion of pixels depends only on the camera motion, and not on the 3D scene structure. In this respect, the problems associated with a panning and tilting camera are much easier than if the camera were mounted on a moving vehicle traveling through the scene.

We ultimately seek to generalize the use of adaptive background subtraction to handle panning and tilting cameras, by representing a full spherical background model. There are two algorithmic tasks that need to be performed: 1) background subtraction: as the camera pans and tilts, different parts of the full spherical model are retrieved and subtracted to reveal the independently moving objects. 2) background updating: as the camera revisits various parts of the full field of view, the background intensity statistics in those areas must be updated. Both of these tasks depend on knowing the precise pointing direction of the sensor, or in other words, the mapping between pixels in the current image and corresponding pixels in the background model. Although we can read the current pan and tilt angles from encoders on the pan-tilt mechanism, this information is only reliable when the camera is stationary (due to unpredictable communication delays, we can not precisely know the pan-tilt readings for a given image while the camera is moving). Our solution to the problem is to register each image to the current spherical background model, thereby inferring the correct pan-tilt values, even while the camera is rotating.

Figure 16: Set of background reference images for a panning and tilting camera.

Maintaining a background model larger than the camera's physical field of view entails representing the scene as a collection of images. In our case, an initial background model is constructed by methodically collecting a set of images with known pan-tilt settings. An example view set is shown in Figure 16. One approach to building a background model from these images would be to stitch them together into a spherical or cylindrical mosaic, however we use the set of images directly, determining which is the appropriate one based on the distance in pan-tilt space. The warping transformation between the current image and a nearby reference image is therefore a simple planar projective transform.

The main technical challenge is how to register incoming video frames to the appropriate background reference image in real-time. Most image registration techniques are difficult to implement in real time without the use of special video processing hardware. We have developed a novel approach to registration that relies on selective integration of information from a small subset of pixels that contain the most information about the state variables to be estimated (the 2D projective transformation parameters). The dramatic decrease in the number of pixels to process results in a substantial speedup of the registration algorithm, to the point that it runs in real-time on a modest PC platform. More details are presented in [8]. Results from a sample frame registration and background subtraction are shown in Figure 17.

Figure 17: Results of background subtraction from a panning and tilting camera. From left to right: 1) current video frame, 2) closest background reference image, 3) warp of current frame into reference image coordinates, 4) absolute value of difference between warped frame and background reference image.

## 3.2 Object Tracking

To begin to build a temporal model of activity, individual object blobs generated by motion detection are tracked over time by matching them between frames of the video sequence. Many systems for object tracking are based on Kalman filters. However, pure Kalman filter approaches are of limited use because they are based on unimodal Gaussian densities that cannot support simultaneous alternative motion hypotheses [14]. We extend the basic Kalman filter notion to maintain a list of multiple hypotheses to handle cases where there is matching ambiguity between multiple moving objects. Object trajectories are also analyzed to help reduce false alarms by distinguishing between legitimate moving objects and noise or clutter in the scene.

An iteration of the basic tracking algorithm is

```
1) Predict positions of known objects
2) Associate predicted objects with current objects
```

```
3) If tracks split, create new tracking hypothesis
4) If tracks merge, merge tracking hypotheses
5) Update object track models
6) Reject false alarms
```

Each object in each frame is represented by the following parameters: 1) $p$ = position in image coordinates; 2) $\delta p$ = position uncertainty; 3) $\vec{v}$ = image velocity; 4) $\delta\vec{v}$ = uncertainty in velocity; 5) object bounding box in image coordinates; 6) image intensity template; 7) a numeric confidence measure and 8) a numeric salience measure.

### Predicting Future Object Positions

Both for computational simplicity and accurate tracking, it is important to estimate the position of an object at each iteration of the tracker. The estimated position is used to cull the number of moving regions that need to be tested. An object's future position in the image is estimated in the typical manner. Given a time interval $\Delta t$ between two samples, the position is extrapolated as

$$p_{n+1} = p_n + \vec{v}_n \Delta t$$

And the uncertainty in the position is assumed to be the original position uncertainty plus the velocity uncertainty, grown as a function of time

$$\delta p_{n+1} = \delta p_n + \delta\vec{v}_n \Delta t$$

These values are used to choose candidate moving regions from the current frame. This is done by extrapolating the bounding box of the object by $\vec{v}_n \Delta t$ and growing it by $\delta p_{n+1}$. Any moving region $R_{n+1}$ whose centroid falls in this predicted bounding box is considered a candidate for matching.

### Object Matching

Given an object region $R$ in the current frame, we determine the best match in the next frame by performed image correlation matching, computed by convolving the object's intensity template over candidate regions in the new image. That is, to evaluate a potential object displacement $d$ for image region $R$, we accumulate a weighted sum of absolute intensity differences between each pixel $x$ in region $R$ and the corresponding pixel $x+d$ in the next frame, yielding a correlation function $C(d)$ as:

$$C(d) = \sum_{x \in R} \frac{W(i,j)\, |I_n(x) - I_{n+1}(x+d)|}{||W||} \tag{1}$$

Here $W$ is the weighting function, which will be described shortly, and $||W||$ is a normalization constant given by

$$||W|| = \sum_{x \in R} W(x) \tag{2}$$

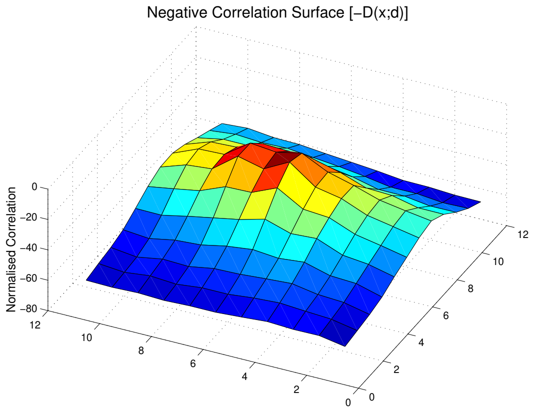



Figure 18: A typical correlation surface (inverted for easier viewing)

Graphically, the results of $C(d)$ for all offsets $d$ can be thought of as a correlation surface (Figure 18), the minimum of which provides both the position of the best match, and a measure of the quality of the match. The position of the best match $\hat{d}$ is given by the argmin of the of correlation surface

$$\hat{d} = \min_{d} C(d)$$

which can be refined to sub-pixel accuracy using bi-quadratic interpolation around $\hat{d}$. The new position of the object corresponding to this match is $p_{n+1} = p_n + d$, and the new velocity estimate is given by $\hat{v}_{n+1} = \frac{d}{\Delta t}$. The quality of the match $Q(R)$ is the value of $\min C(d)$.

Due to real-time processing constraints in the VSAM testbed system, this basic correlation matching algorithm is modified in two ways to improve computational efficiency. First, correlation is only computed for "moving" pixels [18]. This is achieve by setting the weighting function $W$ to zero for pixels that are not moving, and thus not performing any computation for these pixels. For moving pixels, a radial, linear weighting function is used:

$$W(x) = \frac{1}{2} + \frac{1}{2}\left(1 - \frac{r(x)}{r_{\max}}\right),$$

where $r(x)$ is the radial distance, in pixels, from $x$ to the center of the region $R$, and $r_{\max}$ is the largest radial distance in $R$. This has the effect of putting more weight on pixels in the center of the object.

Second, and more significantly, imagery is dynamically sub-sampled to ensure a constant computational time per match. When matching an $n \times m$ size image template, the computation is $O(n^2m^2)$ which rapidly becomes unwieldy for large templates. The notion, then, is to fix a threshold above which size, an image is sub-sampled. Furthermore, we treat the $x$ and $y$ dimensions separately, so that no data is lost in one dimension if it is already small enough for efficient matching. In this case, the threshold is set at 25 pixels, determined empirically to provide a reasonable quantity of data for correlation matching without over-stressing the computational engine. The algorithm is:

```
while (n > 25)
    sub-sample in 'x' direction by 2;
while (m > 25)
    sub-sample in 'y' direction by 2;
```

Of course, physically sub-sampling the imagery is almost as computationally expensive as correlation matching, so this is implemented by counting the number of times sub-sampling should be performed in each direction and selecting pixels at this spacing during the correlation process. For example, an $80 \times 45$ image would be sub-sampled twice in the $x$ direction and once in the $y$ direction making it a $20 \times 22$ image. So, in the correlation process, every $4^{\text{th}}$ pixel in the $x$ direction and every $2^{\text{nd}}$ pixel in the $y$ direction are chosen for matching. The loss in resolution is (almost) made up by the sub-pixel accuracy of the method. This method ensures that the computational complexity of the matching process is $< O(25^4)$. The complexity of the matching as a function of $n$ and $m$ is shown in Figure 19.

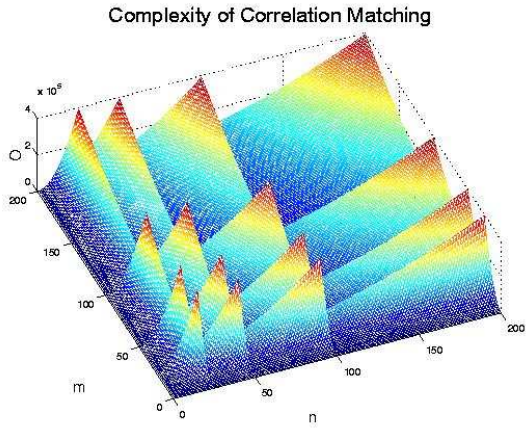



Figure 19: The computational complexity of the correlation matching algorithm with a threshold of 25. Clearly, the complexity is bounded at $O(25^4)$.

**Hypothesis Tracking and Updating**

Tracking objects in video is largely a matter of matching. The idea is, at each frame, to match known objects in one frame with moving regions in the next. There are 5 simple scenarios which might arise:

- A moving region exists that does not match with any known object. In this case, a new object is hypothesized and it's confidence is set to a nominal low value.
- An object does not match any moving region. Either the object has left the field of view, has been occluded, or has not been detected. In this case, the confidence measure of the object is reduced. If the confidence drops below a threshold, the object is considered lost.

- An object matches exactly one moving region. This is the best case for tracking. Here, the trajectory of the object is updated with the information from the new moving region and the confidence of the object is increased.
- An object matches multiple moving regions. This could occur if the object breaks into several independent objects (such as a group of people breaking up or a person getting out of a car), or the detection algorithm does not cluster the pixels from an object correctly. In this case, the best region (indicated by the correlation matching value) is chosen as the new object position, its confidence value is increased, and any other moving regions are considered as new object hypotheses and updated accordingly.
- Multiple objects match a single moving region. This might occur if two objects occlude each other, two objects merge (such as a group of people coming together), or an erroneously split object is clustered back together. This case is a special exception. Here, an analysis must be done of the object trajectories to determine how to update the object hypotheses. Objects merging into a single moving region are each tracked separately. Their trajectories are then analyzed. If they share the same velocity for a period of time, they are merged into a single object. If not, they are tracked separately. This allows the system to continue to track objects that are occluding each other and yet merge ones that form a single object.

Object parameters are updated based on the parameters of the matched new observations (the moving regions). The updated position estimate$p_{n+1}$ of the object is the position calculated to sub-pixel accuracy by the correlation matching process The new velocity estimate $\hat{v}_{n+1}$ calculated during matching is filtered through an IIR filter to provide $\vec{v}_{n+1}$

$$\vec{v}_{n+1} \;=\; \alpha\,\hat{v}_{n+1} \;+\; (1-\alpha)\,\vec{v}_n$$

and the new velocity uncertainty estimate is generated using an IIR filter in the same way

$$\delta\vec{v}_{n+1} \;=\; \alpha\,|\vec{v}_{n+1}-\hat{v}_{n+1}| \;+\; (1-\alpha)\,\vec{v}_n$$

In most cases, the template of the object is taken as the template of the moving region, and the confidence is increased. However, if multiple objects are matched to a single moving region, the templates are not updated. If two objects have come together and are occluding, the template of each could be corrupted by the other if they were updated. The philosophy behind this decision is that, hopefully, two occluding objects will not change their appearance greatly during the occlusion, and tracking will still be possible after the occlusion is finished. Note that even though multiple objects may match to the same moving region, they will not necessarily get the same position estimate because the correlation matching process will match them to different parts of the region.

Any object that has not been matched maintains its position and velocity estimates, and current image template. Its confidence is then reduced. If the confidence of any object drops below a certain threshold, it is considered lost, and dropped from the list. High confidence objects (ones that have been tracked for a reasonable period of time) will persist for several frames; so if an object is momentarily occluded, but then reappears, the tracker will reacquire it. Recent results from the system are shown in Figure 20.


Figure 20: Tracking two objects simultaneously.

**False Alarm Rejection**

A serious issue with moving object tracking is the disambiguation of legitimate objects from "motion clutter" such as trees blowing in the wind, moving shadows, or noise in the video signal. One cue to the legitimacy of an object track is persistence: an intermittent contact is less likely to be a valid object than a persistent one. Another cue is the purposefulness or salience of the trajectory: trees blowing in the wind tend to exhibit oscillatory motion whereas people and vehicles tend to move with a purpose.

The tracking scheme described above automatically deals with the persistence of objects, but special consideration must be made as to the salience of objects. The motion salience algorithm used is based on a cumulative flow technique due to Wixson. Here, the optic flow of moving objects is accumulated over time. However, when the flow changes direction, the accumulation is set to zero. This way, insalient motion, such as that from blowing trees, never accumulates significantly, whereas purposeful motion, such as a car driving along a road, accumulates a large flow.

Because optic flow is computationally expensive, a short cut is used. At each iteration, the displacement $d$ computed by the correlation matching process is taken as an average flow for the object. Initially, three parameters, frame count $c$, cumulative flow $d_{\mathrm{sum}}$ and maximum flow$d_{\max}$ are set to zero. The algorithm for determining motion salience is to cumulatively add the displacements at each frame to the cumulative flow, and increment the frame count. If, at any frame, the cumulative displacement falls to $< 90\%$ of the maximum value (indicating a change in direction), everything is set to zero again. Then, only objects whose displacements accumulate for several frames are considered to be salient. The algorithm is displayed in Figure 21

$d_{\text{sum}} = d_{\text{sum}} + d$
$c = c + 1$
**if** $(d_{\text{sum}} > d_{\max})$
    $d_{\max} = d_{\text{sum}}$
**if** $(d_{\text{sum}} < 0.9 \times d_{\max})$
    $d_{\text{sum}} = 0$
    $c = 0$
    $d_{\max} = 0$
**if** $(c >$ Threshold$)$
    Salient
**else**
    Not salient

Figure 21: Moving object salience algorithm.

## 3.3 Object Type Classification

The ultimate goal of the VSAM effort is to be able to identify individual entities, such as the "FedEx truck", the "4:15pm bus to Oakland" and "Fred Smith". Two object classification algorithms have been developed. The first uses view dependent visual properties to train a neural network classifier to recognize four classes: single human; human group; vehicles; and clutter (Section 3.3.1). The second method uses linear discriminant analysis to determine provide a finer distinction between vehicle types (e.g. van, truck, sedan) and colors (Section 3.3.2). This method has also been successfully trained to recognize specific types of vehicles, such as UPS trucks and campus police cars.

### 3.3.1 Classification using Neural Networks

The VSAM testbed classifies moving object blobs into general classes such as "humans" and "vehicles" using viewpoint-specific neural networks, trained for each camera. Each neural network is a standard three-layer network (Figure 22). Learning in the network is accomplished using the backpropagation algorithm. Input features to the network are a mixture of image-based and scene-based object parameters: image blob dispersedness (perimeter$^2$/area (pixels)); image blob area (pixels); apparent aspect ratio of the blob bounding box; and camera zoom. There are three output classes: human; vehicle; and human group. When teaching the network that an input blob is a human, all outputs are set to 0.0 except for "human", which is set to 1.0. Other classes are trained similarly. If the input does not fit any of the classes, such as a tree blowing in the wind, all outputs are set to 0.0.

Results from the neural network are interpreted as follows:

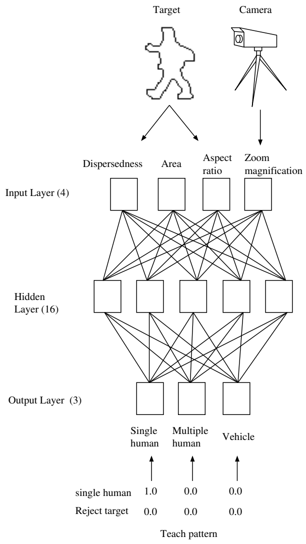



Figure 22: *Neural network approach to object classification.*

| Class | Samples | % Classified |
|---|---|---|
| Human | 430 | 99.5 |
| Human group | 96 | 88.5 |
| Vehicle | 508 | 99.4 |
| False alarms | 48 | 64.5 |
| Total | 1082 | 96.9 |

Table 1: *Results of neural net classification on VSAM data*

```
if (output > THRESHOLD)
  classification = maximum NN output
else
  classification = REJECT
```

This neural network classification approach is fairly effective for single images; however, one of the advantages of video is its temporal component. To exploit this, classification is performed on each blob at every frame, and the results of classification are kept in a histogram. At each time step, the most likely class label for the blob is chosen, as described in [20]. The results for this classification scheme are summarized in Table 1.

We have experimented with other features that disambiguate human from vehicle classes. These could also be incorporated into the neural network classification, at the expense of having to perform the extra feature computation. Given the geolocation of an object, as estimated from its image location and a terrain map (see Section 4.3), its actual width $w$ and height $h$ in meters can be estimated from its image projection. A simple heuristic based on the ratio of these values performs surprisingly well:

$$\begin{array}{llcl} w < 1.1 & h \in [0.5, 2.5] & \Rightarrow & \text{human} \\ w \in [1.1, 2.2] & h \in [0.5, 2.5] & \Rightarrow & \text{group} \\ w \in [2.2, 20] & h \in [0.7, 4.5] & \Rightarrow & \text{vehicle} \\ \textbf{ELSE} & & \Rightarrow & \text{reject} \end{array} \quad (3)$$

Another promising classification feature for a moving object is to determine whether it is rigid or non-rigid by examining changes in its appearance over multiple frames [28]. This is most useful for distinguishing rigid objects like vehicles from non-rigid walking humans and animals. In [21] we describe an approach based on local computation of optic flow within the boundaries of a moving object region. Given the gross displacement $d$ of a moving blob $R$, as calculated in Section 3.2, and the flow field $v(x)$ computed for all pixels $x$ in that blob, it is possible to determine the velocity of the pixels relative to the body's motion $d$ by simply subtracting off the gross motion

$$r(x) = v(x) - d$$

to find the *residual flow* $r(x)$. It is expected that rigid objects will have little residual flow, whereas a non-rigid object such as a human being will exhibit more independent motion. When the average absolute residual flow per pixel

$$A = \sum_{x \in R} \|r(x)\| / \sum_{x \in R} 1 \ .$$

is calculated, the magnitude of its value provides a clue to the rigidity of the object's motion, and over time its periodicity. Rigid objects such as vehicles display extremely low values of $A$ whereas moving objects such as humans display significantly more residual flow, with a periodic component (Figure 23).

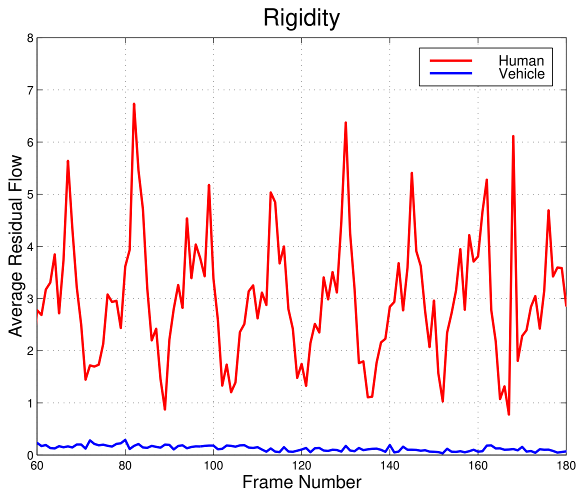



Figure 23: *Average magnitude of residual flow for a person (top curve) and a car (bottom curve), plotted over time. Clearly, the human has a higher average residual flow at each frame, and is thus less rigid. The curve also exhibits the periodicity of the non-rigid human gait.*

### 3.3.2 Classification using Linear Discriminant Analysis

We have developed a method for classifying vehicle types and people using linear discriminant analysis. The method has two sub-modules: one for classifying object "shape", and the other for determining "color". Each sub-module computes an independent discriminant classification space, and calculates the most likely class in that space using a weighted $k$-class nearest-neighbor ($k$-NN) method.

To calculate both discriminant spaces, Linear Discriminant Analysis (LDA) is used. LDA is a statistical tool for discriminating among groups, or clusters, of points in multidimensional space. LDA is often called supervised clustering. In LDA, feature vectors computed on training examples of different object classes are considered to be labeled points in a high-dimensional feature space. LDA then computes a set of discriminant functions, formed as linear combinations of feature values, that best separate the clusters of points corresponding to different object labels. LDA has the following desirable properties: 1) it reduces the dimensionality of the data, and 2) the classes in LDA space are separated as well as possible, meaning that the variance (spread of points) within each class is minimized, while the variance between the classes (spread of cluster centroids) is maximized.

LDA calculations proceed as follows. First, calculate the average covariance matrix of points within each class (W) and between different classes (B)

$$W = \sum_{c=1}^{C} \sum_{i=1}^{n_c} (x_{ic} - \bar{x}_c)\,(x_{ic} - \bar{x}_c)^T \tag{4}$$

$$B = \sum_{c=1}^{C} n_c (\bar{x}_c - \bar{x})\,(\bar{x}_c - \bar{x})^T \tag{5}$$

where $C$ is the number of object classes, $n_c$ is the number of training examples in class $c$, $x_{ic}$ is the feature vector of the $i$th example in class $c$, and $x_c$ is the centroid vector of class $c$. Then, compute the eigenvalues $\lambda_i$ and eigenvectors $b_i$ of the separation matrix $W^{-1}B$ by solving the generalized eigenvalue problem $(B - \lambda_i W) \times b_i = 0$. Assume without loss of generality that the eigenvalues have been been sorted so that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_N$, where $N$ is the dimensionality of the feature space. The eigenvector $b_i$ associated with each eigenvalue $\lambda_i$ provides the coefficients of the $i$th discriminant function, which maps feature vector $x$ into a coordinate in discriminant space. Dimensionality reduction is achieved by only considering the $M < N$ largest eigenvalues (and eigenvectors), thus mapping $N$-dimensional feature vector $x$ into an $M$-dimensional vector $y$ as

$$\underset{M \times 1}{y} = \underset{M \times N}{[b_1 b_2 \ldots b_n]^T} \;\; \underset{N \times 1}{x}$$

In practice, we choose $M$ to be the first integer such that

$$\sum_{i=1}^{M} \lambda_i \;\geq\; .99 \sum_{i=1}^{N} \lambda_i \;.$$

During on-line classification, feature vector $x$ is measured for a detected object, and transformed into a point $y$ in discriminant space. To determine the class of the object, the distance from point $y$ to points representing each labeled training example is examined, and the $k$ closest labeled examples are chosen. These are the $k$ nearest neighbors to $y$. According to the $k$-NN classification rule, the labels of these nearest neighbors provide votes for the label (class) of the new object, and their distance from $y$ provides a weight for each vote. The class of $y$ is chosen as the class that receives the highest weighted vote. Due to the disparity in numbers of training samples for each class, we also normalize the number of votes received for each class by the total number of training examples from that class.

**Shape classification, off-line learning process**

The supervised learning process for object type classification based on shape is performed through the following steps:

1. Human operators collect sample shape images and assign class labels to them. In this experiment, we specify six shape-classes: human (single and group), sedan (including 4WD), van, truck, Mule (golf carts used to transport physical plant workers) and other (mainly noise). We also labeled three "special" objects: FedEx vans, UPS vans, and Police cars. Figures on the following pages show sample input image chips for each of these object types. In total, we collected approximately 2000 sample shape images.
2. The system calculates area, center of gravity, and width and height of the motion blob in each sample image. The system also calculates 1st, 2nd and 3rd order image moments of each blob, along the x-axis and y-axis of the images. Together, these features comprise an 11-dimensional sample vector of calculated image features.
3. The system calculates a discriminant space for shape classification using the LDA method described above.

**Shape classification, on-line classification process**

In the on-line classification phase, the system executes all steps automatically.

1. The system calculates area, center of gravity, width and height of an input image, and 1st, 2nd and 3rd order image moments along the x-axis and y-axis, forming an 11-dimensional vector for the motion blob.
2. The corresponding point in discriminant space is computed as a linear combination of feature vector values.
3. Votes for each class are determined by consulting the 10 nearest neighbor point labels discriminant space, as described above.

### Color classification, off-line learning process

In addition to the type of object determined by blob shape, the dominant color of the object is also classified using LDA. Observed color varies according to scene lighting conditions, and for this reason, a discrete set of color classes is chosen that are fairly invariant (class-wise) to outdoor lighting changes, and the variation in each class is learned using LDA.

1. Human operators segment color samples from training images and divide them into six classes: 1) red-orange-yellow, 2) green, 3) blue-lightblue, 4) white-silver-gray, 5) darkblue-darkgreen-black, and 6) darkred-darkorange. We collected approximately 1500 images under fine weather conditions and 1000 images under cloudy conditions.
2. The system samples RGB intensity values of 25 pixels on each sample image. The system then maps sampled RGB values into (I1,I2,I3) color space values according to the following equations

$$I1 = \frac{(R+G+B)}{3.0} \times 10.0 \tag{6}$$

$$I2 = \frac{(R-B)}{2.0} \times 100.0 \tag{7}$$

$$I3 = \frac{(2.0 \times G - R - B)}{4.0} \times 100.0 \tag{8}$$

   The system averages the calculated (I1,I2,I3) values to get a single 3-dimensional color feature vector for the that image.
3. The system calculates a discriminant space for color classification using the LDA method described above.

### Color classification, on-line classification process

In the on-line classification phase, the system executes all steps automatically.

1. The system measures RGB samples every 2 pixels along the x and y axes of the input motion blob.
2. RGB values are converted to (I1,I2,I3) color space.
3. The corresponding points in discriminant space are computed as a linear combination of feature vector values, and the Euclidean distance to each color class is summed up.
4. Votes for each class are determined by consulting the 10 nearest neighbor point labels in discriminant space, as described above.
5. The color class associated with the shortest total Euclidean distance is chosen as the output color class.

Table 2: *Cross-validation results for LDA classification.*

| | Human | Sedan | Van | Truck | Mule | Others | Total | Errors | % |
|---|---|---|---|---|---|---|---|---|---|
| *Human* | 67 | 0 | 0 | 0 | 0 | 7 | 74 | 7 | 91% |
| *Sedan* | 0 | 33 | 2 | 0 | 0 | 0 | 35 | 2 | 94% |
| *Van* | 0 | 1 | 24 | 0 | 0 | 0 | 25 | 1 | 96% |
| *Truck* | 0 | 2 | 1 | 12 | 0 | 0 | 15 | 3 | 80% |
| *Mule* | 0 | 0 | 0 | 0 | 15 | 1 | 16 | 1 | 94% |
| *Others* | 0 | 2 | 0 | 0 | 0 | 13 | 15 | 2 | 87% |
| | | | | | | | | Avg. | *90%* |

**Results**

The following pages show some sample training image chips for different object types, and some sample output from the classification procedure. Table 2 shows a cross-validation evaluation between objects (columns) and classified results (rows).

The recognition accuracy has been found to be roughly 90%, under both sunny and cloudy weather conditions. Currently, the system does not work well when it is actually raining or snowing, because the raindrops and snowflakes interfere with the measured RGB values in the images. For the same reason, the system does not work well in early mornings and late evenings, due to the non-representativeness of the lighting conditions. The system is also foiled by backlighting and specular reflection from vehicle bodies and windows. These are open problems to be solved.

Trucks: Left

Trucks: Right

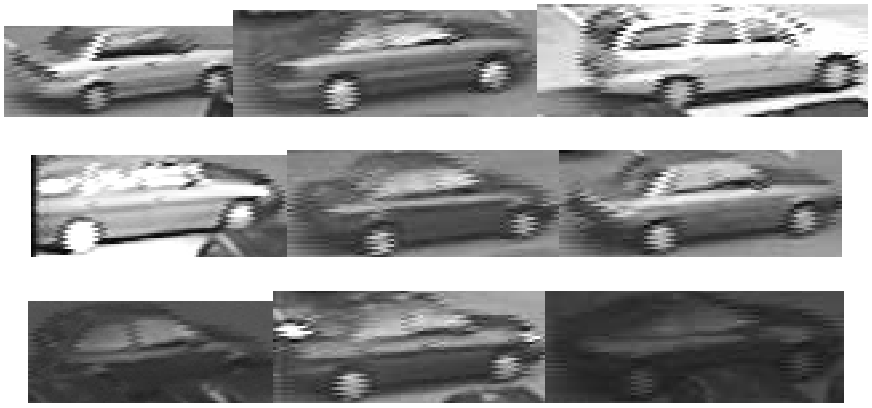
Sample images used for LDA learning : Trucks

Vans : Right

Vans : Left 1

Vans : Left 2

Sample images used for LDA learning : Vans



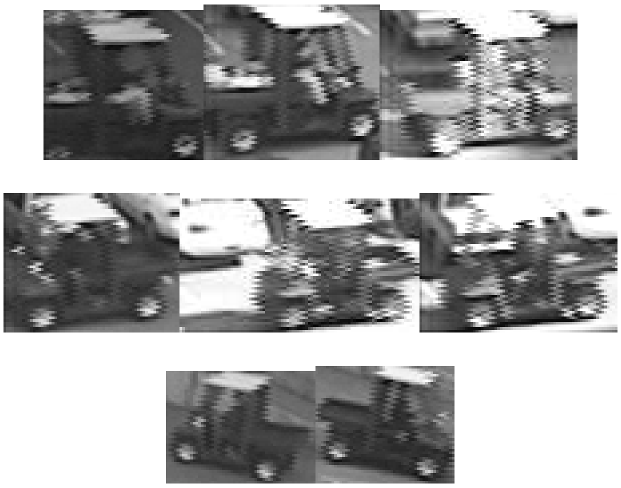
Sedans : Right

Sedans : Left 1

Sedans : Left 2

Sample images used for LDA learning : Sedans

(d) 4WDs



(e) Mules Sample images used for LDA learning : 4WDs and Mules

UPS 1

UPS 2


FedEx

Police cars

Sample images used for LDA learning : Special objects


Some final results of the classification process.

## 3.4 Activity Analysis

After detecting objects and classifying them as people or vehicles, we would like to determine what these objects are doing. In our opinion, the area of activity analysis is one of the most important open areas in video understanding research. We have developed two prototype activity analysis procedures. The first uses the changing geometry of detected motion blobs to perform gait analysis of walking and running human beings (Section 3.4.1). The second uses Markov model learning to classify simple interactions between multiple objects, such as two people meeting, or a vehicle driving into the scene and dropping someone off (Section 3.4.2).

### 3.4.1 Gait Analysis

Detecting and analyzing human motion in real-time from video imagery has only recently become viable, with algorithms like *Pfinder* [29] and $W^4$ [13]. These algorithms represent a good first step to the problem of recognizing and analyzing humans, but they still have drawbacks. In general, they work by detecting features (such as hands, feet and head), tracking them, and fitting them to a prior human model, such as the *cardboard model* of Ju *et al* [16].

We have developed a "star" skeletonization procedure for analyzing human gaits [10]. The key idea is that a simple, fast extraction of the broad internal motion features of an object can be employed to analyze its motion. A simple method is employed to robustly detect extremal points on the boundary of the object, to produce a "star" skeleton. The star skeleton consists of the centroid of a motion blob, and all of the local extremal points that are recovered when traversing the boundary of the blob (see Figure 24).

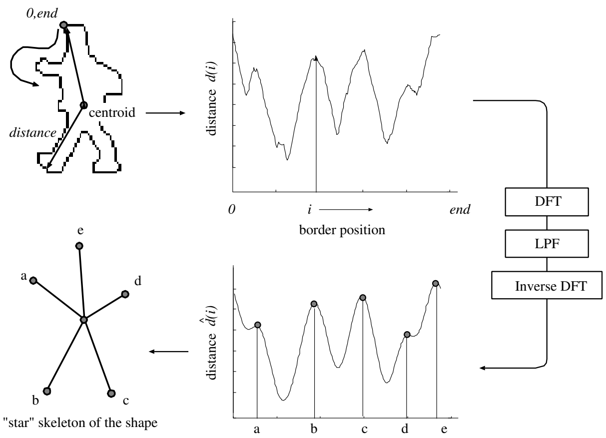



Figure 24: *The boundary is "unwrapped" as a distance function from the centroid. This function is then smoothed and extremal points are extracted.*

Figure 25 shows star skeletons extracted for various objects. It is clear that, while this form of skeletonization provides a sparse set of points, it can nevertheless be used to classify and analyze the motion of different types of moving object.

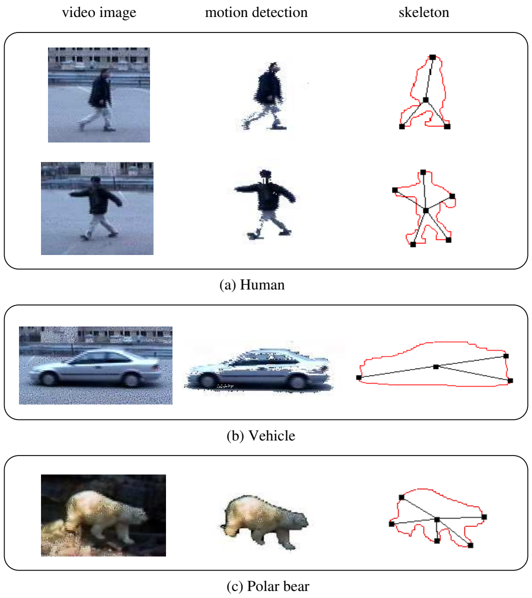



Figure 25: *Skeletonization of different moving objects. It is clear the structure and rigidity of the skeleton is significant in analyzing object motion.*

One technique often used to analyze the motion or gait of an individual is the cyclic motion of individual joint positions. However, in our implementation, the person may be a fairly small blob in the image, and individual joint positions cannot be determined in real-time, so a more fundamental cyclic analysis must be performed. Another cue to the gait of the object is its posture. Using only a metric based on the star skeleton, it is possible to determine the posture of a moving human. Figure 26 shows how these two properties are extracted from the skeleton. The uppermost skeleton segment is assumed to represent the torso, and the lower left segment is assumed to represent a leg, which can be analyzed for cyclic motion.

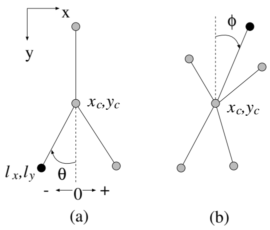



Figure 26: *Determination of skeleton features. (a) $\theta$ is the angle the left cyclic point (leg) makes with the vertical, and (b) $\phi$ is the angle the torso makes with the vertical.*

Figure 27 shows human skeleton motion sequences for walking and running, and the values of $\theta_n$ for the cyclic point. This data was acquired from video at a frame rate of 8Hz. Comparing the average values $\overline{\phi}_n$ in Figures 27(e)-(f) shows that the posture of a running person can easily be

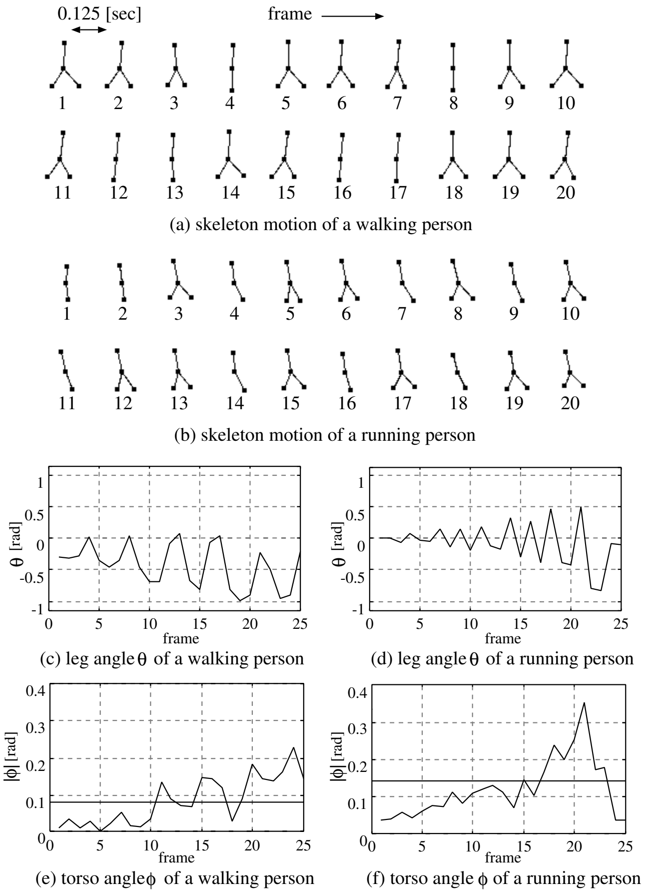



Figure 27: *Skeleton motion sequences. Clearly, the periodic motion of $\theta_n$ provides cues to the object's motion as does the mean value of $\overline{\phi}_n$.*

distinguished from that of a walking person, using the angle of the torso segment as a guide. Also, the frequency of cyclic motion of the leg segments provides cues to the type of gait.

### 3.4.2 Activity Recognition of Multiple Objects using Markov Models

We have developed a prototype activity recognition method that estimates activities of multiple objects from attributes computed by low-level detection and tracking subsystems. The activity label chosen by the system is the one that maximizes the probability of observing the given attribute sequence. To obtain this, a Markov model is introduced that describes the probabilistic relations between attributes and activities.

We tested the functionality of our method with synthetic scenes which have human-vehicle interaction. In our test system, continuous feature vector output from the low-level detection and tracking algorithms is quantized into the following discrete set of attributes and values for each tracked blob

- object class: Human, Vehicle, HumanGroup
- object action: Appearing, Moving, Stopped, Disappearing
- Interaction: Near, MovingAwayFrom, MovingTowards, NoInteraction

The activities to be labeled are 1) A Human entered a Vehicle, 2) A Human got out of a Vehicle, 3) A Human exited a Building, 4) A Human entered a Building, 5) A Vehicle parked, and 6) Human Rendezvous. To train the activity classifier, conditional and joint probabilities of attributes and actions are obtained by generating many synthetic activity occurrences in simulation, and measuring low-level feature vectors such as distance and velocity between objects, similarity of the object to each class category, and a noise-corrupted sequence of object action classifications. Figure 28 shows the results for two scenes that were not used for joint probability calculation.

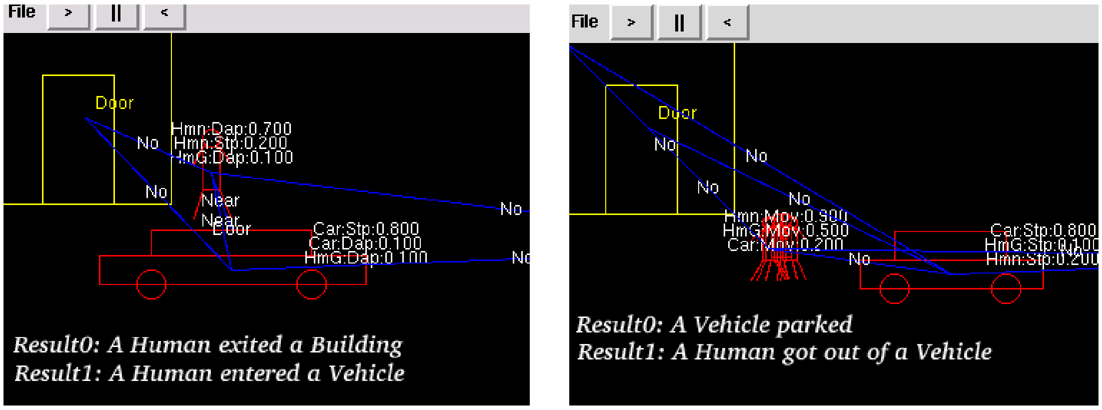



Figure 28: *Results of Markov activity recognition on synthetic scenes. Left: A person leaves a building and enters a vehicle. Right: A vehicle parks and a person gets out.*

## 3.5 Web-page Data Summarization

We have developed a web-based data logging system (see Figure 29). In a high-traffic area, data on dozens of people can be collected in just a few minutes of observation. Each observation consists of color or thermal video from multiple cameras, best view image chips, collateral information such as date and time, weather conditions, temperature, estimated 3D subject trajectory, camera acquisition parameters, and object classification results. In addition to storing data for evaluation and debugging, a data logging system will be necessary when VSAM systems begin 24/7 site monitoring operation.

In our data logging prototype, all observations can be explored by web browsing via CGI through an HTTP server, so that VSAM researchers can access the data from anywhere. There are two ways to view object and activity information. Figure 30 shows a example activity report. The activity report shows labeled events such as a “Car Parked”, or “A Human Entered a Building”,

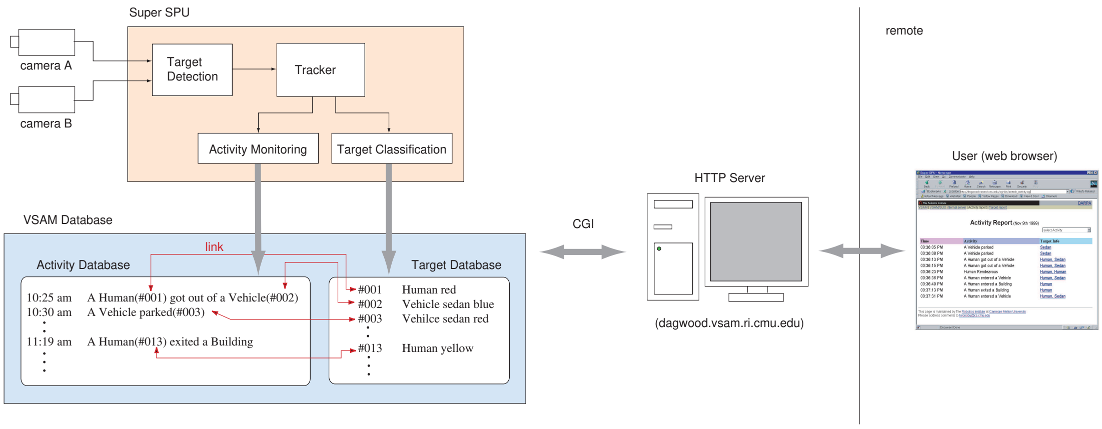



Figure 29: Web-page data summarization system.

sorted by time. If a user wants more detail, a hypertext link brings up a page showing an image chip of the object, along with its class and color information. Figure 31 shows an example object report. All of the objects seen by the system, and the activities to which they are related, are shown on a page, sorted by time of observation. To cut down on information overload, the user can select specific subsets of object classes to view. When the user selects an object, the system automatically brings up a page showing other objects of the same class having similar color features. In this way, it might be possible for a user to detect the same vehicle or person being observed at different places and times around the surveillance site.

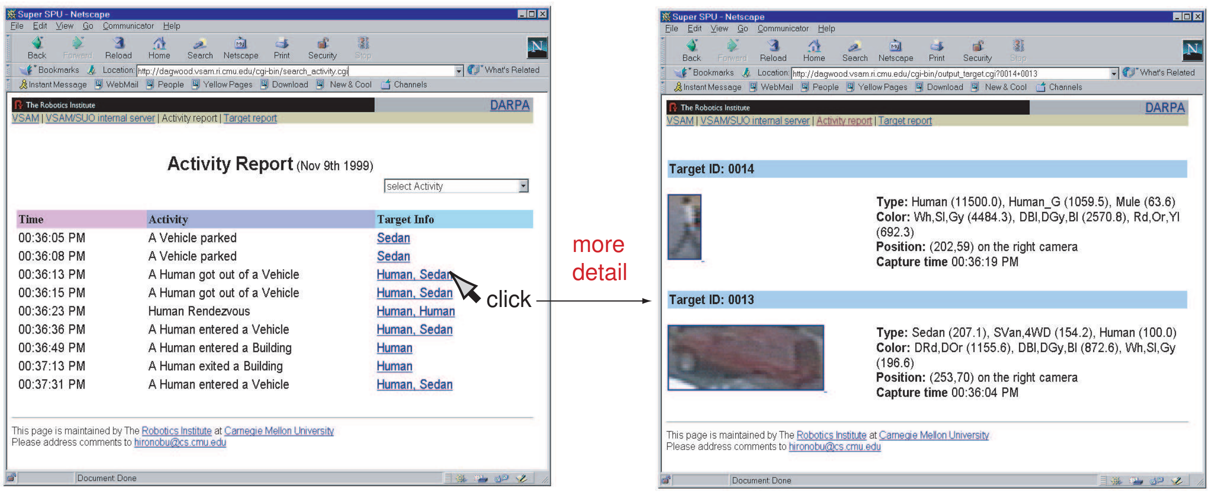



Figure 30: Activity report for web page.

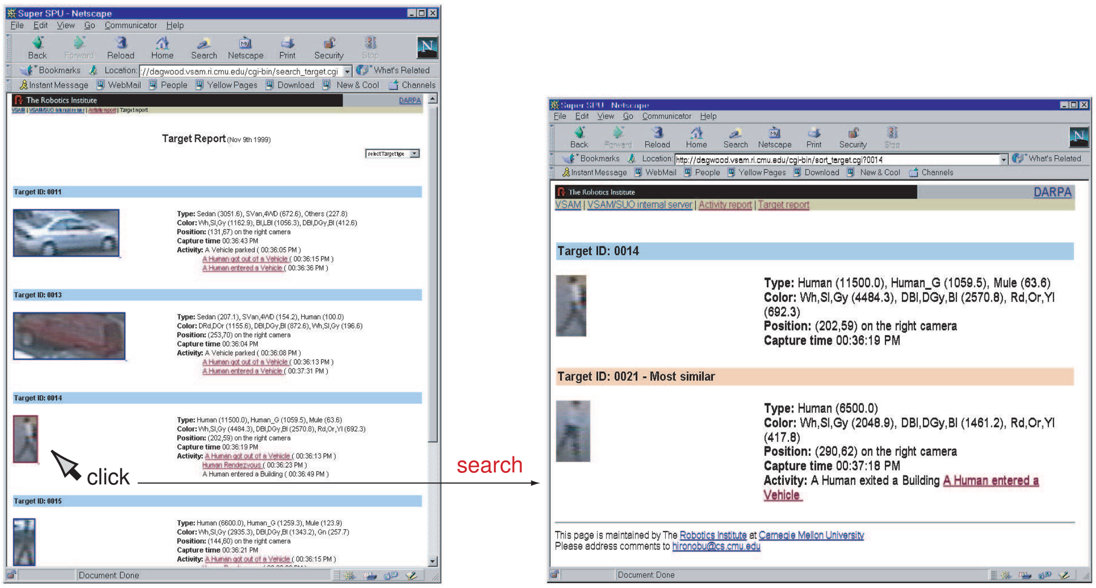



Figure 31: Object report for web page.

## 3.6 Airborne Surveillance

Fixed ground-sensor placement is fine for defensive monitoring of static facilities such as depots, warehouses or parking lots. In those cases, sensor placement can be planned in advance to get maximum usage of limited VSAM resources. However, the battlefield is a large and constantly shifting piece of real-estate, and it may be necessary to move sensors around in order to maximize their utility as the battle unfolds. While airborne sensor platforms directly address this concern, the self-motion of the aircraft itself introduces challenging video understanding issues. During the first two years of this program, the Sarnoff Corporation developed surveillance technology to detect and track individual vehicles from a moving aircraft, keep the camera turret fixated on a ground point, and multitask the camera between separate geodetic ground positions.

### 3.6.1 Airborne Object Tracking

Object detection and tracking is a difficult problem from a moving sensor platform. The difficulty arises from trying to detect small blocks of moving pixels representing independently moving object objects when the whole image is shifting due to self-motion. The key to success with the airborne sensor is characterization and removal of self-motion from the video sequence using the Pyramid Vision Technologies PVT-200 real-time video processor system. As new video frames stream in, the PVT processor registers and warps each new frame to a chosen reference image, resulting in a cancelation of pixel movement, and leading to a "stabilized" display that appears motionless for several seconds. During stabilization, the problem of moving object detection from a moving platform is ideally reduced to performing VSAM from a stationary camera, in the sense that moving objects are readily apparent as moving pixels in the image. Object detection and tracking is then performed using three-frame differencing after using image alignment to register frame $I_{t-2}$ to $I_t$ and frame $I_{t-1}$ to $I_t$, performed at 30 frames/sec. Sample results are shown in Figure 32. Under some circumstances, there is some remaining residual pixel motion due to parallax caused by significant 3D scene structure such as trees and smokestacks. Removing parallax effects is a subject of on-going research in the vision community.

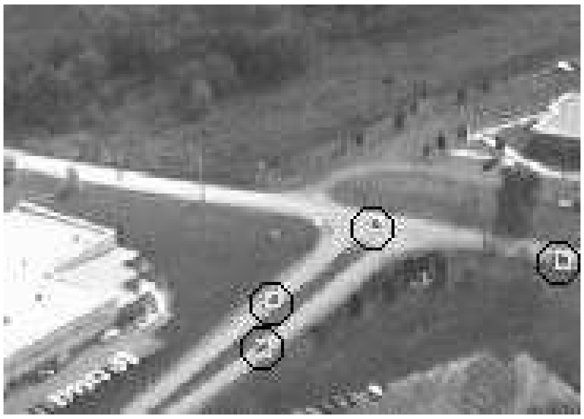


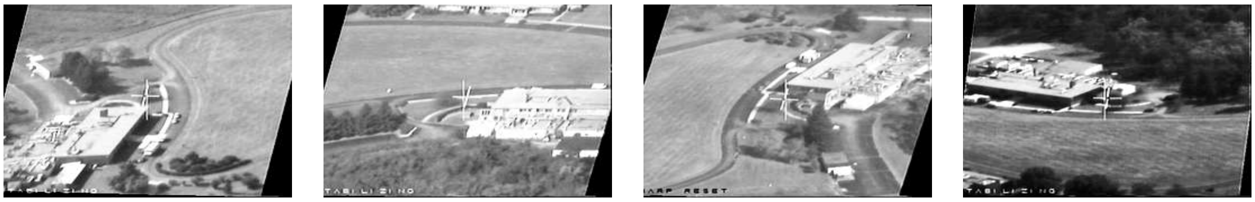
Figure 32: *Detection of small moving objects from a moving airborne sensor.*

### 3.6.2 Camera Fixation and Aiming

It is well known that human operators fatigue rapidly when controlling cameras on moving airborne and ground platforms. This is because they must continually adjust the turret to keep it locked on a stationary or moving object. Additionally, the video is continuously moving, reflecting the self-motion of the camera. The combination of these factors often leads to operator confusion and nausea. Sarnoff has built image alignment techniques [4, 12] to stabilize the view from the camera turret and to automate camera control, thereby significantly reducing the strain on the operator. In particular, real-time image alignment is used to keep the camera locked on a stationary or moving point in the scene, and to aim the camera at a known geodetic coordinate for which reference imagery is available. More details can be found in [27]. Figure 33 shows the performance of the stabilization/fixation algorithm on two ground points as the aircraft traverses an approximate ellipse over them. The field of view in these examples is 3°, and the aircraft took approximately 3 minutes to complete each orbit.

### 3.6.3 Air Sensor Multi-Tasking

Occasionally, a single camera resource must be used to track multiple moving objects, not all of which fit within a single field of view. This problem is particularly relevant for high-altitude air platforms that must have a narrow field of view in order to see ground objects at a reasonable



Fixation on target point A.

Fixation on target point B.

Figure 33: *Fixation on two target points. The images shown are taken 0, 45, 90 and 135 seconds after fixation was started. The large center cross-hairs indicate the center of the stabilized image, i.e. the point of fixation*

resolution. Sensor multi-tasking is employed to switch the field of view periodically between two (or more) target areas that are being monitored. This process is illustrated in Figure 34 and described in detail in [27].

Figure 34: *Footprints of airborne sensor being autonomously multi-tasked between three disparate geodetic scene coordinates.*

# 4 Site Models, Calibration and Geolocation

An automated surveillance system can benefit greatly from the scene-specific knowledge provided by a site model. Some of the many VSAM tasks supported by an accurate 3D site model are:

- computation of object geolocation (Section 4.3);

- visibility analysis (predicting what portions of the scene are visible from which cameras) to allow more effective sensor tasking;
- geometric focus of attention, for example to task a sensor to monitor the door of a building, or specify that vehicles should appear on roads;
- suppression of false alarms in areas of foliage;
- prediction of visual effects like shadows;
- visualization of the scene to enable quick comprehension of geometric relationships between sensors, objects, and scene features;
- simulation for planning best sensor placement and for debugging algorithms; and
- landmark-based camera calibration.

## 4.1 Scene Representations

Figure 35 illustrates the wide variety of scene representations that have been used in the VSAM testbed system over the past three years. Most of the variety is due to work in our first year of effort (1997), where we bootstrapped a representation of the Bushy Run site largely by hand. During the second and third years of the project, performed on the campus of CMU, we used a Compact Terrain Data Base (CTDB) model of campus, which ended up supporting almost all of our algorithmic needs.

**A) USGS orthophoto.** The United States Geological Survey (USGS) produces several digital mapping products that can be used to create an initial site model. These include **1) Digital Orthophoto Quarter Quad (DOQQ)** - a nadir (down-looking) image of the site as it would look under orthographic projection (Figure 35a). The result is an image where scene features appear in their correct horizontal positions. **2) Digital Elevation Model (DEM)** - an image whose pixel values denote scene elevations at the corresponding horizontal positions. Each grid cell of the USGS DEM shown encompasses a 30-meter square area. **3) Digital Topographic Map (DRG)** - a digital version of the popular USGS topo maps. **4) Digital Line Graph (DLG)** - vector representations of public roadways and other cartographic features. Many of these can be ordered directly from the USGS EROS Data Center web site, located at URL `http://edcwww.cr.usgs.gov/`. The ability to use existing mapping products from USGS or National Imagery and Mapping Agency (NIMA) to bootstrap a VSAM site model demonstrates that rapid deployment of VSAM systems to monitor trouble spots around the globe is a feasible goal.

**B) Custom DEM.** The Robotics Institute autonomous helicopter group mounted a high precision laser range finder onto a remote-control Yamaha helicopter to create a high-resolution (half-meter grid spacing) DEM of the Bushy Run site for VSAM DEMO I (Figure 35b). Raw radar returns were collected with respect to known helicopter position and orientation (using on-board altimetry data) to form a cloud of points representing returns from surfaces in the scene. These points were converted into a DEM by projecting into LVCS horizontal-coordinate bins, and computing the mean and standard deviation of height values in each bin.

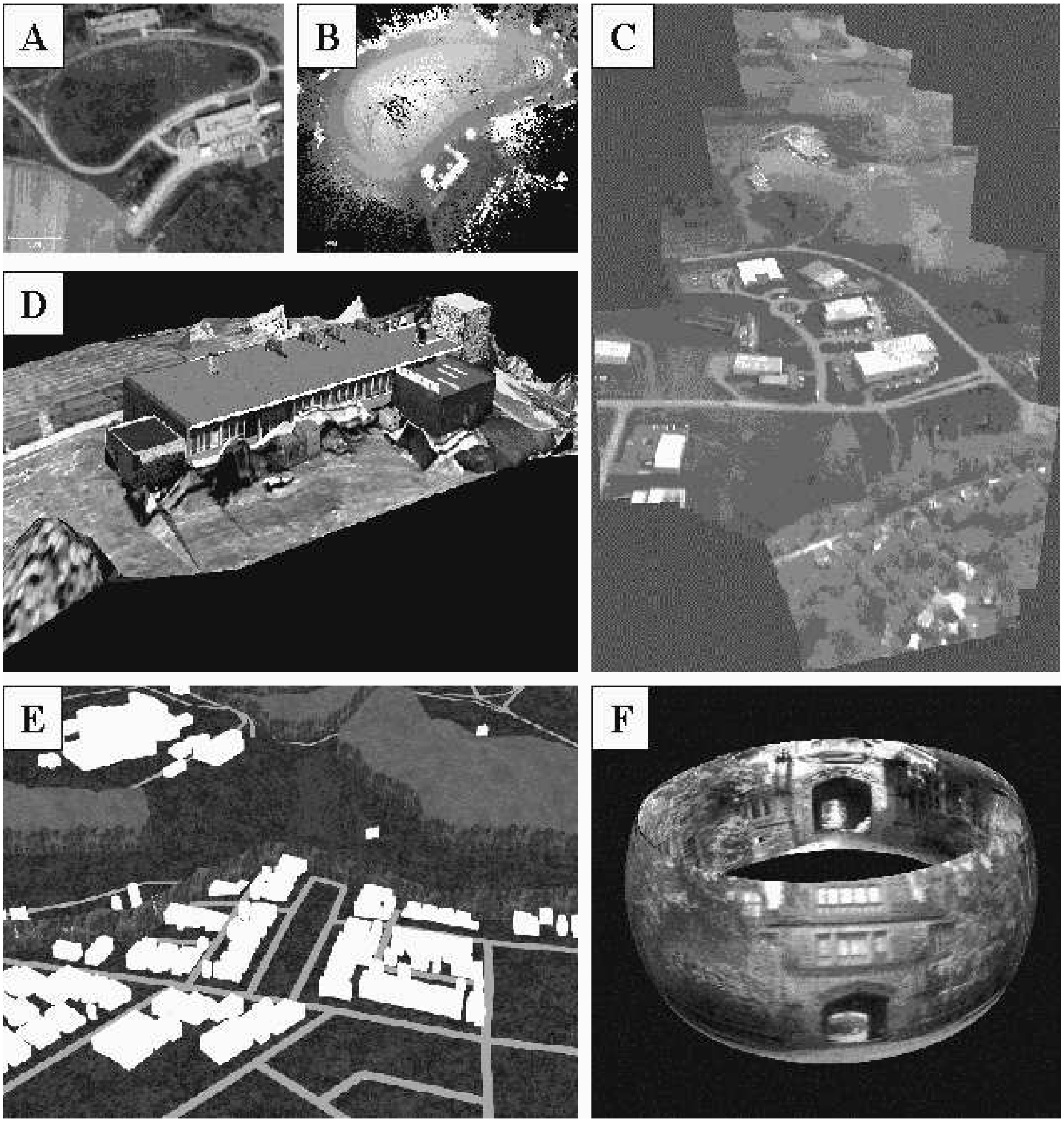



Figure 35: *A variety of site model representations have been used in the VSAM IFD testbed system: A) USGS orthophoto; B) custom DEM; C) aerial mosaic; D) VRML model; E) CTDB site model; and F) spherical representations.*

**C) Mosaics.** A central challenge in surveillance is how to present sensor information to a human operator. The relatively narrow field of view presented by each sensor makes it very difficult for the operator to maintain a sense of context outside the camera's immediate image. Image mosaics from moving cameras overcome this problem by providing extended views of regions swept over by the camera. Figure 35c displays an aerial mosaic of the Demo I Bushy Run site. The video sequence was obtained by flying over the demo site while panning the camera turret back and forth and keeping the camera tilt constant[12, 24, 23]. The VSAM IFD team also demonstrated coarse registration of this mosaic with a USGS orthophoto using a projective warp to determine an approximate mapping from mosaic pixels to geographic coordinates. It is feasible that this technology could lead to automated methods for updating existing orthophoto information using fresh imagery from a recent fly-through. For example, seasonal variations such as fresh snowfall (as in the case of VSAM Demo I) can be integrated into the orthophoto.

**D) VRML models.** Figure 35d shows a VRML model of one of the Bushy Run buildings and its surrounding terrain. This model was created by the $K^2T$ company using the factorization method [25] applied to aerial and ground-based video sequences.

**E) Compact Terrain Data Base (CTDB).** During the last two years, the VSAM testbed system has used a Compact Terrain Data Base (CTDB) model of campus as its primary site model representation. The CTDB was originally designed to represent large expanses of terrain within the context of advanced distributed simulation, and has been optimized to efficiently answer geometric queries such as finding the elevation at a point in real-time. Terrain can be represented as either a grid of elevations, or as a Triangulated Irregular Network (TIN), and hybrid data bases containing both representations are allowed. The CTDB also represents relevant cartographic features on top of the terrain skin, including buildings, roads, bodies of water, and tree canopies. Figure 35e shows a small portion of the Schenley Park / CMU campus CTDB. An important benefit to using CTDB as a site model representation for VSAM processing is that it allows us to easily interface with the synthetic environment simulation and visualization tools provided by ModSAF and ModStealth.

**F) Spherical Representations.** During the second year (1998), VSAM testbed SPU's used the Microsoft Windows NT operating system, which is not supported by CTDB software. For that reason, we explored the use of spherical lookup tables for each fixed-mount SPU. Everything that can be seen from a stationary camera can be represented on the surface of a viewing sphere (Figure 35f). This is true even if the camera is allowed to pan and tilt about the focal point, and to zoom in and out – the image at any given (pan,tilt,zoom) setting is essentially a discrete sample of the bundle of light rays impinging on the camera's focal point. We used this idea to precompile and store a spherical lookup table containing the 3D locations and surface material types of the points of intersection of camera viewing rays with the CTDB site model. During the third year, we changed from Windows to the Linux operating system, a variant of Unix, and could then use CTDB directly on each SPU. This made the spherical lookup tables obsolete.

Three geospatial site coordinate systems are used interchangeably within the VSAM testbed. The WGS84 geodetic coordinate system provides a reference frame that is standard, unambiguous and global (in the true sense of the word). Unfortunately, even simple computations such as the distance between two points become complicated as a function of latitude, longitude and elevation. For this reason, site-specific Cartesian coordinate systems are typically established to handle the bulk of the geometric model computations that must be performed. We have used a Local Vertical Coordinate System (LVCS) [2] with its origin at the base of the PRB operator control center for representing camera positions and for providing an operator display map coordinate system. The CTDB model of campus is based on Universal Transverse Mercator (UTM) coordinates, which provide an alternative Cartesian coordinate system, and which are related to the LVCS by a rotation and translation. Conversion between geodetic, LVCS, and UTM coordinates is straightforward, so that each can be used interchangeably in the system.

## 4.2 Camera Calibration

For a VSAM system to make full use of a geometric site model requires calibrating the cameras with respect to the model. We have developed a set of calibration procedures specifically designed for *in-situ* (meaning "in place") camera calibration. We believe that all cameras should be calibrated in an environment that resembles their actual operating conditions. This philosophy is particularly relevant for outdoor camera systems. Cameras get jostled during transport and installation, and changes in temperature and humidity can affect a camera's intrinsic parameters. Furthermore, it is impossible to recreate the full range of zoom and focus settings that are useful to an outdoor camera system within the confines of an indoor lab.

Some amount of on-site calibration is always necessary, if only for determining the extrinsic parameters (location and orientation) of the camera placement. Unfortunately, outdoors is not an ideal environment for careful camera calibration. It can be cold, rainy, or otherwise unpleasant. Simple calibration methods are needed that can be performed with minimal human intervention.

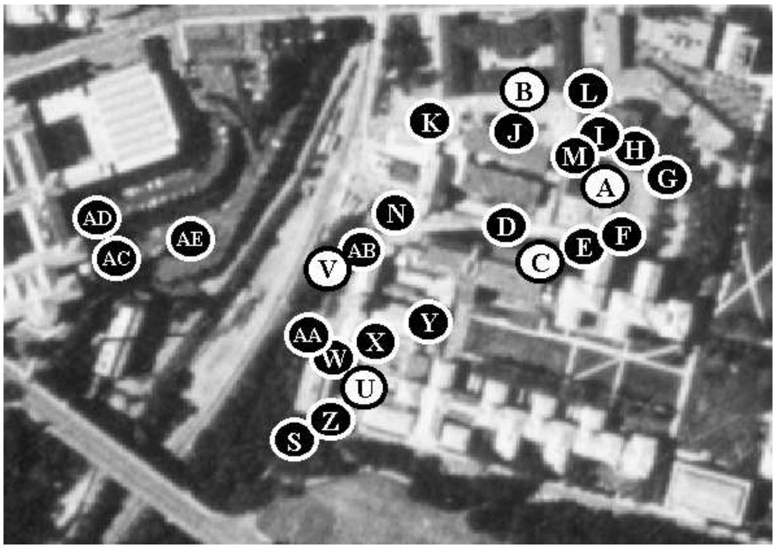
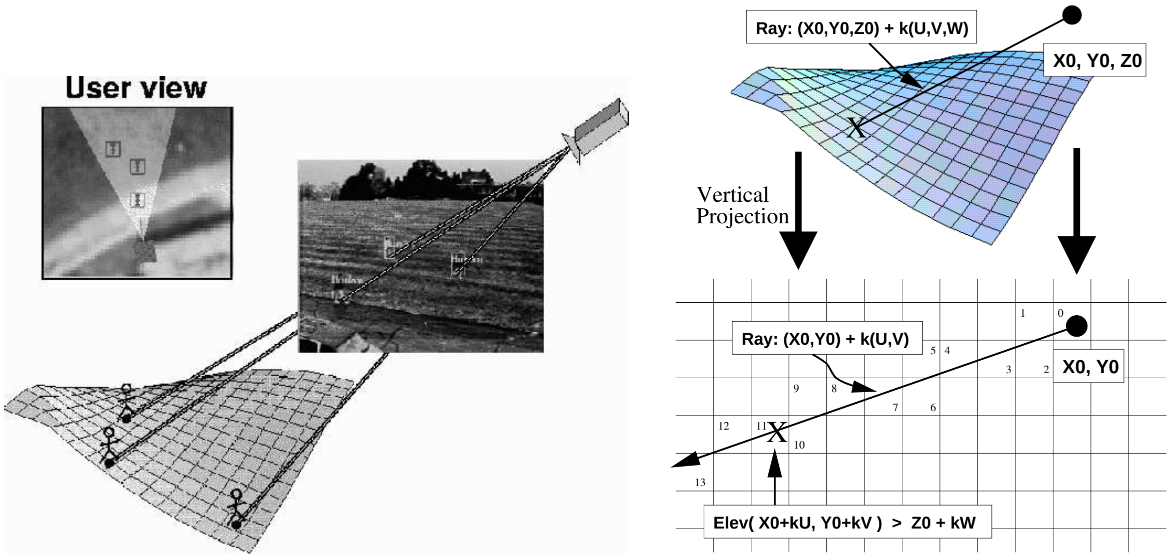



Figure 36: *GPS landmarks measurements for extrinsic camera calibration on the CMU campus.*

We have developed methods for fitting a projection model consisting of intrinsic (lens) and extrinsic (pose) parameters of a camera with active pan, tilt and zoom control. Intrinsic parameters are calibrated by fitting parametric models to the optic flow induced by rotating and zooming the camera. These calibration procedures are fully automatic and do not require precise knowledge of 3D scene structure. Extrinsic parameters are calculated by sighting a sparse set of measured landmarks in the scene (see Figure 36). Actively rotating the camera to measure landmarks over a virtual hemispherical field of view leads to a well-conditioned exterior orientation estimation problem. Details of the calibration procedures are presented in [7].

## 4.3 Model-based Geolocation

The video understanding techniques described in Section 3 operate primarily in image space. A large leap in terms of descriptive power can be made by transforming image blobs and measurements into 3D scene-based objects and descriptors. In particular, determination of object location in the scene allows us to infer the proper spatial relationships between sets of objects, and between objects and scene features such as roads and buildings. Furthermore, we believe that computation of 3D spatial geolocation is the key to coherently integrating a large number of object hypotheses from multiple, widely-spaced sensors.




(a) (b)

Figure 37: *(a) Estimating object geolocations by intersecting backprojected viewing rays with a terrain model. (b) A Bresenham-like traversal algorithm determines which DEM cell contains the first intersection of a viewing ray and the terrain.*

In regions where multiple sensor viewpoints overlap, object locations can be determined very accurately by wide-baseline stereo triangulation. However, regions of the scene that can be simultaneously viewed by multiple sensors are likely to be a small percentage of the total area of regard in real outdoor surveillance applications, where it is desirable to maximize coverage of a large area using finite sensor resources. Determining object locations from a single sensor requires domain constraints, in this case the assumption that the object is in contact with the terrain. This contact location is estimated by passing a viewing ray through the bottom of the object in the image and intersecting it with a model representing the terrain (see Figure 37a). Sequences of location estimates over time are then assembled into consistent object trajectories.

Previous uses of the ray intersection technique for object localization in surveillance research have been restricted to small areas of planar terrain, where the relation between image pixels and terrain locations is a simple 2D homography [5, 9, 19]. This has the benefit that no camera calibration is required to determine the back-projection of an image point onto the scene plane, provided the mappings of at least four coplanar scene points are known beforehand. However, large outdoor scene areas may contain significantly varied terrain. To handle this situation, we perform geolocation using ray intersection with a full terrain model provided, for example, by a digital elevation map (DEM).

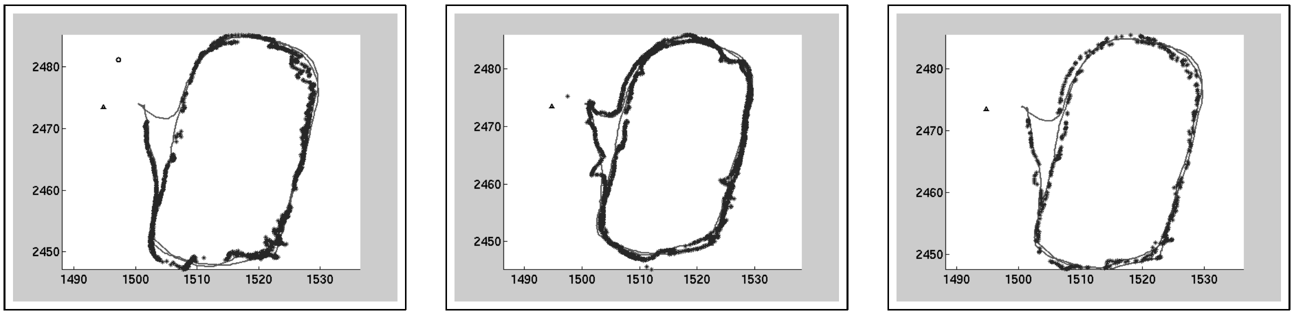
Given a calibrated sensor, and an image pixel corresponding to the assumed contact point between an object and the terrain, a viewing ray $(x_0 + ku, y_0 + kv, z_0 + kw)$ is constructed, where $(x_0, y_0, z_0)$ is the 3D sensor location, $(u, v, w)$ is a unit vector designating the direction of the viewing ray emanating from the sensor, and $k \geq 0$ is an arbitrary distance. General methods for determining where a viewing ray first intersects a 3D scene (for example, ray tracing) can be quite involved. However, when scene structure is stored as a DEM, a simple geometric traversal algorithm suggests itself, based on the well-known Bresenham algorithm for drawing digital line segments. Consider the vertical projection of the viewing ray onto the DEM grid (see Figure 37b). Starting at the grid cell $(x_0, y_0)$ containing the sensor, each cell $(x, y)$ that the ray passes through is examined in turn, progressing outward, until the elevation stored in that DEM cell exceeds the $z$-component of the 3D viewing ray at that location. The $z$-component of the view ray at location $(x, y)$ is computed as either

$$z_0 + \frac{(x - x_0)}{u} w \quad \text{or} \quad z_0 + \frac{(y - y_0)}{v} w \tag{9}$$

depending on which direction cosine, $u$ or $v$, is larger. This approach to viewing ray intersection localizes objects to lie within the boundaries of a single DEM grid cell. A more precise sub-cell location estimate can then be obtained by interpolation. If multiple intersections with the terrain beyond the first are required, this algorithm can be used to generate them in order of increasing distance from the sensor, out to some cut-off distance. See [6] for more details.

**Geolocation Evaluation**

We have evaluated geolocation accuracy for two cameras (PRB and Wean) on the CMU campus using a Leica laser-tracking theodolite to generate ground truth. The experiment was run by having a person carry the theodolite prism for two loops around the PRB parking lot, while the system logged time-stamped horizontal (X,Y) positions estimated by the Leica theodolite. The system also simultaneously tracked the person using the PRB and Wean cameras, while logging time-stamped geolocation estimates from each camera.



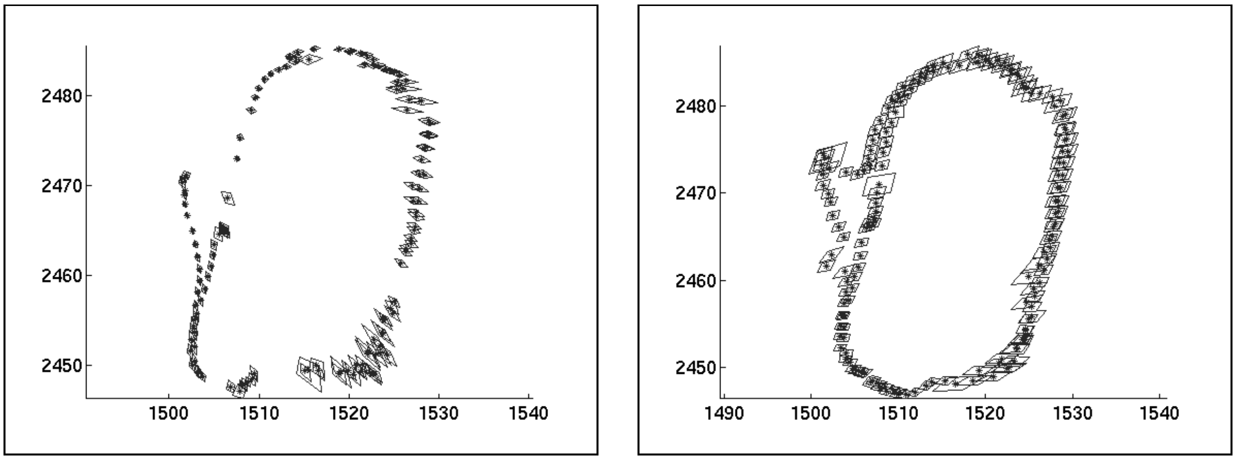

(a) (b) (c)

Figure 38: *Ground truth trajectory overlaid with geolocation estimates from a) PRB camera, b) Wean camera, and c) an average of PRB and Wean estimates. Scales are in meters.*

Figure 38 shows the ground truth trajectory curve, overlaid with geolocation estimates from (a) the PRB camera, (b) the Wean camera, and (c) an average of the PRB and Wean camera estimates for corresponding time stamps. Both cameras track the overall trajectory fairly well. The PRB camera geolocation estimates have large errors at the lower portions and upper right arc of the loop, because the person's feet were occluded by parked vehicles when walking through those areas. The higher elevation and direction of view of Wean camera allowed it to see the person's feet at the lower portion of the loop, so the trajectory is correctly followed there. An error still occurs at the top right of the loop, as the person comes close to two vehicles and is reflected from their shiny surfaces. This pulls the bounding box off the person's feet, and causes an underestimation of their position. Geolocation estimates were only averaged for points with time stamps agreeing to within a small threshold, so there are far fewer points shown in Figure 38c. The effect of averaging is to smooth out many of the high variance portions of both curves, although the overall distance accuracy does not noticeably improve.

Geolocation estimates are computed by backprojecting a point located at the center of the lowest side of the bounding box enclosing a moving blob. The system maintains a running estimate of the variance of this point – the variance is high when the position or shape of the bounding box changes greatly during tracking. The system computes an internal estimate of horizontal geolocation error by projecting an error box of one standard deviation around the image point used for estimation, until it intersects the terrain, thus providing a bound on the error of the geolocation estimate. A subsampling of this set of error boxes is shown in Figure 39, for both cameras. It is interesting to note that during portions of the trajectory where errors are large due to occlusions or reflections, the system is aware that the variance of the geolocation estimate is high.

To determine actual geolocation accuracy, the time stamp of each geolocation estimate was compared to the list of ground truth time stamps to find a suitably close correspondence. For cases where a corresponding ground truth point was found, the horizontal displacement error is plotted in Figure 40, for a) PRB camera, b) Wean camera, and c) the average geolocation computed from PRB and Wean. The mean and covariance of each point cloud were estimated, and the major and




(a) (b)

Figure 39: *Geolocation error boxes computed by the system for trajectory estimates from a) PRB camera and b) Wean camera. Scales are in meters. Compare with Figures 38a and b.*

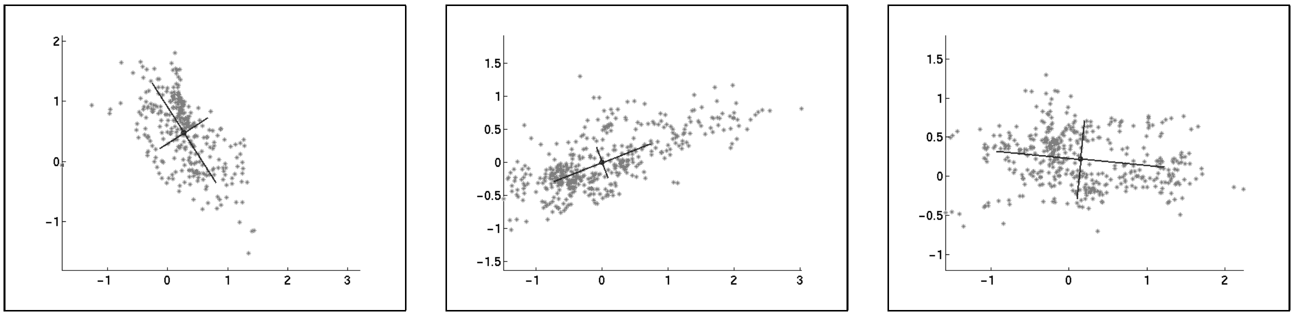
minor axes of the covariance ellipse is overlaid on each plot, with the length of the axes scaled to represent 1.5 times the standard deviation of the point spread along that direction. Numeric standard deviations along each axis are displayed in the following table.

| Geolocation Estimates | max std (meters) | min std (meters) |
|---|---|---|
| PRB | 0.6520 | 0.3139 |
| Wean | 0.5232 | 0.1628 |
| Avg of PRB and Wean | 0.7184 | 0.3337 |

These numbers confirm the observation that averaging geolocation estimates from both cameras is not improving accuracy. It is actually getting slightly worse. Referring again to Figure 40, we see that the center of each error point spread is not at (0,0). We are therefore averaging biased geolocation estimates from each camera, and the noise in each estimate is therefore not cancelling out properly, but rather intensifying. Removing the geolocation bias from each sensor will be necessary to achieve more accurate results from averaging. Possible sources of error are the camera calibration parameters, the terrain model, and small biases in the time stamps produced by each SPU. Nonetheless, standard deviation of geolocation estimates from each camera are roughly on the order of .6 meters along the axis of maximum spread, and roughly .25 meters at minimum. We have confirmed that the axis of maximum error for each camera is oriented along the direction vector from the camera to the object being observed.

## 4.4 Model-based Human-Computer Interface

Keeping track of people, vehicles, and their interactions, over a chaotic area such as the battlefield, is a difficult task. The commander obviously shouldn't be looking at two dozen screens showing



(a) (b) (c)

Figure 40: *Plotted covariances of the horizontal displacement errors between estimate geolocations and ground truth locations for corresponding time stamps. a) PRB camera, b) Wean camera, and c) average of PRB and Wean estimates with corresponding time stamps. Scales are in meters.*

raw video output – that amount of sensory overload virtually guarantees that information will be ignored, and requires a prohibitive amount of transmission bandwidth. Our suggested approach is to provide an interactive, graphical visualization of the battlefield by using VSAM technology to automatically place dynamic agents representing people and vehicles into a synthetic view of the environment.

This approach has the benefit that visualization of the object is no longer tied to the original resolution and viewpoint of the video sensor, since a synthetic replay of the dynamic events can be constructed using high-resolution, texture-mapped graphics, from any perspective. Particularly striking is the amount of data compression that can be achieved by transmitting only symbolic georegistered object information back to the operator control unit instead of raw video data. Currently, we can process NTSC color imagery with a frame size of 320x240 pixels at 10 frames per second on a Pentium II computer, so that data is streaming into the system through each sensor at a rate of roughly 2.3Mb per second per sensor. After VSAM processing, detected object hypotheses contain information about object type, location and velocity, as well as measurement statistics such as a time stamp and a description of the sensor (current pan, tilt, and zoom for example). Each object data packet takes up roughly 50 bytes. If a sensor tracks 3 objects for one second at 10 frames per second, it ends up transmitting 1500 bytes back to the OCU, well over a thousandfold reduction in data bandwidth.

Ultimately, the key to comprehending large-scale, multi-agent events is a full, 3D immersive visualization that allows the human operator to fly at will through the environment to view dynamic events unfolding in real-time from any viewpoint. We envision a graphical user interface based on cartographic modeling and visualization tools developed within the Synthetic Environments (SE) community. The site model used for model-based VSAM processing and visualization is represented using the Compact Terrain Database (CTDB). Objects are inserted as dynamic agents within the site model and viewed by Distributed Interactive Simulation clients such as the Modular Semi-Automated Forces (ModSAF) program and the associated 3D immersive ModStealth viewer.

We first demonstrated proof-of-concept of this idea at the Dismounted Battle Space Battle

Lab (DBBL) Simulation Center at Fort Benning Georgia as part of the April 1998 VSAM workshop. On April 13, researchers from CMU set up a portable VSAM system at the Benning Mobile Operations in Urban Terrain (MOUT) training site. The camera was set up at the corner of a building roof whose geodetic coordinates had been measured by a previous survey [11], and the height of the camera above that known location was measured. The camera was mounted on a pan-tilt head, which in turn was mounted on a leveled tripod, thus fixing the roll and tilt angles of the pan-tilt-sensor assembly to be zero. The yaw angle (horizontal orientation) of the sensor assembly was measured by sighting through a digital compass. After processing several troop exercises, log files containing camera calibration information and object hypothesis data packets were sent by FTP back to CMU and processed using the CTDB to determine a time-stamped list of moving objects and their geolocations. Later in the week, this information was brought back to the DBBL Simulation Center at Benning where, with the assistance of colleagues from BDM, it was played back for VSAM workshop attendees using custom software that broadcast time-sequenced simulated entity packets to the network for display by both ModSAF and ModStealth. Some processed VSAM video data and screen dumps of the resulting synthetic environment playbacks are shown in Figure 41.

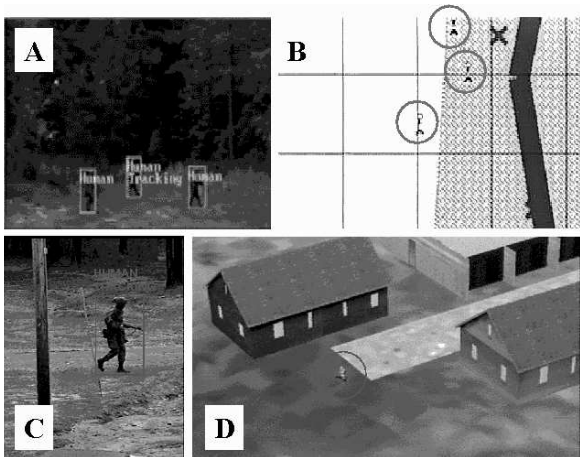



Figure 41: *Sample synthetic environment visualizations of data collected at the Benning MOUT site. A) Automated tracking of three people. B) ModSAF 2D orthographic map display of estimated geolocations. C) Tracking of a soldier walking out of town. D) Immersive, texture-mapped 3D visualization of the same event, seen from a user-specified viewpoint.*

We have also demonstrated that this visualization process can form the basis for a real-time immersive visualization tool. First, we ported object geolocation computation using the CTDB onto the VSAM SPU platforms. This allowed estimates of object geolocation to be computed within the frame-to-frame tracking process, and to be transmitted in data packets back to the OCU. At the OCU, incoming object identity and geolocation data is repackaged into Distributed Interactive Simulation (DIS) packets understood by ModSAF and ModStealth clients, and re-broadcast (multicast) on the network. At that point, objects detected by the SPUs are viewable, after a short lag, within the context of the full 3D site model using the ModStealth viewer (Figure 42).

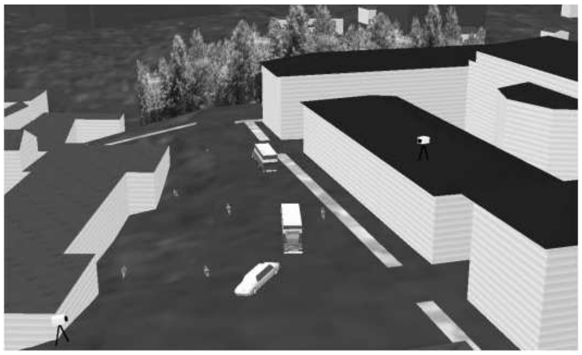


Figure 42: *Real-time, 3D ModStealth visualization of objects detected and classified by the VSAM testbed system and transmitted via DIS packets on the network.*

# 5 Sensor Coordination

In most complex outdoor scenes, it is impossible for a single sensor to maintain its view of an object for long periods of time. Objects become occluded by environmental features such as trees and buildings, and sensors have limited effective fields of regard. A promising solution to this problem is to use a network of video sensors to cooperatively track objects through the scene. We have developed and demonstrated two methods of sensor coordination in the VSAM testbed. First, objects are tracked long distances through occlusion by *handing-off* between cameras situated along the object's trajectory. Second, wide-angle sensors keeping track of all objects in a large area are used to task active pan, tilt and zoom sensors to get a better view of selected objects, using a process known as *sensor slaving*.

## 5.1 Multi-Sensor Handoff

There has been little work done on autonomously coordinating multiple active video sensors to cooperatively track a moving object. One approach is presented by Matsuyama for a controlled indoor environment where four cameras lock onto onto a particular object moving across the floor [22]. We approach the problem more generally by using the object's 3D geolocation (as computed in the last section) to determine where each sensor should look. The pan, tilt and zoom of the closest sensors are then controlled to bring the object within their fields of view, while a viewpoint independent cost function is used to determine which of the moving objects they find are the specific object of interest. These steps are described below.

Assume that at time $t_0$ a sensor with pan, tilt value $(\theta_0, \phi_0)$ has been tasked to track a particular object with 3D ground location $X_0$ and velocity $\dot{X}$. Given a function $G(X)$ that converts a ground coordinate to a pan, tilt point (determined by camera calibration), the object's location $X_0$ is converted to a desired sensor pan, tilt value $(\theta_d, \phi_d) = G(X_0)$. The behavior of the pan, tilt unit is approximated by a linear system with infinite acceleration and maximum velocity $(\pm\dot{\theta}, \pm\dot{\phi})$ as

$$\begin{aligned} \theta(t) &= \theta_0 \pm \dot{\theta}(t - t_0) \\ \phi(t) &= \phi_0 \pm \dot{\phi}(t - t_0) \end{aligned} \tag{10}$$

Substituting the desired sensor pan, tilt $(\theta_d, \phi_d)$ into the left hand side of this equation and solving for $(t - t_0)$ yields a prediction of the acquisition time, that is, how long it would take for the pan, tilt device to point at the object's current location. However, the object will have moved further along its trajectory by that time. This new object position is estimated as

$$X(t) = X_0 + \dot{X}(t - t_0) \tag{11}$$

This predicted object position is then converted into a new desired sensor pan, tilt, and the whole procedure iterates until the time increments $(t - t_0)$ become small (convergence) or start to increase (divergence). This algorithm guarantees that if it converges, the sensor will be able to reacquire the object.

An appropriate camera zoom setting can be determined directly given a desired size of the object's projection in the image. Knowing the classification of the object $C$ (as determined from Section 3.3), we employ the heuristic that humans are approximately 6 feet (2m) tall and vehicles are approximately 15 feet (5m) long. Given the position of the object and the sensor, and therefore the range $r$ to the object, the angle $\rho$ subtended by the image of the object is approximately

$$\rho = \begin{cases} \tan^{-1}\frac{2}{r}, & \text{human} \\ \tan^{-1}\frac{5}{r}, & \text{vehicle} \end{cases}$$

Knowing the focal length of the sensor as a function of zoom, as determined from camera calibration, the appropriate zoom setting is easily chosen.

Once the sensor is pointing in the right direction at the right zoom factor, all moving objects extracted are compared to the specific object of interest to see if they match. This need to re-acquire a specific object is a key feature necessary for multi-camera cooperative surveillance. Obviously viewpoint-specific appearance criteria are not useful, since the new view of the object may be significantly different from the previous view. Therefore, recognition features are needed that are independent of viewpoint. In our work we use two such criteria: the object's 3D scene trajectory as determined from geolocation, and a normalized color histogram of the object's image region. Candidate motion regions are tested by applying a matching cost function in a manner similar to that described in Section 3.2.

An example of using multi-sensor hand-off to track a vehicle as it travels through campus is shown in Figure 43. This diagram shows the continuous, autonomous tracking of a single object for a distance of approximately 400m and a time of approximately 3 minutes. In Figure 43(a) two sensors cooperatively track the object. At the time shown in Figure 43(b) the object is occluded from sensor 2, but is still visible from sensor 1, which continues to track it. When the object moves out of the occlusion area, sensor 2 is automatically retasked to track it, as shown in Figure 43(c).

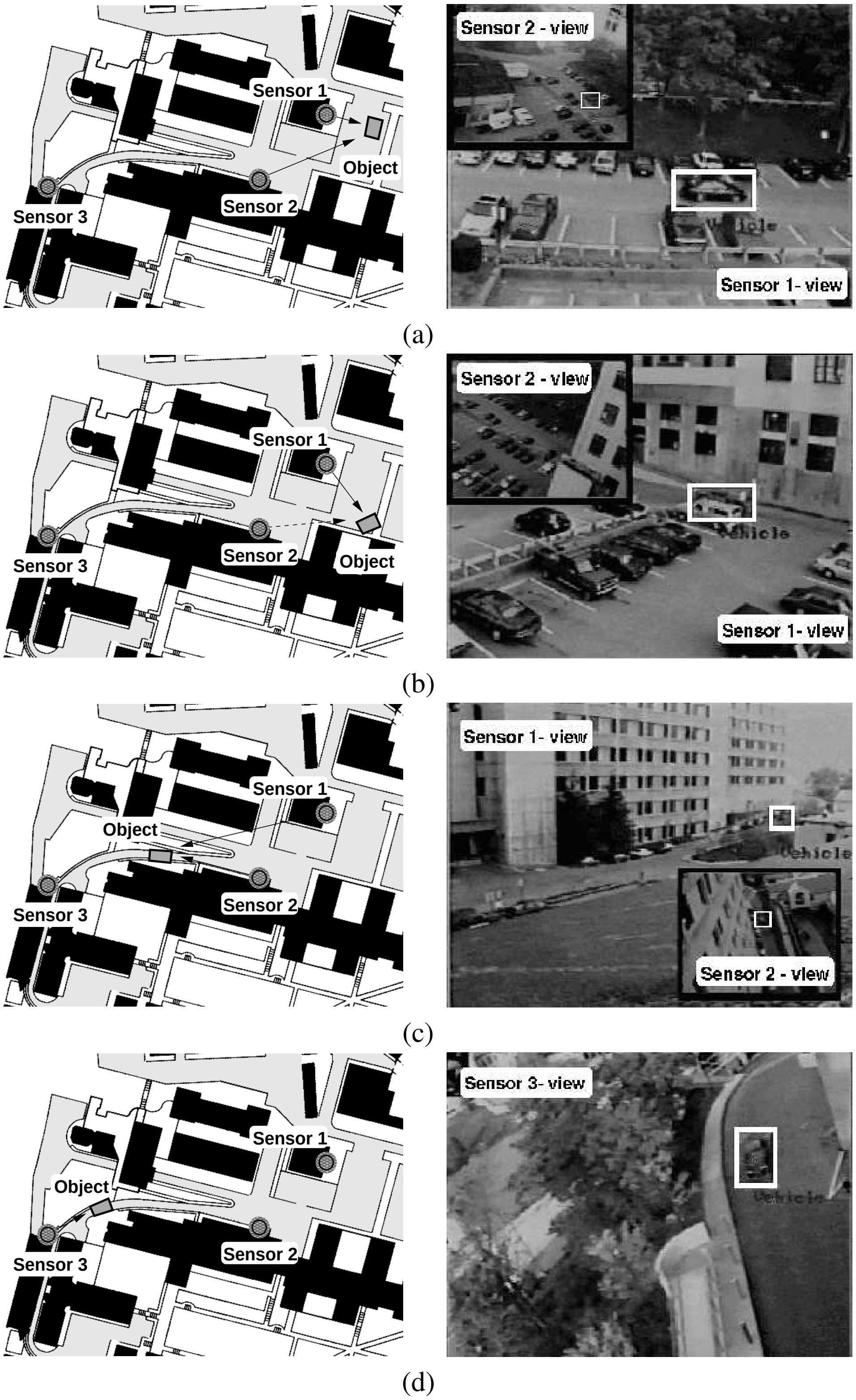



Figure 43: *Cooperative, multi-sensor tracking (see text for description).*

Finally, when the object moves out of the field of regard of both sensors, a third sensor is automatically tasked to continue surveillance, as shown in Figure 43(d). By automatically managing multiple, redundant camera resources, the vehicle is continuously tracked through a complex urban environment.

## 5.2 Sensor Slaving

A second form of sensor cooperation is sensor slaving. We use the term *sensor slaving* to denote using one wide field of view camera to control a second, active camera to zoom in and actively follow the subject to generate a better view. The motivation is to keep track of all objects in the scene while simultaneously gathering high-resolution views of selected objects. A camera slaving system has at least one master camera and one slave camera. The master camera is set to have a global view of the scene so that it can track objects over extended areas using simple tracking methods such as adaptive background subtraction. The object trajectory generated by the master camera is relayed to the slave camera in real time. The slave camera, which is highly zoomed in, can then follow the trajectory to generate close-up imagery of the object.

Slaving is a relatively simple exercise if both cameras are calibrated with respect to a local 3D terrain model. We have shown in Section 4.3 that a person's 3D trajectory can be determined to reasonable accuracy (roughly 1 meter of error for a person 50 meters away) by intersecting backprojected viewing rays with the terrain. After estimating the 3D location of a person from the first camera's viewpoint, it is an easy matter to transform the location into a pan-tilt command to control the second camera. Figure 44 shows an example of camera slaving. A person has been detected automatically in the wide-angle view shown in the left image, and a second camera has been tasked to move slightly ahead of the person's estimated 3D trajectory, as shown in the right image.

For cameras located far apart geographically, it is obvious that we need to have very good camera calibration, and an accurate 3D site model. We have also developed a sensor slaving method that works for closely located cameras. This method requires only image-based computations (no geolocation computation or extrinsic camera calibration). Furthermore, intrinsic parameters are needed only by the slave camera, which has to determine the pan/tilt angles needed to point towards each pixel in the image. The basic idea is to form a mosaic by warping the master camera view into the pixel coordinate system of the slave camera view (Figure 45). Image trajectories of objects detected in the master view can then be transformed into trajectories overlaid on the slave camera view. The slave camera can then compute the pan-tilt angles necessary to keep the object within its zoomed field of view.

Figure 44: *Example of camera slaving. Left: wide-angle view in which a person is detected. Right:a better view from a second camera, which has been tasked to intercept the person's estimated 3D path.*

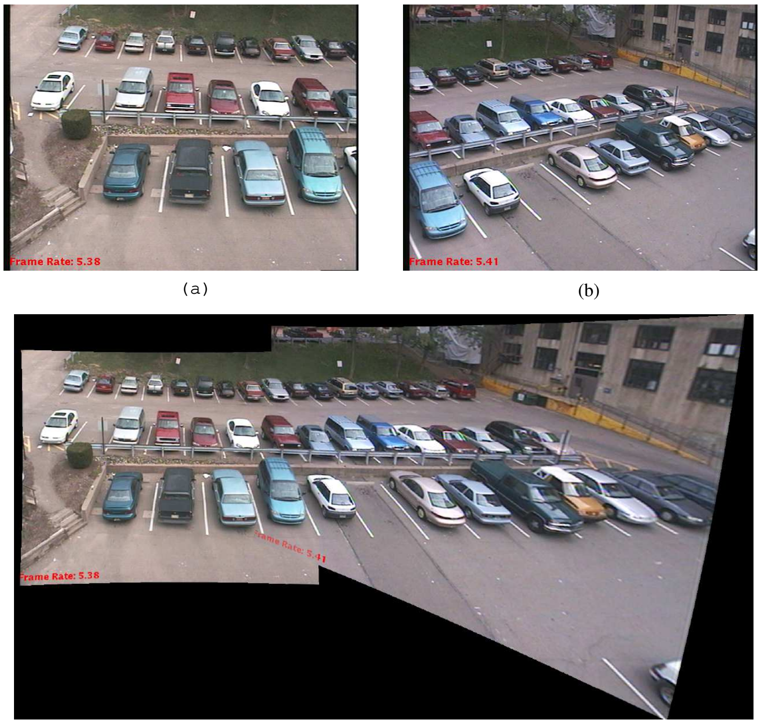



Figure 45: *(a) and (b) are images taken from a slave camera and master camera, respectively. (c) shows the master camera view warped into the pixel coordinate system of the slave camera view, to form an image mosaic. Image pixels are averaged directly in overlapping region.*

# 6 Three Years of VSAM Milestones

The current VSAM IFD testbed system and suite of video understanding technologies are the end result of a three-year, evolutionary process. Impetus for this evolution was provided by a series of yearly demonstrations. The following tables provide a succinct synopsis of the progress made during the last three years in the areas of video understanding technology, VSAM testbed architecture, sensor control algorithms, and degree of user interaction.

Although the program is over now, the VSAM IFD testbed continues to provide a valuable resource for the development and testing of new video understanding capabilities. Future work will be directed towards achieving the following goals:

- better understanding of human motion, including segmentation and tracking of articulated body parts;
- improved data logging and retrieval mechanisms to support 24/7 system operations;
- bootstrapping functional site models through passive observation of scene activities;
- better detection and classification of multi-agent events and activities;
- better camera control to enable smooth object tracking at high zoom; and
- acquisition and selection of "best views" with the eventual goal of recognizing individuals in the scene.

Table 3: Progression of Video Understanding Technology

| Video Understanding | 1997 Demo Results | 1998 Demo Results | 1999 Demo Results |
|---|---|---|---|
| Ground-based moving target detection (MTD) and tracking | Multiple target detection, single target step and stare tracking, temporal change, adaptive template matching | Multi-target MTD and trajectory analysis, motion salience via temporal consistency, adaptive background subtraction | Layered and adaptive background subtraction for robust detection, MTD while panning, tilting and zooming using optic flow and image registration, target tracking by multi-hypothesis Kalman filter |
| Airborne MTD and tracking | Stabilization / temporal change using correlation | Real-time camera pointing based on motion plus appearance, drift free fixation | (N/A) |
| Ground-based target geolocation | Ray intersection with DEM | Ray intersection with SEEDS model | Geolocation uncertainty estimation by Kalman filtering, domain knowledge |
| Airborne target geolocation | Video to reference image registration | Fine aiming using video to reference image registration in real-time | (N/A) |
| Target recognition | Temporal salience (predicted trajectory) | Spatio-temporal salience, color histogram, classification | Target patterns and/or spatio-temporal signature |
| Target classification technique | Aspect ratio | Dispersedness, motion-based skeletonization, neural network, spatio-temporal salience | Patterns inside image chips, spurious motion rejection, model-based recognition, Linear Discriminant Analysis |
| Target classification categories | Human, vehicle | Human, human group, vehicle | Human, human group, sedan, van, truck, mule, FedEx van, UPS van, police car |
| Target classification accuracy (percentage correctly identified) | 87% Vehicle, 83% Human (small sample) | 85% (large sample) | $>$ 90% (large sample) |
| Activity monitoring | Any motion | Individual target behaviors | Multiple target behaviors: parking lot monitoring, getting in/out of cars, entering buildings |
| Ground truth verification | None | Off-line | On-line (one target) |
| Geolocation accuracy | 5 meters | 2 meters | $<$ 1 meter |
| Camera calibration | Tens of pixels | Fives of pixels | Ones of pixels |
| Domain knowledge | Elevation map and hand-drawn road network | SEEDS model used to generate ray occlusion tables off-line | Parking area, road network, occlusion boundaries |

Table 4: Progression of VSAM Architecture Goals

| VSAM Architecture | 1997 Demo Results | 1998 Demo Results | 1999 Demo Results |
|---|---|---|---|
| Number of SPUs | 3 | 8 | 12 |
| Types of Sensors | Standard video camera with fixed focal length | Standard video camera with zoom, omnicamera | Static color and B/W cameras, color video cameras with pan, tilt and zoom, omnicamera, thermal |
| Types of SPU and VSAM nodes | Slow relocatable, airborne | Fast relocatable, fixed-mount, airborne, visualization clients | Super-SPU handling multiple cameras, web-based VIS-node |
| System coverage | Rural, 0.1 $km^2$ area ground-based, 3 $km^2$ airborne coverage | University campus, 0.3 $km^2$ area ground-based, airborne coverage over 9 $km^2$ urban area | Dense coverage of university campus, 0.3$km^2$ ground-based area of interest |
| Communcation architecture | Dedicated OCU/SPU | Variable-packet protocol | |

Table 5: Progression of VSAM Sensor Control

| Sensor Control | 1997 Demo Results | 1998 Demo Results | 1999 Demo Results |
|---|---|---|---|
| Ground sensor aiming (hand-off and multitasking) | Predetermined handoff regions | 3D coordinates and signatures, epipolar constraints, occlusion and footprint databases | Camera-to-camera handoff, wide-angle slaving |
| Air sensor aiming | Video to reference image registration for landmark points | Video to reference image registration for landmark points | (N/A) |
| Ground / Air interaction | Human-directed to predetermined locations | OCU-directed to target geolocation | (N/A) |
| SPU behavior | Single supervised task (track target) with primitive unsupervised behavior (look for target) | Single-task supervision (track activity) with unsupervised behavior (loiter detection) | Multi-task supervision for activity monitoring and complex unsupervised behavior (parking lot monitoring) |

Table 6: Progression of User Interaction

| User Interaction | 1997 Demo Results | 1998 Demo Results | 1999 Demo Results |
|---|---|---|---|
| Site model | USGS orthophoto and DEM, LIDAR, real-time mosaics | Compact Terrain DataBase (CTDB), spherical mosaics, aerial mosaic | Improved CTDB model |
| Site model function | Visualization and geolocation | **Off-line:** Demo scenario planning, after-action review, algorithm evaluation, ground-truth verification. **On-line:** Relocatable sensor planning and geolocation, occlusion analysis, target geolocation | **Off-line:** Sensor placement and planning, virtual SPU for scenario perturbation analysis. **On-line:** Ground-truth verification, dynamic visualization and system tasking |
| System tasking by user (user interface) | 2D point-and-click camera control for sensor-based tasking | 2D point-and-click camera control for region and target-based tasking. | Tracked-object specification, 3D interactive activity and event-based tasking |
| Visualization | Overlay current target and sensor positions on orthophoto, live video feeds | Target and sensor information shown on GUI display, computer switchable live video feeds, ModStealth | ModStealth visualization of sensor network, video archiving and replaying of significant events |
| WebVSAM | None | Java-based visualization nodes | Indexed web access to activity report, live internet access to VSAM network via Web visualization nodes |

## Acknowledgments

The authors would like to thank the U.S. Army Night Vision and Electronic Sensors Directorate Lab team at Davison Airfield, Ft. Belvoir, Virginia for their help with the airborne operations. We would also like to thank Chris Kearns and Andrew Fowles for their assistance at the Fort Benning MOUT site, and Steve Haes and Joe Findley at BDM/TEC for their help with the the CTDB site model and distributed simulation visualization software.

# References

[1] C. Anderson, Peter Burt, and G. van der Wal. Change detection and tracking using pyramid transformation techniques. In *Proceedings of SPIE - Intelligent Robots and Computer Vision*, volume 579, pages 72–78, 1985.

[2] American Society of Photogrammetry ASP. *Manual of Photogrammetry*. Fourth Edition, American Society of Photogrammetry, Falls Church, 1980.

[3] J. Barron, D. Fleet, and S. Beauchemin. Performance of optical flow techniques. *International Journal of Computer Vision*, 12(1):42–77, 1994.

[4] J. Bergen, P. Anandan, K. Hanna, and R. Hingorani. Hierarchical model-based motion estimation. In *Proceedings of the European Conference on Computer Vision*, 1992.

[5] K. Bradshaw, I. Reid, and D. Murray. The active recovery of 3d motion trajectories and their use in prediction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(3):219–234, March 1997.

[6] R. Collins, Y. Tsin, J.R. Miller, and A. Lipton. Using a DEM to determine geospatial object trajectories. In *Proceedings of the 1998 DARPA Image Understanding Workshop*, pages 115–122, November 1998.

[7] R.T. Collins and Y. Tsin. Calibration of an outdoor active camera system. In *Proceedings of the 1999 Conference on Computer Vision and Pattern Recognition*, pages 528–534. IEEE Computer Society, June 1999.

[8] F. Dellaert and R.T. Collins. Fast image-based tracking by selective pixel integration. In *ICCV99 Workshop on Frame-Rate Applications*, September 1999.

[9] B. Flinchbaugh and T. Bannon. Autonomous scene monitoring system. In *Proc. 10th Annual Joint Government-Industry Security Technology Symposium*. American Defense Preparedness Association, June 1994.

[10] H. Fujiyoshi and A. Lipton. Real-time human motion analysis by image skeletonization. In *Proceedings of the 1998 Workshop on Applications of Computer Vision*, 1998.

[11] Geometric Geodesy Branch GGB. *Geodetic Survey.* Publication SMWD3-96-022, Phase II, Interim Terrain Data, Fort Benning, Georgia, May 1996.

[12] M. Hansen, P. Anandan, K. Dana, G. van der Wal, and P. Burt. Real-time scene stabilization and mosaic construction. In *Proc. Workshop on Applications of Computer Vision*, 1994.

[13] I. Haritaoglu, Larry S. Davis, and D. Harwood. $W^4$ who? when? where? what? a real time system for detecing and tracking people. In *FGR98*, 1998.

[14] M. Isard and A. Blake. Contour tracking by stochastic propagation of conditional density. In *Proceedings of the 1996 European Conference on Computer Vision*, pages 343–356, 1996.

[15] Institute for Simulation & Training IST. *Standard for Distributed Interactive Simulation – Application Protocols, Version 2.0.* University of Central Florida, Division of Sponsored Research, March 1994.

[16] S. Ju, M. Black, and Y. Yacoob. Cardboard people: A parameterized model of articulated image motion. In *Proceedings of International Conference on Face and Gesture Analysis*, 1996.

[17] T. Kanade, R. Collins, A. Lipton, P. Anandan, and P. Burt. Cooperative multisensor video surveillance. In *Proceedings of the 1997 DARPA Image Understanding Workshop*, volume 1, pages 3–10, May 1997.

[18] T. Kanade, R. Collins, A. Lipton, P. Burt, and L. Wixson. Advances in cooperative multi-sensor video surveillance. In *Proceedings of the 1998 DARPA Image Understanding Workshop*, volume 1, pages 3–24, November 1998.

[19] D. Koller, K. Daniilidis, and H. Nagel. Model-based object tracking in monocular image sequences of road traffic scenes. *International Journal of Computer Vision*, 10(3):257–281, June 1993.

[20] A. Lipton, H. Fujiyoshi, and R.S. Patil. Moving target detection and classification from real-time video. In *Proceedings of the 1998 Workshop on Applications of Computer Vision*, 1998.

[21] Alan J. Lipton. Local application of optic flow to analyse rigid versus non-rigid motion. In *ICCV99 Workshop on Frame-Rate Applications*, September 1999.

[22] T. Matsuyama. Cooperative distributed vision. In *Proceedings of DARPA Image Understanding Workshop*, volume 1, pages 365–384, November 1998.

[23] H. S. Sawhney, S. Hsu, and R. Kumar. Robust video mosaicing through topology inference and local to global alignment. In *Proc. European Conference on Computer Vision*, 1998.

[24] H. S. Sawhney and R. Kumar. True multi-image alignment and its application to mosaicing and lens distortion. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 1997.

[25] C. Tomasi and T. Kanade. Shape and motion from image streams: factorization method. *International Journal of Computer Vision*, 9(2), 1992.

[26] K. Toyama, J. Krumm, B. Brumitt, and B. Meyers. Wallflower: Principles and practice of background maintenance. In *Proc. International Conference on Computer Vision*, pages 255–261, 1999.

[27] L. Wixson, J. Eledath, M. Hansen, R. Mandelbaum, and D. Mishra. Image alignment for precise camera fixation and aim. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 1998.

[28] L. Wixson and A. Selinger. Classifying moving objects as rigid or non-rigid. In *Proc. DARPA Image Understanding Workshop*, 1998.

[29] C. Wren, A. Azarbayejani, T. Darrell, and Alex Pentland. Pfinder: Real-time tracking of the human body. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):780–785, 1997.