A Master-Slave System to Acquire Biometric Imagery of Humans at Distance

Xuhui Zhou, Robert T. Collins, Takeo Kanade, Peter Metes
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
{xuhui, rcollins, tk, metes}@cs.cmu.edu

ABSTRACT
The Distant Human Identification (DHID) system is a master-slave, real-time surveillance system designed to acquire biometric imagery of humans at distance. A stationary wide field of view master camera is used to monitor an environment at distance. When the master camera detects a moving person, a narrow field of view slave camera is commanded to turn to that direction, acquire the target human, and track them while recording zoomed-in images. These zoomed-in views provide meaningful biometric imagery of the distant humans, who are not recognizable in the master view. Based on the lenses we currently use, the system can detect and track moving people at distances up to 50 meters, within a 60^o field of regard.

Keywords
Video Surveillance, Master-Slave, Motion Detection, Real-Time Tracking, Biometric Imagery

1. INTRODUCTION
Video surveillance is already prevalent in our daily life. We can see surveillance cameras in banks, airports, stores and parking lots. Most commercial surveillance systems only provide recorded video for review after crimes or accidents happen. Therefore, there is growing interest in developing real-time systems with automated video understanding algorithms that provide prompt alert of events while they are happening.

Extensive research has been conducted in automated surveillance, especially in moving object detection and tracking [1], [2], [3], [5], [6]. The Video Surveillance and Monitoring (VSAM) system [1] is a multi-camera system that allows a single operator to monitor activities in a cluttered environment using a distributed network. W^4 [2] is a real-time system for detecting and tracking people and their body parts in video imagery. Pfinder [3] is a real-time system for tracking a person, using a multi-class statistical model of color and shape to obtain a 2D representation of head and hands over a wide range of viewing conditions.

We are developing a testbed system and algorithms for acquiring biometric imagery of non-cooperative subjects (neither actively cooperating nor trying to deceive) in an outdoor environment. A typical scenario is monitoring a parking lot, or building perimeter, to acquire identifiable imagery of all the people passing through that area. The biometric imagery consists of short gait sequences and relatively high resolution views of each person's face, suitable for use by gait and face recognition algorithms developed separately.

The goals of wide area monitoring and biometric image acquisition are in conflict with each other. To identify people at a distance, we need to use a highly zoomed camera. But, with the zoom increase comes a decrease in overall field of view, so we can only monitor a smaller scene. The main challenge in acquiring biometric data at a distance is not optics, but camera control. We cannot just point a static sensor at the scene and hope to obtain focused, high-resolution imagery of a person. Instead, we must actively control the pointing angle and zoom of the camera.

To solve the dilemma above, this paper proposes a Distant Human Identification (DHID) system based on a master-slave camera architecture. A static, wide field of view master camera is used to monitor a wide area at a distance. When the master camera detects a moving human, an active narrow field of view slave camera is commanded to turn to that direction, acquire the human target, and track them while taking high resolution biometric imagery. Unlike traditional master-slave systems where the master camera completely controls the slave pointing angle (e.g. [1]), keeping a highly zoomed camera pointing at a moving person requires a level of pointing accuracy that is not achievable from calibration alone. That is, we cannot simply use scene coordinates estimated from the master camera to control the pan and tilt of the slave camera when it is zoomed in to view a person's face. We instead use control signals from the master camera only initially, to get the person within the slave camera's field of view, and then perform real-time processing on the slave camera video to keep the active camera centered on the moving target.

This hybrid master-slave camera system uses a combination of motion detection algorithms to acquire and track moving people. Conventional approaches to moving object detection and tracking include background subtraction [1], [2], [3], [4], temporal differencing [7], optical flow [8], and color-based blob tracking [15][16]. Our approach uses a combination of all of the above. The static master camera uses adaptive background subtraction to detect all moving objects in the scene, and determines whether they are people or vehicles. After the master detects a new person, the moving slave camera uses a combination of optical flow stabilization and temporal differencing to acquire the human

IWVS'03, November 7, 2003, Berkeley, California, USA.

target. After detection is achieved in the slave camera view, the slave switches into an active tracking mode. The tracking is based on color blob tracking using the mean-shift algorithm.

The following sections will give a detail description of the DHID system. In Section 2, we will give an overview of the architecture of the system. We will address how master and slave camera coordinate with each other. In Section 3 and Section 4, we will describe in detail the algorithms used in master camera processing and slave camera processing. In Section 5, we will show some implementation results and discuss possible future work.

2. SYSTEM OVERVIEW

The DHID system consists of three parts: a stationary wide field of view Master Camera, a moving narrow field of view Slave Camera and a hardware Pan-Tilt-Unit (PTU) to which the slave camera is mounted. Each of these three components is controlled by a separate computer, connected to the others through a standard TCP/IP network, as shown in Figure 1.

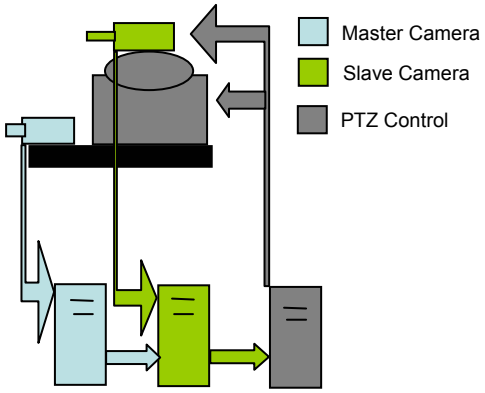


Figure 1: Logical layout of DHID system

Mirroring the hardware, the software of the system consists of three standalone processes on the three computers: Master Camera Process, Slave Camera Process and Pan-Tilt-Zoom Process. These three processes communicate with each other through a three-tiered architecture, as shown in Figure 2.

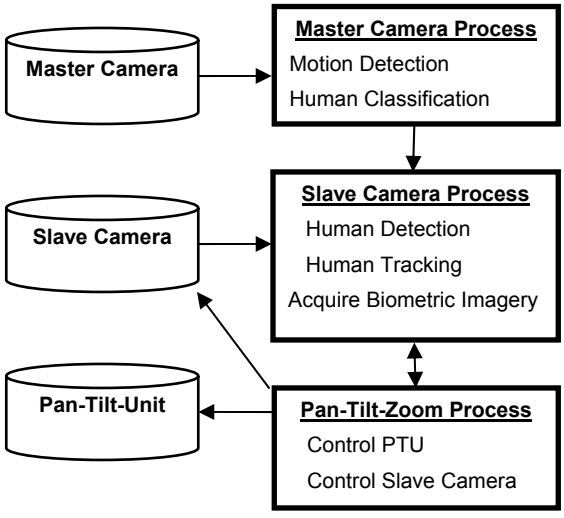


Figure 2: Three-Tier architecture of DHID system

The Master Camera Process is the starting tier of the system. It continuously grabs RGB video from the master camera, detects moving objects, and determines which of them are people. The system is currently designed to detect and track a single moving person instead of a group. If there are multiple candidates on the screen, one of them is selected for tracking based on criteria described in Section 3. After detecting a person, the Master Camera Process computes their dimension, speed, and position, and passes these parameters to the Slave Camera Process.

The Slave Camera Process is the middle tier of the system. It is a two-thread process. One thread listens to the Master Camera Process for parameters alerting it to a detected person. The other main thread grabs and processes video from the slave camera to independently detect and track this person. The Slave Camera Process determines parameters (e.g. Pan, Tilt, Speed of PTU and Zoom, Focus of Slave Camera) and passes them to the Pan-Tilt-Zoom Process. During the detection phase, these parameters are sent from the Master Camera Process, allowing the master camera to take control of the Pan-Tilt angle. During the active tracking phase, these parameters are computed by real-time slave camera tracking, allowing the slave camera to servo on the moving target.

The Pan-Tilt-Zoom Process is the end tier of the system. It resides on a real-time operating system (we use VXWorks) and controls the PTU and Slave Camera. This process accepts command parameters from the Slave Camera Process to move the PTU and set the zoom, focus and iris of the Slave Camera.

3. MASTER CAMERA PROCESS

The main task of the Master Camera Process is to monitor a wide area, detect moving humans automatically, and compute Pan-Tilt positions for the Slave Camera to "look at". This section provides an overview of our algorithm to detect moving objects, distinguish humans from other objects, and calibrate the geometric master-slave relationship.

3.1 Motion Detection at Pixel Level

Detecting moving objects in a video stream is a traditional research problem in computer vision. Our approach is adaptive background subtraction. The basic idea is to maintain a statistical background model at every pixel. When the intensity of a pixel in the current frame differs above a threshold from the background model, it is labeled as foreground. Foreground pixels are then grouped together to indicate potential moving objects.

One intuitive approach is to get a weighted average of all the pixel values in the past frames as in the following equation (1).

$$B(t) = \begin{cases} I(t) & ;t = 0 \\ I(t) * \omega + B(t-1)(1-\omega) & ;t > 0 \end{cases} \quad (1)$$

Where $B(t)$ is the background model value at time t, $I(t)$ is the image pixel value at time t, and ω is a weight parameter between 0 and 1.

The difficulty and the trick are how to choose the weight ω. If the weight is small, the background model detects well but adapts too slowly to properly "absorb" an object that has stopped in the scene. The extreme of this case is no update at all when ω = 0. If the weight is too large, the background adapts promptly but can be easily corrupted by moving objects. There will be either a long

trail following a moving object, or a hole within the moving object. The extreme of this case is two-frame differencing when ω = 1.

Inspired by the Wallflower Algorithm [4], we use two background models, but instead of a predicted history and actual history as used in Wallflower, we use a combination of the weighted background models above. Model One uses a small weight for updating. Model Two use a large weight in updating, but only updates those pixels whose locations are determined to be background by Model One. The pixels that are considered to be background in Model One will not belong to any moving object, and therefore they will not corrupt the background in the quickly updating Model Two. To detect motion, each pixel in the current frame is checked against both background models. If the pixel is foreground in both models, we declare that it is foreground.

Since our background model is adaptive with time, we also have built an additional mechanism in our background model to adapt to sharp changes in the overall scene (e.g., the sun comes out from cloud cover). When the number of moving foreground pixels exceeds a certain threshold, both background models are reset to the current frame.

3.2 Human Classification at Region Level

After classifying foreground change at the pixel level, we perform a median filter to reduce "salt and pepper" noise. We then group pixels into blob regions and determine a class label (human vs. other) for each foreground blob [5], [6]. For efficiency, we use a very simple model of the expected appearance of a human, consisting of a center-surround rectangular kernel (see Figure 3). The rectangle is chosen to have a size and aspect ratio consistent with known dimensions of upright humans in the scene. A central rectangle of positive weights is surrounded by an outer ring of negative weights. This is to enforce that a cluster of foreground pixels of the right shape must be surrounded by a ring of non-foreground (i.e. background) pixels.

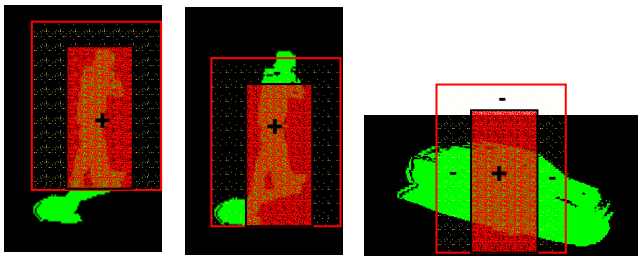


Figure 3: Template matching cases. (Left) High score (Middle and right) Low score.

We convolve the shape kernel over the foreground image and threshold the convolved results. After flood fill with 8-connect, we get a set of candidate blobs classified as moving, upright humans. As shown in Figure 3, it is clear that we can distinguish human from cars easily because of the penalty imposed by the negative weights in the outer ring of the shape kernel. Meanwhile, the kernel does a good job of locating the human upper body, and ignoring shadows below the feet.

After locating all the possible human blob candidates, we need to choose one for passing to the slave camera. First of all, the operator can manually click on one subject, which will always take priority. Otherwise, the system is in an automatic choosing mode. At the first frame, we place the target focus of attention cursor at the center of screen. In the following frame, if there is no detected person, the cursor stays. When a person appears, the focus of attention cursor switches to that person. If there are multiple people found, the cursor will go to the nearest one. In the case of a moving human, the cursor will follow that person since the target human will always appear to be the closest object from the cursor position in the last frame due to minor movement in consecutive frames.

Future work on multiple people tracking will explore how to multitask between multiple candidates to ensure that views of each person are tracked accordingly, and that the appropriate target is selected among multiple candidates.

3.3 Calibration of Master-Slave Correlation

After getting the coordinate of the target human in the master camera image, we need to move the slave camera to point at and further track the person. The purpose of master-slave calibration is to determine the geometric relationship between the master camera image pixel coordinates $M(X,Y)$ and the Pan-Tilt angles of the slave camera $S(P, T)$.

First, we collect a series of sample master pixel locations M_i (X_i, Y_i) $(i = 1,2,3,...n)$. For each pixel M_i, which is related to an actual point P_i in the surveillance scene, we manually move the slave camera to center the slave image at P_i and record the corresponding slave pan-tilt angles S_i (P_i, T_i). After this process, we can get n pairs of master-slave correlations (M_i,S_i) $(i = 1,2,3,...n)$.

Then, for every other pixel point M_j in the master image, we calculate the related slave angles S_j as a linear interpolation of the slave angles (S_1,S_2) of the closest two sample points (M_1,M_2) according to equation (2).

$$S_j = S_1 + (S_2 - S_1) * \frac{M_j - M_1}{M_2 - M_1} \qquad (2)$$

Due to the errors of approximating a nonlinear mapping with linear interpolation, and the mechanical bias of the PTU, this process just gives us a coarse registration between the master and slave camera coordinate systems. However, this is accurate enough to bring the target object within the slave camera field of view, which is sufficient since further image processing on the slave camera can detect and center the target.

In the implementation of the calibration, we adopt an aggressive approach. After the first two points, we can predict the pan-tilt angles of the remaining points in the scene. If the calculated angles of a newly marked point match with the actual angles within a certain threshold, the calibration is finished. If not, this point is used as another sample point, and the calibration process continues.

4. SLAVE CAMERA PROCESSSING

The Slave Camera Process is responsible for detecting, tracking and acquiring biometric imagery of the person for later identification. Since the slave camera has a narrower field of view

than the master camera, it will have a higher-resolution view of the target human, so tracking should be easier. On the other hand, the slave camera is also continuously moving, which actually makes the tracking more difficult. This section will describe in detail the algorithms for detection and tracking from the moving slave camera. A hybrid approach to detection and tracking is used. For detection, sparse optic flow is used to register adjacent frames for frame differencing and motion history accumulation. This serves to find the moving person within the moving image. At this point, a color histogram-based appearance model is formed and a mean-shift tracker is used from then on to track the moving person while actively servoing the PTU to keep that person centered within the field of view.

4.1 Target Detection Phase

As mentioned in the last section, our coarse calibration procedure is only precise enough to bring the target human within the field of view, but not accurate enough to place them at the center point of the slave camera image Therefore, we need to detect the target object in the slave camera based on the person's motion as well as the blob dimension and speed information passed from the Master Camera Process. Since the slave camera is a moving camera, we cannot use a background model to detect motion anymore. Instead, we use a combination of Frame Differencing and Motion History [12], [13]. First, we register every two consecutive frames through sparse optic flow. Then, any pixel that differs above a certain threshold between the two registered frames is determined to be a foreground pixel. After that, we add several consecutive foreground images to form a Motion History Image (MHI). Finally, we convolve the MHI with a shape kernel (as described previously) to detect the target person.

4.1.1 Frame Registration

We use sparse optic flow computed by the KLT (Kanade-Lucas-Tomasi) Tracker [9], [10], [11] to register two consecutive frames (*f1, f2*). *N* features *P*(i) (i=1,2,3,…*N*) in *f1* are selected as 7 by 7 patches with high intensity variation in both X and Y directions (e.g. corners). The KLT's gradient ascent procedure then finds the corresponding positions of these N features in the second frame. Since the slave camera only has pan-tilt movement, over a small time step we can approximate it by a translation motion $d = [d_x, d_y]^T$ applied to each patch center $X=(x,y)$: $P_{f1}(X) \rightarrow P_{f2}(X+d)$. For every patch, the problem is to find the translation d that minimizes the dissimilarity between $P_{f1}(X)$ and $P_{f2}(X+d)$, which is to solve the equation (3)(4)(5) iteratively in a Newton-Raphson style minimization [11].

$$T * \begin{bmatrix} d_x \\ d_y \end{bmatrix} = a \tag{3}$$

$$T = \iint_W \begin{bmatrix} g_x^2 & g_x g_y \\ g_x g_y & g_x^2 \end{bmatrix} \omega(x,y)\,dxdy \tag{4}$$

$$a = \iint_W [P_{f2}(x,y) - P_{f2}(x,y)]\omega(x,y) \begin{bmatrix} g_x \\ g_y \end{bmatrix} dxdy \tag{5}$$

Where g_x, g_y is the gradient value, *W* is the given feature window, and $\omega(x)$ is a Gaussian weighting function to emphasize the central area of the window.

From the steps above, we can get *n* ($0<n<N$ due to lost features) pairs of corresponding features and their translations $\overrightarrow{d_i}$ (i=1,2,…,n). As shown in the example in Figure 4(c) (d), there are two major clusters of translation, corresponding to background and foreground motion respectively.

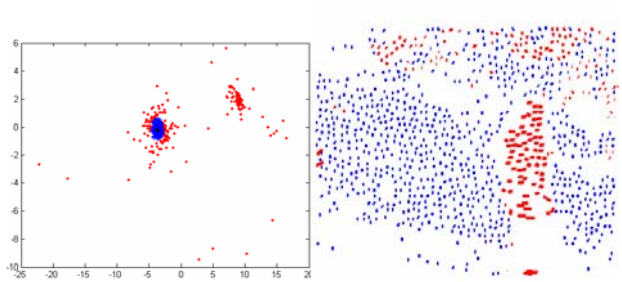
We use a least-median-of-squares (LMedS) algorithm [14] to find the largest cluster of translations: $\overrightarrow{d_m}$ as shown in equation (6)(7)(8).

$$lmeds = \min_{i=1}^{n} \{median(R_i^2)\} \tag{6}$$

$$R_i = |\overrightarrow{d_j} - \overrightarrow{d_i}|^2 \ (j = 1,2,3,...,n) \tag{7}$$

$$\overrightarrow{d_m} = \arg_d(lmeds) \tag{8}$$

(a) (b)



(c) (d)

Figure 4: A frame registration example.

(a)(b) A pair of consecutive frames. (c) The clustering results. (d) The correspondent patches. Blue are background and red are foreground.

Since d_m is the most common translation, it should correspond to the average translation of background patches. In Figure 4(c), patches whose translation differs with $\overrightarrow{d_m}$ within boundary $B=3*\sqrt{lmeds}$ are labeled blue (thick) versus red patches (thin) that do not fall in that cluster. The statistics result shows that 85 percent of the n tracked patches are clustered as background (blue). As can be seen in the figure, this clustering coincides well with the distinction between the two different motions of the background and foreground "layers" of the scene.

On the other hand, this clustering alone is not accurate and sufficient enough to delineate the whole foreground layer. First, patches are sparse. Second, not all the background patches are exactly clustered within the bounds of cluster *B* (e.g. the red dots around blue cluster in Figure 4(c)). Therefore, the next step of frame difference is necessary.

4.1.2 Motion History and Template Matching

After getting an estimate of the background translation d_m, we translate the first frame by d_m so that background pixels are aligned with each other in both frames. The aligned background pixels are then removed by simple frame subtraction and thresholding. A typical problem of this approach, however, is that although it removes the background, it does not completely detect the foreground object but only the leading and trailing edges. We solve this problem by adding several consecutive foreground frames together to form a Motion History Image (MHI) [12], [13]. The decay process of the Motion History representation serves to fill in a more complete description of the foreground object.

Next, we convolve the MHI to segment the target human object with a rectangular center-surround shape template similar to the one used in the Master Camera Process. The advantage here is that we know roughly the dimensions of the target person based on the information of Master Camera. Once the detection process detects a target object within the same neighborhood for a number of consecutive frames, we flag it as a successful detection and trigger the tracking phase

4.2 Target Tracking Phase

Since human beings are non-rigid objects, we use the Mean-Shift algorithm for tracking them based on a color histogram appearance model [15], [16], [17]. In the first frame, we calculate the histogram model. In subsequent frames, we shift the blob object to a new location whose histogram best matches the template. After that, we adjust the scale of the blob object and continue tracking.

4.2.1 Histogram Model

To acquire a color histogram appearance model, the initial position and dimensions of the 2D target object blob obtained from the previous detection phase are used. For every pixel in the blob, we do a color space conversion from RGB into C1 = B-G, C2 = G-R, C3=R+B+G. We use this color space because it allows us to emphasize chrominance features C1, C2 more than intensity feature C3. Then, we sample C1, C2 into 8 bins and C3 into 4 bins [18], [19]. Every pixel in the blob will give a vote to a bin in the 8*8*4 histogram.

4.2.2 Mean Shift Tracking

In subsequent frames, our task is to find the target location whose neighborhood color density function is most similar to the model's density function. This is equivalent to maximizing the Bhattacharyya Coefficient associated with the model and candidate distributions, which can be achieved by mean shift iterations [15], [16].

From the start position *X=(x,y)* of the candidate blob, we compute an offset ΔX as in equation (9) iteratively.

$$\Delta X = \frac{\sum_W K(Y-X)\omega(Y)(Y-X)}{\sum_W K(Y-X)\omega(Y)} \quad (9)$$

Where *W* is the blob window around the position *X, Y=(x,y)* is the coordinate of pixel (*P*) in *W*, $\omega(Y) = \sqrt{H_m(i)/H_c(i)}$ is a sample weight at *P* with color *i* which shows the histogram density similarity of color *i* between model and candidate, and *K* is a suitable kernel function.

It is proved that the shifting process will converge to a new position X', which corresponds to a local mode in the candidate position.

After locating the matched target blob, we then adjust the tracking scale by $\pm 10\%$ and choose the scale yielding a better Bhattacharyya Coefficient. This will help us adapt to scale changes when the target object moves closer or further. When calculating the Bhattacharyya Coefficient at this stage, we enhance the algorithm used in [14] with an approach similar to the shape kernel approach used in Section 3.2. We use an center-surround difference mask to enforce that the target object (foreground) in the current tracking window is surrounded by a ring of non-target (background) pixels. First a rectangle ring (outer) is place around the current tracking window (inner). Then, the Bhattacharyya Coefficient in equation (10) is calculated over the inner window (B_i) and outer ring (B_o) respectively. The final Combined Bhattacharyya Coefficient (B') is the normalized difference between B_i and B_o, as shown in Equation (11)

$$B = \sum_W \sqrt{m(u) \bullet p(u)}; \quad u = c(x,y) \quad (10)$$

$$B' = \frac{B_i}{\|m\| * \|p_i\|} - \frac{B_o}{\|m\| * \|p_o\|} \quad (11)$$

where *W* is the image area of computation. *m(u)* is the model histogram at color *u*, *p(u)* is the target histogram at color *u, u* is the color at pixel *(x, y)* within area *W*, and P_i, P_o are the target histograms in the inner window and outer ring.

5. RESULTS AND DISCUSSION

5.1.1 Implementation Results

Our hardware implementation of the DHID system is shown in Figure 5. The master camera appears on the left and the slave camera is on the right. The slave camera is mounted on a Pan-Tilt Unit. The hardware profiles are listed in Table 1.

The software is written in C++. The running speed of the system is shown in Table 2. It is limited by the frame rate of the NTSC frame grabber (30 frame/second). The Detection phase is slower than the tracking phase because of the feature tracking in the frame registration step. The image resolution used by master and slave camera is 640*486. In some parts of the implementation (such as the KLT tracker), images are down sampled to speed up the process.

Based on the lenses we currently use, the system can detect and track moving humans at a distance of 50 meters within a 60^{o}field

of regard (which is bounded by the width of the fixed master camera field of view). The success rate in our tracking experiments, performed overlooking a parking lot, has been above 95 percent. We are currently building a second system with upgraded hardware that will operate at further distances.

Table 1: Hardware Profile

Process	Camera	Computer
Master Camera Process	JVC TK-C1380	Pentium 2.6GHz, 1G Ram Windows XP
Slave Camera Process	Sony DXC-9000	Pentium 2.4GHz, 1G Ram Windows XP
Control Process	———	Pentium 1GHz, 256M Ram VXWorks.

Table 2: System Running Speed

Process	Frame Rate (f/s)
Master Camera	24
Slave Camera	20 (detection), 25(tracking)

Some sample images of detection and tracking results are shown below. Figure 6 is a Master Detection example. The image on the left is the original image and image on the right shows the detected foreground regions. There are two detected objects, denoted by the red bounding boxes. The target object is the blob on the right, shown with an overlaid red crosshair. Figure 7 is the related example in the Slave Camera view. Again, the image on the left is the original image and the image on the right is the detected foreground. The detected target is the target object in Figure 6. We can see clearly that the image from the slave camera could be used to identify the person, even though it is unrecognizable from the master camera view. Figure 8 is a Master-Slave tracking example. The master images are shown on the top row and slave images are shown on the bottom row. We just show frames 0, 50, 100 from left to right.

5.1.2 Future Work

First, in the Master Camera Detection module, we use a simple rectangular shape kernel to detect upright humans. In the future, we plan to collect more experimental data to build a better Human Contour Template.

Second, we plan to incorporate multiple people tracking in our system. We should implement the functionalities to distinguish a single person vs. a group of people, and to select a target among multiple candidates based on an intelligent strategy.

Third, during Slave Camera Detection, we use motion-compensated frame differencing to detect the moving target from a moving camera. Actually, the master camera process has much more information than just target position, such as the color and shape of the target, which the slave camera could also use during its detection phase. We could take advantage of this information by communicating between the Master Camera Process and the slave Camera Process, provided that the increased network traffic does not degrade the real-time system performance.

Finally, while tracking target humans in the Slave Camera, we plan to build a database of biometric imagery from each person passing through the scene. These images and video clips will be used by face and gait recognition algorithms to determine the identity of each person.

6. ACKNOWLEDGMENTS

This work was supported by the DARPA HumanID program under ONR contract N00014-00-1-0915.

Figure 5: Actual DHID System

Figure 6: Master camera detection example.

Figure 7: Slave camera detection example.

(a). Frame 0 **(b) Frame 50** **(c) Frame 100**

Figure 7: Master-Slave tracking example (Frame 0, 50, 100)

7. REFERENCES

[1] R. Collins, A. Lipton, H. Fujiyoshi, and T. Kanade Algorithms for cooperative multisensor surveillance Proceedings of the IEEE, Vol. 89, No. 10, October, 2001, pp. 1456 - 1477.

[2] I. Haritaoglu, D. Harwood, and L. S. Davis, "W4: Real-time surveillance of people and their activities," IEEE Trans. Pattern Anal. Mach. Intell., vol. 22, pp. 809–830, Aug. 2000.

[3] C. R. Wren, A. Azarbayejani, T. J. Darrell, and A. P. Pentland, "Pfinder: Real-time tracking of the human body," IEEE Trans. Pattern Anal. Mach. Intell., vol. 19, pp. 780–785, July 1997.

[4] K. Toyama, J. Krumm, B. Brumitt, and B. Meyers, "Wallflower: Principles and practice of background maintenance," in Proc. Int. Conf. Computer Vision, Corfu, Greece, 1999, pp. 255–261.

[5] G. L. Foresti, "Real-time system for video surveillance of unattended outdoor environments," IEEE Trans. Circuits Syst. Video Technol., vol. 8, p. 697, Oct. 1998.

[6] J. Heikkila and O. Silven, "A real-time system for monitoring of cyclists and pedestrians," in Proc. 2nd IEEE Int. Workshop Visual Surveillance, Fort Collins, CO, June 1999.

[7] P.L. Rosin and T. Ellis, “Image difference threshold strategies and shadow detection,” in Proc. British Machine Vision Conf., 1995, pp. 347–356.

[8] J. Barron, D. Fleet, and S. Beauchemin, “Performance of optical flow techniques,” Int. J. Comput. Vis., vol. 12, no. 1, pp. 42–77, 1994.

[9] B. D. Lucas and T. Kanade. An Iterative Image Registration Technique with an Application to Stereo Vision. International Joint Conference on Artificial Intelligence, pages 674-679, 1981.

[10] C. Tomasi and T. Kanade. Detection and Tracking of Point Features. Carnegie Mellon University Technical Report CMU-CS-91-132, April 1991.

[11] J. Shi and C. Tomasi. Good Features to Track. IEEE Conference on Computer Vision and Pattern Recognition, pages 593-600, 1994.

[12] J.Davis and A.Bobick. TheRepresentation and Recognition ofAction Using Temporal Templates. MIT Media Lab Technical Report 402, 1997.

[13] J.Davis and G.Bradski. Real-TimeMotion Template Gradients Using Intel Computer Vision Library. IEEE ICCV'99 FRAME-RATE WORKSHOP, 1999.

[14] P.J. Rousseeuw and A.M. Leroy. Robust Regression and Outlier Detection. John Wiley & Sons, New York, 1987

[15] R. Collins Mean-shift Blob Tracking through Scale Space Computer Vision and Pattern Recognition (CVPR'03), IEEE, June, 2003.

[16] Comaniciu, D., Ramesh, V. and Meer, P., “Real-Time Tracking of Non-Rigid Objects using Mean Shift,” IEEE Computer Vision and Pattern Recognition, VolII, 2000, pp.142-149.

[17] Comaniciu, D., Ramesh, V., Meer, P., “The Variable Bandwidth Mean Shift and Data-Driven Scale Selection,” International Conference on Computer Vision, Vol I, pp.438-445.

[18] S. Birchfield. Elliptical Head Tracking Using Intensity Gradients and Color Histograms. IEEE Conference on Computer Vision and Patten Recognition, Santa Barbara, California, June 1998

[19] M. Swain and D. Ballard. Color Indexing. International Journal of Computer Vision, 7(1):11-32 1991.