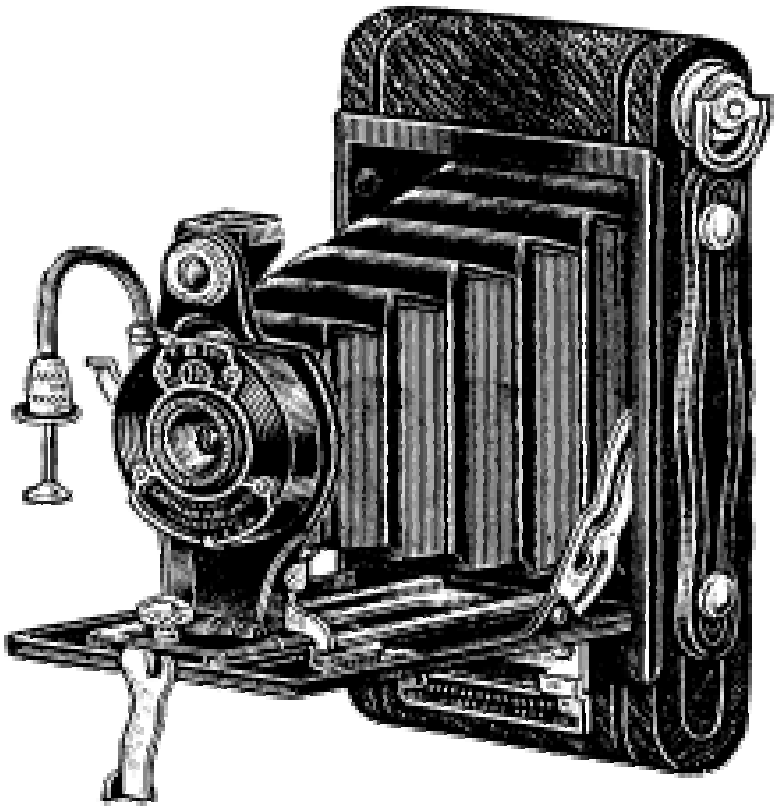
18-551 Digital Communications and Signal Processing Systems Design

LIGHTS, CAMERA, …ACTION!



Group 12: Ammar El Seed, Girish Jattani, Mona Zhe

1. Introduction

In the music, theatre, and film industries, getting the right shot is everything. Specifically, adjusting cameras to keep the singer/actor in the right view can make or break the experience. For a cameraman at a concert, it is a difficult task to operate the lift that moves him around in the set whilst trying to track the target person.

1.1 The Problem

Our goal, therefore, is to design an autonomous system to identify and track a person, so that a camera with many degrees of freedom can always orient itself towards the face of the person. The applications of this system extend throughout society- we could use it to film a live concert or theatre production without the need of a cameraman, for surveillance, or tracking specific players at sporting events. For such a system to have this capability, we outline the following tasks:

1) Recognize the presence of an object in a confined area
2) Distinguish an object as human or nonhuman
3) Decide the direction of the movement of that person (for example approaching / going away / left / right).

1.2 The Solution

We propose a silhouette-based body-shape recognition algorithm[1] that takes in an input frame and compares it to a background frame. One of the advantages of this method is that it implicitly looks for the macroscopic features that classify a silhouette as a human. To give a brief overview of our system, the input and background frame are first converted from RGB to grayscale (color is not needed). The frames are then differenced and the resulting image is thresholded to give rise to a binary image. This binary image then undergoes morphological filtering, blob coloring and if any blobs of interest are found, the regions of interest undergo statistical analysis.

[1] Based on Kuno and Watanabe's Automated Detection of Human for Visual Surveillance System

We implemented the blob detection, the morphological filtering and the human recognition algorithms on the DSP board leaving only preprocessing stages such as downsampling the high resolution BMP files to the PC. In the next section, we will go over in more detail the overview of the system.

1.3 Prior Work

Human detection has been for a long time a research interest to many academics. Some used silhouette-based methods to recognize humans, others used skin-color to find humans and still others studied the gait cycle of walking humans. In terms of previous 551 projects, only one project (Spring 2002 – Face Detection for Surveillance) is somewhat related. Even then, the similarities are superficial; that project involved detecting human faces in static pictures using a skin-color method. Unlike our project, it did not deal with video and it was never fully ported to the EVM. Thus our project is unique in that it is (we think) the first time that a human detection algorithm is implemented in hardware.

1.4 Constraints

We set constraints on our project to make it more feasible for us to finish it in the limited time that we have. These constraints revolve around the assumptions that we made. These assumptions include:

a) Environment does not change much, with minimal light variation.
   - This assumption is vital to the functioning of background subtraction, i.e. the frame differencing mentioned earlier will fail if the background changes dramatically. The same holds for lighting- if illumination changes then the amount of noise in the background subtraction will increase as well

b) Camera remains stationary – this is just specific to our project implementation.

c) Sufficient lighting available in the scene and a considerable difference in contrast between the background and any moving foreground entities (including humans)

2. System Overview

Our system can be broken up into 3 separate stages:

1) Online processing on the PC side
2) Online processing on the EVM side
3) Display of results graphically into a GUI

STAGE 1: Online processing on the PC side

The PC takes in as input the current frame and the background frame. It then does the following:

1) Downsampling – the initial resolution of the images is much higher than what we require so we decided to reduce the sizes of the images before further processing.
2) Background subtraction – differencing between current frame and background frame
3) Thresholding for binarization

STAGE 2: Online processing on the EVM side

The EVM takes in the binary image and does the following:

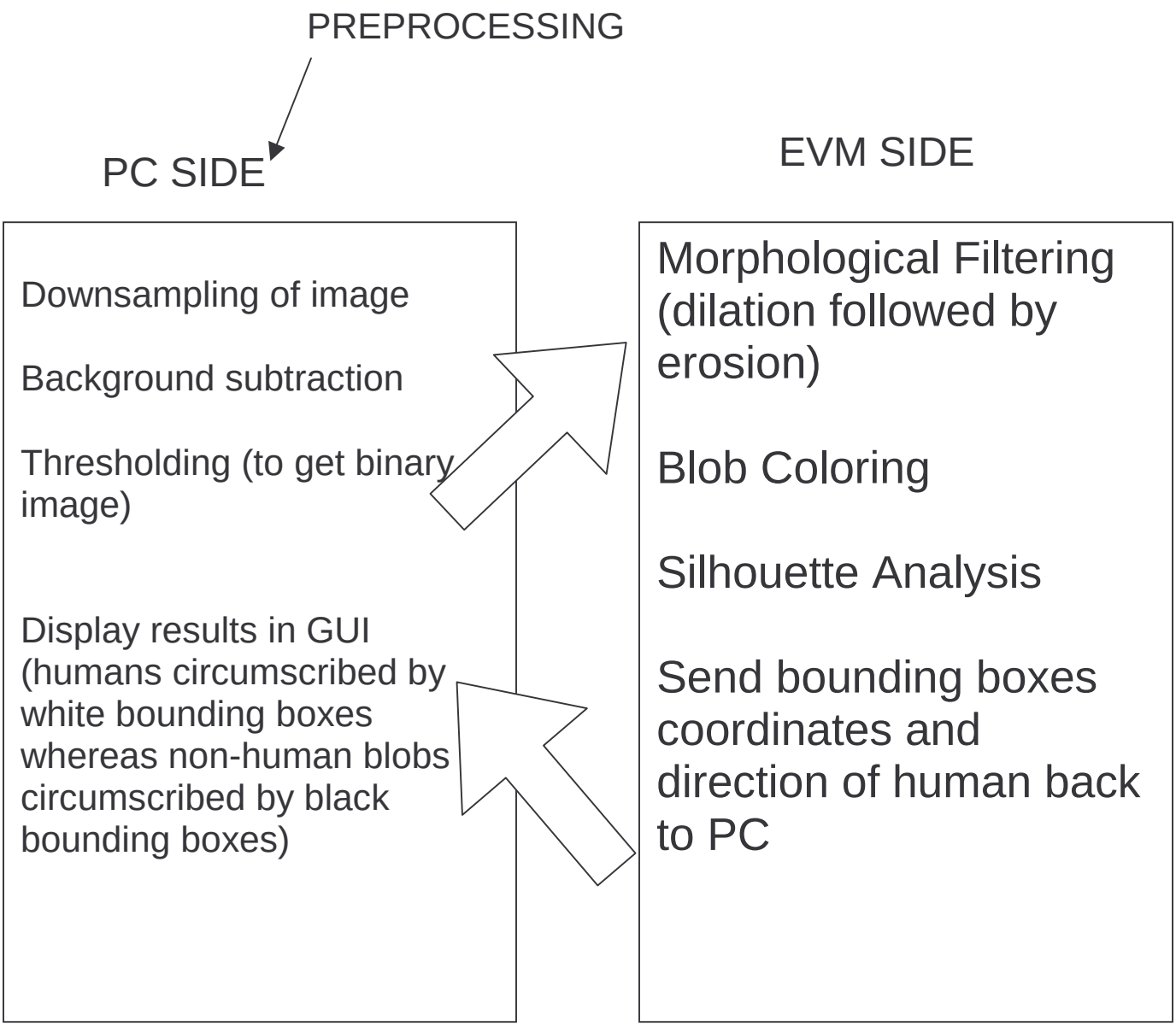
1) Performs morphological filtering (dilation followed by erosion)
2) Blob Coloring (i.e. the EVM finds all the blobs in the frame)
3) Silhouette Analysis (Used to determine if the blob in question is human or not)

4) Send bounding box coordinates of each blob found to PC

STAGE 3: Display of results graphically into a GUI

The PC takes in the bounding box coordinates and the respective (colored) bmp file as input, it then displays the bmp file but superimposed on it are the circumscribing rectangles around all the blobs found previously.

The following is a flowchart summarizing the above:



In the next section we will discuss our MATLAB implementation of our project and the results we got out of using this implementation on frames taken from the MOBO Database.

3. MATLAB Implementation

MATLAB proved very helpful when it came to simulating our project. Many of the functions that we were to later implement for our project were already provided, for example, imdilate() and imerode() performed dilation and erosion respectively. MATLAB also proved a good test bed for trying different structural elements to come up with better silhouettes.

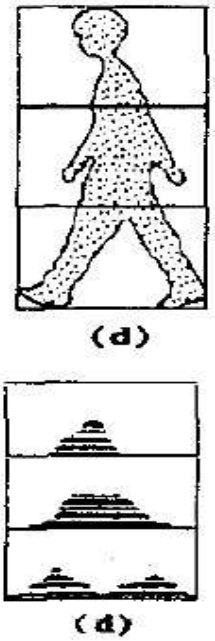
We wrote a function that called on provided routines to first read in the input and background frames, then to perform conversion to grayscale followed by what Kuno and Watanabe called the **extraction function**. The extraction function is used to aid in determining the threshold value for binarization. This is especially important if the brightness changes due to the moving foreground object (i.e. new object in scene) are small.

After that, we carry out the morphological filtering using a tall and thin structural element on the output binary image. In order to find the desirable silhouette, we call on *bwlabel* and use the heuristic that if any humans are present, then chances are the human(s) are the largest blobs.

Now that we have the desired blob, we can carry out silhouette analysis on it. In the next section, we will go into depth what silhouette analysis exactly entails and we will also look into the results of running our MATLAB implementation on the MOBODatabase.

3.1 Statistical Analysis

Assuming that we have a silhouette circumscribed by a bounding box, we can start to carry out statistical analysis. This is done by first dividing the bounding box enclosing the silhouette vertically into three equal segments, segment A – head, B – torso and C for the legs. The **projection histogram** of each segment then undergoes statistical analysis. The projection histogram is defined as a vector with length equal to that of the number of columns in the bounding box; each element in the vector is the sum of all white pixels in the respective column. The following is a graphic example:



The top diagram is an example of a silhouette of a person walking left across the frame.

Note that the projection histogram for segment C (the bottom one) has two heaps. This directly corresponds to the two legs. Similar remarks can be made about the head and torso sections. Also note that the heaps in both the head and torso segments are distributed mostly halfway between the two vertical edges of the bounding box.

The above observation will prove critical in determining whether or not the silhouette is human, as will be discussed in the next section.

According to the Watanabe paper, the following quantities need to be extracted from the projection histograms (note that this method only uses information from one frame):

I – Mean of each segment

The mean is defined here (as it is in the Watanabe paper) as the centroid of the projection histogram for a single segment. This value represents exactly where the white pixels are in the highest concentration along the horizontal axis. That is, the mean represents roughly where, relative to the left and right boundaries of the bounding box, are the white pixels mostly

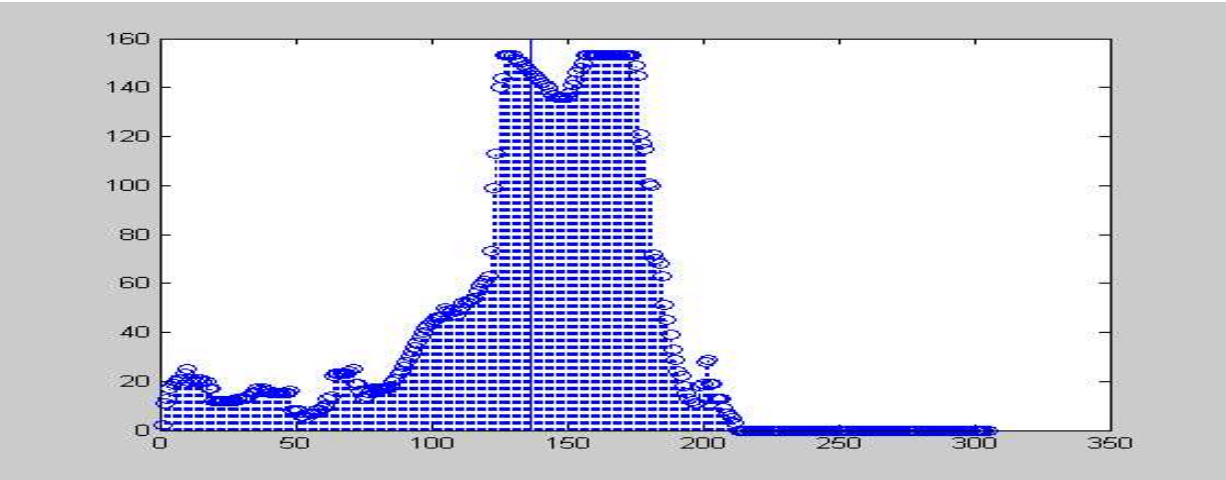
concentrated. The mean takes on values between 0 and 1, where 0 represents the extreme left (i.e. all the white pixels are on the leftmost column) and 1 represents the extreme right.
The formula for the mean is given by the following equation:

$$mean = \frac{\sum NumIndex \times NumWhitePixels}{SumTotalWhitePixels}$$

where *NumIndex* indicates the column number, *NumWhitePixels* represents the total number of white pixels in that column and *SumTotalWhitePixels* is the summation of all the white pixels in the projection histogram of that particular segment.

This number is then further normalized to the width of the bounding box. In other words, the mean is further divided by the number of columns in the bounding box. The reason why we need to normalize this mean value is to account for different sized bounding boxes, depending on how close the person is to the camera the size of the bounding box will get wider as the person approaches and vice versa.

The following shows an example of a projection histogram extracted from section B of one silhouette with the mean (before normalizing it) indicated by a solid line:



In other words the center of mass lies somewhere around index number 140. Hence half of the white pixels in this segment are on either side of the 140th column.

II – Comparison of the standard deviations of the segments

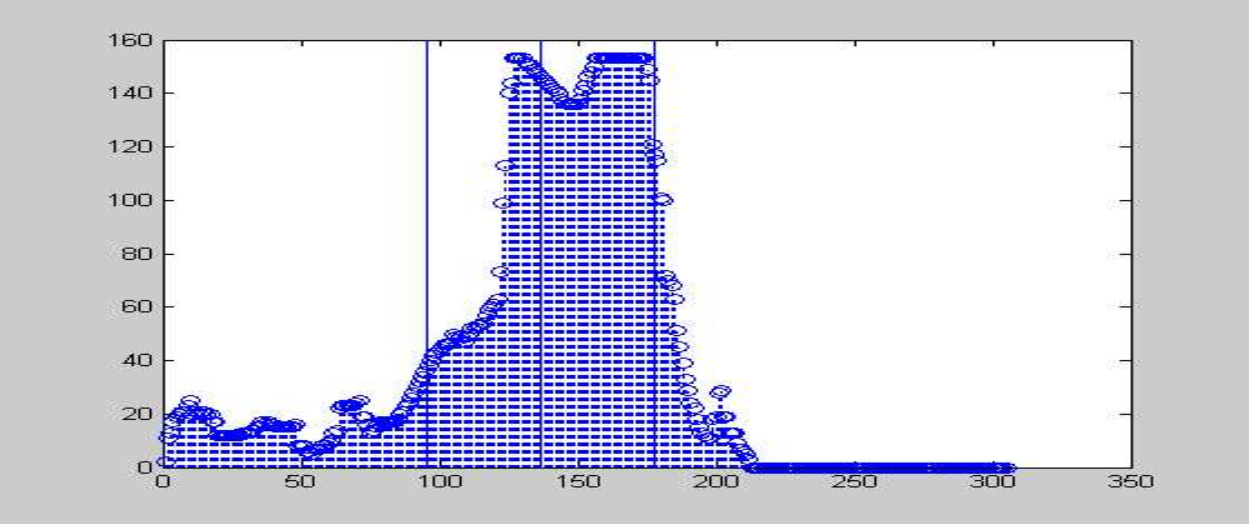
According to the Watanabe paper, the differences of the **standard deviation** between the projection histogram of segment A and segment B, and differences of the standard deviation between the projection histograms of segments B and C reflect the general form of humans.

The standard deviation per segment is calculated using the following formula:

$$std = \sqrt{\frac{\sum (mean - NumIndex)^2 \times (NumWhitePixels)}{SumTotalWhitePixels}}$$

where mean refers to the quantity given by the first formula (i.e. before being normalized). *NumIndex*, *NumWhitePixels* and *SumTotalWhitePixels* are as defined above.

The standard deviation shows the relative distribution of the white pixels around the mean value, as the following diagram shows:

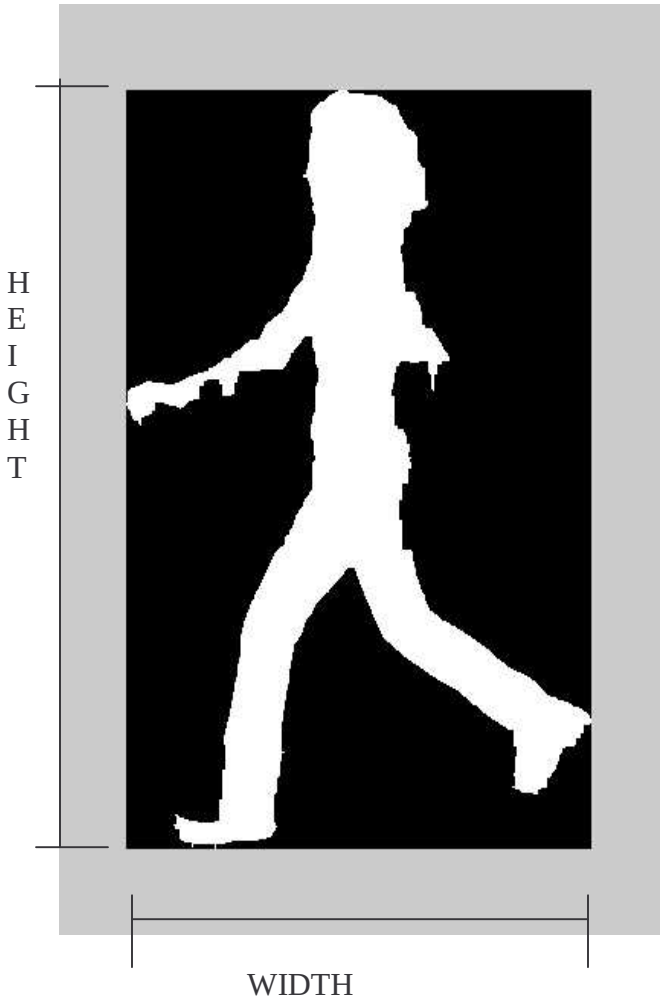


The two lines to the left and right of the mean represent the lines x = mean - std and x = mean + std.

Now once these calculations are done for each segment, the difference is taken between stdA (standard deviation of segment A) and stdB. The difference between stdB and stdC is also calculated.

III – Aspect Ratio

The aspect ratio is calculated by dividing the number of rows in the circumscribing bounding box by the number of its columns. This number will roughly reflect the height to width ratio of the silhouette:



For upright walking humans, we expected the aspect ratio to vary over a restricted range.

In the next section we will go over training the algorithm and will show how the results we got affected which quantities we ended up using to carry out human detection.

3.2 Database Analysis of training set

For our training set, we used a subset of the huge MOBODatabase; we used frames of three different people, in 6 different poses each, with 300 or more frames per pose. So that amounted to a bit more than 5000 frames. Here are a few examples:

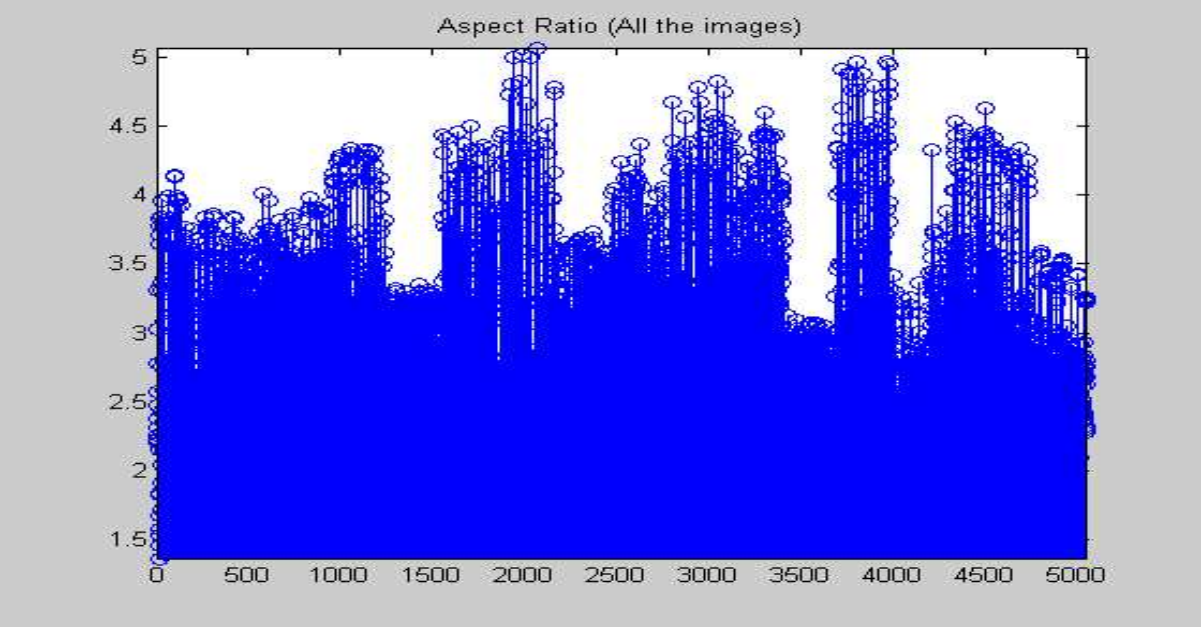
In order to go through each frame in one batch automatically we wrote a MATLAB M-file that loops through all the images in the batch calling the function described earlier in this section. The values for all the aforementioned quantities were stored in long arrays.

Here are the results we accumulated [2]:

[2] Note: each figure represents results accumulated across all the people, across all the different poses

Results for Aspect Ratio:

This next figure represents the numbers we got for aspect ratio:



Analysis of Results

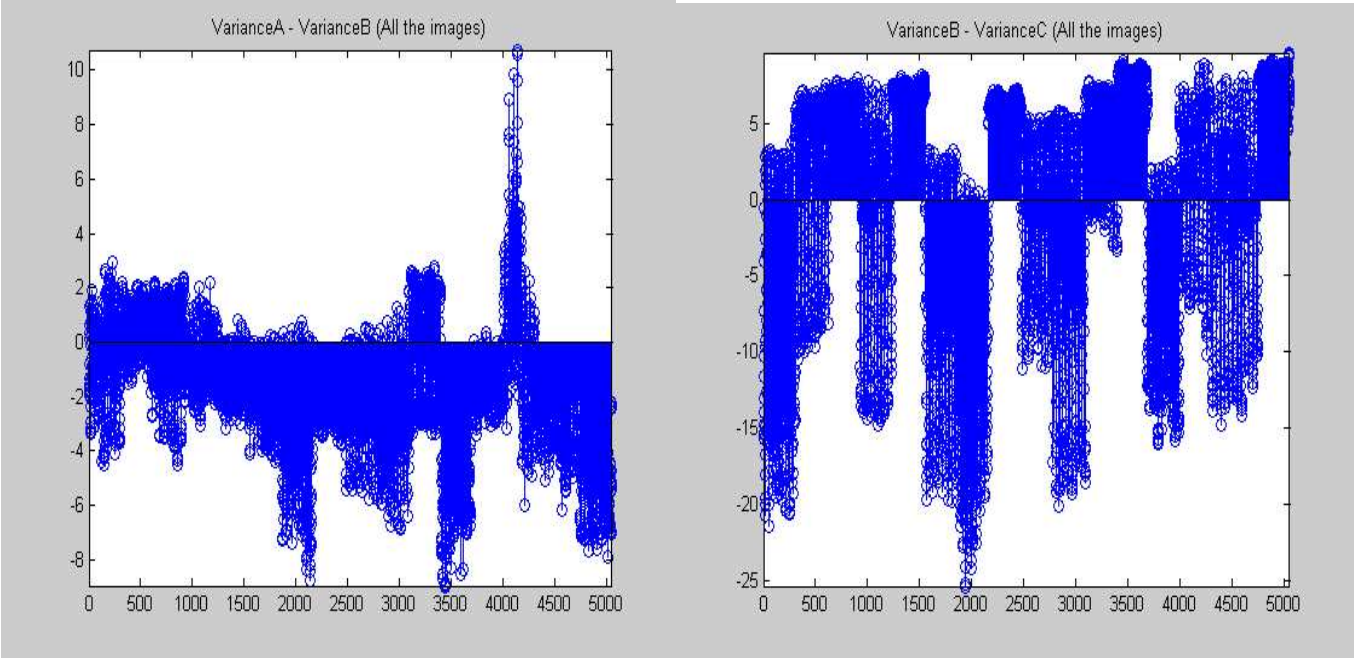
The average value of aspect ratio = 2.9004

with standard deviation (MATLAB defined standard deviation; not the above) = 0.6557

As anticipated the aspect ratio varied within a constrained range with most of the values lying only .6557 away from the mean. We thus concluded to use aspect ratio as a rough criterion for human detection.

Results when comparing the standard deviations of the different segments:

The following figure shows the results we got when we calculated the differences between the variances[3] of the two top segments:



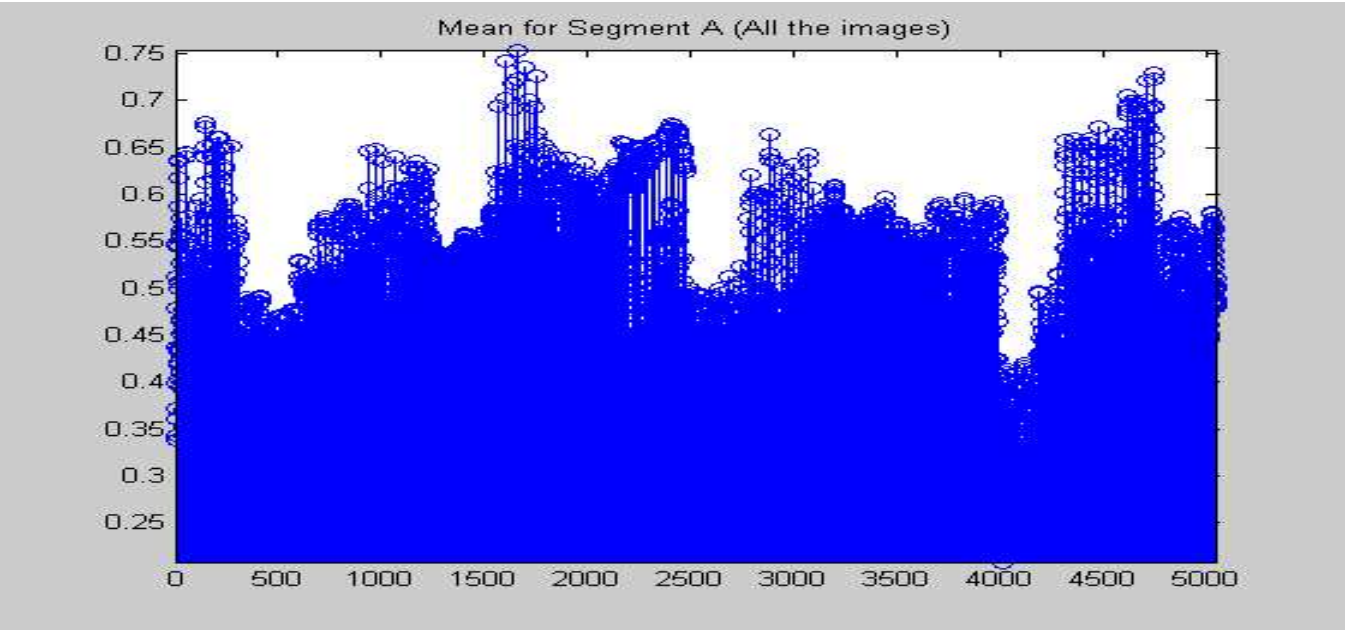
Analysis of Results

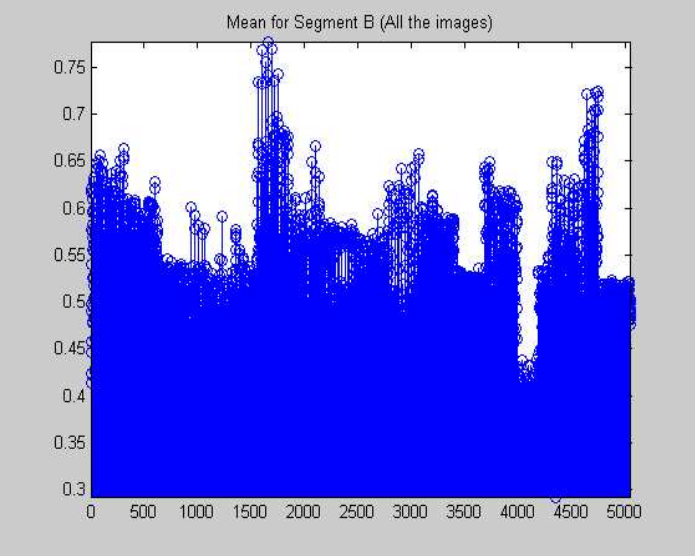
Studying these results, we realized that not only do these values oscillate considerably over different people and poses, but they also oscillate uncontrollably over one person, over one pose! That was a clear indication that this differencing will not serve us well as a good biometric. We thus decided to not use it.

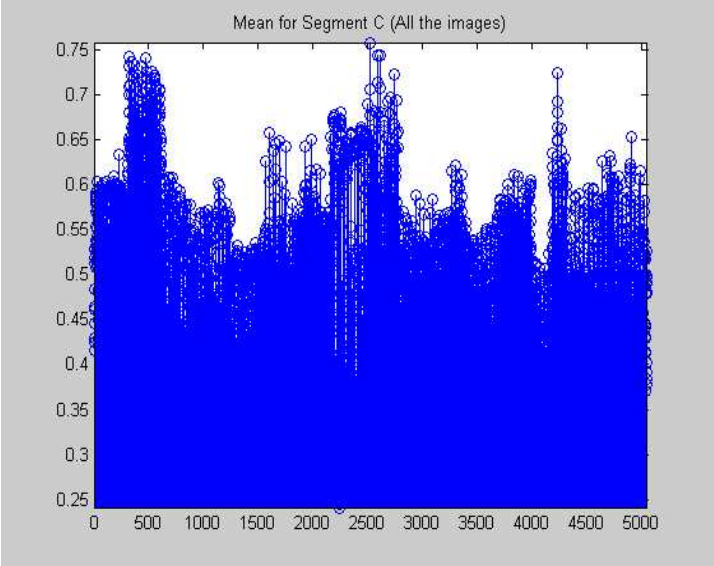
[3] We ran into problems trying to plot the small values of standard deviation. So we decided to use variance (standard deviation) ^ 2 instead.

Results from the Mean values of each segment:

The following figure shows us the values we got for the means of all three segments:







Results Analysis

On average, the mean for each segment was around 0.5, that is, the number of white pixels were distributed almost evenly across the middle of each segment. Since these values were

independent of pose or person, we found these mean values to be good metrics for human detection.

In the next section we will discuss how we ported our code to C and more importantly to the EVM. We will also highlight the changes that had to be made for our project to work on the EVM.

4.1 Algorithmic Motivations and Implementation Decisions

The Watanabe paper is extensively utilized as a guide throughout the implementation of the project. However, our implementations on the EVM deviate from those outlined in the paper and are derived, instead, from our own Matlab training set (the Mobo database) results. In general, we found that the details presented in the paper were not sufficient to form a concrete basis for grey scaling/image thresholding, morphological processing (dilation and erosion), and blob coloring. In order to resolve these issues with the paper, we decided to create a testing program that would allow us to test and observe single frame results based on varying thresholds for each respective function. Once these values were determined, we could then shift our efforts to examining multiple successive frames. Therefore, the overall algorithm for human detection and motion tracking was conceived from scratch.

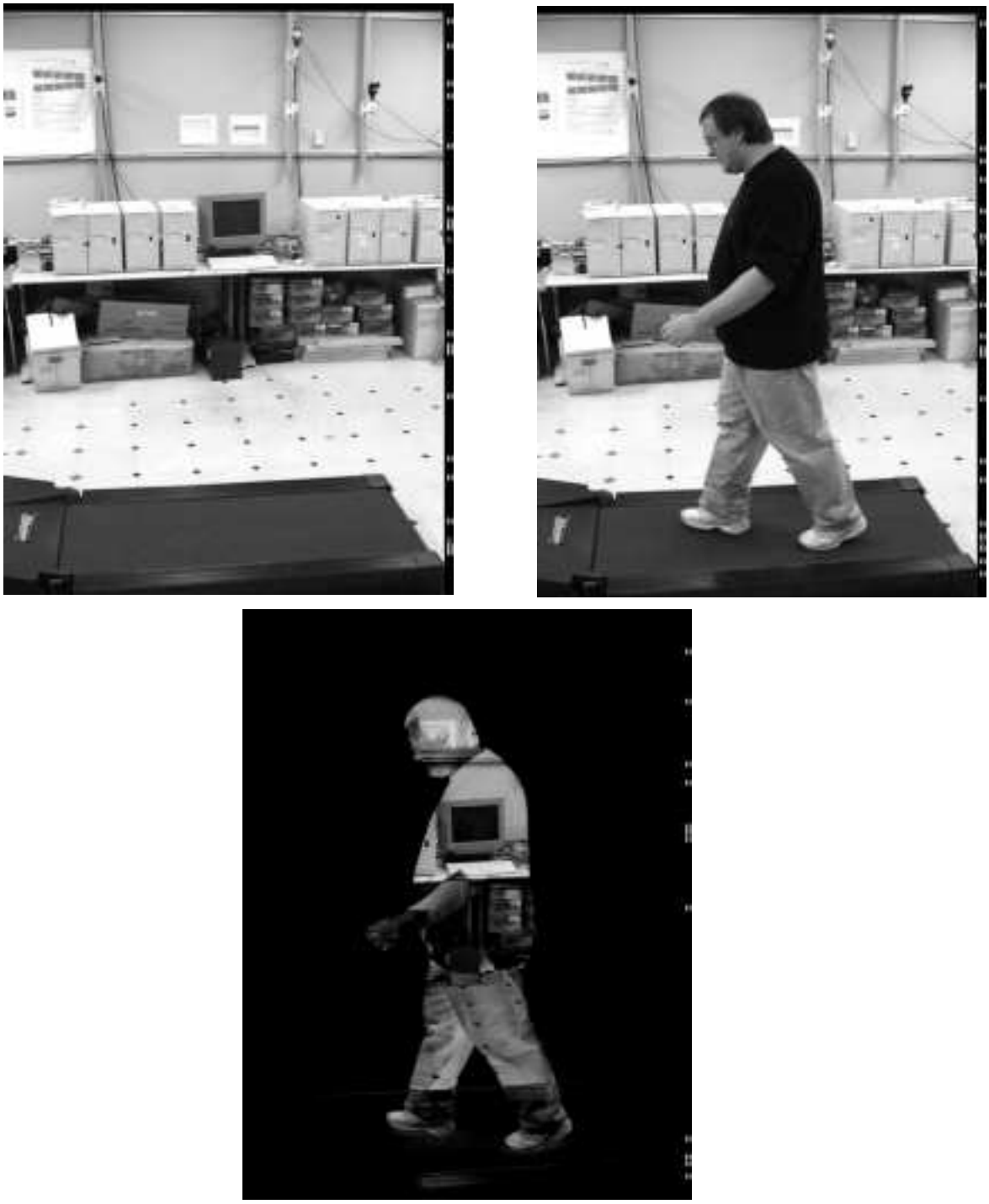


Figure: Results of a database image undergoing background subtraction. (Clockwise from top left: the grayscale background, the grayscaled scene to be analyzed, the output of background subtraction)

Morphological Processing

Simply processing the frame difference between the background and the current frame would be a quick way to figure out where objects occur in the foreground, however blob coloring results would be horrendous because the images are still highly segmented and jagged. Initial image subtraction results from our test program verified these assertions as they contained significant noise in the form of the background interfering with the foreground color. Take for example the sample run from our training set on the previous page- note that the monitor in the background creates a large hole in the center of the man's stomach. Similarly, if the man were in a different position relative to the monitor, he may even be segmented in two different blobs. We would therefore need an additional level of processing which would remove the variability caused by similarities in the background color and the foreground color.

As with any morphological processing, great care must be taken to choose the right structuring element. A significant performance tradeoff occurs with a more complex and/or larger structuring element versus a smaller and/or lesser complex one. A bad structuring element may also *enhance* noise to the point where it could affect results by causing false positives. Another factor we considered in the design of the element was the aspect ratio data, which approximated that the lowest possible ratio of height to width (on average) for a human was 2.2. Therefore, modeling a structuring element along these parameters would assist in recreating the overall human structure. Combining this reasoning with our previous logic, we settled on a 9x9 pixel grid, where the middle four columns were 'on', yielding an element with a height to width ratio of (9 / 4) = 2.25, just barely over our analysis' recommendation.

The actual implementation of the dilation and erosion filters- searching for the maximum and minimum pixels that surround a given pixel and reside in the structural element is a very costly implementation. The code will *always execute* the inner most loops. However, the dilation can be interpreted in a manner that avoids executing the inner most loops only when absolutely necessary. Our dilation algorithm takes every on pixel in the image and superimposes the structuring element around it. Thus, when a pixel is

off, there is no need to iterate through the structuring element to find the maximum value. If a pixel is on, then the maximum value at any points which correspond to neighbors (as specified by the structuring element) will be turned on at some point. Otherwise, if a pixel is off, it cannot possibly contribute to the maximum value to pixels that are its neighbors as defined by the structuring element. In terms of loop structure, the inner loops never occur if a pixel is off. Similarly for the erosion, there is no need to search for a pixel’s neighbors if it is off already, the most that can happen to it is that it is turned off again.

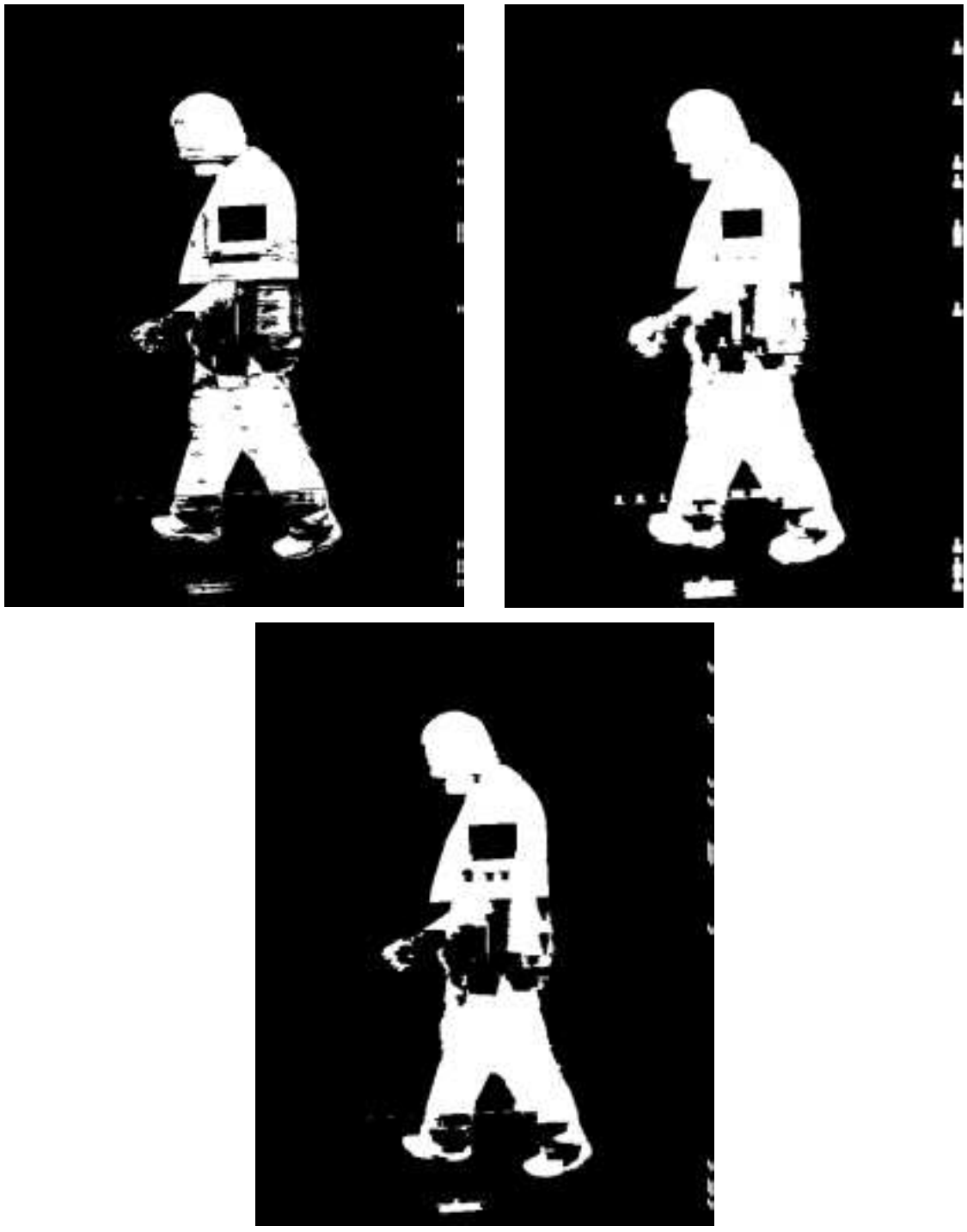


Figure: Results of dilation, erosion on a binary image, with the 4x9 structuring element. (Clockwise from top left: original binary image, dilation result, and erosion result)

Blob Coloring

Now that the image had some consistent structures in it- e.g. ‘blobs’, the next step in the processing of this image was to come up with an efficient method of labeling these clusters of pixels. Each of these blobs could then be evaluated as a potential candidate for a human by our statistical analyzer. Standard blob coloring algorithms involve run length encoding or recursive region growing. Our blob coloring works as follows:

Sum up the number of on pixels for each column and store them in an array called colSum. So colSum[w] corresponds to the number of on-pixels in column w.

1) Traverse through the colSum vector. If an entry (colSum[w]) has enough on-pixels, meaning that it exceeds our threshold for a column sum, then we store the column number w as the start of a possible blob (call it wStart). Once we encounter a colSum[w] less than the threshold, then we store the index number w as end of the blob (call it wEnd). If the difference between wEnd and wStart is below our defined width threshold, we consider it to be too skinny and we then discard that blob. Otherwise, we store its starting point and endpoint in an array of possible blobs that we want to examine (i.e. carry out silhouette analysis) later on.
2) We repeat the steps above until we reach the end of colSum.
3) We come back, this time, examining the horizontal threshold. Also, we only sum up as far as the starting point and the endpoint of the blob stored earlier on from the column analysis. If the resulting blob is thick enough, we box it up, and send its coordinates to segmentation analysis for it to determine if the blob in question is human or nonhuman.

Motion Analysis

Now that we have found a human, we will try to track the movements of the human across sequential frames. We felt that focusing on the relative motion of the center of the bounding box across multiple frames will give us the results we needed. In order to figure out whether or not a person is moving across the frame (i.e. left or right), we compare the x-coordinate of the center of the bounding box in the previous frame with

the x-coordinate of the current frame. If the difference is greater than a certain threshold, we then use the sign of the difference to figure out whether the person moved to the left or to the right. If however the subtraction gave rise to a number that is smaller in magnitude than the threshold, then we compare the y-coordinate values of the two centers (i.e. the center of the bounding box in the previous frame and the center of the bounding box in the current frame). We used the heuristic that if the person is approaching, then the y-coordinate of the center of the bounding box in the current frame will be greater than the y-coordinate of the center of the bounding box in the previous frame. Likewise, if the person is moving away from the camera, then the y-coordinate of the center of the bounding box will decrease between the previous and the current frame.

Formats and Conversion

As part of the preprocessing, it was necessary to convert the input video files from the camera into a series of images. The camera used (Canon Powershot A70) stores videos in the AVI format, but the C++ image library used could only read in Windows Bitmaps. To remedy this, we used Adobe Premiere 6 LE, which had the option of outputting an AVI file to a sequence of Bitmaps. These Bitmaps were then transferred over onto the PC with the EVM for use during our demonstration.

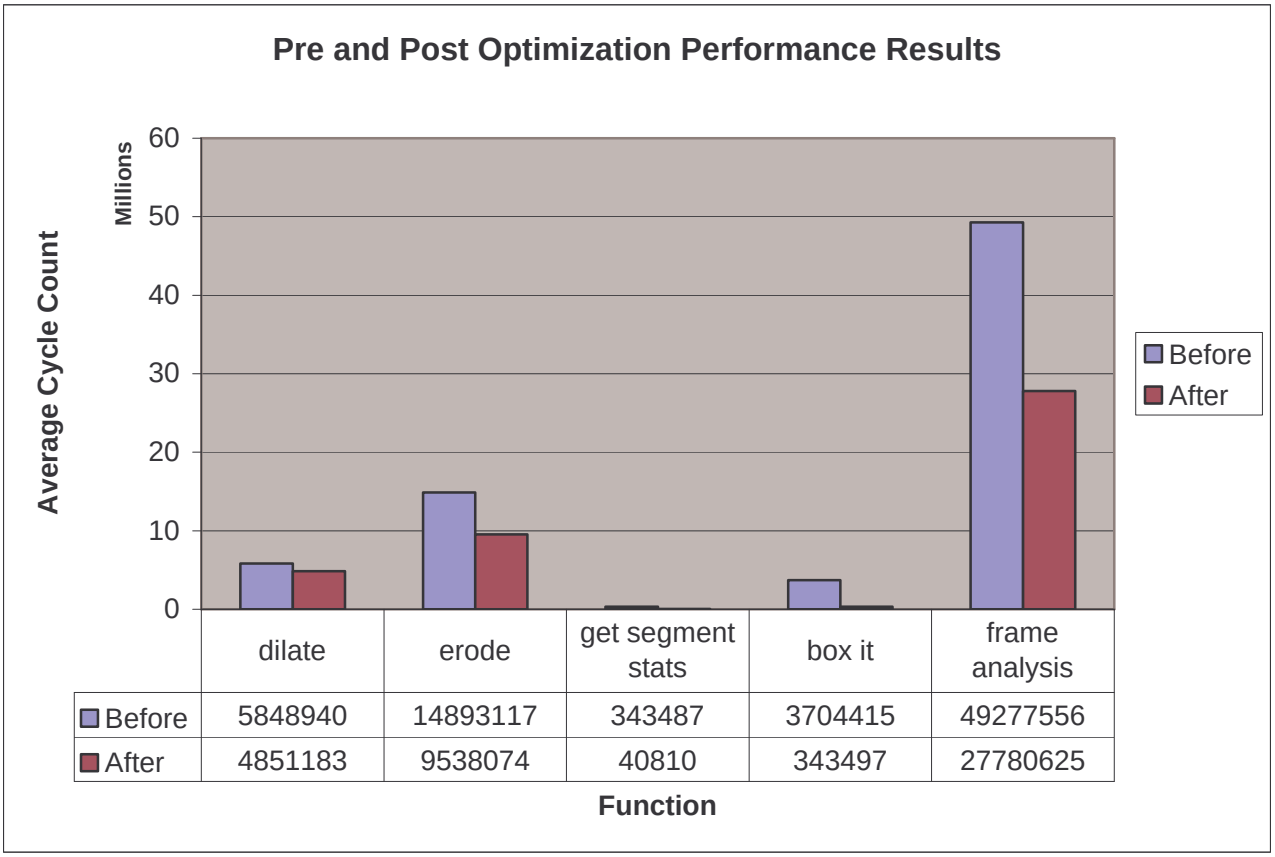
Adobe Premiere also provided the ability of resampling the video, which was necessary because the camera's sampling rate was 30 frames a second. In order to provide some realism and remove unnecessary processing, we decided that 5 frames a second would be sufficient enough for the motion of a human to be tracked reliably. At the same time, we would be able to supply near real time results because the delay from processing would span the gap in frame updates (5 frames/second is equivalent to .2 seconds per frame).

Optimizations

After identifying which algorithms we would use and testing their correctness, we needed to optimize the code. We chose to evaluate our optimization on a few different levels- project/compiler, file, memory, and function. Looking at each in turn allowed us to systematically improve the performance of the code overall.

At the project level, our initial settings were set to only register level. This level of optimization only attempts to reuse registers more efficiently. Under Code Composer we changed the project build options to enable file level optimization ("-O3"), which is the highest level of compiler optimization. Enabling this level of optimization allows register level optimizations as well aggressive loop unrolling (to minimize looping/branching overhead), elimination of global common sub expressions (cutting down on repetitive calculations), and removal of dead and non-effectual code. As shown in the table, changing the level of optimization resulted in the most significant performance benefit. <See figure of overall results below> Another interesting property of the Code

Composer compiler is that the code's execution speed isn't necessarily inversely proportional to code size, especially at the file optimization level. Thus, the fastest performing code is also the smallest, which yields the added benefit of assisting in fitting all of the code onto the EVM's on chip memory. At the memory level, this allows for the fastest possible instruction fetch times, since the code resides the shortest possible distance away from the CPU.



	dilate	erode	get segment stats	box it	frame analysis
Before	5848940	14893117	343487	3704415	49277556
After	4851183	9538074	40810	343497	27780625

Our initial goal was to make the code small enough so that it would fit within the 64-kilobyte limit of the on chip memory. Therefore, the initial focus was away from optimizations. However, once the goal of code size was attained, the next step was to look at the most frequently executed code. We were not surprised to find that our get_pixel was called the most frequently, because the morphological filtering, segment analysis, and most importantly, the blob finding algorithm use it extensively. To remedy

this, we decided to force get_pixel to be inlined. Inlining functions in general tends to increase the overall code size, but because we had ample space left in the on chip code section, we were able to utilize this. This helped reduce the amount of cycles tremendously because get_pixel was almost always being called from within a loop.

The next objective was to optimize the higher level and algorithmic approaches in our code. On a function-by-function basis, a few consistent optimizations were attempted. The first of which was manually unrolling loops which iterated over the entire image, or at other known blocks. For example, in our “blob-coloring” function box_it(), the calculation of column pixel sums was unrolled nine times, and the calculation was forced to accumulate in three different registers. These two techniques combined were used again in the dilate_image() and get_seg_stats() functions.

Specifically, for the image dilation, we made use of the fact that the filter was rectangular, and eliminated unnecessary iterations of the inner loop, and unrolled the rest out. The filter’s structuring element consists of a rectangle standing up, with four columns of on pixels. The loop was organized to access the filter in row major order so that the inner loop could be unrolled to all four of its columns in parallel. Also, repetitive calculations are forced to temporary variable, which improves the chances of the compiler assigning a register to it, thus cutting down on the number of memory accesses.

In terms of the get_seg_stats function, instead of iterating three times in a generic loop for each segment, we chose to unroll them immediately. The memory access pattern across the loop body will therefore remain consistent and prevents the pathological case of memory accesses to each segment where one segment’s data overwrites another segment’s data in the cache.

Finally, the last technique used was the parallelization of calculations. This was realized in the ‘blob coloring’ function box_it. Recall that box_it functions by initially calculating every column’s total on pixels, which are then used to find continuous runs of on pixels

horizontally. Dividing the task of summing between three registers allows for parallelization because previous cycle's calculations will be relatively independent of the current calculations. This in turn also takes advantage of the Very Long Instruction Word architecture that the EVM has adopted because more instructions can be fed into the EVM's ALU's for *concurrent* processing, which in this case is the accruing of an on pixel count.

Analysis of Results

To test our human detection algorithms, we chose the following videos for testing purposes:

The video MechE_lab1 was used, which depicts a person walking from right to left, and then back. We used it to see verify that our algorithm could detect a human walking across the camera. The major issue with this video was the noise created by the reflections in the metal tools hanging on the wall behind the human. We found that about 72% of the time, it was able to identify and track the human. The most common cases of false negatives occurred when the person made a large stride in a direction. This stride caused the bounding box to stretch in width, causing it to almost become a square (since the height decreases as well). As a result, it fails the aspect ratio test and the algorithm determines that the blob is non-human. Another interesting complication is when the person is about to leave the scene, that is, when he is around the edge of the frame. It is possible for a blob to be found at the edge of a scene (possibly on a human) and in some rare cases, the aspect ratio falls within the range for humans, and the mean tests for all three sections pass as well. In these cases, the algorithm will return that a human blob was identified, even though it does not box up the entire human figure. Other times, however, it will see that it's only half of a human figure if it *is* a human, so it would output nonhuman because half of the body's mass is inside the frame and the rest is outside while the person is walking side to side. In this case, then, mean test fails, and is usually due to the other leg not being present to balance out those values in the last segment (the bottom third of the human).

In the video MechE_lab4, the person is still walking across the frame with the same background. However, this time, he is wearing a very light shirt. In fact, it was so light that it almost blends in with the grayish background. The result of this, unfortunately, was not so good, but is expected because the difference between the background and the foreground falls below the threshold. For this video the success rate was only 43%. Most of the time, the only part of the body that was detected was the person's head, which was relatively darker than the rest of the body. When only the head was detected, it would immediately fail the aspect ratio metric. In this case, the width of the bounding box is correct, but our algorithm does not scan all the way down the human if it encounters a row in the bounding box that is below the threshold. Although stopping the vertical scanning for blobs can hinder the detection of humans, it can also produce more accurate silhouettes because it limits out the effects of shadows or other blobs that are near the human, as is the case in the next video.

In the beginning of MechE_lab6, a person is pushing a stool across the frame, but later stands up and walks towards the camera. We constructed this odd scenario, because according to our algorithm, we wanted to check our prediction that it would not identify the human and the stool as being human *together as one blob*. Indeed, as we thought, it did not see the person and the stool as a human. Both the aspect ratio and the means test failed the required range. However, when the person pushes the stool away and begins to stand up, the output does indicate that a human is detected, and it also verifies that the direction of the person is towards the camera.

The last video taken in the Mechanical engineering lab was MechE_lab2. This video shows someone sweeping the floor with a broom. In this scenario our algorithm is able to separate the bottom of the broom and the human as *two* separate blobs, and identifies each correctly- the man as a human and the broom as a nonhuman. Because of the nature of the action of sweeping the floor, the bounding boxes vary very widely in ratio- sometimes the person is standing straight up and at others he bending and reaching out. Nevertheless, we found a success rate of 87% for this particular test. Unless the person bends down or reaches for something in an extreme manner, which

severely messes up the aspect ratio, it will generally detect the object as human, even though he had a broom in his hands. The broom itself does not pass any of the tests, so it is being correctly classified as a nonhuman as it should be.

Another interesting video, called Steps, was taken in front of HH. Here we tested out a scene in which more than one person is present. Another intriguiging aspect of this film, was that this scene was outdoors in daylight- a highly variable environment. At the point of peak activity in the film, there are 5 people in the scene. Over time, person A walks obliquely in front of the camera back and forth. For most of the time, it can accurately identify the human. Person G also walks about in front of the camera. Here the success rate is 75%. However, a brick wall takes up some room in front of the camera. So when G walks up too close to the wall, his leg disappears underneath that wall and his aspect ratio is thrown off, which immediately comes back as nonhuman. However, once he gets to a point where enough of his legs show from behind the wall, we are able to correctly classify him as human. Person S wears a striped shirt, but is wearing a jacket and a pair of pants that have almost exactly the same color as the background. For the time that is a part of the scene, he does get identified as being an object, but not as a human. It is not until he moves close to the camera that the system can classify him as human. Finally, there are two other people very close together in the background who exit Hammerschlag's front entrance (which also changes the background scene for a few frames). When they first appear, the system is able to detect them as objects, but cannot identify them asbe human. After approximately 6 frames, the two gets boxed together as one, because they are about 20 to 30 feet away from the camera and are at a higher elevation (near the top of the stairs). These results are expected because they reside in the same columns as the doors swinging shut behind them. As stated before this does represent a change in the background and causes the probability that those columns are examined for blobs to increase.

In the Ammar video, a person is walking straight up to the camera, turns and then walks away. Here, the success rate is 89%. Here the threshold that is used to determine if a pixel is on or off (so that the image from background subtraction can be converted to a

binary image) gets lowered to 15. This is because the contrast between the background the new scenes is smaller. This shouldn't be too big of a deal, because we could write a program that analyzes the best threshold given a scene and its original background. The results, however, does show that the algorithm can handle someone who is approaching or leaving the camera.

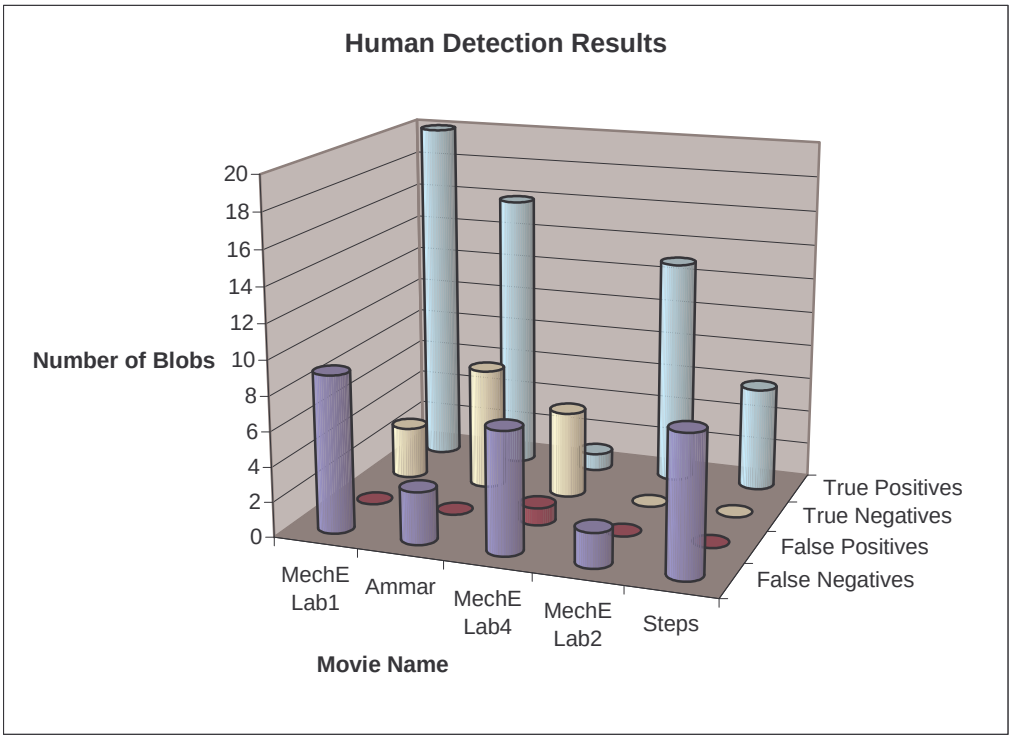
In the Box video, there is a moving box that is pushed by a person whom does not show up in the video. This is used to prove that if there is nonhuman object present in a scene, the system can accurately identify it as nonhuman. We have a 100% success rate for this video.

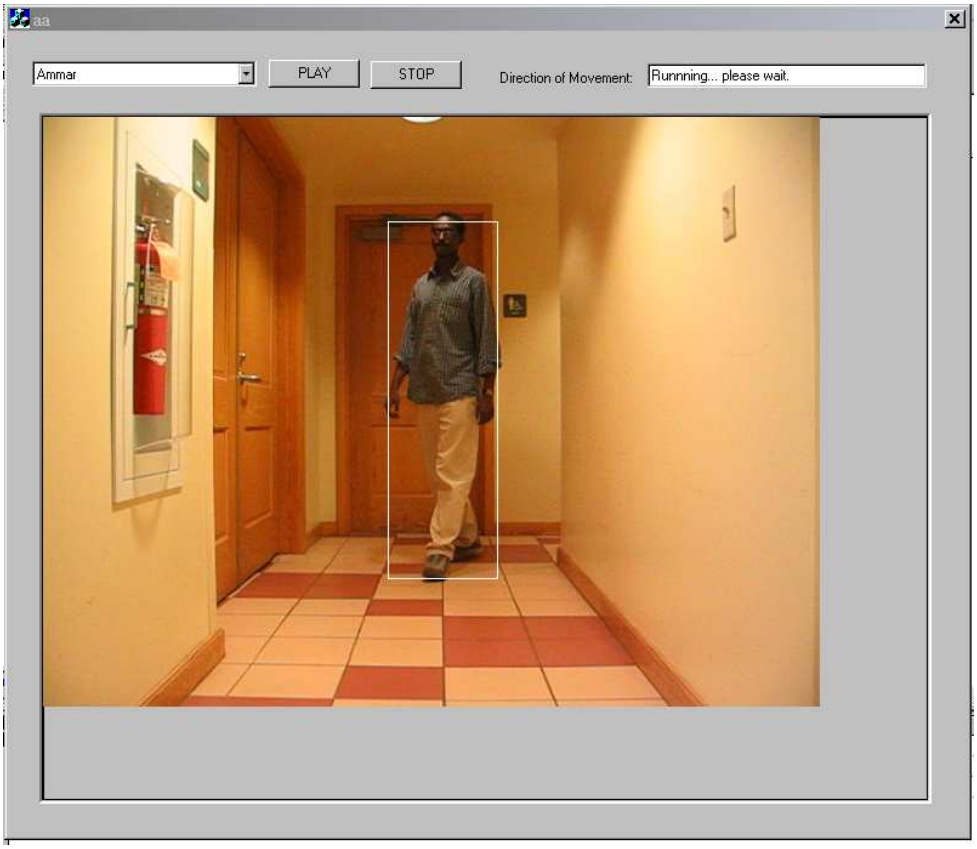
Finally, in the Mona_Smirk video, there is someone who is initially moving across the camera, but then starts approaching. Even though the system can lose track of where the human is in some frames, it does, overall, conclude that the human is moving across and then approaching the camera.

So overall, to conclude the most common problems we had in identifying human were the following:

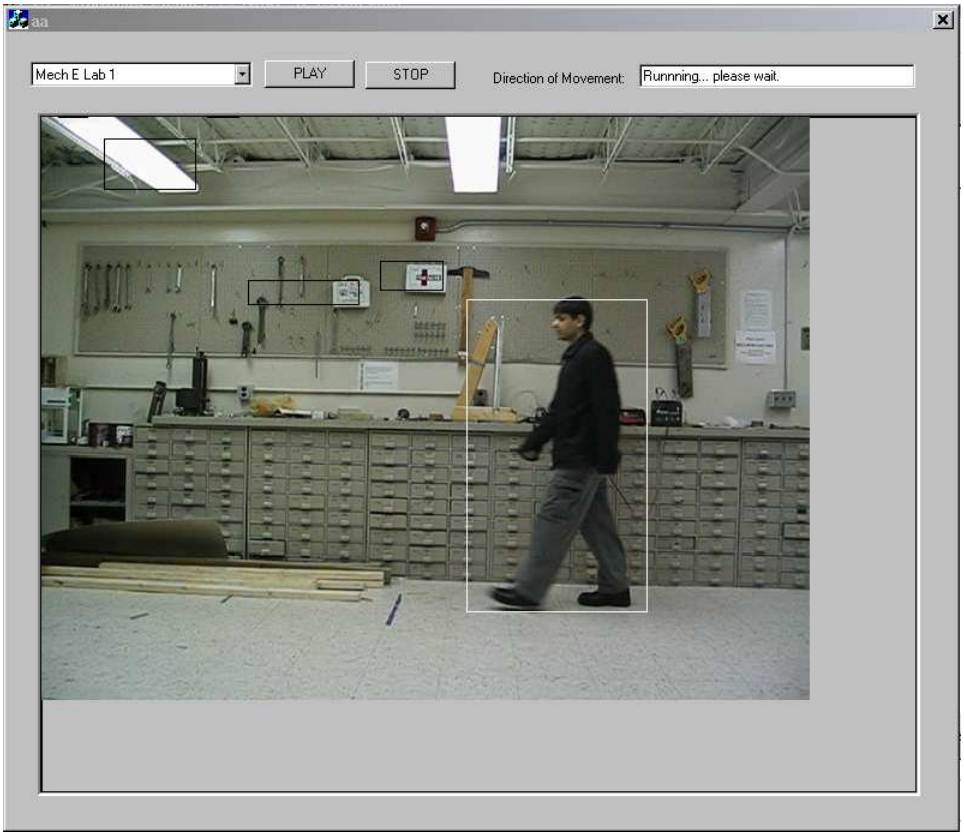
1) If the figure has a small contrast in respect to the background, the system will not accurately define the object as human
2) If the figure gets cut off, the system will not see it as human.
3) If the person is in a pose such that it fails the range of the aspect ratio, the system will obviously not see it as human.

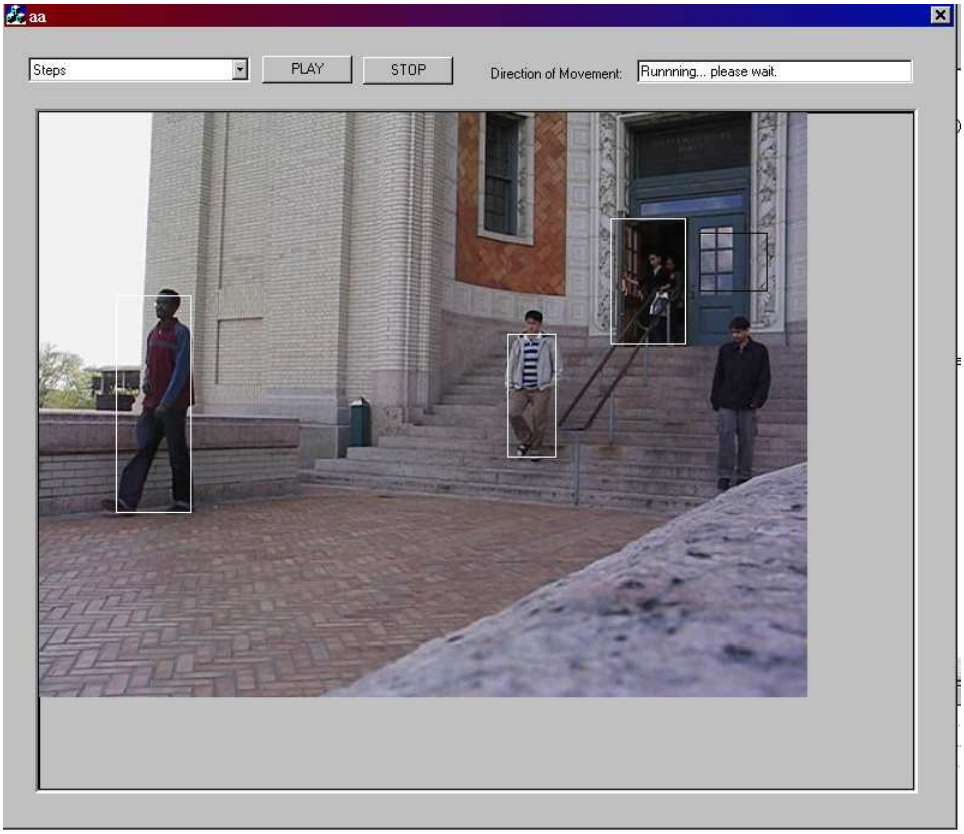
In summary the results over these videos can be summarized as follows (we omitted the video with the chair since we didn't know whether the human and chair should desirably be classified as a human or nonhuman):

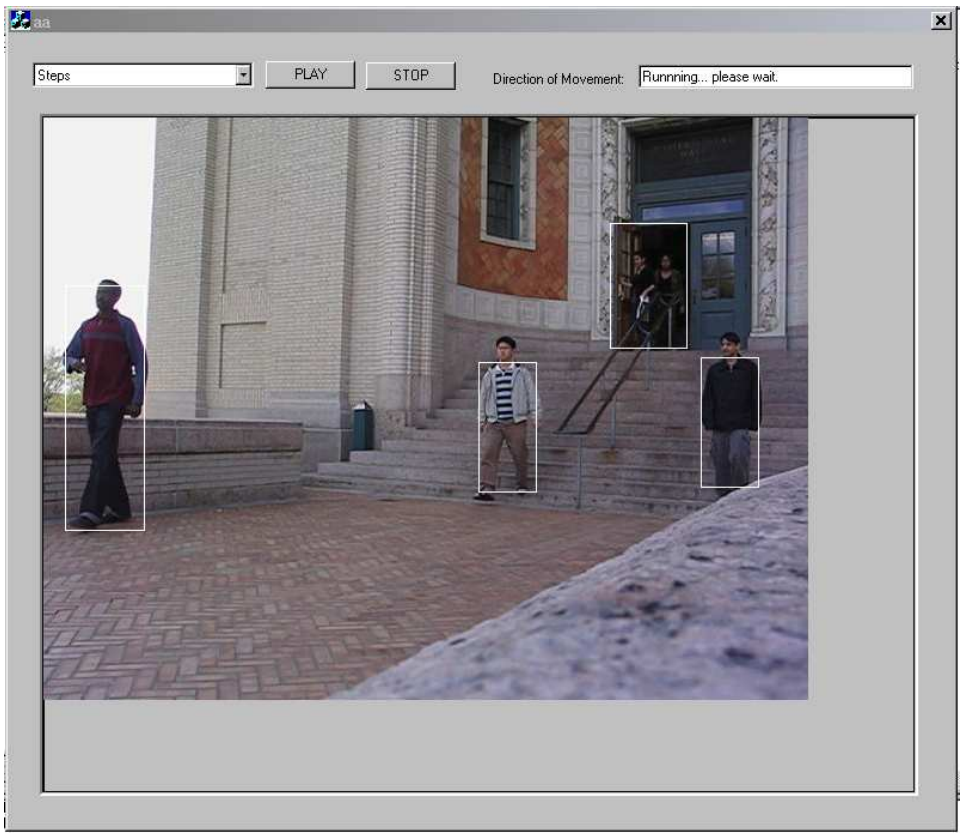


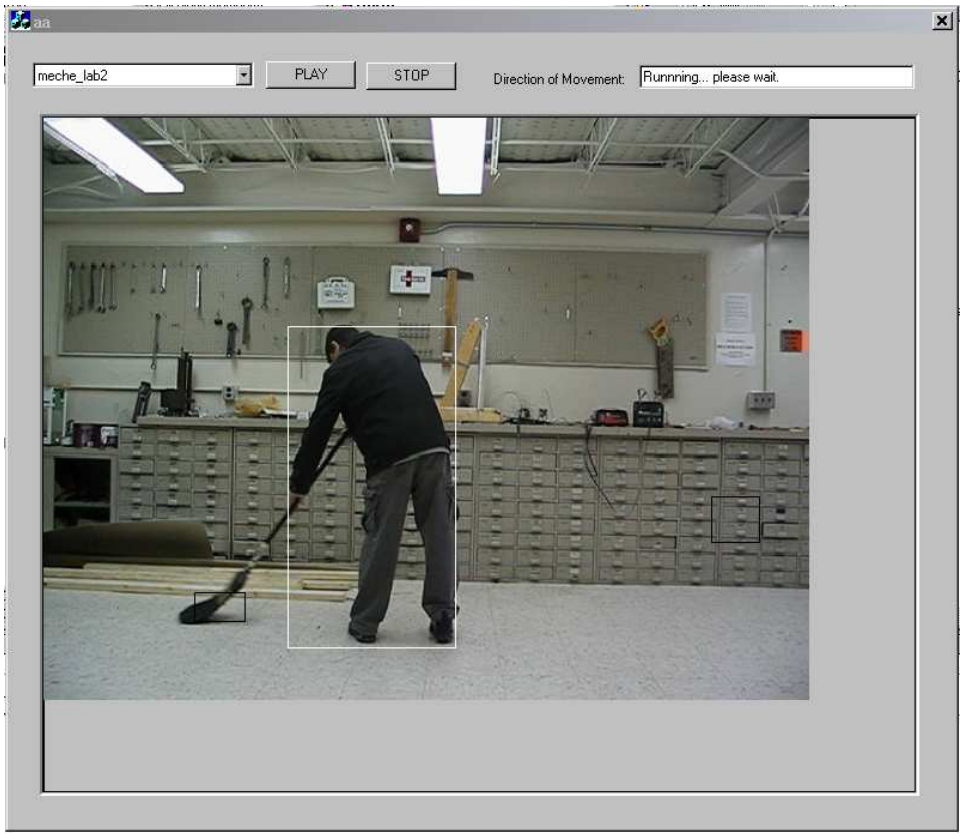

aa
Ammar
PLAY
STOP
Direction of Movement:
Runnning... please wait.

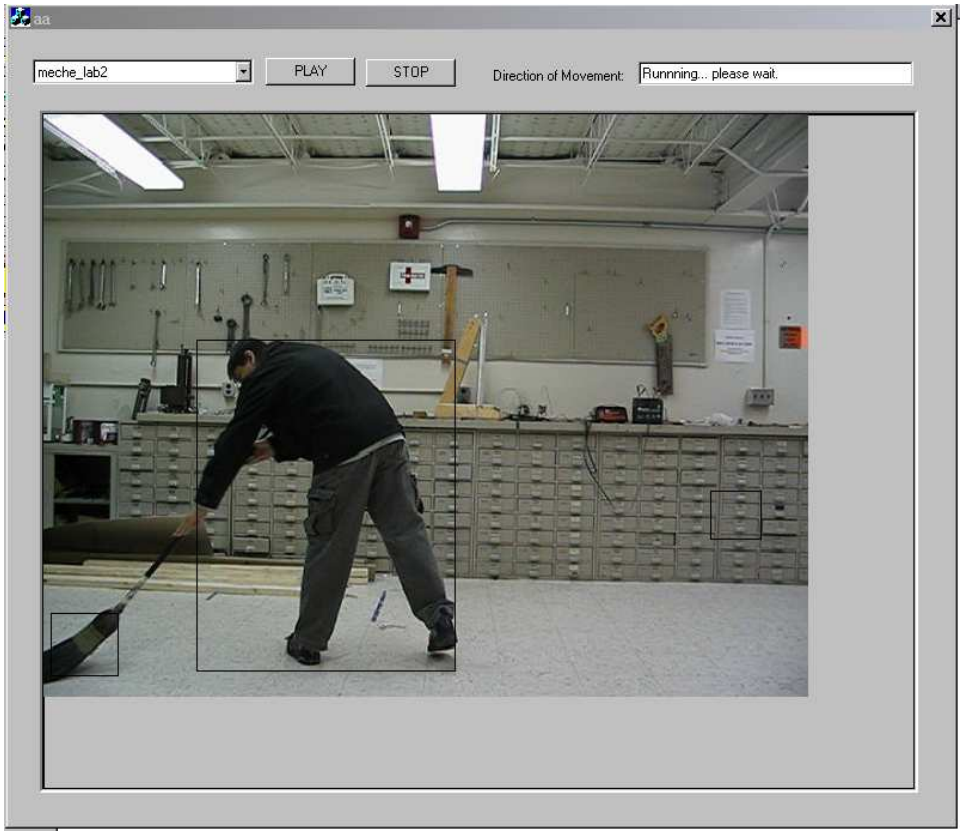

aa
Ammar
PLAY
STOP
Direction of Movement:
Runnning... please wait.

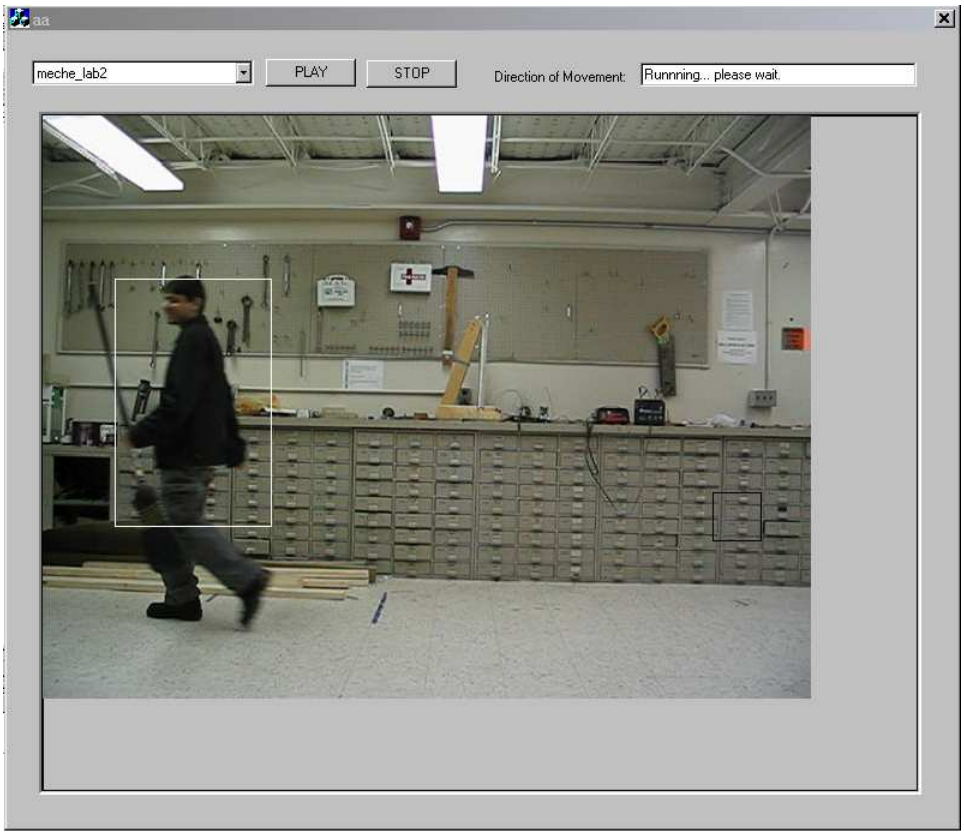

aa
Mech E Lab 1
PLAY
STOP
Direction of Movement:
Runnning... please wait.

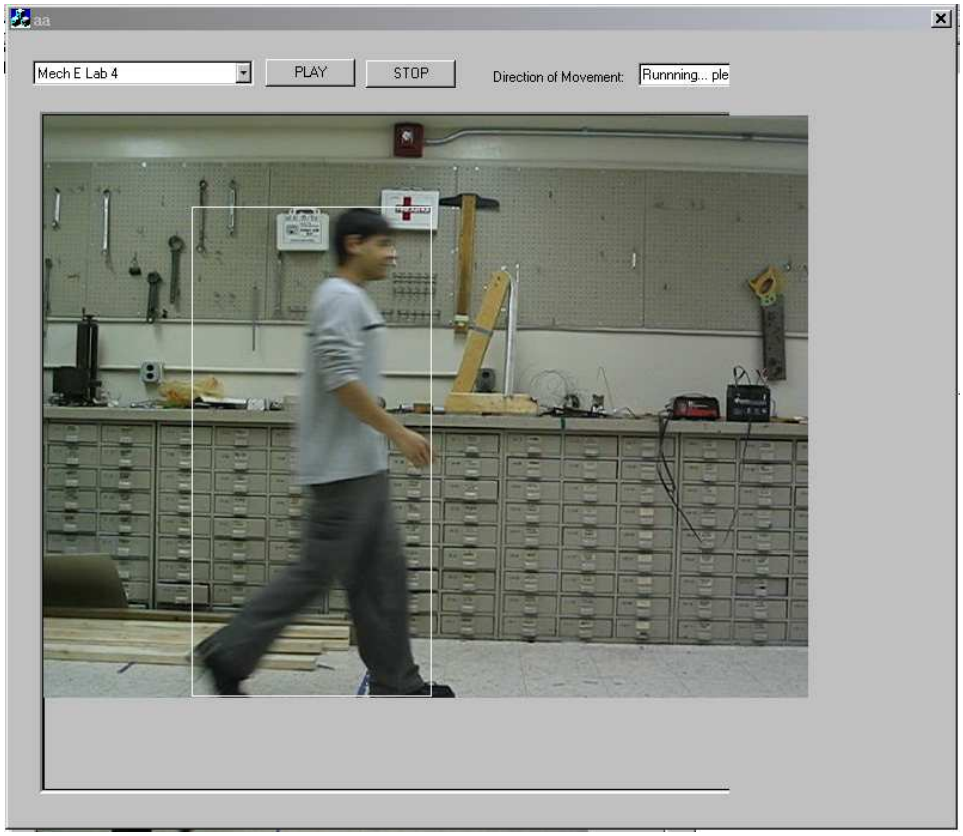

aa
Steps
PLAY
STOP
Direction of Movement:
Runnning... please wait.

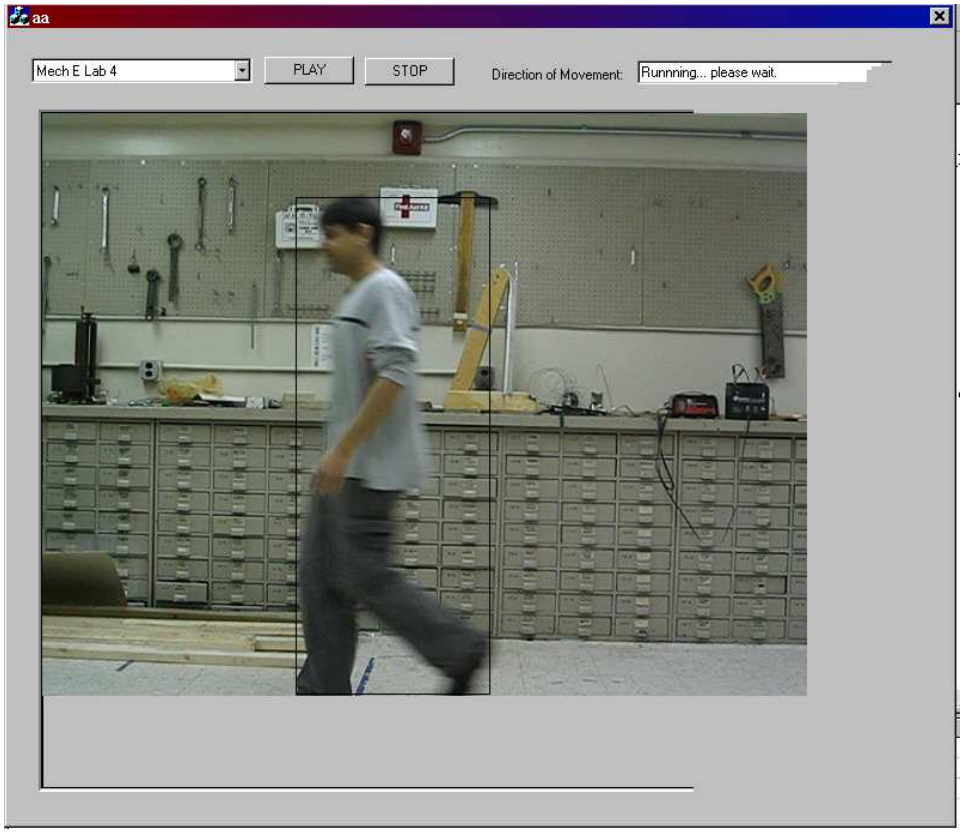

aa
Steps
PLAY
STOP
Direction of Movement:
Runnning... please wait.

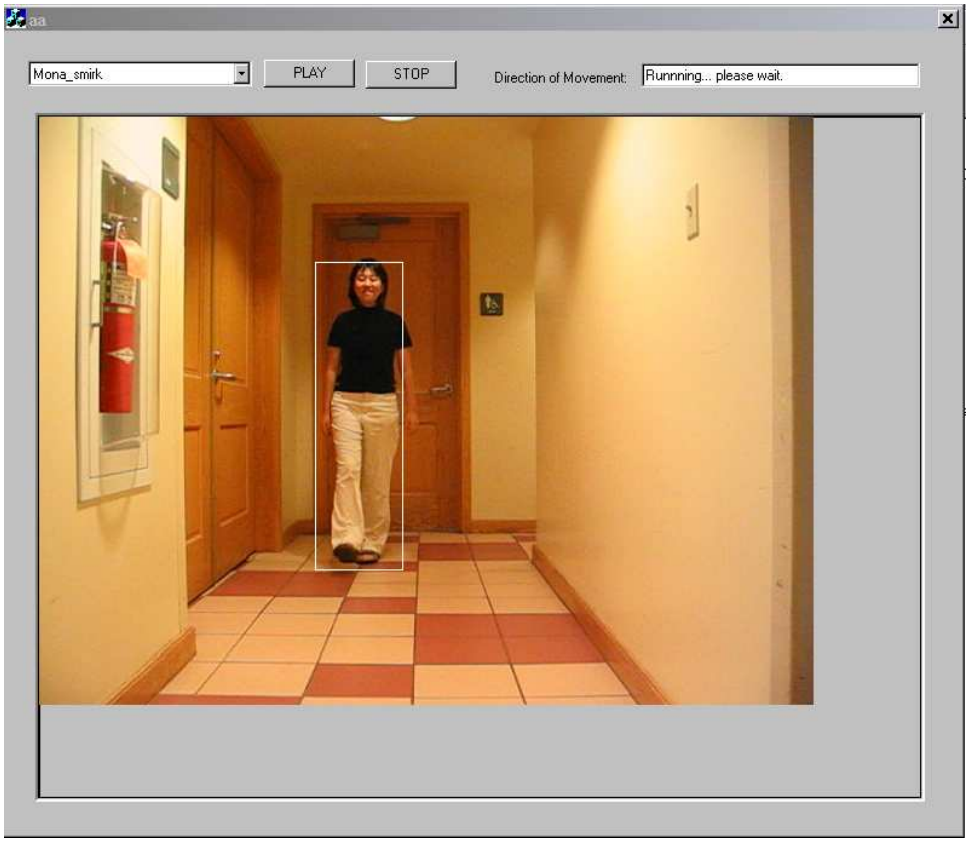

aa
meche_lab2
PLAY
STOP
Direction of Movement:
Runnning... please wait.

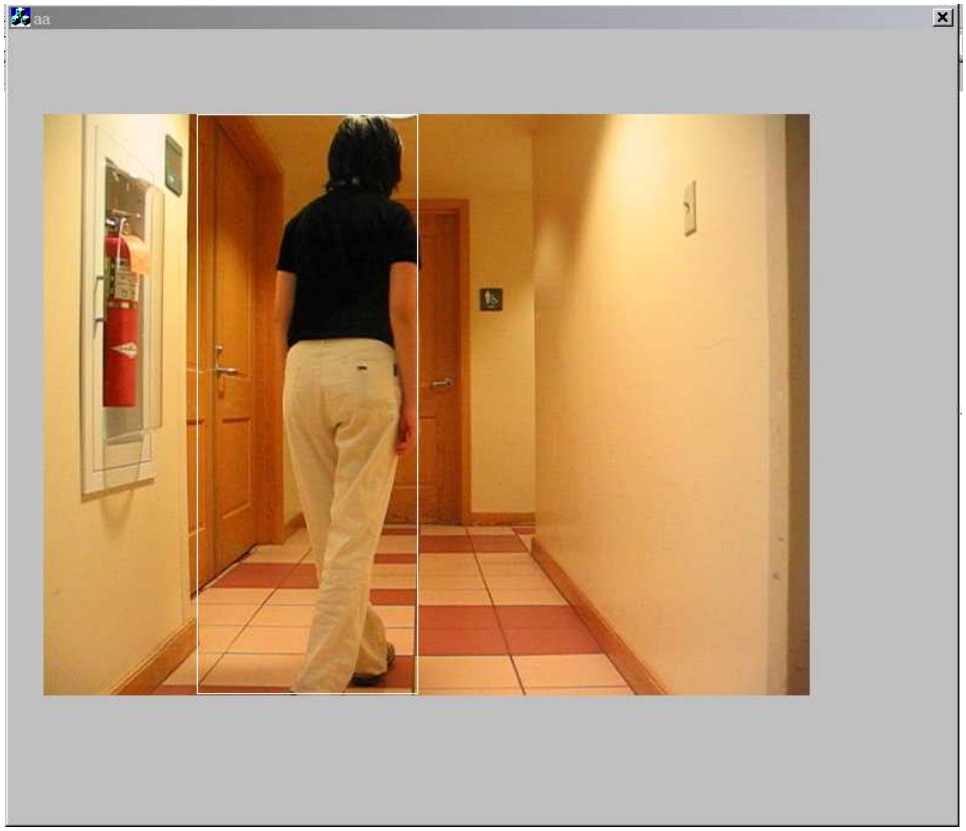

aa
meche_lab2
PLAY
STOP
Direction of Movement:
Runnning... please wait.


aa
meche_lab2
PLAY
STOP
Direction of Movement:
Runnning... please wait.


aa
Mech E Lab 4
PLAY
STOP
Direction of Movement:
Runnning... ple

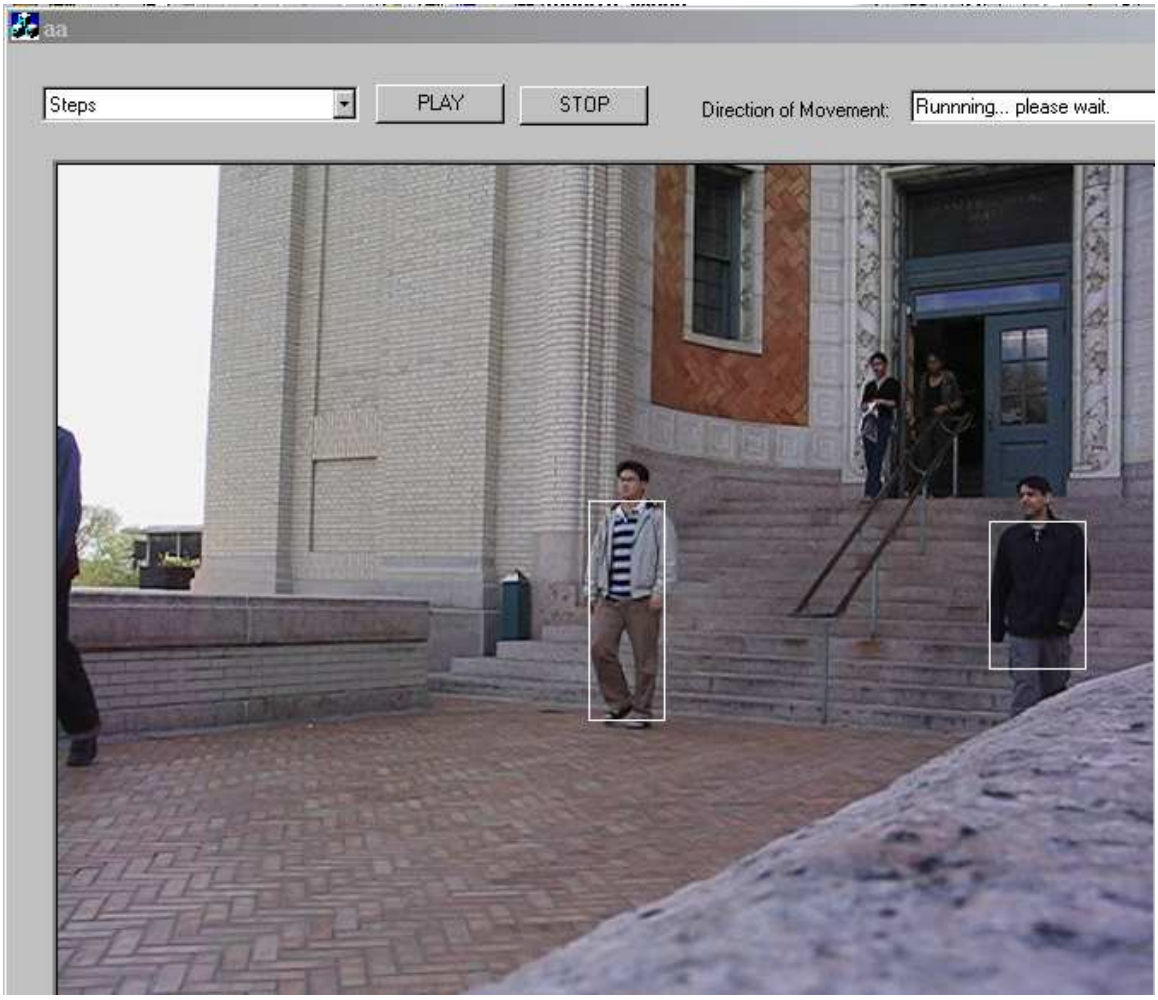

aa
Mech E Lab 4
PLAY
STOP
Direction of Movement:
Runnning... please wait.


aa
Mona_smirk
PLAY
STOP
Direction of Movement:
Runnning... please wait.


aa

GUI (Demonstration System)

For our demo, we used Visual C++ to create a user-friendly GUI that using a drop-down box enables the user to pick the movie to be analyzed. After highlighting one of the selections, the user presses the “play” key. This loads the relevant .bmp files (already extracted from the AVI). Once the loading is done, and after the user clicks OK on a couple more message boxes the modified .bmps will then be displayed as shown below:



white boxes – indicate that the circumscribed silhouette is that of a human
black boxes - indicate that the object detected is not human

Acknowledgements

We would like to thank Prof. Casasent for his valuable feedback, that made things flow much more smoothly. We would also like to thank Ralph Gross for generously sharing his MOBODatabase with us.

References

Algorithm References

Automated Detection of Human for Visual Surveillance System

Authors: Kuno, Watanabe, Shimosakoda and Nagakawa
-1996 IEEE Proceedings of ICPR '96

Silhouette-based Human Identification from Body Shape and Gait

Authors: R. Gross, R. Collins and J. Shi
- *Int'l Conference on Face and Gesture, October 2002*

MATLAB References

http://www.mathworks.com/company/events/archived_webinars.jsp

We found the above website very helpful as it showed us how to carry out different types of morphological processing on images using MATLAB. We adapted our solution to closely recreate the results of such morphological processing.

C References

Code for GUI (Visual C++)

All of the following are links to websites that have sample code

http://www.coding-zone.co.uk/cpp/articles/140101imgproc.shtml
http://www.codeguru.com/Cpp/G-M/bitmap/displayingandsizing/article/php/c4905
http://forums.devshed.com/archive/t-74626