PERSON DETECTION AND TRACKING USING BINOCULAR LUCAS-KANADE FEATURE TRACKING AND K-MEANS CLUSTERING

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Electrical Engineering

by
Christopher Thomas Dunkel
August 2008

Accepted by:
Dr. Stanley Birchfield, Committee Chair
Dr. Adam Hoover
Dr. Richard Brooks

Abstract

In this thesis, we present the design and implementation of a method for real-time person detection and tracking. Many current methods for detecting and tracking people rely on color contrast or movement to segment the image. Using color, however, requires the target and the background to be significantly different, and motion segmentation requires the target to be in constant motion relative to the background, often requiring stationary cameras. Pattern detection methods have also been applied to the problem of detecting pedestrians, but these approaches are slower and require stationary cameras to function. The method we present in this work does not require a color difference or constant motion to operate. We use Lucas-Kanade features to track feature points between left and right images, producing a sparse disparity map which is then segmented through the application of k-means clustering. We apply a Viola-Jones face detector to determine which, if any, of the resulting feature clusters represent a trackable person. This algorithm is tested using two identical standard cameras mounted on a mobile robot platform. Results are presented demonstrating detection and tracking of a person in several different situations, including partial occlusion and self-occlusion.

Dedication

For the future, and the promise it holds.

Acknowledgments

I would like to acknowledge the guidance, encouragement, and especially patience of my adviser, Dr. Stanley Birchfield, without whom I would not have been able to complete this work.

I would also like to extend great appreciation to my committee, Dr. Adam Hoover and Dr. Richard Brooks, for their time and consideration.

Special thanks goes to my good friend and fellow student Jason Schwier for his great programming and editing suggestions, as well as to Lisa Fuller for agreeing to help with the experiments.

Finally, I would like to thank my parents for all their encouragement, and for helping me to believe that I can accomplish anything.

Table of Contents

List of Figures

Chapter 1

Introduction

From as far back as ancient times, mankind has attempted to mimic nature using technology. This is especially true of robots and intelligent systems. There are descriptions of fantastic machines built by early engineers that could mimic human behavior, even to the extent of playing musical instruments [12]. However, these machines were nothing more than wood, metal, and paint. They had no sensors to provide feedback about the outside world and they could not provide the true level of interaction that people have desired from a robotic companion. It was not until modern times that the technology necessary to provide this kind of interaction has been available. Even so, we are far from creating the robots of our dreams, but new research is being performed every day to make that fantasy become a reality.

As research into robotics (and specifically image processing) has continued, it has become apparent that there are many low-level tasks that humans accomplish automatically that are very difficult for artificial systems to reproduce. One of these tasks is person detection. From the time we are infants, our minds are trained to recognize people and faces. This has been offered as one reason why humans are so adept at seeing faces in nature. Occurrences such as the man in the moon or the face on Mars can attest to this. To a computer, however, a face or a person is no different than any other collection of pixels. The purpose of this thesis is to outline an approach to the problem of person detection and tracking in real time. A successful implementation of this technology would open up a wide variety of tasks to robots and intelligent systems. For instance, a robot equipped with person detection could more easily plot a course through a crowded warehouse or factory. Person detection could allow robots to track a single person, following them with heavy equipment or with documents or tools needed by the individual. Such a robot could even serve as a tireless security guard, patrolling hallways for intruders. As person tracking technology becomes more widespread it will become faster, cheaper, and easier to implement, opening the doors to new levels of intelligent systems.

1.1 A Brief History of Person Detection

The idea of creating a method for artificial systems to detect and track people, specifically using image processing, is not new. This section discusses a few of the approaches that have been used to accomplish this goal. These include color segmentation, optical flow segmentation, dense stereo matching, and systems that use some combination of various methods.

1.1.1 Color Segmentation

The concept behind color segmentation is a simple one: use the color or other appearance information within the image frame to separate various objects. The task of implementing this idea, however, is more complex. One approach, used by Sidenbladh et al. [21], involves using binary skin classification to locate faces within the image. A closed-loop controller is then used to maintain tracking of the person by cameras mounted on a mobile robot. A more statistical approach comes from Tarokh and Ferrari [22], who use clothing color to segment the image. Statistical analysis is then used to determine which color blobs belong to a person and which do not. A more sophisticated approach is discussed by Kwon et al. [14], who use color histograms to locate a person in a pair of stereo images followed by triangulation between the images to find the distance between the camera and the tracked person. Schlegel et al. [20] also use color histograms, however they combine the histogram data with an edge-based detector to improve robustness of the algorithm at the expense of computation time.

Color segmentation methods work well in many situations. By requiring only a small amount of information from each frame in order to function, the algorithms remain fast and efficient. Unfortunately, this efficiency comes at a price. For instance, color-based methods usually require that the person to be tracked wear a different color from the background or from other people in the scene. Most face-based trackers demand that the person face the camera during the entire tracking procedure. All of these techniques are vulnerable to changes in lighting, as different lighting can drastically change the camera's perception of color.

1.1.2 Optical Flow

Optical flow is another common approach to person tracking that avoids many of the problems faced by color segmentation. The basic optical flow approach is that features in the frame are tracked over time to determine the relative velocity of objects within the scene. Another method is then applied to this data to locate or track a person. Piaggio et al. [17] approaches the problem by thresholding the velocity data and assuming that the person moves differently from the background. Chivilò et al. [6], on the other hand, views any velocity changes as a disturbance to be minimized by regulation (i.e. moving the camera or the robot to minimize the relative motion of the person, thereby providing tracking).

By not relying on color information, optical flow does not have to deal with the problems of similar target and background colors, or lighting changes in the environment. Optical flow approaches, however, are subject to drift as the person moves within the frame and may have issues with out-of-plane rotation and movement. Also, because the models are velocity-based, any sudden changes in velocity within the scene, such as travel over rough terrain, could cause issues with the tracking of the person by introducing a spike in erroneous velocity values. Optical flow could also encounter issues when dealing with scenes with multiple moving objects, as this would make the velocity map much more complex.

1.1.3 Dense Stereo Matching

The idea of dense stereo matching is based on the way in which humans perceive depth in a scene. This approach, used by Beymer and Konolige [3], tracks dense features in a pair of stereo frames to construct a 2D plane model of objects within the scene. Using odometry information, the motion of the background is estimated, allowing background objects to be subtracted from the image. The assumption is that the person is what remains after subtraction. A Kalman filter is applied to track the person after this initial detection.

As with the previous methods discussed, stereo matching bypasses many of the problems of both color segmentation and optical flow. The system does not use color information, and therefore does not require the user to wear anything special in order to be tracked. By not using velocity, the problem of multiple moving objects, or drift due to robot motion, is manageable. Tracking the disparity of the scene allows the system to differentiate between near and distant objects. Background subtraction using robot odometry, however, could fall prey to slippage of the wheel encoders, and the assumption that the person and background have different motion models could make it difficult for the system to detect a stationary person.

1.1.4 Pattern Detection

Pattern detection is very popular in the field of pedestrian detection. The process works by training a pattern classifier on several samples of known people. The classifier then searches the live images for similar patterns and flags those that match the classifier's model of a person. Viola et al. [26] present a pedestrian detection system that makes use of both pattern and motion information. This approach is based on the work of Viola and Jones [25] on detecting faces. Several classifiers, each based on a variation of a Haar wavelet, are trained using test sequences of pedestrians. AdaBoost is used to combine the classifiers into a strong classifier for both the pattern and motion of pedestrians. These classifiers are then applied to regions of live images and detect the person using only motion and intensity information.

Much success has been achieved with pattern detection. The algorithm presented above is able to reliably detect pedestrians using very low resolution images (as small as 20×15 pixels) in poor-visibility conditions such as rain or snow. However, pattern detection tends to be slow (≈ 4 frames per second) and requires the cameras to be stationary.

1.1.5 Other Approaches

Of course, many attempts to solve this problem do not fall neatly into the categories above. Ran and Zheng [18] use motion estimation on a sub-pixel level, not to detect a person directly, but to increase the speed of computing a dense correspondence between two stereo images. Their algorithm produces excellent results in a variety of lighting conditions and can differentiate between multiple people. The system, however, must be implemented using stationary cameras. Based on the stated results, any movement by the cameras could destroy the program's ability to perform background subtraction. Furthermore, the system relies on motion to segment the image, and thus the target must move constantly.

Taking 3D modeling to an extreme, Munkelt et al. [16] attempt to detect people in a video by calculating correspondence between features in the image and a 3D model of the internal structure of the target. They present a complex, articulated model of a person and show how it can be matched to a person in an image using color and 3D information. 3D scene information is provided by either a stereo camera setup, or through matching basic shapes in the image with basic shapes in the model and determining depth and relative position information from this.

Although this approach is ambitious in its scope, there are many tasks that a faster, simpler algorithm might perform better. For instance, there is no mention of this algorithm being able to run in real time. The person must also wear special targets in order to fit the person model, and it is doubtful if the system could deal with occlusion or multiple people in close proximity.

Finally, one of the more novel approaches found is that of Inamura et al. [10]. Their paper presents a method of person detection and tracking that combines motion detection, edge detection, and color detection with a voice recognition system. The robot and the user actually speak to one another, with the robot requesting data and using the responses to customize its method of searching. The system uses binocular vision to estimate the distance to the person using Extended Zero Disparity Filtering (EZDF) [19].

Although certainly novel, this method seems cumbersome. The process of telling the robot what you are currently wearing and what you are doing each and every time you enter the frame could quickly become tiresome. There is also no way for this robot to deal with a situation in which people are simply passive objects to be avoided rather than active participants in the tracking process. Relying on color and edge-based methods of tracking the person, the algorithm is also susceptible to the same issues as all other color-based methods.

1.2 A Hybrid Approach to Person Detection

Our approach to the problem of person detection and tracking is an attempt to mitigate many of the issues discussed above in a real-time system. Our system is heavily motivated by the work of Chen and Birchfield [5] and combines elements of dense stereo matching and optical flow to increase the robustness of the algorithm. Implemented on a stereo camera platform, our system uses sparse Lucas-Kanade features [15, 23] to develop a quick disparity map of the scene, which is then segmented using a data-clustering method. The person is detected using a combination of the feature clusters resulting from segmentation and the output of a Viola-Jones face detector.

To begin, a pair of stereo images are recorded. Lucas-Kanade feature points are selected uniformly on the left image. These are tracked to the right image to give us a sparse disparity map of the scene. Consistency checks are used to eliminate points that do not track properly. The feature tracker is also used to track points forward in the frame sequence, giving us continuity from frame to frame. The movement of each feature from one frame to the next is stored as delta values to be used later.

Once the tracking process is complete, the location of each feature point is known in six dimensions: the position (x and y) the disparity (d) and the change in each of those values from the last frame to the current frame ($\Delta x, \Delta y$, and Δd). The list of all valid feature points is segmented using an implementation of the k-means clustering algorithm. The k-means algorithm minimizes intra-cluster variance through a simple iterative algorithm.

The first step is to determine the number of clusters into which the data will be sorted. An initial mean value for each of these clusters must then be chosen. This may either be random, or through some process decided by the user. Secondly, the distance between each feature point and each mean value is calculated. The feature point is “assigned” to the cluster whose mean is the closest. Once all feature points have been assigned in this fashion, the mean values are recalculated to match the actual mean of the points in the given cluster. The last two steps are repeated until no feature points change cluster assignments, or the movement of the means falls below some predefined threshold.

The k-means algorithm provides us with a data set in which each feature point has been segmented into a different cluster, each with a unique set of attributes. At this stage, the Viola-Jones face detector [24] is run on the stereo frame pair. If a face is detected in both the left and right frames of the image, the algorithm declares that a face has been detected and the face’s location in x and y coordinates, as well as its disparity d, are stored.

The determination of whether a person has been detected is made using information from both the k-means algorithm and the face detector. Each cluster found by k-means is tested against a set of criteria based on the size and location of the face found by the face detector. The criteria involve the cluster having a certain relative location and disparity to the detected face. If a cluster passes all given criteria, then it is declared to be a person and the system enters tracking mode.

Tracking the person, although still challenging, is simpler than detecting the person. For example, the face is required to detect a person but not to track. After a person is found, but before the system enters tracking, any points that do not meet certain criteria are removed from consideration, allowing the tracker to deal only with essential features. These features are tracked by the Lucas-Kanade feature tracker, with the person cluster attributes updated by the k-means algorithm. During tracking the features are checked against a set of criteria established on the location and size of the detected face. It was stated earlier that a detected face is not required to track the person. Nonetheless, the Viola-Jones detector is run periodically during tracking to establish a probable person location with which to check the tracked points. In the event that no face is detected (i.e. the person has turned away from the camera) the last-known face location is updated based on the movements of the mean of the person cluster.

A person is lost when the number of tracked feature points drops below a certain threshold. Points may be lost either through Lucas-Kanade tracker error, or the point failing the criteria for the tracked person. When a person is lost, the system exits tracking mode and re-enters detection mode.

This approach to person detection and tracking has several advantages over the alternatives listed in Section 1.1. We use only gray-level image information and therefore do not require that any special color be worn by the person to be tracked or that the background and person be significantly different colors. We are only interested in the relative distance between the person and the background, rather than the absolute distance between camera, person, and background, and therefore there is no need to calibrate the cameras or have a specialized setup—any stereo system will do. By including both velocity and disparity information in the algorithm, our method is robust to scenes in which the person may not move, or where the person may not be the only moving object. Additional objects can also enter and exit the scene with little or no difficulty. By keeping the feature points sparse, computing time and complexity are reduced, resulting in a real-time algorithm that is robust to many of the issues encountered by previous methods.

1.3 Thesis Structure

Chapter 2 of this thesis provides a detailed description of our approach to the problem of person detection, with the approach for person tracking discussed in Chapter 3. Chapter 4 reviews the equipment, both hardware and software, that was used to design and implement this method, including the robot hardware and the methodology used to control the robot. Experimental results, including comparisons of our approach with various other methods, are presented in Chapter 5. Chapter 6 contains concluding remarks and suggestions for possible future work in this area.

Chapter 2

Person Detection

The task of detecting a person in a dynamic scene is a daunting one. With no *a priori* information, how does the system distinguish between a person and any other object in the scene. The answer is that a few assumptions must be made about the nature of people within the standard environment. We assume that the person will initially face the camera, and therefore the presence of a face is a key indicator of person location. We assume that the person will begin at a different disparity than the background or else be moving differently from the background. Finally, we assume that the person is standing straight and not leaning at an extreme angle. These assumptions allow our system to detect a person in environments where the methods discussed in Section 1.1 may have difficulty or fail altogether.

The sections that follow detail our approach to the problem of person detection, including how the above assumptions allow us to accomplish this task.

2.1 Lucas–Kanade Feature Detection

The foundation of our approach to person detection and tracking is the Lucas-Kanade feature detection and tracking algorithm developed by Lucas and Kanade in 1981 [15]. Lucas-Kanade was originally designed as an efficient method of image matching. The improvement of Lucas-Kanade over previous image matching methods is its speed. Previous methods of image registration were costly to compute and took far too long to be considered real time. Lucas and Kanade, however, greatly reduced the processing cost of image matching by limiting the number of potential matches

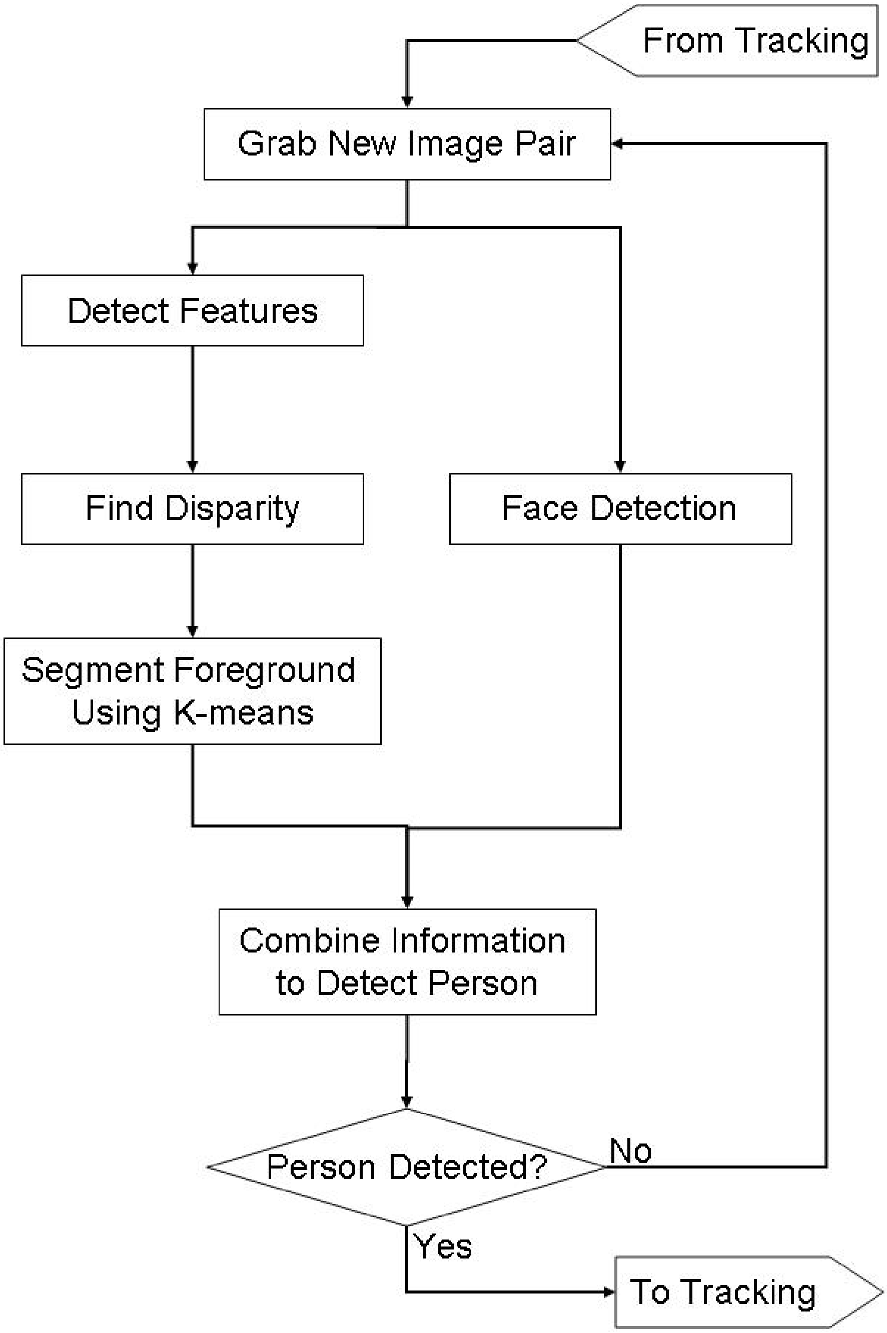


Figure 2.1: Program Flow for Person Detection.

between the images. This speed makes Lucas-Kanade ideal for our algorithm.

In order for the Lucas-Kanade method to work, it must first be determined if a feature point is trackable or not. A feature point is determined to be trackable if a window of size $n \times n$ pixels centered on the prospective feature has two large and similar-valued eigenvalues. This implies that the point is on a corner feature or is in an area with sufficient texture to be reliably tracked. This feature criteria also gives the algorithm the ability to select good features automatically from the image, or have features specified by the user. Once features have been selected, either by the user or by the algorithm itself, they are tracked by minimizing the L_2 norm error between the gradient of the current point location and the possible point location in the new frame. This assumes that the image has changed little between the two frames. In the case of person detection, this is a fairly safe assumption due to the fact that the algorithm is designed to run in real time, much faster than the normal speed of a person.

Our algorithm makes use of approximately 150 feature points selected in the image by the Lucas-Kanade tracker. When selecting features, the algorithm attempts to place them uniformly throughout the scene. This ensures that the features represent the scene effectively and that good spacing is maintained between features. This process of selection is repeated every ten frames during the detection phase in order to account for any new objects or people that may have entered the scene.

2.2 Creating a Sparse Disparity Map

Our algorithm makes use of Lucas-Kanade feature tracking in two ways. One of these is to track points from one frame sequence to the next. The other is to track points from the left frame to the right frame. For this step we assume rectified cameras. By subtracting the feature x-location in the left frame x^{LF} from the corresponding feature x-location in the right frame x^{RF}, we are able to calculate a disparity value d for the feature point.

$$d_i = |x_i^{LF} - x_i^{RF}| \tag{2.1}$$

In order to guard against incorrect disparity readings, a left-right consistency check is implemented. Features found in the right frame are tracked back to the left frame. By subtracting the left-right disparity d^{LR} from the right-left disparity d^{RL}, we arrive at an error value ϵ_d. If ϵ_d is below a pre-defined threshold, the disparity for the point is declared accurate and the point is kept. If ϵ_d is above the threshold, the point is labeled as lost and is excluded from use in future calculations.

$$\epsilon_d = |d^{RL} - d^{LR}| \tag{2.2}$$

$$\epsilon_y = |y^{LF} - y^{RF}| \tag{2.3}$$

Because disparity is based entirely on pixel location difference in the x-direction, a vertical y-direction consistency check is implemented as well. This is done using equation 2.3. If the error value ϵ_y is below a certain threshold, then the feature point passes the vertical criteria. This step is designed to eliminate possible error due to a point drifting to a location that may pass the disparity check in the x-direction but be significantly different in the y-direction.

Once this process of disparity calculation has been completed for every point, the list of feature points is ready for segmentation by clustering. For the first frame, each data point will only contain three dimensions to be clustered: the x, y, and d dimensions. After the first frame, the Δx, Δy, and Δd dimensions will be added.

2.3 Image Segmentation through Clustering

The concept of data clustering arises from research in the field of pattern recognition. There are a huge number of applications for clustering algorithms, especially in the areas of marketing and economic data analysis. In its simplest form, data clustering is no more than a way to group similar information. The interesting part is in the type of information being clustered.

In our case, the data being clustered are feature locations. This data is represented in six dimensions: the three spatial dimensions, (x, y, d), and three corresponding velocity dimensions, $(\Delta x, \Delta y, \Delta d)$. Due to the high dimensionality of the data, as well as the speed required to maintain the real-time nature of the algorithm, only relatively simple clustering methods were considered. These are detailed below.

2.3.1 K-means

K-means clustering is the simplest of the clustering methods considered. The benefit of k-means is its simplicity, speed, and accuracy. K-means is an iterative algorithm developed to minimize variance within clusters of data points. The process works by selecting a starting state, in this case the number of clusters N that we would like the data divided into, and starting means, μ_n, for each cluster. The starting means serve as a guess of where the actual cluster means may be. Unfortunately, the results of the k-means algorithm depends heavily on these starting means, and so they must be chosen wisely.

Once starting means have been chosen, the iterative portion of the algorithm can begin. First, the distance between each feature point X_i and each cluster mean μ_n is calculated. The feature point is assigned to the cluster with the closest mean. Once this has been done for all feature points, the means are recalculated based on the new group membership. This process of finding the distance and calculating means is repeated until no feature points change cluster assignments, or until the movement of the means between iterations falls below a given threshold. A detailed description of how k-means is applied to our case follows.

Each feature point is defined as $X_i = (x_i, y_i, d_i, \Delta x_i, \Delta y_i, \Delta d_i)$, where (x_i, y_i) is the feature position in the 2-D image plane, d_i is the disparity, and $(\Delta x_i, \Delta y_i, \Delta d_i)$ is the change, in pixels per frame, of each of those values. For the first pair of frames, the $(\Delta x_i, \Delta y_i, \Delta d_i)$ values are set to zero.

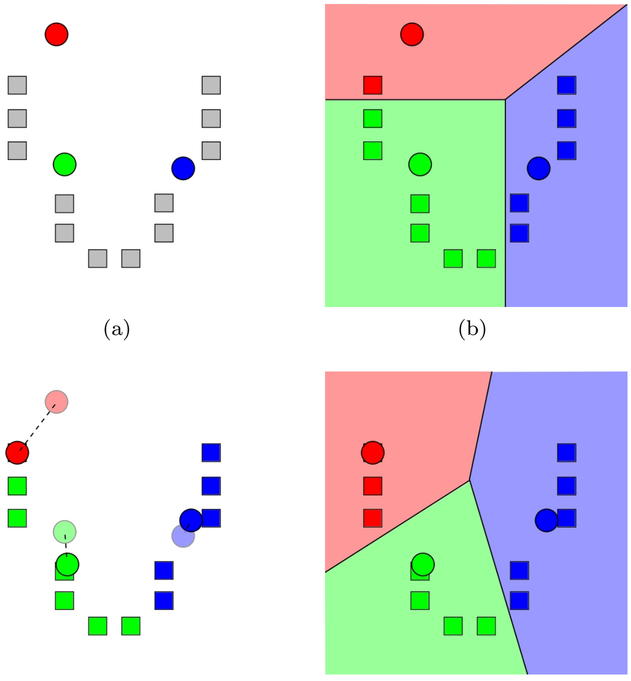
To begin, the set of feature points is divided into N clusters, where N is selected by the user depending on the data set and the number of clusters desired. The value $N = 3$ was chosen for our implementation. These clusters will serve as the starting guess for the actual cluster means. Since we are interested in segmenting foreground and background objects, it was decided to base our initial clusters on disparity value. The mean disparity of all feature points, μ_d, was calculated from these initial cluster assignments as follows:

$$\mu_d = \frac{1}{k} \sum_{i=0}^{k} (d_i) \tag{2.4}$$

Where k is the total number of feature points. Using μ_d, the points are divided into two major groups: points where $d_i \geq \mu_d$ and points where $d_i < \mu_d$. This second group is divided in half vertically to give us our third group. The means, (μ_x, μ_y, μ_d), for each of the resulting point groups are then found, giving us the necessary starting means for the algorithm. At this point the standard deviations, $(\sigma_x, \sigma_y, \sigma_d)$, of each group are calculated according to equation 2.5. The necessity of this will be explained shortly.

$$\sigma_x = \frac{1}{k}\sum_{i=1}^{k}(x_i - \mu)^2 \tag{2.5}$$

The heart of the k-means algorithm is iterative and works in two distinct steps. The first step involves finding the distance between each point, X_i for $i = 1 \ldots k$, and each mean, μ_n for $n = 1 \ldots N$. The task of finding the distance between points raises the question of what method of calculating distance should be used. Once the distance between a given point and each mean has been calculated, the point is labeled as belonging to the group with the closest mean. This is done for all points in the data set.


(a)

(b)

(c)

(d)

Figure 2.2: A simple k-means example.

Now that each point has been assigned to a group, the means, (μ_x, μ_y, μ_d), of each group are calculated just as in Equation (2.4). The previous distance step is then repeated using the new means. This cycle of assignment and recalculation results in clusters with minimal variance after only a few iterations.

An example of k-means can be seen in Figure 2.2. In Figure 2.2a the goal is to divide the data set, represented by squares, into three clusters. The starting means for the clusters have already been chosen and are represented by circles. Moving to Figure 2.2b, each piece of data in the set has been assigned to a cluster based on its distance to one of the three means. Figure 2.2c demonstrates the second half of the iteration, recalculating the mean values based on the new cluster membership, and Figure 2.2d completes the cycle by showing the new cluster assignments based on these new means.

2.3.2 Distance Measures

As discussed in the section above, implementing the k-means algorithm begs the question of how are we to measure the distance between our data points. The common answer would be to use Euclidean distance, just as we would with any standard problem. However, there are instances where Euclidean distance is not the optimal choice, and this is one of those instances. This section reviews both distance measures and their application to our algorithm.

2.3.2.1 Euclidean Distance

The Euclidean method for calculating the distance between two points is a fundamental equation for any student in math or a related field. It is attractive from a computation perspective because the equation is fast and simple. Also, it requires nothing beyond the locations of the two points between whom we would like to find the distance.

$$D_E = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (d_i - d_j)^2} \tag{2.6}$$

The equation for Euclidean distance (Equation (2.6)) between points (x_i, y_i, d_i) and (x_j, y_j, d_j) in $\Re^3$ looks almost trivial. Even with the addition of three more terms, as is required in our algorithm, the task is simple. However, the problem with Euclidean distance is that it treats all dimensions equally. Somtimes this is desired. However, because the distribution of points in our algorithm changes dramatically from dimension to dimension we risk rendering one or more dimensions statistically insignificant. Mahalanobis distance presents a solution to this problem.

2.3.2.2 Mahalanobis Distance

The method for calculating Mahalanobis distance (Equation (2.7)) is similar to Euclidean distance, but factors in the variance of the data points in each specific dimension.

$$D_M = \sqrt{\frac{(x_i - x_j)^2}{\sigma_x^2} + \frac{(y_i - y_j)^2}{\sigma_y^2} + \frac{(d_i - d_j)^2}{\sigma_d^2}} \tag{2.7}$$

By dividing the x, y, and d terms by their variances (σ_x^2, σ_y^2, and σ_d^2 respectively), dimensions with a high variance are weighted lower in the distance calculation than dimensions with a low variance. This has the effect of ensuring that dimensions are not penalized for having a low variance. As an example, in our algorithm the data set has a much higher variance in the x and y dimensions than in the d dimension. This is simply due to the nature of disparity and the camera set-up being used. If Euclidean distance were to be used, the disparity component of the point locations would factor very little into their distances from each other. By using Mahalanobis distance, the x and y components of the points location are scaled down, allowing the disparity component to contribute. It is for this reason that Mahalanobis distance was chosen over Euclidean distance for our algorithm.

2.3.3 Fuzzy c–means

The Fuzzy c-means algorithm [11] is nearly identical to the k-means algorithm. The difference is in how the data points are assigned to clusters. In k-means each feature point is assigned to the cluster with the closest mean value. In fuzzy c-means, points can be assigned to multiple clusters simultaneously. The degree to which each point is in a particular cluster is determined by the distance from the point in question to the mean of the cluster in question. This also allows points to still have an effect on a cluster without being completely within that cluster.

The central equations of fuzzy c-means (Equations (2.9) and (2.10)) are, as expected, very similar to those for k-means. However, there are some very important additions. For one, each point X_i requires a weighting value u to determine its membership in the n^{th} cluster. These weights are usually defined to sum to one:

$$\sum_{n=1}^{N} u_n(X_i) = 1 \tag{2.8}$$

where N is defined as the total number of clusters. Just as in k-means, fuzzy c-means requires the selection of initial cluster means. These are usually chosen uniformly at random, but can be selected in any fashion the user may deem appropriate. Once the weights of each feature have been calculated, however, the new means are determined by Equation (2.9) below.

$$\mu_n = \frac{\sum_X u_n(X_i)^m X_i}{\sum_X u_n(X_i)^m} \tag{2.9}$$

Once the centers of the clusters have been chosen, the weights for each feature point can be calculated. Each feature point X_i has a weight value u_n for each cluster n in the system. Although slightly more complex than the simple distance calculation for k-means, the fuzzy c-means weight calculation is related to the inverse of the normalized ratio between the distance from the given point to the current mean, and the distance from the given point to all other means. This is shown in Equation (2.10) below, where μ_n is the mean of the n^{th} cluster and $d(\mu_n, X)$ is the distance between point X and μ_n using the distance measure of choice. In this case m is simply a normalizing parameter, with the restriction that $m > 1$.

$$u_n(X) = \frac{1}{\sum_j \left(\frac{d(\mu_n,X)}{d(\mu_j,X)}\right)^{2/(m-1)}} \tag{2.10}$$

As with k-means, the process of clustering using fuzzy c-means is iterative. The process of determining feature weights (Equation (2.10)) and using those new weights to determine new means (Equation (2.9)) is repeated until the change in weight values between iteration t and iteration $t+1$ falls below a certain threshold ϵ (see Equation (2.11)).

$$|u_n^{(t)} - u_n^{(t+1)}| < \epsilon_u \tag{2.11}$$

Unfortunately, fuzzy c-means falls prey to the same problem as k-means: a heavy dependence on starting conditions. If the initial cluster means are not chosen wisely there is no guarantee that the results from fuzzy c-means will be accurate or even meaningful.

Fuzzy c-means was not implemented as part of our algorithm due to the fact that k-means provided acceptable results. However, using a soft-assignment clustering method is certainly an avenue that can be explored in the future. Because of the similarity of the two methods, it would be interesting to see if any great improvement could be achieved through the use of fuzzy clustering rather than standard k-means. Although it is doubtful that the use of fuzzy c-means would eliminate any current problems, it may provide a measure of added robustness to the detection algorithm.

2.3.4 Expectation Maximization (EM)

The Expectation Maximization (EM) algorithm [7] is a general statistical approach to the computation of maximum-likelihood estimates in data sets with incomplete data. As with k-means and fuzzy c-means, EM is an iterative algorithm, alternating between an expectation (E) step and a maximization (M) step. As with the other algorithms explored here, initial values for the missing data must be provided. In our case the missing data is the cluster assignment for each feature point.

Unlike the k-means and fuzzy c-means algorithms, which are based on distance measures, the EM algorithm is based on the probability density functions for each cluster. Because each feature point X_i will have a probability weight for each cluster u_n, and because these weights need to be kept independently for future calculation, a series of "support maps" are created to manage this data. There will be one support map for each cluster involved, where each pixel in the support map contains the probability weight that the given pixel is part of the given cluster. Because this is an iterative algorithm, some kind of starting values must be supplied for these support maps. This can be handled in one of two ways: the weights can be randomly assigned to points or the probability of the points' membership in each cluster can be estimated from nearby features. Once these initial weights have been chosen, the iterative process can begin.

As stated above, the EM algorithm consists of two distinct steps: the E-step and the M-step. These two steps are discussed in detail below, with an emphasis on how this method is applied to the problem of data clustering.

2.3.4.1 The E-Step

The purpose of the expectation, or E, step is to determine the expected value of each feature point in relationship to each cluster. Because EM is a general algorithm and is not specifically designed for any one application, the user must provide his own equations to make the method work. Each cluster can be estimated as a Gaussian with parameters $\theta_l = (\mu_l, \sigma_l)$. Therefore, the weighted probability that any feature point X_i is a member of group n is

$$\alpha_n^{(t)} p_n(X_i|\theta_i^{(t)}) \tag{2.12}$$

where $\alpha_n^{(t)}$ is the weighting value at iteration t for the i^{th} pixel in the n^{th} cluster. The result of this equation will be stored in the support map for cluster n at the location of the i^{th} pixel. It should be noted that, for the first iteration of the algorithm, the $p_n(X_i|\theta_i^{(t)})$ will be the initial probability value chosen by the user for that pixel and cluster. Once the new values for each pixel have been calculated using Equation (2.12) the values must be normalized. This is accomplished using

$$\sum_{n=1}^{N} \alpha_n^{(t)} p_n(X_i|\theta_i^{(t)}) \tag{2.13}$$

where N is the total number of clusters. The value at location i in each support map is then divided by the result of Equation (2.13). This should normalize the values to agree with Equation (2.14), where x_{ni} is the pixel value at point i in the n^{th} support map.

$$\sum_{n=1}^{N} x_{ni} = 1 \tag{2.14}$$

Following normalization, the data is ready for the maximization step.

2.3.4.2 The M-Step

The maximization, or M, step will take the values calculated in the E step and use them to prepare the algorithm for the next iteration. This involves calculating new values for the cluster parameters. Specifically, the probability weights α_n, the cluster means μ_n, and the cluster standard deviations σ_n. This step is simpler than the E step above, and can be accomplished in three calculations, listed in Equation (2.15), below.

$$\begin{aligned}
\alpha_n^{(t+1)} &= \frac{1}{N} \sum_{n=1}^{N} p(m|X_n, \Theta^{(t)}) \\
\mu_n^{(t+1)} &= \frac{\sum_{n=1}^{N} X_n p(m|X_n, \Theta^{(t)})}{\sum_{n=1}^{N} p(m|X_n, \Theta^{(t)})} \\
\sigma_n^{(t+1)} &= \frac{\sum_{n=1}^{N} p(m|X_n, \Theta^{(t)}) \left\{(X_n - \mu_n^{(t)})^2\right\}}{\sum_{n=1}^{N} p(m|X_n, \Theta^{(t)})}
\end{aligned} \tag{2.15}$$

In the context of these equations, $p(m|X_n, \Theta^{(t)})$ is the value stored in the i^{th} feature location of the n^{th} support map, and N is the total number of clusters (and support maps).

Once the M step is complete, the E step is repeated with the new cluster parameters and new probability weights are calculated for each feature point. This process is continued until the change in weights falls below some threshold ϵ_α. How we want to interpret this probability data is up to us. The benefit of the EM algorithm when applied to clustering is that it can be used as either hard-assignment, like k-means, or soft-assignment, like fuzzy c-means. For hard assignments, the algorithm will simply assign the feature point to the cluster with the highest probability weight associated with it. For soft assignments, the algorithm will keep the various probability weights stored in the support maps and use that as a measure of the extent to which the given feature is a member of the given cluster.

2.3.4.3 Conclusions about EM

Although the most complex of the algorithms examined, EM does have many benefits. Because EM is a general algorithm, and the internal equations must be changed to suit the given data set and desired results, it is far easier to adjust the internal workings of EM than to do the same to k-means or c-means, which are both based on fundamental, unchangeable assumptions. EM is flexible in that it can be altered to provide hard or soft cluster assignments for feature points. Also, because it is based on a probability measure, rather than a distance measure, EM is robust to outliers that have little statistical significance for the given cluster. In this way, EM is very similar to fuzzy c-means, with the added benefit of allowing the user to select and configure the fundamental equations upon which the specific implementation is based.

However, EM has its drawbacks. In spite of being less dependent on initial conditions than k-means or c-means, EM is still only guaranteed to find a local minima, and it is therefore still possible to get poor results with poor starting conditions. One solution for this is exactly what is implemented in our current algorithm. That is, start with intelligent initial conditions rather than purely random ones. Another possibility is to run the algorithm multiple times with various starting conditions and simply select the result with the best fit to the data. This would be difficult to implement in a real time application, however, due to the increased processing associated with running the algorithm multiple times a frame.

A second problem arises from EM's reliance on probability measures. For any given feature point, the probability weight for a given cluster may be very small. However, problems have been known to arise if it is assumed that very small weights are equivalent to zero. Obviously, this problem would only arise in a soft-assignment situation, as the smallest weight would be insignificant in a hard-assignment case. No current solutions are proposed for this problem, except to simply be very cautious about rounding weight values to zero.

In the end EM clustering was not implemented because it was simply not necessary. The results achieved using k-means were sufficient for our purposes.

2.3.5 K-means++

As discussed in Section 2.3.1, one of the major weaknesses of the k-means algorithm is in its initialization. The user must know, a priori, the number of data clusters to be expected and the general location of the means of those clusters. Poor starting conditions can lead to serious errors in clustering, and even failure of the algorithm altogether. This dependence on starting conditions is critical to our problem of using k-means to segment an image and is the reason for the creation of the method discussed in Section 2.3.1 to derive reasonable starting means and cluster locations. Of course, because k-means is such a popular clustering method, and because the initialization problem is critical to a huge array of applications, other methods of finding good starting means have been created. One of these is K-means++ [2].

$$Weight = \frac{D(x\prime)^2}{\sum_{x \in X} D(x)^2} \tag{2.16}$$

K-means++ is an algorithm for improved, autonomous, initial mean selection. Usually, initial centers are chosen uniformly throughout the data set. Many times data points themselves are chosen as the initial means, simply as a computationally easy solution. K-means++ selects starting centers randomly, but with more thought applied than the uniform selection: each point's chance of being selected is based on a specific probability calculation (see Equation (2.16)). This probability calculation, known as D^2 weighting, is simply added to the k-means algorithm in place of the random mean selection. The steps are as follows:

1. Choose an initial center n_1 uniformly at random from X, where X is the set of all data points.

2. Choose the next center n_i, selecting $n_i = x\prime \in X$ with probability $\frac{D(x\prime)^2}{\sum_{x \in X} D(x)^2}$, where $D(x)$ is the shortest distance from a data point x to a previously-chosen center.

3. Repeat step 2 until a total of N centers have been chosen, where N is a pre-determined number.
4. Proceed with the standard k-means algorithm.

By using the distance of each point to the previously-chosen center points, k-means++ allows the k-means algorithm to start with an intelligently-selected group of centers. This not only reduces the number of iterations needed for k-means to converge on a solution but greatly increases the chances that k-means arrives at a valid answer.

After careful consideration, it was decided not to use k-means++ to augment our algorithm. This decision was based on the fact that we do have some good *a priori* knowledge of how the set of feature points should be divided. This knowledge was used to develop the algorithm that is currently used to select starting clusters and means. K-means++, although an excellent method for choosing random cluster means, does not take advantage of this knowledge.

2.3.6 The Chosen Method

Each of the three clustering methods presented in this section have positive and negative aspects. The idea of soft or fuzzy cluster assignments is tempting, since that allows the system to more closely approximate reality. However, with increased accuracy comes increased complexity. The fuzzy c-means algorithm is not much more complex than k-means but suffers from the same dependence on initial conditions. EM is slightly less dependent on initial conditions due to its alternate approach, but the problem is by no means solved. In addition, the equations governing EM are set by the user, introducing another possible dimension of uncertainty.

For these reasons, and because of its simplicity, k-means we select as the algorithm for segmentation. Given the results that have been achieved using this simple method, it would be interesting to see how a more complex method would improve the effectiveness of our person detection algorithm, but such an investigation is outside the scope of this thesis. The largest problem with k-means, its reliance on starting conditions, has been mitigated through the use of intelligent selection of starting criteria. There are many other ways to handle this problem, one of which is discussed in Section 2.3.5.

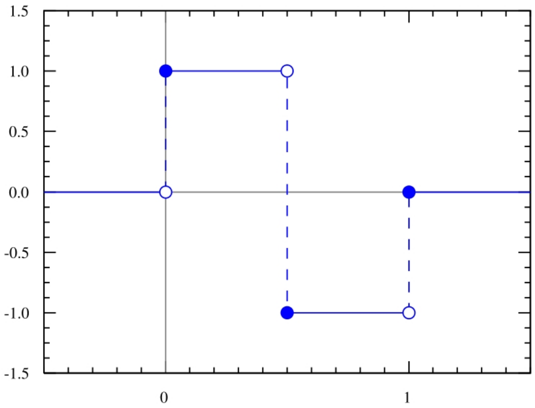


Figure 2.3: A Haar Wavelet.

2.4 Face Detection using Viola-Jones

While object segmentation using k-means goes a long way toward accomplishing our goal of person detection, there is still one question remaining. How do you differentiate between clusters that are people and clusters that are not? To solve this problem, a Viola-Jones face detector is used.

The Viola-Jones method of face detection uses integral images to rapidly compute and evaluate features that resemble Haar basis functions (see Figure 2.3). The algorithm then uses AdaBoost [9] to select a small number of important features and create a classifier. A cascade structure is used to rapidly evaluate all possible face locations within an image and arrive at the most likely solution or solutions. This method performs well at 15 frames per second, which makes it sufficient for our purposes. The specific use of the face detector as part of our algorithm is described below.

Every 10 frames during the detection phase of the algorithm, the face detector is run on the left frame of the stereo pair. If a face is found, the detector is then run on the right frame. If at least one face is found in both the left and right frames of the stereo pair, then the face moves on to the next level of tests. Just as with the feature points, it can not be assumed that the faces detected in each frame are the same face. A disparity check very similar to that used with the feature points is performed. The face location in the left frame F^{LF}, where $F = (x_F, y_F)$, is subtracted from the face location in the right frame F^{RF}. The absolute value of this gives us a disparity value, d_F (see Equation (2.17)). Because the face is not actually "tracked" from the left frame to the right frame, the left-right consistency check used for the feature points will not work. Instead, the calculated disparity value is checked against a reasonability threshold. This threshold represents a reasonable limit to what the disparity of the face might be. If the calculated value exceeds this limit, then the value is not reasonable and the face is declared invalid. The face location is also subjected to the same vertical difference test as the feature points (see Equation (2.18)). If the value of y_{diff} exceeds the accepted value, the face is discarded.

$$d_F = |x^{RF} - x^{LF}| \tag{2.17}$$

$$y_{diff} = |y^{LF} - y^{RF}| \tag{2.18}$$

If the face passes both tests, then the program continues with a face location in (x_F and y_F) and a disparity value (d_F). Along with this information, the hight and width of the face are saved. All of this data will be used to determine which of the clusters found during segmentation might be a person. This process is described in Section 2.5.

It was stated previously that the Viola-Jones algorithm is only run every 10 frames. This is because, although the system can run independently at 15 frames per second, it is by far the slowest part of our algorithm, and to run it every frame would introduce considerable lag. The location of any found faces are kept until the next time the face tracker is run, in order to reduce the possibility of missing the person during one of the frames when the face tracker is not run.

2.5 Data Fusion: Combining Viola-Jones with Clustering to Detect a Person

In the previous section, the problem of combining the face detector and segmentation data was touched upon. This section discusses that problem in detail and outlines the thought process behind the chosen criteria.

The basic question to be answered here is how do we define a person given the information at hand. To begin to answer this question, several factors are considered. First, what kind of data is available to the detector? Once a face has been detected and segmentation has been performed, the mean location of all clusters and the location of the face are the two major pieces of information that the detector can make use of. Secondly, what is the most efficient way to make use of this information and achieve accurate person detection in real time? The answer to this question is slightly more complex.

The algorithm needs to run in real time, and therefore a quick-and-dirty method of eliminating clusters from consideration is needed. To achieve this, it was decided that the best criterion for a person is that a person is an independent cluster, and the mean of that cluster must lie in a bounding box below the detected face. This is the first criterion that must be met for a cluster to be considered as a person. The borders of the bounding box are chosen using the size of the detected face as an indicator of probable body size. The boundary criteria are shown in Equation (2.19), where $F = (x_F, y_F, d_F)$ is the center point location of the detected face, F_W and F_H are the width and height respectively of the detected face, d_b is the disparity range of the bounding box, and I_H is the image frame height.

$$\begin{gathered} x_F - (1.5)(F_W) < \mu_x < x_F + (1.5)(F_W) \\ y_F + (0.5)(F_H) < \mu_y < I_H \\ d_F - d_b < \mu_d < d_F + d_b \end{gathered} \tag{2.19}$$

Next, the algorithm checks to see how many points in the given cluster fall within the bounding box. Any points outside the bounding box are declared lost. If not enough points remain then the person cannot be reliably tracked, and the cluster fails. If sufficient points are found within the box, then the cluster is labeled as a person and the program enters tracking mode. In the event that more than one cluster mean satisfies the bounding box criteria, the cluster with the greatest number of points within the box will be labeled the person.

This method should easily handle more complex situations, such as multiple detected faces or multiple people in the image. In the event of multiple faces, the above criteria will be tested for every face. If a unique cluster is found to satisfy the criteria for each unique face, then multiple people will be found and tracked.

2.6 Removing Outliers using Boundary Parameters

During the process of detecting and tracking people there are many reasons why a feature point needs to be removed from consideration. The point may be lost by the Lucas-Kanade tracker. It may be removed from consideration because it did not pass the disparity test. Or, it could be removed because it fell outside the bounding box during person detection. However, the point cannot simply be forgotten about or deleted from the point list because the point list needs to maintain synchronization with the point list used by the Lucas-Kanade tracker. While the program is running, all feature points are kept in a vector of custom structures, known as the PointList. Each structure contains the point's location in $\Re^6$ space, $(x, y, d, \Delta x, \Delta y, \Delta d)$, the point's current cluster assignment, and the status of that point (new, tracked, or lost, represented by 1, 0, or -1 respectively). The first hurdle that a point must pass to avoid being lost is from the tracker itself. The Lucas-Kanade tracker is not perfect and may lose points due to occlusion or changes in the environment or lighting conditions. If a point survives the tracking stage it must then pass the left-right disparity consistency check discussed in Section 2.2. The majority of the points that make it past this stage will be eliminated after a person has been detected. At this point, any feature that is not both a member of the chosen cluster and within the person bounding box will be labeled as lost. This allows tracking to proceed with a minimum number of points to manage. Points can also be eliminated during tracking. Any point that strays outside the bounding box, or that is lost due to tracking error, will be eliminated from further calculation. Once a point is lost, it cannot be recovered without generating an entirely new set of features. To handle the movement of people and objects in and out of the frame during detection, a new set of feature points is recalculated periodically, so feature loss is not a significant problem.

Because each point must be kept in the point list, even if it is declared lost, the point cannot simply be deleted. When a feature point fails one of the criteria for tracking and is marked lost, only a few things change. Regardless of the status of the point, the update function will continue to update the point's location within the image based on the results from the Lucas-Kanade tracker and the disparity calculation. However, if the point is lost, the update function will set the delta values for the point to zero. This will keep the point from registering an erroneous delta value if and when the point is re-included into the program calculations. Lost points are excluded from program calculations by a simple conditional statement included in each function that excludes any point with a status indicating that the point has been lost.

Chapter 3

Person Tracking

Detecting the person is only one half of the problem. To be effective, the algorithm must be able to track the person as well. Our approach to tracking the person is modification of our approach to detecting the person. After the person is detected, but before the system enters tracking mode, all points that either belong to the wrong cluster, or that fall outside the given bounding box, are set as lost in the point list (see Section 2.6). Any points that remain after this step are used in tracking.

The tracking process consists of the same basic steps as detection. First, a new pair of stereo frames are received from the cameras. Lucas-Kanade feature tracking is used to track all remaining features from the previous set of frames to this new set. The disparity for the points is calculated, including removing any points that do not pass the disparity check. The face detector is run every 10 frames to locate any faces within the image. Any detected faces are used to determine where the person should be and what points are valid. Between runs of the face tracker, the location of the face is updated based on the movement of the person cluster mean. The k-means algorithm is run on the remaining set of tracked points to recalculate mean and standard deviation values. Finally, the number of successfully tracked points are monitored to ensure that the person is being tracked. If, at any point, the number of tracked points falls below a given percentage, the tracker will lose the person and return to detection mode.

This chapter contains a detailed description of these steps, including information on alternate methods considered.

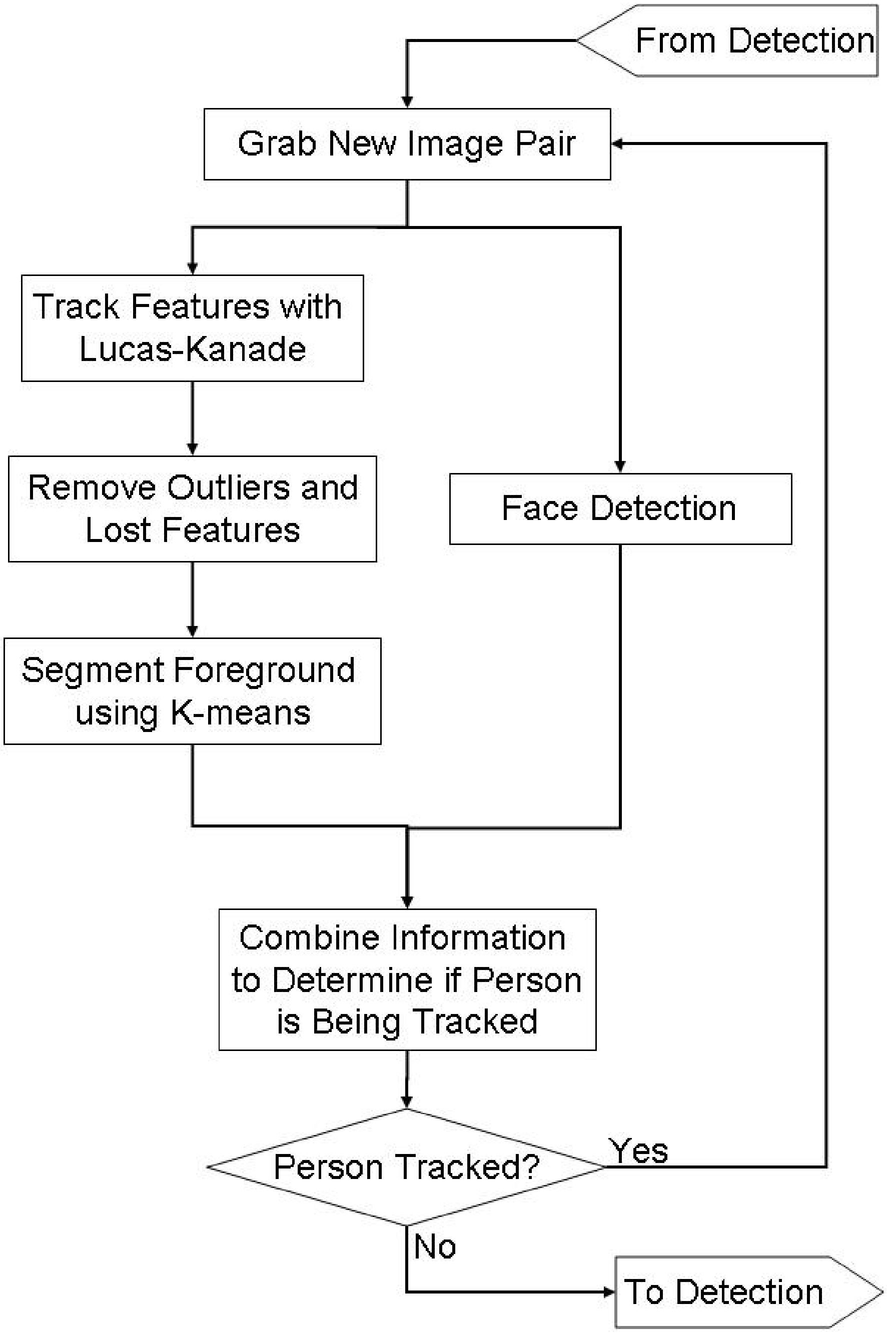

From Detection
Grab New Image Pair
Track Features with Lucas-Kanade
Remove Outliers and Lost Features
Face Detection
Segment Foreground using K-means
Combine Information to Determine if Person is Being Tracked
Person Tracked?
Yes
No
To Detection

Figure 3.1: Program Flow for Person Tracking.

3.1 Lucas-Kanade Applied to Person Tracking

Because the tracking phase of the algorithm assumes that the person has already been found, the majority of the work is turned over to the Lucas-Kanade feature tracker. Once all points not on the detected person are removed during the last step of the detection phase, the Lucas-Kanade algorithm is relied on to effectively track the remaining points from frame to frame, as well as to maintain up-to-date disparity values. This works just as in the detection phase, allowing the algorithm to remove any points that do not match the disparity of the tracked person or that do not pass the consistency check. In addition, all tracked points will be checked against the person "bounding box" discussed in Section 2.6 (see Equation (2.19) for box parameters).

3.2 Removal of Feature Points During Tracking

An important part of the person tracking process is ensuring that the points being tracked still accurately represent the location of the person. To ensure this, a set of criteria are enforced on the set of tracked points. These criteria are exactly the same as those seen in Equation (2.19) in Section 2.6. Because this "bounding box" against which the tracked feature points are compared is based on the location and size of the face, a method for estimating face location, even without a positive detection from the face tracker, was found. This involves updating the last known face location based on the movement of the mean value of the person cluster from frame to frame. This approach has been found to provide adequate results, and is expanded upon in the next section.

As stated in previous sections, any feature points that do not meet the specified criteria for tracking will be marked as lost in the point list. The only way for these points to recover is for the person to be lost by the tracker. If this occurs, the program will re-enter the detection phase and the Lucas-Kanade tracker will select a new set of feature points.

3.3 Augmentation with Viola-Jones

The Viola-Jones face detector plays a slightly different role in tracking than it does in detection. For detection, the person's face must be seen by the system. Once the person has been located, however, the face is no longer required. The face detector is still run during tracking, however, to provide an added level of robustness to the algorithm. If any faces happen to be discovered during tracking, the program will update the location of the face using this new data and check the tracked points against the boundaries based on this data. This ensures that the feature points haven't drifted due to occlusion or any other tracking error.

A problem arises, however, in the fact that the face detector is only run once every 10 frames. This is due to the relatively high processing time required by the face detector interfering with the real-time nature of the program. However, an up-to-date face location is required if the tracked feature points are going to be checked for accuracy. The solution to the problem of locating the face between face detector runs is a motion estimator implemented in our system. This estimator updates the location and size of the face based on the movements of the mean of the person cluster. This is done by first calculating several delta values (Equation (3.1)) which represent the change in the location of the mean from frame $t-1$ to t in the x, y, and d dimension. These equations are designed so that any move toward the origin, $(0,0,0)$, will be seen as a negative change. Therefore, any change is added to the current location of the face (Equation (3.2)).

$$\begin{aligned} \Delta x &= \mu_x^{(t)} - \mu_x^{(t-1)} \\ \Delta y &= \mu_y^{(t)} - \mu_y^{(t-1)} \\ \Delta d &= \mu_d^{(t)} - \mu_d^{(t-1)} \end{aligned} \tag{3.1}$$

To update the size of the face, the change of the mean in the disparity dimension, Δd, is scaled by two and added to the current height and width of the face (Equation (3.3)).

$$\begin{aligned} x_F^{(t)} &= x_F^{(t-1)} + \Delta x \\ y_F^{(t)} &= y_F^{(t-1)} + \Delta y \\ d_F^{(t)} &= d_F^{(t-1)} + \Delta d \end{aligned} \tag{3.2}$$

$$\begin{aligned} F_W^{(t)} &= F_W^{(t-1)} + 2\Delta d \\ F_H^{(t)} &= F_H^{(t-1)} + 2\Delta d \end{aligned} \tag{3.3}$$

3.4 Losing a Person and Returning to Detection

The determination of whether or not the person is being tracked reliably is dependent on the number of valid feature points currently being tracked by the algorithm. When the system makes the switch from detection to tracking, a large portion of the feature points are “lost” by the program. These include any point which is not a part of the person cluster or that does not fall within the criteria listed in Equation (2.19).

During tracking, points can be lost in several ways. First, they can be lost by the Lucas-Kanade feature tracker. Secondly, they may fail the disparity test. And lastly, they may drift from the person to the background and fail the criteria listed above. If the number of tracked points falls below a certain threshold, in this case 25% of the original number tracked, then the algorithm considers the person lost and the system returns to detection.

Chapter 4

System Overview

Engineering, by its very nature, is the process of turning theory into reality. Experiments must be run on, and are sometimes defined by, the equipment at hand. This section details the equipment used to design, create, and implement our method. This includes all computer hardware and software, as well as the camera and robotic equipment and our method for controlling the robot.

4.1 Computer Hardware and Software

Our algorithm was designed and written using a Dell Inspiron 700m laptop (Figure 4.1a), with 496 MB of RAM and an Intel Pentium Centrino 1.6 GHz processor. The operating system was Windows XP Service Pack 2. The Integrated Development Environment (IDE) used was Microsoft Visual Studio C++ 6.0. Many of the supporting algorithms used, including the Lucas-Kanade feature tracker and the Viola-Jones face detector, are part of Blepo [4], a free Computer Vision library developed by our lab that incorporates the Intel OpenCV library [1].

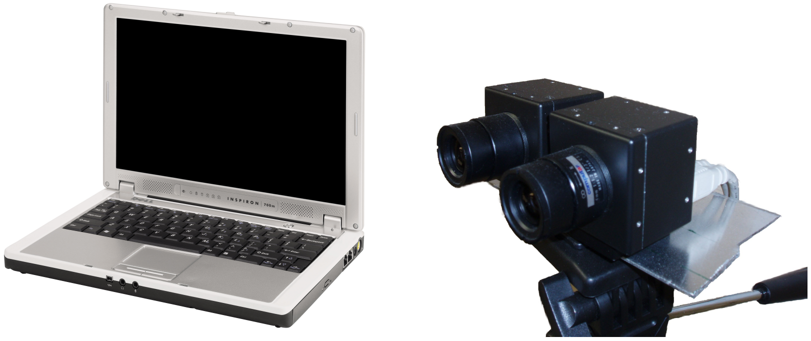
This computer provided enough processing power to effectively run the algorithms needed for our program while remaining small enough to fit easily on top of our robotic base during testing.

4.2 Stereo Camera Hardware

Rather than use an integrated stereo camera setup, two identical ImagingSource DFK 21F04 CCD cameras were used (Figure 4.1b). Each camera is equipped with with a Computar 4mm 1:12

1/3” lens. These were secured at a distance from each other of approximately 6cm from center to center using an aluminum plate, bolted to a standard aluminum tripod. The cameras ran off a single 8-32 VDC power supply and were connected to the computer using an IEEE 1394 Firewire interface. The cameras were daisy-chained together, so only one connection to the computer was required. For the experiment, the cameras were set to output a 320×240 resolution YUV format frame at a rate of 30 fps (frames per second).

The tripod and cameras are simply placed on top of the robotic base, with the cameras significantly above the level of the mobile robot (see Figure 4.2b). This allows the cameras to more easily see the person’s face during the detection process and avoid being blocked by low obstacles during tracking.



(a) Dell Inspiron 700m

(b) Stereo Camera Setup

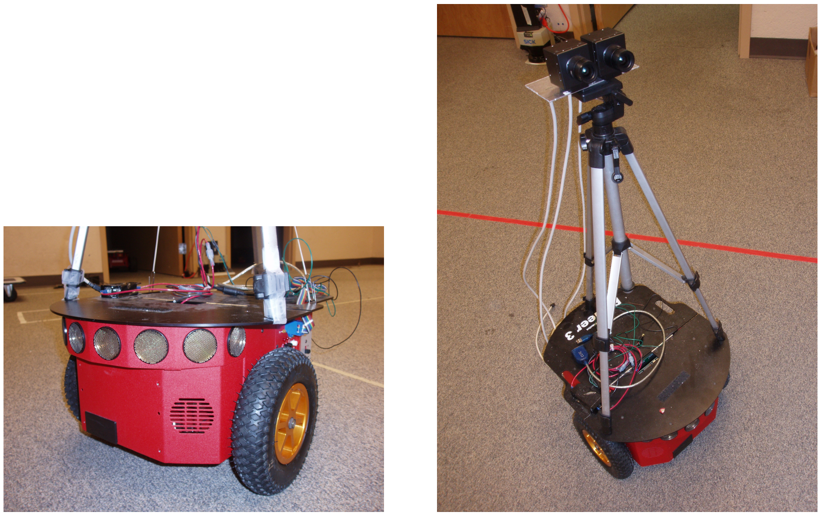
Figure 4.1: The Two Critical Segments of the Person Detection System.

4.3 Mobile Robotic Platform

The robotic base used for our system is a Pioneer P3-DX mobile robot from ActivMedia Robotics (see Figure 4.2b). ActivMedia Robotics specializes in mobile robot platforms for both commercial and research applications. The P3-DX is a 3-wheeled model with an integrated forward-looking sonar array (see Figure 4.2a). The robot has two main drive wheels and one non-powered caster for stability. Steering is accomplished by running the wheels at differential velocities.

The Pioneer line of ActivMedia robots are operated through a standard RS-232 interface and the ActivMedia Robotics Interface for Application, or ARIA, software API. ARIA is designed as a simple interface for software to access and control all aspects of the robot, including the sonar array, as well as common attachments, such as pan/tilt cameras and laser scanners. Our program does not currently call for additional sensors, but such things could be implemented later if needed for tasks such as object avoidance. Our setup can be seen in Figure 4.2.

Our program makes use of ARIA's functions for setting the robot's forward and rotational velocity. There is a hardware limit to both of these values, but our program institutes software limits to ensure that no harm comes to the robot or observers due to accidental collisions at excessive speed.



(a) Pioneer P3-DX Detail

(b) Pioneer with Stereo Camera Setup

Figure 4.2: Our Robotic Platform.

4.4 Robot Interface

The final stage of the design process was to interface our program with a mobile robot base. As was mentioned above, the robot's API, ARIA, allows the user to interface their program with the robot as well as a wide array of sensors and devices. However, for our system, simply sending movement commands to the robot is sufficient.

The primary output from the person tracking system is a coordinate in (x, y, d) space. The robot interface makes use of the x and d values of this coordinate. When tracking begins, the disparity of the person is recorded. During tracking, the current disparity value is checked against this original value and a proportional (p) controller is used to move the robot to minimize the difference between these values. A p-controller is also used to move the robot to minimize the distance between the current x-value of the person and the x-value of the center of the frame.

The equations for the forward velocity v and rotational velocity ϕ of the robot are provided in Equation (4.1), where C_f is the forward velocity gain, C_r is the rotational velocity gain (50 and 0.75 respectively), P_d is the disparity of the person, and P_x is the x-location of the person.

$$\begin{aligned} v &= C_f(d_i - P_d) \\ \phi &= C_r(\frac{F_{width}}{2} - P_x) \end{aligned} \tag{4.1}$$

A p-controller was chosen over more complex control methods due to the ease of implementation and the relative simplicity of the system being controlled. The results achieved using this controller are more than adequate to allow the robot to follow a tracked person. The one key to this control method is the tuning of the gains. In our case, the gains were tuned through a process of trial and error. Badly-tuned gains will result in poor tracking of the person and poor following by the robot. For the safety of the robot and the user, the forward velocity and rotational velocity were limited within the program. The algorithm sends a signal for the robot to stop moving as soon as the person is lost by the tracking system.

Chapter 5

Experimental Results

This section describes the various situations in which our system was tested, both live and recorded. Observations of the system's performance in these different situations are given. In order to establish a basis of comparison, the results of our method are discussed along with the probable results from both the system described in [5] and basic color-segmentation techniques. Lastly, we discuss how the robot was controlled in order to allow following of the tracked person by the system.

5.1 Experiment Overview

Our system was tested using both live situations and previously recorded video. The reason for the two different testing methods was to demonstrate the system's ability to achieve repeatable results, as well as to verify performance in real-world situations.

5.1.1 Recorded Video

During the process of testing the system, two video sequences were recorded to be used for testing. The first video features a single person starting off camera, entering the frame, and then simply walking forwards and backwards while facing the camera, allowing the system to test the ability to track a person near objects of similar disparity. The second video features a person standing stationary in the frame while a second person walks around her, both looking at the camera. This tests the system's ability to deal with occlusion of the tracked person.

There are many benefits to using previously recorded video. Previously recorded video allows repeatability during testing, as well as control over the testing environment.

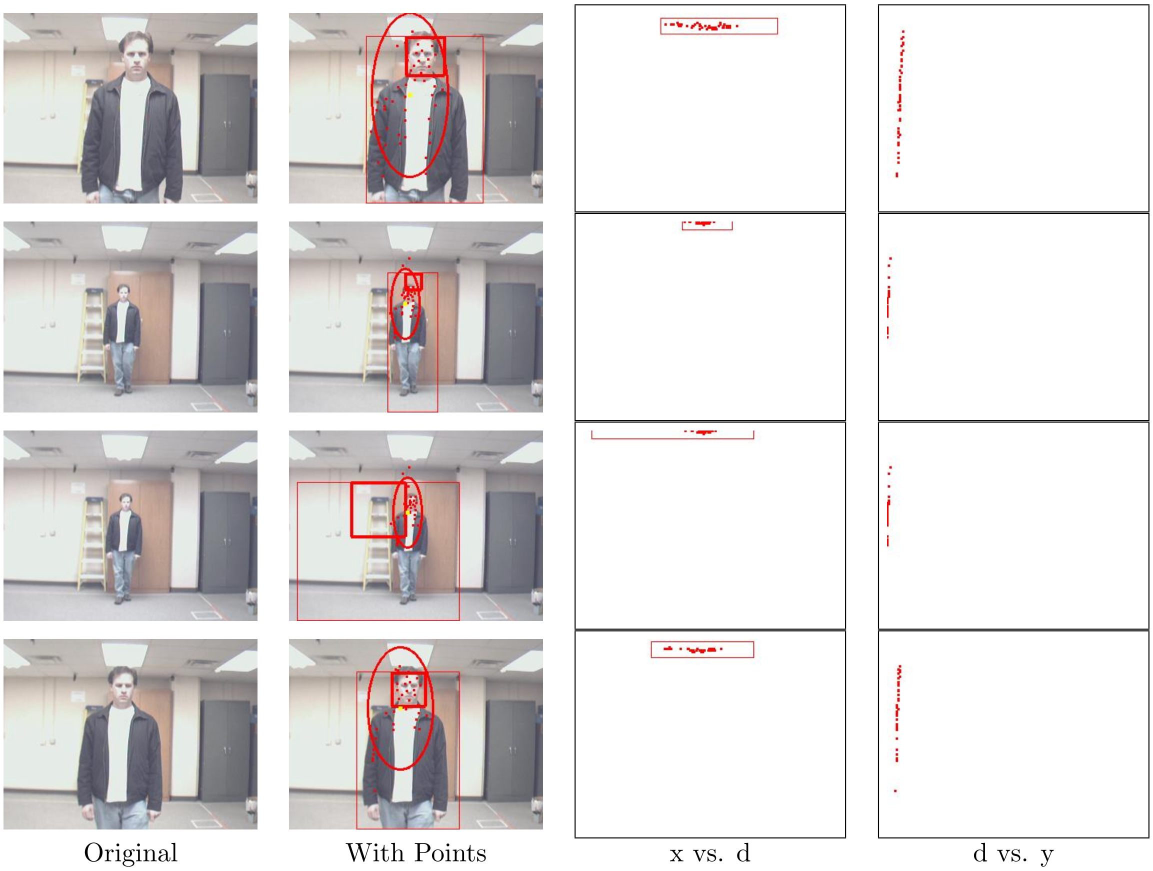


Figure 5.1: Recorded Frameset Results: Back-and-Forth Test.

The results from the back-and-forth and occlusion recorded video tests are displayed in Figures 5.1 and 5.2 respectively. Note that for both of these figures, the first column is the unaltered image from the left camera, the second column shows the addition of feature points, face, and bounding box overlays, the third column presents the feature points plotted as a function of x versus d with x as the horizontal axis, and the fourth column displays the feature points plotted as a function of d versus y with d as the horizontal axis. In Figure 5.1, the first row occurs at frame 110 of the video stream, when the tracker has successfully found the person within the image. The second row, occurring at frame 400, shows that the person was still successfully tracked even when close to the background. The third row demonstrates the issue of false positives from the face detector. At frame 410, the face detector mistakes the top of the ladder and the paper on the wall for a face. This is due to the person's actual face being too far away for the detector to acquire it, and also to the shape of the ladder and paper approximating a face well enough to provide a false match. However, during testing there were many times when the face detector would find false faces within the background. This problem, and possible solutions, are discussed in Chapter 6. The tracker recovers the person's face and resumes successful tracking by frame 670 (final row).

Figure 5.2 displays similar results for the occlusion test. Unfortunately, the two people in the scene were too far away from the cameras for the face detector to get a positive match initially. However, by frame 380 (first row), the person had been acquired. Although successful, this frame demonstrates another error of the face tracker, since it has detected the subject's chest, rather than his face. Again, this error has been corrected by frame 450 (second row) and the tracking continues. The person moves laterally across the image successfully by frame 630 (third row), and finally ends up partially occluded behind the subject standing in the middle of the frame in the fourth row. Although the tracked subject was never fully occluded, the partial occlusion in row four does demonstrate that the tracker is robust to this situation.

5.1.2 Live Video

Although testing on previously recorded video streams provides valuable data and feedback, however the system must function well in live environment tests. These tests involve mounting the cameras and tripod to the top of a mobile robot (see figure 4.2b). The system is used not only to track a person within the scene, but to follow the tracked person within a given area. A selection of frames, both with feature points and without, can be seen in Figure 5.3.

Although not as useful for testing purposes as the recorded video, live testing is essential to determine how well the system operates in realistic situations. In Figure 5.3, the robot tracks a target around an enclosed area. During this test run, the robot was able to track the person successfully a majority of the time. Problems with loss of the person were either caused by quick robot or subject motion, resulting in loss of the tracked feature points by the Lucas-Kanade algorithm, or through detection of false faces within the image. This second problem is particularly apparent when testing in the selected enclosed area, as a large number of objects are mounted to the walls and have the potential to confuse the Viola-Jones algorithm.

Figure 5.3 shows results from several different situations encountered during a single live test. This figure follows the same column structure as the previous figures. In the first row, the

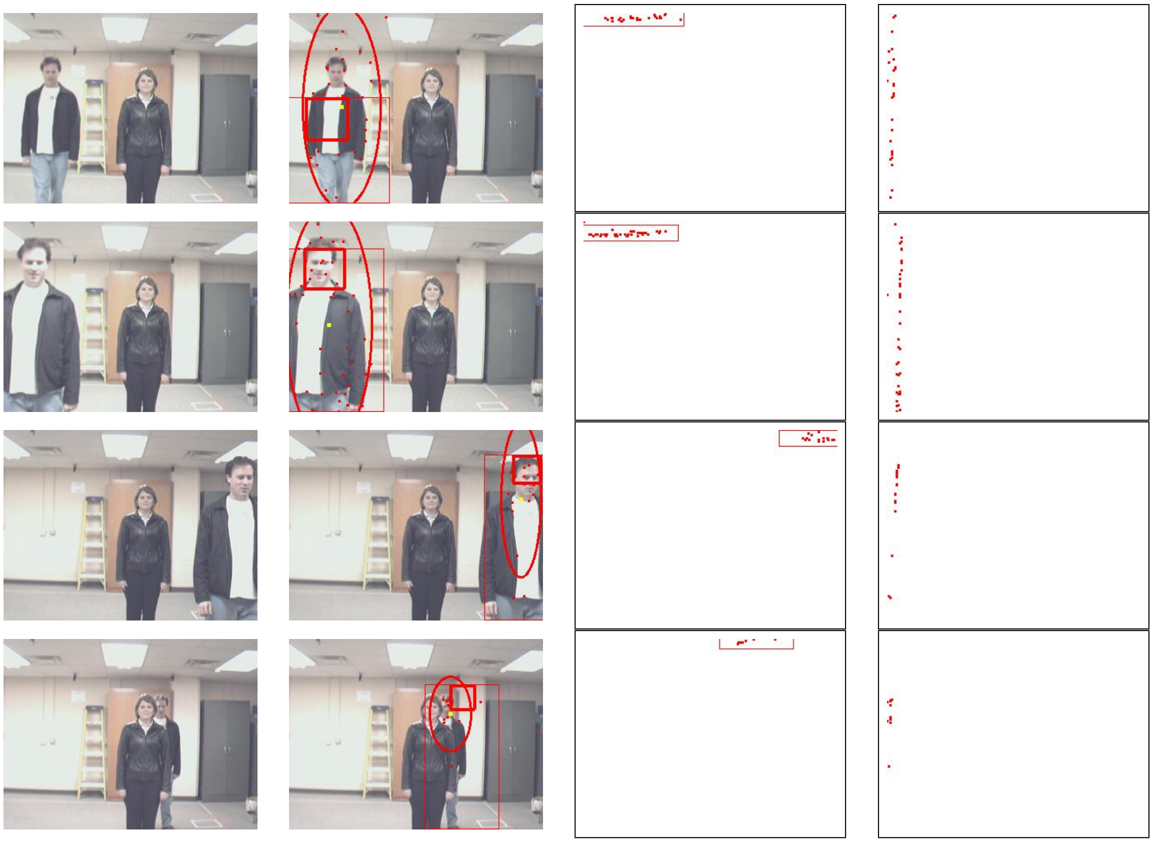


Figure 5.2: Recorded Frameset Results: Occlusion Test.

algorithm has detected a person within the scene and has entered tracking mode. The person is well represented by the tracked feature points and the face is easily detected by the Viola-Jones tracker. The frames in the second row occur several seconds later in the sequence. The person has moved significantly within the room and, although a few feature points have been lost, the person remains tracked. In the third row of frames the person is moving quickly relative to the camera and has become blurred slightly. This has led to a noticeable thinning of tracked feature points, but the tracking algorithm maintains a valid person position.

The last two rows of the figure demonstrate two special cases. The penultimate row shows a face detection error similar to the one discussed earlier in Section 5.1.1. Rather than a ladder, the face detector has incorrectly selected a grouping of background clutter as the face. This has led to a loss of tracking and the person will have to be recovered. The last row demonstrates the tracker's ability to maintain a person's location without the aid of the face.

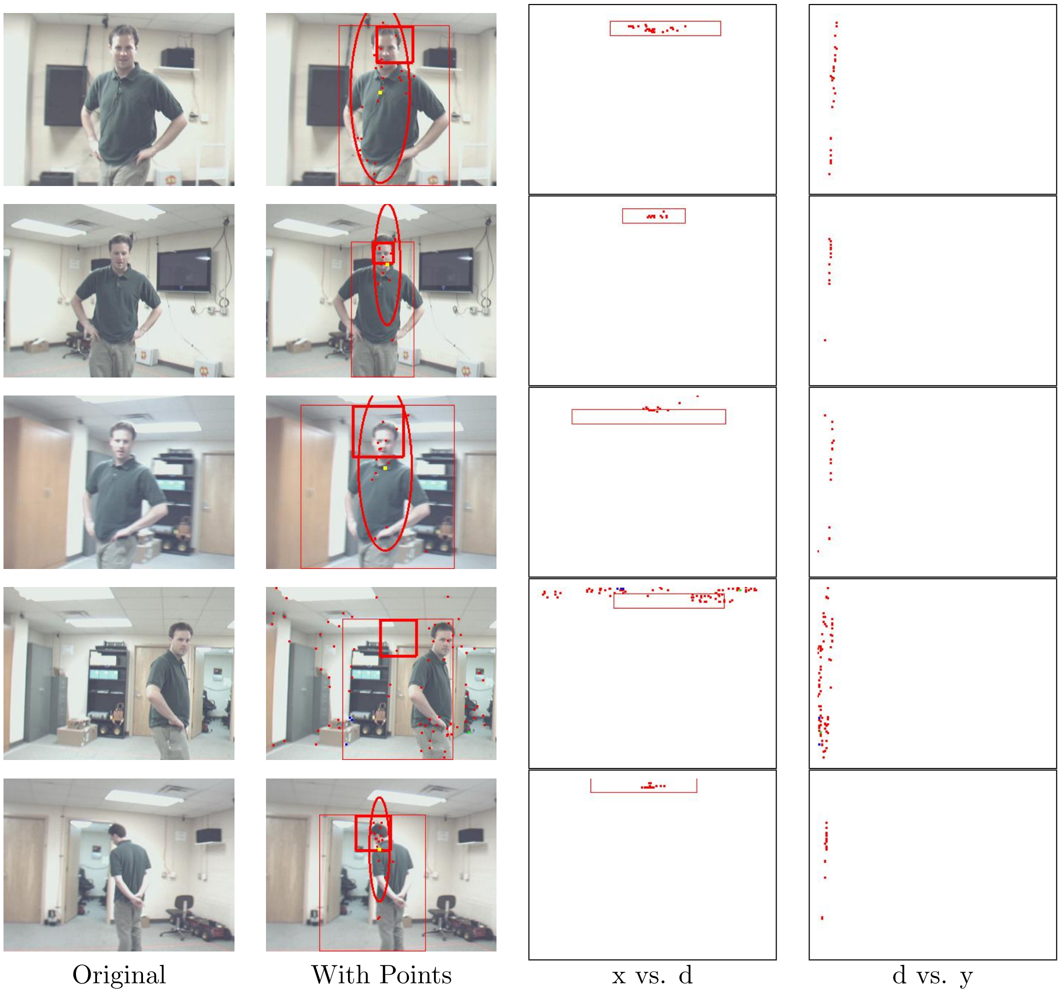


Figure 5.3: Live Tracking Results: Various Situations and Results from a Walk around the Lab.

5.1.3 System on a Moving Platform

Along with testing the system using live video, it was decided to test the system with non-stationary cameras. This is achieved by mounting the entire system on a mobile robot base (see Figure 4.2). The robot is controlled by the system using a simple proportional control scheme described in Section 4.4. To test the ability of the system to detect and track a person during movement of the system, including vertical movement due to uneven surfaces and blurring of the background during turning, the person walked out of our lab, down a hallway, and back again. During this test the person was lost once while rounding a corner, but remained tracked throughout the remainder (Figure 5.4).

Figure 5.5 shows two frames from the robot tracking sequence. The left frame shows the

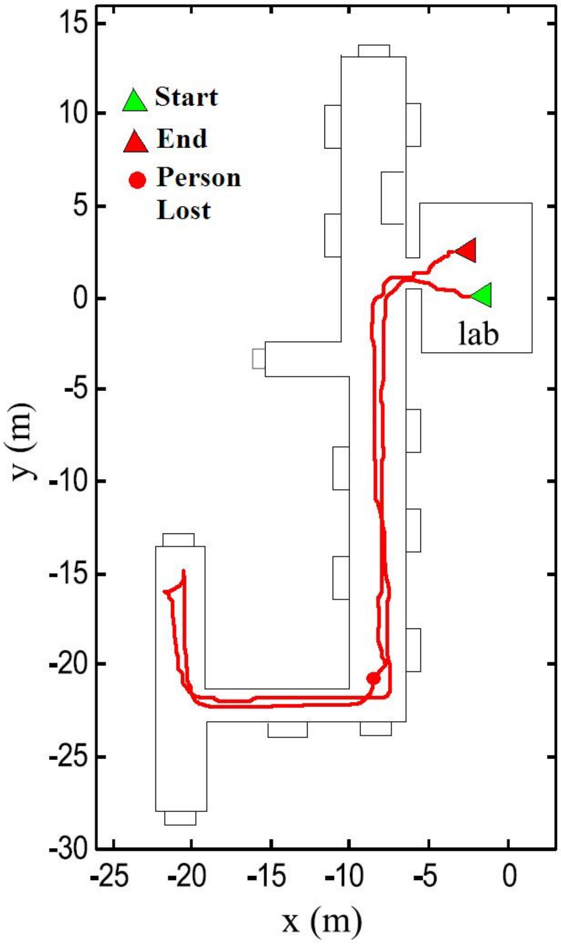


Figure 5.4: Path of the system and robot from the lab and back again.

tracked person against a complex background. The addition of low-contrast clothing increases the difficulty of tracking the person. The right frame shows the same person tracked in a different section of the hallway, this time with another person present.

5.2 Quantitative Results

All of the results presented up to this point have been qualitative. That is to say that they have relied on the subjective evaluation of the reader to determine if the system has performed adequately or not. In this section we present quantitative results from the tracking of a person within

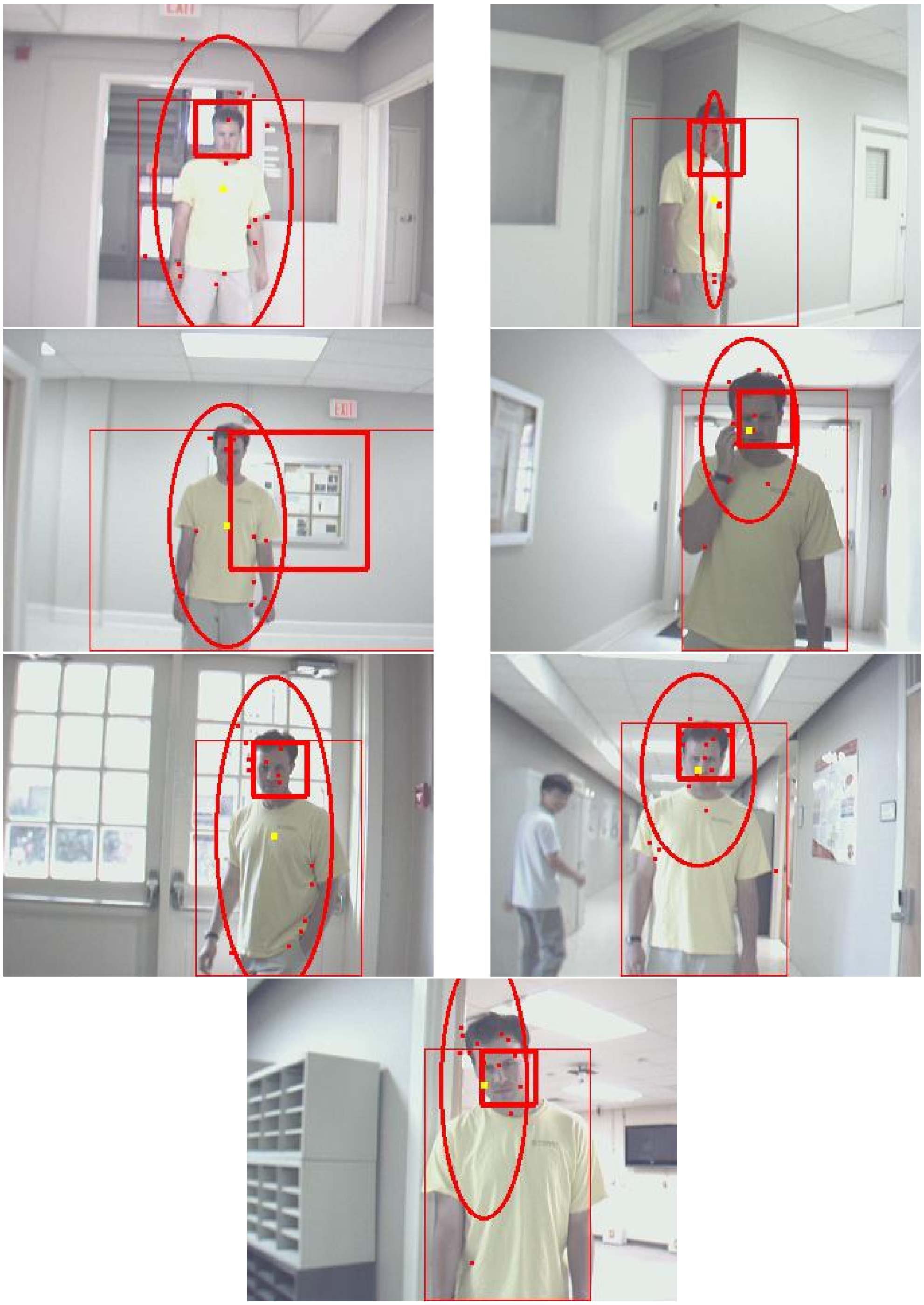


Figure 5.5: Frames from live test through lab hallways shown in Figure 5.4, above. The frames start toward the beginning of the test and are arranged in order from left to right, bottom to top. Note the low contrast between the subject and the background, as well as situations of back lighting and the presence of other people in the scene.

a scene. The following tests were performed on the previously recorded video sequences presented in Figures 5.1 and 5.2, known as the Back and Forth Test and the Occlusion Test, respectively. These tests were not performed on live data due to the challenges of recording the live data and still allowing the system to operate in real time. In each frame of the two sequences, the actual location of the person was selected and compared with the location calculated by the system. The Euclidean distance, in pixels, between these two points was then calculated and defined as the person location error. The following paragraphs discuss the results of these tests and what those results might indicate.

The first set of graphs, presented in Figure 5.6, shows us the person location error rate for each frame during the entirety of each sequence. Note that segments of the graph that show a zero error rate indicate that the person was not tracked during those frames, and therefore meaningful data could not be obtained. In the top graph of Figure 5.6 we can see the error data from the Back and Forth Test. There is a large spike in error immediately following the detection of the person, however this settles out to a reasonable level shortly afterward. Here we can see the error dip toward the middle of the sequence and then rise again toward the end. Because this sequence is of the target walking away from the camera and then back toward the camera, it can be hypothesized that this gradual decrease in error is due to the decrease in the size of the target. This causes the feature points and the actual person center to become closer together, decreasing error. This is supported by the gradual increase of error as the person walks toward the camera again.

The bottom graph in Figure 5.6 shows data from the Occlusion Test sequence. Because the target moves laterally with respect to the camera during this sequence the tracker has more difficulty in maintaining feature points and therefore peaks in location error are far more frequent. In this test the person is not detected for quite some time. As mentioned earlier, this is due to the target being too far away for the face detector to make a positive identification. We theorize that the spikes in this error graph are due to changes in the face location during the running of the face tracker. Even a minor change in face location may lead to points being eliminated through the bounding box criteria, thus changing the calculated person location and increasing error. It can be seen, however, that during the periods of sustained stable tracking toward the end of the sequence error tends to decrease over time. This leads us to believe that with the addition of a more stable face detector very accurate tracking could be achieved.

The second set of graphs, presented in Figure 5.7, show the change in average error as the face detector is run more or less frequently. As stated previously, the face detector is run only once every 10 frames due to the relatively slow speed of the detector compared with the rest of the system. The previously recorded sequences, however, do not require the system to run in real time and therefore allow us to run the face detector far more frequently. In both of the cases presented the face detector is run once every 1, 2, 5, 10, 15, and 20 frames.

The top graph presents the results from the Back and Forth Test sequence. These results are close to what we would intuitively expect. That is, when the face tracker is run more frequently, error is lower than if the tracker is run less frequently. It is interesting to note, however, that the lowest error is achieved at a rate of once every 10 frames. The situation is even more curious when examining the data in the bottom graph. This data is from the Occlusion Test sequence, and is completely counter-intuitive. The possible explanation for both of these aberration is the same. Due to the face tracker's high false-positive rate, it is very likely that when running the tracker at higher frequency, the possibility of an erroneous detection also increases. This would, at best, cause points to be lost and increase the person location error. Running the tracker less frequently reduces the chances of a false detection during one or two frames.

5.3 Tracking Evaluation Approaches

The problem of detecting and tracking people within a scene has generated a lot of interest in recent years. Such a wide range of approaches have been tried, however, that developing a way to test them all against one another has become a problem all its own. The IEEE Workshop on Performance Evaluation of Tracking and Surveillance, or PETS, has approached this problem by giving all interested parties access to a single data set on which to test their system. These data sets have primarily focused on issues with tracking and surveillance in airports, monitoring such things as unattended bags (see Figure 5.8). Krinidis et al. [13] take a slightly different approach and propose a database of audio and visual data as a standard for testing of tracking systems. Their database covers a range of situations, from a single person in a simple scene to multiple people moving in different directions. Each of these are repeated for various lighting conditions.

Both of these attempts to provide a unified data set for testing are valid, however they are not useable for our case because they were recorded with a single, stationary camera. The PETS data set has the added challenge of containing people that are small compared with the size of the frame, as well as very crowded scenes. Since these are not the types of situations in which our system is designed to work, it is doubtful if meaningful results could be achieved. The Krinidis data set shows promise in the sense that the types of scenes represented are similar to those in which we are interested. With the addition of stereo data, such a database would be well suited for our testing purposes.

5.4 Experimental Observations

Overall, the system performed well with both the recorded and live data sets. The results achieved with the previously recorded data were slightly more consistent, but this could be attributed to the relative simplicity of these sequences compared to the live tests. This difference in results could also be attributed to robot movement response times and to false positives from the face tracker. The room in which the system was tested contains various posters and objects mounted to the walls, artificially inflating the false positive rate. Once this happens, a large number of the tracked feature points may be eliminated due to movement of the bounding box, resulting in a greatly increased chance the person will be lost by the system.

These issues, as well possible solutions and improvements to deal with them, will be discussed in detail in the following chapter.

5.5 Algorithm Comparison

Here we will discuss the difference in performance between our algorithm and that presented by Chen and Birchfield in [5], as well as to general color-based methods such as [21, 22, 14, 20].

The methods presented in both this thesis and in [5] are very similar. Both use stereo vision and Lucas-Kanade features to create a sparse disparity map of the image. The difference is in the segmentation of the disparity map. Our method makes use of k-means clustering, whereas Chen and Birchfield rely on motion estimation of both the background and the person combined with Random Sample Consensus (RANSAC) [8]. The motion of the background and the person is accomplished through computation of a 4×4 projective transformation matrix. RANSAC is applied to the background motion estimation in order to reduce outliers and arrive at a consistent estimate. Just as in our algorithm, a Viola-Jones face tracker is used to estimate the disparity of the person.

The strength of the algorithm proposed by Chen and Birchfield is in its ability to detect people in situations where the person's motion is different from the motion of the background, even if the person and the background may be at similar disparities. Our algorithm, on the other hand, does not require relative motion between the person and the background. A person may be detected even if both they and the background are completely stationary.

Based on this information, we can infer that Chen and Birchfield's algorithm would have performed well during the sequence presented in Figure 5.3. During this sequence the person was almost always in motion and alternated between facing toward and away from the robot. We can also infer that their algorithm would be able to handle situations where the motion model for the person and background are similar due to their use of disparity.

A more common method with which we have compared our algorithm is a color-based segmentation method. As discussed in Section 1.1.1, the major problem with color segmentation arises from the restrictions placed on what the tracked target must wear and the environments in which the algorithm can operate. This is due to the fact that color-based algorithms tend to be very sensitive to changes in lighting, or situations in which the color of the person is very similar to the color of the background.

Although the results shown in Figures 5.1, 5.2, and 5.3 all show targets wearing colors significantly different from the background, it can be inferred that a color-based method would have difficulty in these situations if the person were wearing a lighter color. Even with the darker color of the target's shirt in Figure 5.3, the background is filled with objects of various colors that would likely confuse a color-based tracker.

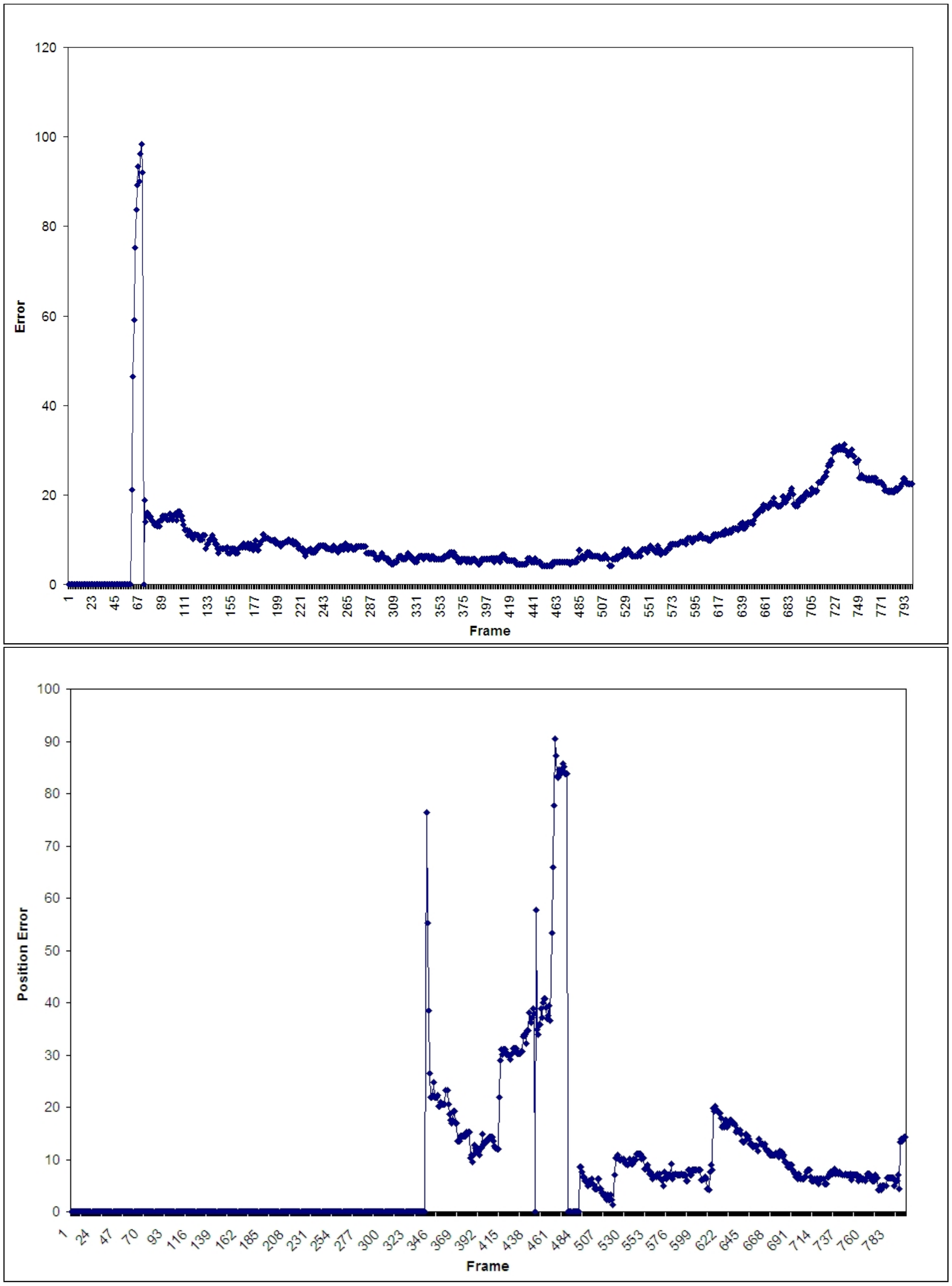


Figure 5.6: Error between the actual person location and the tracked person location, in pixels. The top graph shows data from the Back and Forth Test in Figure 5.1, while the bottom graph shows data from the Occlusion Test in Figure 5.2. Both tests were run with the face detector applied once every 10 frames. Note that segments of the graph that show zero error are frames in which the person was not tracked, and therefore effective data could not be gathered.

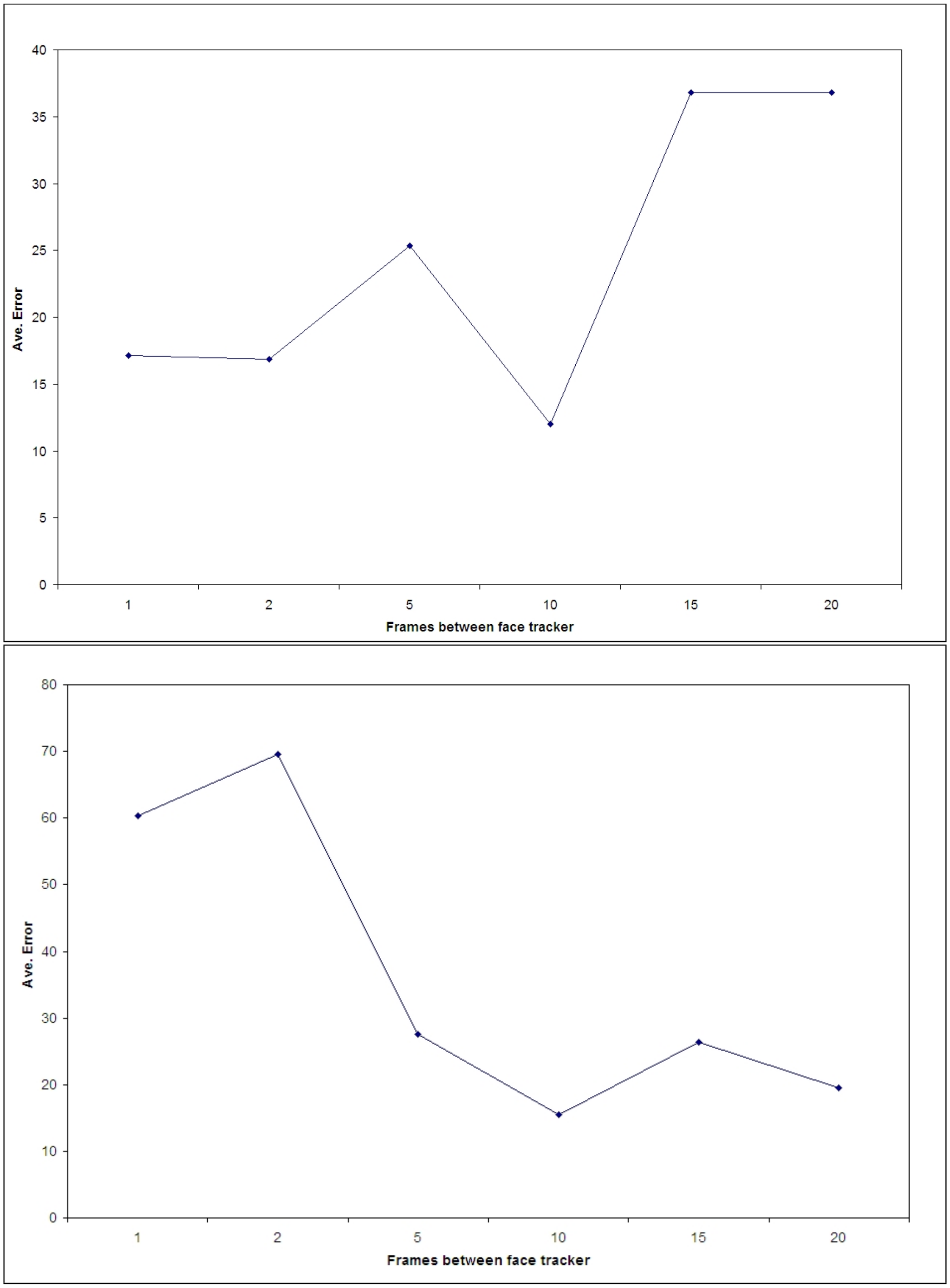


Figure 5.7: Average person location error rates for the same two video sequences with the frequency of the face tracker varied. The top graph represents the change in performance of the Back and Forth Test, and the bottom graph represents the change in performance of the Occlusion Test. The frequency of the face tracker was varied from once per frame to once every 20 frames.

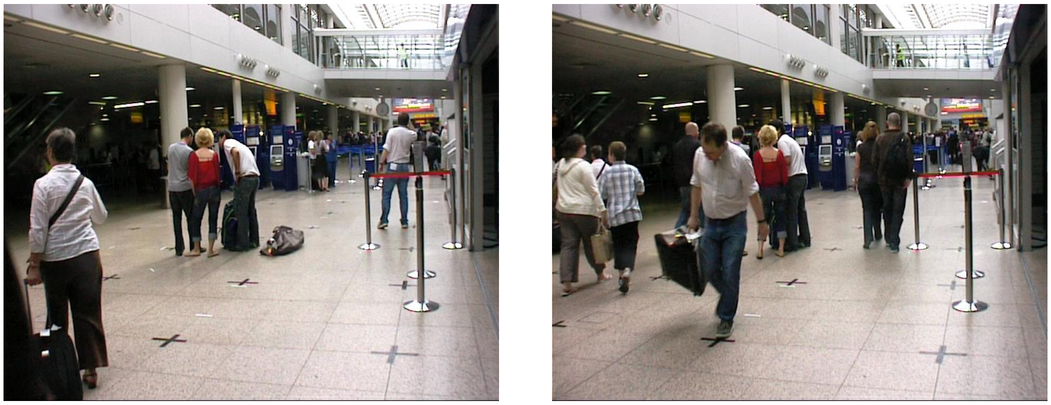


Figure 5.8: Two frames from the PETS 2007 data set. In the left frame the couple in the center has been distracted by a man asking for directions. In the right frame we see a man escaping after having stolen their bag. This is an example of the types of situations the PETS participants are interested in detecting.

Chapter 6

Conclusions and Discussion

The problem of detecting and tracking a person in real time within a dynamic scene is not a simple one. Many methods of accomplishing this goal have been tried, with varying levels of success. The method presented here represents a new variation on these methods: a combination of several approaches, including face detection, feature tracking, and stereo image segmentation. Chapter 5 reviewed the tests we performed with the system, both during algorithm development and afterwords. From these results we see that, while our system performs well in many situations, improvements could be made.

This chapter reviews some possible solutions to problems identified within the algorithm, as well as possible improvements that could be made to the current program. Future work that could stem from this research is also discussed, followed by some final thoughts regarding this project and the future of person detection and robotics.

6.1 Possible Solutions and Improvements

In the previous chapter several problems with our algorithm were identified. These include such issues as detecting a person beyond the range of the face detector, detecting a person when they are at a similar disparity to the background, and losing a person due to false face detection during tracking. Most of these problems are caused by limitations in the methods used. For instance, a person cannot be detected beyond a certain distance from the camera because a face is required for detection and the Viola-Jones face detector cannot detect a face if it is too small. This section discusses each of these problems and proposes possible solutions.

6.1.1 Detecting Distant People

The problem here is that our algorithm is unable to detect people who are too far away from the camera. This stems from the program's reliance on the face tracker to detect a person within the image. Once the face becomes too small within the scene, the Viola-Jones face detector can no longer reliably find faces.

A possible solution for this problem would be to limit the program's reliance on the face tracker. Perhaps a blob-based method could be used on a smaller area to determine if the area represented by each cluster is a person or not. There may also be other, scalable versions of a face tracker that could be used with the algorithm.

6.1.2 Detecting People Near the Background

This is a problem that any disparity-based algorithm must deal with. Because the segmentation of the image relies mostly on the person being a different distance from the camera than other objects in the scene, issues arise in situations were the person may be standing very close to the background. In our algorithm, k-means will be unable to segment clusters belonging to objects very near the background due to a lack of difference in disparity.

One attempt to solve this was to add delta values to the data set processed by the k-means clustering algorithm (see Section 2.3). However, the delta values calculated seem to be too small to cause any great effect in segmenting the image. This problem could possibly be solved by scaling the delta values. This would increase the recorded velocity values of each individual feature point, allowing the clustering algorithm to more effectively differentiate between objects with different object motions. Another approach would be to augment the k-means clustering with a motion estimation method like the one used by [5]. This approach estimates the motion of the background using a 4×4 transform matrix. Once the motion of the background is known, points that match that model can be eliminated from consideration for the person.

6.1.3 Eliminating False Face Detection

One major problem encountered during testing was the loss of the person due to detection of "faces" in the background. This would cause the person bounding box to shift dramatically, possibly eliminating a majority of the points tracked on the person.

The only way to solve this problem completely is to either eliminate the face tracker as part of the algorithm and find another classifier to help differentiate person clusters from non-person clusters. This option was explored briefly in Section 6.1.1. A less drastic step would be to impose a set of criteria on any new faces discovered during tracking. The original intent of placing so much weight on face location was to avoid the possibility of error due to feature point drift. Through testing it seems apparent, however, that the feature tracker is the more accurate of the two. It might be wiser to check that any new faces fall within a reasonable distance of the tracked person before allowing them to eliminate feature points.

6.2 Future Work

As was noted at the beginning of this thesis, the development of a process for detecting and tracking people in real time is only a starting point for future work into expanding the roll of robots and intelligent systems in our lives. This section discusses some possible future work that could be used to improve this algorithm, or to further the applications of such a program.

6.2.1 Other Clustering Methods

The algorithm presented here made use of the k-means clustering process only. This was due, in part, to the simplicity of the approach, as well as to show that the idea has merit. This is, by no means, the only approach that could be used. Section 2.3 discussed two other approaches: fuzzy c-means and Expectation-Maximization, that could be used for this task. It is possible that one of these methods could produce more accurate results than were achieved with k-means. This is especially true given the noise level of the data set being used. It would be very interesting to see what improvement would result from the substitution of either of these alternate methods into the framework outlined in this paper.

6.2.2 Person Classification

In our approach the person was simply classified as a point cluster that met a certain set of simple criteria. This was largely due to our goal of maintaining a simple approach to the problem. As humans, we recognize people as far more than a blob with a face. It seems the logical next step to combine our method with a more accurate and complete person model. Image segmentation through clustering could still provide a way to tell if an object of interest is within the scene, but a more complete person model, perhaps incorporating color, shape, or texture modeling, could replace the face tracker as the sole method of distinguishing person from non-person.

6.2.3 Person Data Collection

During the introduction, the idea of using a robot with person detection and tracking as a security guard was mentioned. With the addition of face recognition, or even face data storage, this could be a possibility. Conceptually, it would be simple to save the image data found by the face tracker, or even data for the entire person. With that information, the robot could keep a record of people it had met and treat new and old people differently depending on the situation. This would also make tracking multiple people within the same scene much easier. With additional functionality like this, robots could more easily plan routes in busy areas, interact with multiple people in a work environment, or alert the authorities if anyone unknown enters its range. Of course, with the development of faster hardware and more intelligent software, many of these ideas could become reality in the future.

6.3 Final Thoughts

This thesis began with talk of the dreams that have driven humanity to develop robots and intelligent systems. The ability to detect and track people is a fundamental step toward making those dreams reality. The ultimate robot is one that can not only perform tasks with speed and accuracy but also work with people on a personal level. Imagine the increase in productivity if an operator could simply tell a robot what it needed, in human terms. Or the increase in safety if a robot could see the people around it and adjust to avoid disasters.

Here we have presented only a small step toward that goal. Even with the issues discussed in the last chapters, this project has shown that it is possible to create a simple, intuitive algorithm for detecting and tracking people in a dynamic environment. Improvements can certainly be made, but we have shown that the idea is possible, and it will be fantastic to see what happens with technology such as this in the years to come.

Bibliography

[1] Intel OpenCV library, http://www.intel.com/research/mrl/research/opencv/.

[2] D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathmatics, 2007.

[3] D. Beymer and K. Konolige. Tracking people from a mobile platform. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2001.

[4] S. Birchfield. Blepo computer vision library, http://www.ces.clemson.edu/∼stb/blepo/.

[5] Zhichao Chen and Stanley T. Birchfield. Person following with a mobile robot using binocular feature-based tracking. In *Proceedings of the IEEE Conference on Intelligent Robots and Systems (IROS)*, October 2007.

[6] G. Chivilò, F. Mezzaro, A. Sgorbissa, and R. Zaccaria. Follow-the-leader behaviour through optical flow minimization. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2004.

[7] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B (Methodological)*, 39(1):1–38, 1977.

[8] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[9] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, pages 148–156, 1996.

[10] T. Inamura, T. Shibata, Y. Matsumoto, M. Inaba, and H. Inoue. Finding and following a human based on on-line visual feature determination through discourse. *Proceedings of the 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 348–353, October 1998.

[11] Wikimedia Foundation Inc. Cluster analysis. http://en.wikipedia.org/wiki/Data_clustering.

[12] Wikimedia Foundation Inc. History of robots. http://en.wikipedia.org/wiki/History_of_robots.

[13] M. Krinidis, G. Stamou, H. Teutsch, S. Spors, N. Nikolaidis, R. Rabenstein, and I. Pitas. An audio-visual database for evaluating person tracking algorithms. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pages 237–240, 2005.

[14] H. Kwon, Y. Yoon, J. B. Park, and A. C. Kak. Person tracking with a mobile robot using two uncalibrated independently moving cameras. In *Proceedings of the International Conference on Robotics and Automation*, 2005.

[15] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, pages 674–679, 1981.

[16] Olaf Munkelt, Christof Ridder, David Hansel, and Walter Hafner. A model driven 3D image interpretation system applied to person detection in video images. *14th International Conference on on Pattern Recognition*, pages 70–73, 1998.

[17] M. Piaggio, R. Fornaro, A. Piombo, L. Sanna, and R. Zaccaria. An optical-flow person following behaviour. In *Proceedings of the 1998 IEEE ISIC/CIRA/ISAS Joint Conference*, pages 301–306.

[18] Yang Ran and Qinfen Zheng. Multi moving people detection from binocular sequences. *Proceedings of the 2003 IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 297–300, 2003.

[19] S. Rougeaux, N. Kita, Y. Kuniyoshi, S. Sakane, and F. Chavand. Binocular tracking based on virtual horopters. *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pages 2052–2057, 1994.

[20] C. Schlegel, J. Illmann, and H. Jaberg. Vision based person tracking with a mobile robot. In *The British Machine Vision Conference*, 1998.

[21] H. Sidenbladh, D. Kragić, and H.I. Christensen. A person following behaviour for a mobile robot. *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*, pages 670–675.

[22] M. Tarokh and P. Ferrari. Robotic person following using fuzzy control and image segmentation. *Journal of Robotic Systems*, 20(9):557–568, September 2003.

[23] Carlo Tomasi and Takeo Kanade. Detection and tracking of point features. Technical Report CMU-CS-91-132, Carnegie Mellon University, April 1991.

[24] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, 2001.

[25] Paul Viola and Michael J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.

[26] Paul A. Viola, Michael J. Jones, and Daniel Snow. Detecting pedestrians using patterns of motion and appearance. *International Journal of Computer Vision*, 63(2):153–161, 2005.