2010

Disjoint Inter-Camera Tracking in the Context of Video-Surveillance

Trevor Montcalm
University of Windsor

DISJOINT INTER-CAMERA TRACKING IN THE CONTEXT OF VIDEO-SURVEILLANCE

by
Trevor Montcalm

A Thesis
Submitted to the Faculty of Graduate Studies
through School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada
2010

DISJOINT INTER-CAMERA TRACKING IN THE CONTEXT OF VIDEO-SURVEILLANCE

by

Trevor Montcalm

APPROVED BY:

Dr. Severien Nkurunziza
Department of Mathematics and Statistics

Dr. Ziad Kobti
School of Computer Science

Dr. Bubaker Boufama, Advisor
School of Computer Science

Dr. Robin Gras, Chair of Defense
School of Computer Science

7 Sept 2010

Author's Declaration of Originality

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

Abstract

Disjoint intra-camera tracking is the task of tracking objects across video-surveillance cameras that have non-overlapping views. Unlike the closely related task of single-camera tracking, Disjoint intra-camera tracking is difficult due to the gaps in observation as an object moves between camera views. To solve this problem, an intra-camera video-surveillance system builds an appearance profile of the objects seen in its camera, and matches these appearance profiles to achieve the effect of tracking.

This thesis demonstrates two novel ideas that improve Disjoint intra-camera tracking. The first is to use a Zernike moment based shape feature for objects observed in a scene, used to describe the shape of an object in a compact, reliable form. The second is to dynamically weigh the Zernike moment shape feature with other standard features to achieve better tracking results. Weighting emphasis is given to better features, more stable features, more recent values of features, and features that have been reliably translated from a different video camera. This weighting is used both during appearance aggregation and object comparison.

This thesis is dedicated to my parents, friends, family and supervisor that all helped to make this work possible.

Acknowledgements

Dr. Bubaker Boufama has been an ideal supervisor throughout my thesis and graduate degree. Patience, understanding and guidance all contributed greatly to this work.

Contents

List of Figures

List of Tables

1

Introduction

Object tracking in the context of video surveillance is the task of detecting and monitoring objects of interest in video surveillance cameras. These objects can be people, pets, automobiles, anything of interest to a security operator monitoring the video cameras. For the relative short history of video surveillance, this task has been performed exclusively by human security operators, often watching multiple camera feeds at once. This is a tedious and error-prone task.

Automated or intelligent video surveillance is the process of using a computer system to assist a security operator in monitoring camera feeds[31]. It can also help with off-line forensic investigations, sifting through hours of security video automatically. A topic of recent research in the past 10 to 15 years, early works first focused on detecting the objects, with later works getting into tracking and even high-level behavioral recognition for scene description. In addition, research started with the basic application of one security camera, expanding to multiple cameras typical of a video surveillance network.

The basic task of object tracking is the matching of objects on a frame by frame basis. This forms the basis of existing single camera object tracking methods. A key assumption

that can be used is the strong temporal consistency between frames. Only a fraction of a second passes between frames, and therefore the objects cannot move very far or change very much. This assumption can also be used for multiple cameras viewing the same scene, where the cameras are overlapping.

Disjoint Inter-Camera Object Tracking is the task of object tracking when the cameras do not share a common scene, that is the camera views do not overlap. Due to this, there is no temporal consistency between cameras. In the non-observed space between camera views, an object can stop, reverse direction, or even change its appearance. These conditions make classical object tracking methods not applicable.

Disjoint Inter-Camera Object Tracking builds on top of existing object tracking methods to track objects across separate camera views. First objects must be detected and tracked within the scope of each camera, or each scene of overlapping cameras, with conventional intra-camera object tracking methods[8]. As objects move between these independently surveyed areas, an inter-camera object tracking algorithm must make decisions on ”who is who” and ”what is what” to the objects that arrive in the scenes.

Comparing frames of video to solve this task is inefficient and noise-sensitive. The common method to undertaken to solve ”who is who” is the method of appearance profile building and matching. Digital profiles of observed objects are built, and then used to compare to newly observed objects. This thesis proposes a new method to build and compare appearance profiles. Contributions include using Zernike Moments as a shape descriptor, and combining this and several other features in a dynamic way as an appearance profile.

Object tracking comes very naturally and intuitively to humans. For computers however, it is a difficult task for various reasons[31]. Occlusion, when objects are concealed by line of sight, is difficult simply because the system cannot see each object independently.

The input image quality of video surveillance cameras is also a source of challenge. They are often noisy, blurry, and have low-resolution, due to their goals of quantity over quality. And finally, like many tasks in computer vision, inter-camera object tracking has the inherit problem that there is not a 100% absolute solution to the task of object tracking. Even humans can be deceived sometimes and lose track of objects in security videos.

The layout of the rest of this thesis document is as follows. Chapter 2 presents research into previous works focused on inter-camera object tracking. Chapter 3 follows with the proposed method chosen for this thesis. Chapter 4 provides some real-life results of the proposed method, with chapter 5 providing a conclusion.

2

Related Work

2.1 Basics of Intelligent Video Surveillance

Throughout the history of intelligent video surveillance, a general pattern has emerged in the manner in which intelligent video surveillance is accomplished. The following sections provide a summary of these steps, along with other notable different possibilities in implementation. First, section 2.1.1 defines the input and output of intelligent video surveillance systems. Section 2.1.2 shows the next stage of processing called Background Subtraction, which classifies each pixel in an input frame into either background or foreground. Next is Blob Formation in section 2.1.3, where foreground pixels are grouped together and filtered. Section 2.1.4 describes intra-camera object tracking, the base single-camera tracking algorithm that inter-camera tracking builds on. Finally, section 2.1.5 describes the multiple camera tracking algorithm that can track objects across disjoint camera views.

2.1.1 Input and Output

The basic hardware for any intelligent video surveillance system is the network of cameras, video monitors and video recording devices. These devices span from the early beginnings of video surveillance with analogue cameras, to the current digital cameras and digital storage options of modern computer hardware. Each camera provides a video stream, commonly called a video feed.

The input to an Intelligent Video Surveillance system is a number of video feeds, just like a human operator watching a number of monitors would see. A video feed is a streaming sequence of pictures, a certain number of them every second. Each picture is called a frame, and the rate at which these pictures are taken is called the frame-rate. Each frame consists of a two-dimensional MxN grid. Each cell in this grid is called a pixel, which can have a single greyscale value for its brightness or multiple brightness values representing its colour.

The output of an intelligent video surveillance system can vary, depending on the goal of the intelligent video surveillance system. It may be a high-level behavioural recognition based description of the scene[17]. It may be a warning system, alerting human operators of certain key-conditions, for example detecting abandoned packages in a public area[3]. Most systems simply output a list of objects and their properties, such as location. This information can then be used for other higher-level purposes.

Intelligent Video Surveillance algorithms have been designed for both calibrated and uncalibrated cameras. Calibrated cameras require initial set up and calibration, but they provide more realistic data values. Calibration is often paired with a homography so that objects true height and location on the ground plane can be computed[23, 30]. Multiple homographies can be computed for precise tracking through occlusions[19]. Work has also

been done on alleviating manual calibration with some automatic elements[38].

Some intelligent surveillance algorithms are designed specifically for stereo cameras[26]. Stereo cameras can provide an additional input value for each pixel of each frame: a depth value. This extra information can be used to help with occlusion and provide more realistic 3D object locations.

Intelligent video surveillance can cover a wide variety of situations and scopes. The most basic case is an indoor, controlled lighting environment that has movement restrictions due to hallways and rooms. Many authors have tackled this case with success, and used the environment to their advantage by exploiting redundancy in pathways[8]. Others have gone further, with open indoor spaces that have variable sun-lit rooms[28], and also into wide field outdoor situations[25, 14]. These areas are more challenging due to their freedom of movement and variable lighting.

In any case, the range of video surveillance is limited only by the combination of a cameras 2D resolution and its field of view. An object must occupy some number of pixels in the image, because of the inherit nature of the video surveillance algorithms input. If the view is very wide or an object is small and far away, the object will be too small to be noticed. If the view is too narrow or the object is too large, the object will occupy the cameras entire image, and no information can be discerned.

2.1.2 Background Subtraction

Background subtraction is the process of classifying each pixel in a frame of video as either foreground or background. Foreground pixels belong to items of interest in the scene, and background pixels are part of the unchanging, static background. This process is done for each frame of video from each camera in the system. This natural parallelism can be

exploited in the design of intelligent video surveillance systems[38].

Background subtraction is also an important step in additional applications such as human-computer interaction and traffic monitoring. Due to this, there are many different background subtraction algorithms, and a complete survey on this topic can be found in [27].

As the name implies, background subtraction involves building some sort of background model and then subtracting the current frame of video from that background model to derive a foreground/background image. The simplest example of this is frame differencing[28]. The background model is simply the previous frame. When subtracted, any pixel with a non-zero value (changed pixel) is counted as foreground, and pixel with a zero value (not changed) is background. This method has a low complexity, but is sensitive to noise and loses foreground information too quickly. To illustrate, if a large object with a uniform colour is moving slowly, only the edges of the object will change pixels for each frame, and an outline of the object will appear instead of its full shape.

A step up from this are algorithms that build a true background image in some statistical manner. This background image is the same as if the camera's field of view were empty of objects. This solves the problem of time with frame differencing: an object, even if not moving or moving slowly, will still be detected. The hard part is building a background image. One method is to compute a median of each pixel over many past frames[1]. Thresholding or clustering algorithms can be applied to the subtracted image to give a more stable pixel classification.

The most prominent background subtraction algorithm is the Gaussian Mixture Model[25]. This algorithm models each background pixel by a number of guassian values. It is highly resistant to shadow and background artifacts. This method has been further improved by

using a variable number of gaussian values per pixel[40].

In some special cases, background subtraction may not be suitable as a first stage pixel classifier. In the specific case of low-light nighttime tracking, a contrast based method can work better[11].

2.1.3 Blob formation

Blob formation is the process of deciding which groups of foreground pixels in the background subtracted image could be objects in the scene, and which are noise. Though it has varying names[10], in this thesis each group of foreground pixels called a blob. Its input is a foreground image mask from background subtraction, and its output is a list of blobs and the pixels each blob is comprised of. Blob formation is generally split into three steps: Morphological Operations, Connected Component Analysis, and then size filtration[6, 1, 35, 31].

Morphological operations smooth out the subtracted image, removing the very fine details of objects that are noise or background subtraction artifacts. This entails first applying a dilation operation, followed by an erosion operation[1]. Mask sizes can vary depending on the resolution of the camera, though they typically are either 3x3 for a fine grained approach, or 5x5 for more blurring.

Next a grouping operation like Connected Component Analysis (CCA)[1] is applied. This operation is the main blob-forming operation, deciding which foreground pixels belong together as one possible object (blob). CCA is the most popular choice, though a Bayesian classifier is possible[26].

Foreground pixels are not the only choice for grouping. In [11], the foreground image is divided into small blocks. Each block containing some foreground pixels is compared with their neighbours, forming groups. In effect, this is CCA applied to the foreground blocks

instead of individual foreground pixels.

Size Thresholding is the last operation. It simply trims out blobs that are too small to be considered as individual objects[1]. These blobs are mostly noise blobs or fragmented blobs from an occluded real object in the scene. In the latter case, trimming out these smaller blobs is not of concern since there is most likely a larger blob for the rest of the object.

2.1.4 Intra-Camera Object Tracking

Object tracking is the task of matching blobs on a frame by frame basis to decide which blobs are real object, and which blobs are just noise. The simplest example is using a Kalman filter and comparing the location of the blobs. If a blob has moved by only small distance, it is considered the same real-life object. This works because the time between frames is very small ($1/framerate$), the objects cannot move very far during this time.

More complex is the idea of using multiple pieces of information, or features, to help in matching blobs[1, 25]. This can provide a more robust set of information to use for blob matching. A list of commonly used features in intelligent video surveillance is provided in section 2.2. How these features are compared and used is discussed in section 2.2.3.

Object tracking defined in this way is performed on a per-camera basis. This data association can be exploited to scale up to larger and wider camera networks[8]. Each camera can have a dedicated processing unit responsible for all the object tracking duties, minimizing traffic on the interconnecting network.

2.1.5 Inter-Camera Object Tracking

Inter-camera object tracking is the task of matching objects between cameras that have a disjoint (separate) field of view[14, 15, 13, 8, 6, 29, 5, 23, 16, 28, 30, 4, 18]. Inter camera tracking works on top of the aforementioned single camera object tracking methods[15, 14, 8]. The input to an inter-camera tracking algorithm is the set of objects observed in each camera. The output of an inter-camera tracking algorithm is a set objects that can exist in any camera, called global objects in this thesis.

Explicit or implicit, most authors use the concept of features to accomplish inter camera object tracking. By comparing features of objects, the system can make intelligent decisions about what objects are the same between cameras, and which are different. In essence, the task of inter-camera tracking therefore becomes the task of solving ”who is who” and ”what is what” across cameras.

Defined in this way, inter-camera tracking is very similar to intra-camera tracking discussed in Section 2.1.4. They differ only in scope of time (frame-rate based versus variable time) and in applicability of features.

2.2 Features for object tracking

There are many possible features used to describe objects. This section will cover some well-known and reliable examples like colour histogram and location, and also cover some more unique features. Most papers only use two to five features[30], and some only use one[16].

Features can be evaluated in a number of ways. The most basic approach is to simply use each feature equally[1, 28]. Other more advanced methods exist, like giving empha-

sis to certain features as matching statistics are built up[8], or using a probability density function[13].

To start, paragraph 2.2.1 describes some basic geometric features in use for object tracking. Paragraph 2.2.2 then describes some appearance features like colour histogram. Finally, paragraph 2.2.3 describes the ways in which features are used to match objects in both single camera tracking and in inter-camera tracking.

2.2.1 Geometric Features

This section describes features that are based on geometry. Location is the best example of a geometric feature. In intra-camera object tracking, it is also the most reliable and intuitive feature. Location is also often used as a key output of a tracking algorithm, answering the question ”where are the objects?”

Location can either be a 2D pixel location[6] or an actual real-life location on the ground plane of the scene being observed[30]. 2D location is most often computed as the centroid of an object[1]. The reason for this over the more obvious ”centre of the bounding box” approach is that the centroid is more stable during object deformation, for example when a human walks with swaying arms.

3D location requires calibrated cameras and a homography computed for the cameras[30]. Also possible is using multiple cameras to view the same scene, allowing full 3D coordinates to be extracted from the observed objects[21]. Papers of this type tend to focus on occlusion handling, rather than extending tracking to cover disjoint scenes like inter-camera object tracking.

Velocity is another geometric feature in use[1]. Similar to using a Kalman filter, the velocity is an interpolated feature, based on the length of time between frames and an

average of some sort of the difference between successive object locations. Velocity can be a good distinguishing feature for object tracking due to the laws of physics. An object, in the time between frames, cannot change their velocity by very much. Similarly for inter-camera object tracking, if the views are close enough this property still holds to a lesser degree.

Location is a very accurate feature for single camera object tracking, however this is not the case for inter-camera tracking. The exit location of one camera does not necessarily correspond to the entry location of another camera, due to the separate nature of the camera views. Even if a separation distance is known, an object can fool the tracker by simply taking a different path.

Nevertheless, location and velocity can still be useful features for inter-camera tracking. Often location is combined with time information[30]. This time and location information can be used to exploit redundancy in paths or environmental restrictions between the cameras[8] and predict the velocity if the distance between the cameras are known. Also, if these distances are small, location and time can be strong enough to be the sole matching feature of an intelligent video surveillance system[30]. Location can also be a good choice for a first stage filter, leaving other more CPU intensive and precise features to finer inspection[39].

2.2.2 Appearance Features

Appearance features are features that describe an objects visible appearance. Appearance features provide a digitized answer to the question ”What does the object look like?”. The largest of all the feature categories, appearance features cover a wide array of methods to record an objects appearance. The first paragraph starts off with some basic shape features.

Next is the Colour Histogram feature, a very popular and often used feature that describes an objects colour. Colour and Shape Blend describes some more advanced appearance features that blend and fuse colour and shape into unique appearance descriptors. Finally, Texture Data introduces an appearance feature that uses the texture information of an object.

Basic Shape

Basic shape features in this thesis are defined as any shape feature that can be represented by a single numerical value. The most common and prevalent example of this is the height feature. Both height in pixels[1] and height in real-world units[23] are used in object tracking. Width is another shape feature used[1]. Like height, it can be pixel based or real world, depending on the camera calibration of the video system. Lastly, the size feature represents how large the object is in the scene, represented by the number of pixels belonging to the object[1].

Height and width are stable features when the objects observed have a rigid structure. Motor vehicles are a good example of this trait. If the objects can deform, these features tend to loose their value. In the case of tracking human beings, width is slightly worse than height due to arm motion as a human walks. However, both can easily be invalidated if someone bends over or sits down. Size is more tolerant of this effect, but not immune.

Feature discernability for basic shapes is highly dependent on the camera's view and size of objects. Due to their simple one-number nature, basic shape features tend to work well for objects with large differences in height, width, or size. Unfortunately, this is not usually the case in many real-world scenarios. Camera viewing angle and proximity influence these features much more than actual differences in the height, width and size of objects in reality.

For this reason, basic shape features are not well suited for use as primary features for matching. They are typically used as part of an overall feature vector[1], or as a first stage, rough filter that is not treated as accurate[12].

Colour Histogram

The colour histogram feature of an object is a way to describe the overall colour distribution of an object. Each pixel that belongs to an object is placed into a certain colour bin. These bins are then normalized.

The colour histogram is rotation and scale independent. This is because the colour histogram only records what colours an object has, and not where they are. Scale independence is achieved by normalizing the bins: the histogram records the colour ratio, and not the actual amounts. Comparing histograms is done with standard histogram comparison methods like Correlation and Chi-Square distance.

The colour histogram is the most often used and most relied upon feature in object tracking[1, 6, 13, 5]. It is a highly stable feature, emphasized by the fact that most objects do not change their colour significantly during observation by a video surveillance system. It also has a strong discernability between different objects. This is evident by the fact that objects being tracked in video surveillance applications can have a wide variety of colours. In addition, even with similar colours, the colour distribution can vary. An example of which is two people both wearing white shirts: one is long sleeve, the other is a T-shirt.

There are negative aspects to the colour histogram. Though steady to the human eye, variable lighting and shadows can change the observed colours of an object. These slight changes can lead to objects being miss-matched and incorrectly identified. In [24], objects are assumed to be human, and multiple ellipses in a human shape are used to filter out

shadows. In addtition, in the realm of multiple cameras, each camera may have a different colour response due to it's manufacturing process. This only adds to the slight variation of colours objects may have.

To solve this problem some authors have chosen to reduce the colour sensitivity of their intelligent video surveillance algorithms. In [6], Consensus-Colour Conversion of Munsell colour space (CCCM) is used to reduce all of RGB space to just 11 basic colours. (Similar to a pack of crayons.) This very rough colour space is used only initially in tracking to tolerate large differences in lighting between separate cameras. An advantage to this reduced colour space is robustness against noise and illumination changes, along with decreased computational requirements. A side effect of using a colour space this coarse is that objects with somewhat similar colours become grouped into the same category. (Light green and dark green become just green.)

Another technique is to only use the colours that primarily dominate an object. [23] uses K-means clustering on the colours in an objects histogram. This produces a simplified colour histogram that trims out noisy and errant colour values. This technique offers a more middle-ground compromise between full RGB spectrum and the limited CCCM colour space. In addition, instead of a universal reduced set of colours, each reduced set of colours is computed for the object being observed, leading to a more true representation.

For the specific issue of different inter-camera colour sensitivities, some authors have chosen to compute Colour Transfer Functions that translate the colours observed in one camera, to the equivalent colours in another camera. Colour Transfer Functions are described in detail in section 2.3.2.

Colour and Shape Blend

Some authors have chosen to create new features that blend shape properties with colour properties. These new features are often the sole feature in use during tracking, since they blend a variety of information. When to name a feature as new, or instead as a unique combination of existing features, is often a subjective matter of interpretation.

One example of such a feature is presented in [18]. Each object has a circle placed around it. Then, for a series of N control points placed on this circle, concentric circles are expanded around each point into the object. Each ring has a gaussian model for the colour, and a counting of the number of edge pixels. The number of control points N and the number of rings around each control point can be varied. The feature is scale, rotation and translation invariant by normalizing the data values and using enough control points.

A similar idea is used in [4]. This paper deals exclusively with tracking human beings. An edge detector is first used on the pixels that comprise an object, and then a number of feature points are placed along the edges uniformly. For each feature point, a 5x5 pixel window is drawn around it, and the first k dominant colours are recorded, along with the angle and distance in comparison to the persons head (topmost edge of object).

Texture Data

Texture can be used as a final decider between similarly shaped objects with similar colours. In [39], Local Binary Patterns are chosen as the method of texture representation. A histogram of the LBP is used as the sole feature for tracking. LBP was chosen due to it's capability for texture recognition and low complexity, since in this work the system runs directly on a programmable Digital Signal Processor (DSP).

2.2.3 Using the features to match

Once there is a way to represent an object, there must be a way to compare the representations of objects to decide ”who is who”. This is done by comparing features and combinations of features. In single camera tracking, features from blobs in each frame are compared to the features from known objects in the scene. In inter-camera tracking, features from previously visible objects are compared to the set of objects in all cameras.

Different features can be used in different tracking situations. For example, geometric and basic appearance features like colour histogram can be used for single camera tracking, while inter-camera tracking may only use appearance features. This may be due to little geometric correspondence between the cameras. For example, an outdoor area where the cameras are far apart.

In many works, there is only one all-encompassing feature to represent objects[16]. Often a large and complicated feature, a comparison method will also be described for the feature. Most of the computational work is done during feature comparison, with less work being devoted to sorting out match scores and object assignment.

In contrast, other methods have many features that are combined and weighed to give an overall score, again with each feature having its own comparison method[1, 8]. All the features may be averaged together for an average score, or certain features may contribute more or less to the average score.

Regardless of method, the overall match scores are used to make intelligent decisions on ”who is who” for the objects in the scene. Most authors use some kind of reasoning in their matching to enhance the tracking system with basic human knowledge, an example being how the same object cannot appear in two different locations at the same time. Some authors accomplish this using probabalistic models[18, 8]. Some simply use a matching

matrix, and find the best candidate on a frame by frame basis[1].

More advanced, in [12] a complex neural cost function system is employed to perform the object tracking. Well formulated, the system design is in two stages. The first stage uses a bipartite graph with a simple cost function of the features of height, width, and colour histogram. This first stage acts as a filter to efficiently trim the search results down. Secondly, a Self-Organizing Map is used to perform a more detailed comparison, able to detect subtle differences and use them for accurate tracking. This second stage must be hand-trained before use.

2.3 Challenges of Intelligent Video Surveillance

Object tracking in video surveillance can be a difficult task even for a human operator. Occlusions can hide objects away from view temporarily, differences in colour can change appearances of the same object in different views, low-quality cameras can make objects too blurry to distinguish, and sometimes cloud changes or lighting changes can even cause good cameras to have temporary lapses in quality. This section explains these challenges and the ways authors handle them.

2.3.1 Occlusion

Common to all object tracking, occlusion is when a foreground object is hidden from view. Occlusion is a frequently occurring problem, with many research papers being devoted entirely to this topic. There are two types of occlusion: foreground and background. Foreground occlusion is when two objects are in the same location from the camera's point of view, for example two people crossing paths. Background occlusion is when a foreground

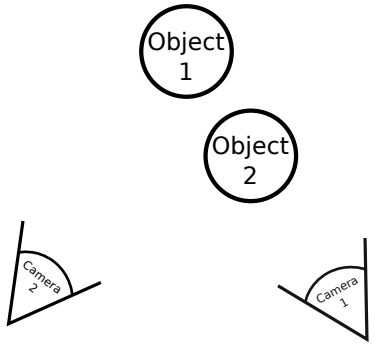

Figure 2.1: Occlusion: Camera 1 has objects occluded, while view of Camera 2 sees two distinct objects

object is hidden by a background part of the scene, for example a person walking behind a wide tree trunk.

The simplest form of occlusion handling is using a Kalman filter on an objects location and velocity [1, 18, 9]. This helps in two ways. First, foreground occlusion can be predicted with the video surveillance system noting that occlusion is happening. Secondly, and more important, the path of objects can be estimated throughout the occlusion, and correct identities can be recovered thereafter. For most purposes, this works well, but the system can be fooled if objects change their trajectory during the unobservable periods of occlusion.

Certain environmental conditions of the intelligent video surveillance system can be used to help with occlusion. In [7], an edge detector and Bayesian classifier are used to pick out the familiar shape of a humans head and torso. The reasoning behind this is that most often, even with heavy occlusion, a persons body may be occluded, but not their head and torso. The method in [7] also assumes an indoor environment with cameras mounted at ceiling level.

Others have used multiple cameras viewing the same scene to overcome occlusion [21, 38]. The idea behind this is simple: if one camera sees two objects occluded, a different camera viewing the same scene at a different angle will not. (Figure 2.1) By combining

multiple views together into a common 2D ground plane homography, simple features like location can still be used to reliably track objects.

In [19], this is taken one step further with the idea of using multiple scene planes (ground plane homographies). Each ground plane has separate occupancies, effectively changing from one 2D location for each object to several 2D locations for objects. The idea behind this is that objects (humans) might change shape, but individual limbs and torso will not occupy the same space.

Stereo cameras, if applicable, can be used to help sort out depth information during occlusions[26, 10]. Using this information occlusions can be more readily handled, similar to using multiple cameras viewing the same scene.

2.3.2 Camera Colour Response

Even with identical hardware, the colour appearance of an object in one camera may not be the same as in another camera, highlighting the issue of varying colours between cameras. Figure 2.2 shows an example of this effect. A number of approaches have been taken by authors trying to mitigate this effect and it's results on object tracking.

Previously mentioned, one approach is to use a reduced colour space[29, 23]. On top of this, in [5] an Incremental Major Colour Spectrum Histogram Representation(IMCSHR) is used. This simply widens and normalizes the MCSHR for greater tolerance of inter-camera colour changes.

Another idea is to translate the colours from one camera to another, proposed independently by many authors. Referred to by different names, in this summary they are collectively called Colour Transfer Functions.

To start, in [6] and [8] a matrix is used as a colour transfer function between cameras.

Figure 2.2: Difference in colour response between two cameras. This hallway has the same color of paint, yet the cameras pick up much different colors.

It is an RGB quantization matrix, initially set to the identity matrix where no colours are changed. As objects are matched between the cameras, Singular Value Decomposition is used to compute a more accurate colour translation matrix between the two cameras. Named Inter-Camera Colour Calibration in this paper, this approach has no initial training required and can be incrementally updated with time, adjusting to changing conditions.

A slightly more advanced approach has been taken in [15] and [14]. Called the Brightness Transfer Function (BTF), it is formulated from the concept of Bi-Directional Reflectance Distribution Function from physics. This method requires explicit hand-matching of objects between cameras to train and initialize the BTFs. A probability function is then built that represents the amount of brightness difference between each view in each colour channel. These functions depend on a number of factors like object material properties, scene luminance, and camera parameters, a high number of dimensions. Principal Components Analysis is then applied on the set of all functions for a particular pair of cameras and colour channel, reducing it's number of dimensions to a lower number.

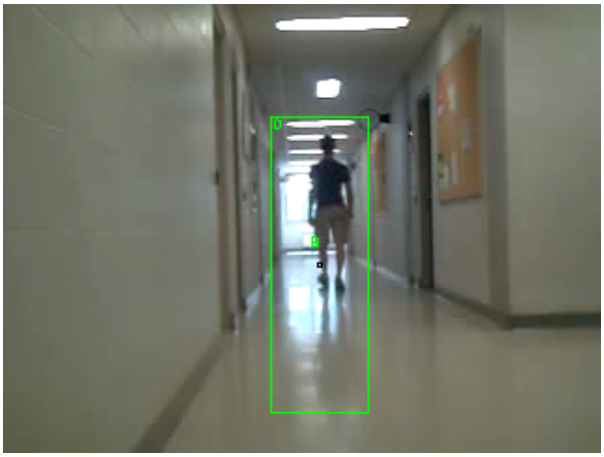

Figure 2.3: Poor image quality. Notice the low-resolution and blurred appearance of the walking man.

Extending this work, in [5] the name Colour Transfer Function is used. The eigenvalue decomposition of the colour component mean and covariances between matches is computed for an initial transfer function. Iteratively improving the CTF, Least Median of Squares and variable weighting is used to built the final more robust CTF.

2.3.3 Poor Image Quality

Intelligent video surveillance algorithms often do not have good quality input. Figure 2.3 shows an example, aquired from a low-quality handheld video camera. This section describes some of the methods authors have used to overcome poor image quality in a video surveillance systems input.

An often encountered situation in intelligent video surveillance is low light, nighttime tracking. This is a class of video input where there is little contrast and mostly darker colours, with little colour differences. To overcome this, in [11] a local contrast algorithm is used instead of background subtraction. In addition, foreground areas are not formed per-pixel as in most background subtraction methods, but rather in m$\times$n sized blocks.

Image quality in video surveillance can be sporadic and variable. In [23], a Bayesian classifier works to identify major errors in image foreground and background segmentation. Using this information, the object profiler builder can trim out bad frames that do not represent the object, and build a more robust and true profile of each object.

2.4 Conclusion

This chapter has presented a survey of the methods used for object tracking. Both the more conventional intra-camera and newer inter-camera methods have been presented. A bottom-up layout was used, starting with the hardware common to all intelligent video surveillance systems like cameras, video monitors, and video recording devices. Background subtraction methods were then presented, along with the blob-formation algorithms that distinguish possible objects on a frame-by-frame basis. Intra-camera object tracking was then introduced, which compares blobs over a series of frames to decide whether or not they represent true objects. Finally, inter-camera object tracking works on top of existing intra-camera methods to decide if appearances of objects in one camera are previously seen objects in another camera.

All existing methods must distinguish objects in some manner. Though terminology varies, the data used to distinguish objects is called features in this survey. They can be as simple as 2D location, or as complex as texture representation and shape descriptors. The most common and robust feature found was the colour histogram, describing the colour distribution of an object.

Some common challenges of video surveillance were also presented in this chapter. Occlusion is when objects are partially or fully hidden from view due to line of sight. Difficult and very common, entire papers have been devoted solely to resolving this challenge. An-

other challenge, notably for inter-camera object tracking, is differences in camera colour response due to lighting or manufacturing differences. This can make colour comparisons between cameras less beneficial. Finally, common to all of intelligent video surveillance, poor image quality arises when the cameras being used are low-resolution or low optical quality. These can generate poor images that hinder an object tracking methods ability to distinguish objects.

Many varied methods have been implemented in the relatively short history of intelligent video surveillance. With this information, the proposed method presented in this thesis has used the best ideas and portions from other works, along with new methods that form the contribution of this thesis. The next chapter presents this proposed method.

3

Proposed Method

3.1 Overview

This chapter describes the choices and innovations applied to the state of the art in Disjoint Camera video surveillance. The proposed method is a combination of all the best components from the research undertaken along with additional functionality that forms the core of this thesis work. With this in mind, the steps and processes involved follow the same pattern as mentioned in section 2.1. First, section 3.2 explains the base algorithm steps like background subtraction and blob forming. Next, section 3.3 explains the single camera object tracking method used in each camera view, with section 3.4 explaining the features computed and used in this step. Finally, section 3.5 explains the proposed Inter-Camera Object Tracking method.

3.2 Basic Algorithm

The basic algorithm for Inter-Camera Object Tracking takes frames of video from a single camera and produces blobs, per-frame, that might represent objects. This involves two steps: Background Subtraction and then the blob-formation step. This section describes the chosen algorithms to accomplish this task.

3.2.1 Adaptive Gaussian Mean Mixture Background Subtraction

Using the results of a recent survey of background subtraction methods[27], the Adaptive Gaussian Mean Mixture method was chosen as the ideal background subtraction algorithm[40]. This algorithm builds on top of previous Gaussian Mean Mixture works. To fully explain the algorithm, some background on Gaussian Mean Mixtures is needed.

First proposed in [32], Gaussian Mean Mixture background subtraction works by representing the background model as a mixture of gaussians, one mixture for each pixel:

$$\sum_{m=1}^{M} \pi_m K(x;\mu_m,\Sigma_m)$$

where M is the number of gaussian components, π_m is the estimated weight of the m^{th} guasian at time t, K is the kernel function in use, x is the pixel value at time t, μ_m is the mean value of the m^{th} gaussian at time t, and σ is the covariance matrix of the m^{th} gaussian at time t. Multiple guassian components are chosen to better reflect the reality that multiple surfaces can appear in a pixel. K is a gaussian probability density kernel:

$$K(x;\mu_m,\Sigma_m) = \frac{1}{(2\pi)^{\frac{D}{2}}} \frac{1}{|\Sigma_m|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu_m)^T \Sigma_m^{-1}(x-\mu_m)}$$

where D is the dimension of the data, μ are the estimates of the means and σ are the

estimates of the variances that describe the Gaussian components. The estimated mixing weights denoted by π are non-negative and add up to one. For computational speed reasons, it is assumed that the red, green and blue channels each are independant and have the same variances, leaving:

$$\Sigma_m = \sigma_m^2 I$$

where I is the identity matrix.

To update the models for new data, the following recursive equations are used:

$$\begin{aligned} \pi &\leftarrow \pi + \alpha(o^{(t)} - \pi), \\ \mu &\leftarrow \mu + o^{(t)}(\frac{\alpha}{\pi})\delta, \\ \sigma &\leftarrow \sigma + o^{(t)}(\frac{\alpha}{\pi})(\delta^T\delta - \sigma^2) \end{aligned}$$

where δ is the difference between the current frame and the mean, and α is the set learning rate of the system. The background model is represented by the largest B clusters. The value of α determines how fast a foreground object that is now stationary becomes the background. A sample value for α is $1/T$, with T being the time adaption period (number of frames).

Adaptive Gaussian Mean Mixture background subtraction adds onto this method a way of dynamically determining the number of gaussian modes to use. The primary reason for doing this is because the number of gaussians is equal to the number of clusters in K space, which is the number of different surfaces that can appear in a pixel. In reality, this number is never a constant amount, and this method of background subtraction aims to better match

the number of surfaces to gaussian modes.

The main equation modified in Adaptive Gaussian Mean Mixture background subtraction is the updating equation for the mixing weights:

$$\pi \leftarrow \pi + \alpha(o^{(t)} - \pi) - \alpha c_T$$

The new variable c_T represents the ratio of the number of frames necessary for a surface to be considered present over the time adaptation period. When this amount is exceeded, an additional gaussian mode is added into the background model. The math for c_T is derived from using Dirichlet priors on the mixing weights [40]. In their implementation, they define c_T as 0.01.

After each update, the mixing weights π are normalized to one. To start, only one gaussian mode is used. This, in effect, causes the first frame processed to become the background model, as it is the largest (and only) mode.

3.2.2 Blob Forming Algorithm

Based on [1], the Blob Forming algorithm consists of three separate steps applied sequentially to the background subtracted image: Gaussian smoothing, Connected Component Analysis, and Blob merging with size thresholding.

Gaussian Smoothing

The first operation performed on the background subtracted image is a gaussian smoothing operation. This eliminates the fine grained noise from the background subtracted image and produces a smoother image that better represents the objects of interest. The gaussian kernel is a 5 by 5 pixel size. After this operation, pixels will now be spread into various

greyscale values. A linear threshold is then applied. Any pixel that is below the greyscale value of 150 is set to the background (0), else the pixel is set to the foreground (255).

This approach differs from [1] in that morphological operations are not used. It was found that morphological operations tend to inflate and destroy smaller elements of detail in the image. As an example, if a 3x3 dilation and then a 3x3 erosion is performed, this effectively lowers the resolution of the foreground objects to 3x3 blocks. Smaller details can be completely eliminated or artificially inflated to 3x3 size. For this reason the Gaussian smoothing operation with a simple constant threshold was used.

Connected Component Analysis

The largest and most obvious step of blob formation is to group the foreground pixels together into regions. Like many other authors, Connected Component Analysis was used to perform this action.

A naive, recursive algorithm best describes how Connected Component Analysis works. Each pixel is examined for neighbours. If a pixel has a neighbour to the immediate left, right, up or down, this region grows to include that pixel. Recurse on the pixels neighbours to discover more pixels that belong to this region.

The above procedure describes 4-way connectivity, where only the direct 4 neighbours of a pixel can contribute to building a region. 8-way connectivity considers a pixels full 8 neighbours. Any neighbour can contribute to the region. 8-way connectivity produces a lesser number of larger regions.

The recursive Connected Component Analysis algorithm is not an optimal choice. Due to its recursive nature, the key operation of determining if a pixel should be added to the current blob (*add*) can be performed multiple times on the same pixel. The system stack

Algorithm 1 Connected Component Analysis

```
1:  for all pixels p do
2:      if p is foreground ∧ p not in a region then
3:          Increment region number
4:          RecursiveCCA(p)
5:      end if
6:  end for
7:
8:  RecursiveCCA(p):
9:  set p to current region number
10: for all pixel p neighbours n do
11:     add = p is foreground ∧ p not in a region
12:     if add then
13:         set n to current region number
14:         RecursiveCCA(n)
15:     end if
16: end for
```

size can frequently be exceeded due to its deep recursion on large foreground regions. This was discovered with an implementation of this algorithm, to provide a correct reference for other implementations to compare against. Other alternatives exist, such as the Two-Pass algorithm described in [34].

Blob merging

Blob merging is the process of merging together blobs that are considered close enough. This is done to overcome background subtraction faults and hairline separations of the same objects. An example of this is when a person walks behind a staircase with thin, metal poles: the blob gets split into several pieces only a couple of pixels apart.

Before any merging takes place, a blob size filter is first applied to clear out any blobs that are too small to be considered real objects. Any blob that has less pixels than 0.1% the camera's number of pixels is trimmed out.

The operation of merging blobs together is relatively straight-forward: Simply re-label the blobs pixels as belonging to the same blob. The real challenge is therefore the process of deciding which blobs to merge. Two conditions are checked to decide if blobs should be merged. For each blob, an encompassing bounding box is computed:

- If the blobs bounding boxes overlap, merge.
- If the blobs are ”close” in both the x-axis and the y-axis, merge.

”close”, similarly, is defined by meeting either of the following conditions:

- If the blobs are within 3 pixels of each other
- If the blobs are within 25% of the width/height

The implementation of these rules works by applying these conditions and merging blobs iteratively until the number of blobs becomes stable.

3.3 Intra-Camera Object Tracking

As mentioned in section 2.1.4, Intra-Camera Object Tracking is the task of matching blobs on a frame by frame basis, discovering which blobs correspond to real life objects and tracking their movement through time. This can also be referred to as Single Camera Object Tracking, due to the fact that blobs belong to a single camera. It can be thought of as the task of solving ”Who is Who”.

The Intra-Camera Tracking algorithm is the next step in the pipeline of processing after blob-formation. It has only 2 inputs: The list of existing objects, the list of blobs in the current frame.

Like many other works, the single camera object tracker works by comparing blobs in the current frame to existing objects that are known to be in the scene. This follows a similar pattern set forth in [1]. Features are used to represent properties of objects and blobs. Grouped together, they are called a feature vectors, representing the overall appearance of an object or blob. Feature Vectors are what's used to facilitate comparison between objects and/or blobs. A complete description of the features used and how they are compared is provided in section 3.4. What's important to note is that a pair of feature vectors can be compared to generate a score of how similar they are. In this work, a score is normalized between the values of 0 (least similar possible) and 1 (exactly the same).

To begin, the Intra-Camera Tracking algorithm first computes the feature vector of each blob in the current frame. In addition, for each existing object a matching feature vector for comparisons to blobs is built. This matching feature vector is the average of the past 5 observed feature vectors, excluding the location feature. Then the similarity score between each blob/object combination is computed and stored.

These two operations form the bulk of the processing required by the Intra-Camera Tracking algorithm. Fortunately, these tasks are easily parallelized. Each blob is independent, and as such the feature vector computation can be spread across threads. After this, each object/blob comparison is independent, and again can be spread across threads. These two techniques can be used to help speed-up this process.

Beyond the feature vector computation and similarity comparison, the Intra-Camera object tracking algorithm is mostly a conditional logic computation. Six different object scenarios are possible:

- A new object appears
- An object continues to be tracked

- An object disappears
- Multiple objects occlude
- Multiple objects merge
- An object splits into two objects

The algorithm presented in this thesis is simpler than other works due to its focus on Inter-Camera tracking. As such, each scenario isn't explicitly handled. The sixth item "An object splits into two objects" is simply handled by the first "A new object appears" scenario. These scenarios are explained in the following:

New Object

An object is decided to be a new if a blob has no match to any existing object. Due to noise and errant blobs, following this rule verbatim produces a very large number of errant objects. To prevent this, a new object is created with the state "possible". If this "possible" object is successfully tracked for a number of frames, the object is labelled "verified" and is considered a real-world object.

Continue Object

The most typical scenario, this is when an existing object is matched to a blob and continues to be tracked. This blob must only be matched to this one particular object (else there is some kind of occlusion), and this blob must be similar enough to be considered a match. The hard threshold is that the similarity score must be greater than 0.85.

Lost Object

This is when a real-life object has left the cameras vision. Note that this is possible by either leaving the field of view or by moving through a door. Like the New Object scenario, if this rule is followed verbatim every time an object doesn't match a blob high enough, or is occluded with a background element for only a single frame, the object will cease to be tracked. Again a two-state solution with a threshold is applied: First objects with no match are assigned the "lost" state. If the object is not matched to a blob for 10 consecutive frames, the object is then considered truly lost and removed from memory.

Object Occlusion

When multiple objects best matched blob, with a high enough similarity, are the same, this is labelled as object occlusion. This state can also be assigned with each objects bounding box overlap. The overlap must be 60% or higher, and the object being overlapped is not currently matched high enough to any other blob.

Object Merging

An example of this is two people finding each other and then walking together. Using the information learned so far in this document, at first the two objects will be labelled as occluding. (Either due to matching to the same blob, or via significant overlap). Unlike regular occlusion, these objects will not separate. Eventually the Intra-Camera Object Tracker needs to decide when these objects are now together. This is accomplished with a constant threshold of 50 frames (1.8 seconds). An arbitrary object is picked as the single label, and the other objects are considered lost.

Object splitting

Objects can split in a variety of ways, one being a person drops a briefcase and walks away. As mentioned before, this case is relatively simple in that a new blob will be formed for the new object.

With all of these scenarios, many conditions arise that aren't explicitly mentioned in this text. For example, if a "lost" object is matched successfully again to a blob, the object then becomes valid again. Noting all of these small conditions in textual form is not feasible due to the size of this paper. Instead, the following pseudocode diagram of the entire Intra-Camera Tracking process should provide enough detail:

Algorithm 2 Intra-Camera Tracking

```
1:  for all blobs b do
2:      Compute Feature Vector of b
3:  end for
4:  for all blob and object combination do
5:      Compute similarity score
6:  end for
7:  for all Objects o do
8:      b =Find Blob with highest similarity
9:      if b ≥ 0.85 then
10:         Label object o with b as best match
11:     else
12:         Label object o with no match
13:     end if
14: end for
```

During times of foreground object occlusion, two or more real-life objects overlap each other. When this happens, both objects get merged into one large, single blob. A simple approach to handle this scenario was employed. Occlusion is only predicted using Kalman filters on the objects position and velocity. The Intra-Camera Tracking algorithm will then label the objects as occluding until the period of occlusion is over.

For background object occlusion the time threshold for lost objects works well enough. During the period of occlusion the object will be considered lost, but its position will be updated with a Kalman filter so that after occlusion the object will still match up. (Assuming a constant velocity.)

Occlusion handling is a difficult enough task in itself that many works are dedicated to its solution. For the purposes of this thesis however, which has a focus on Inter-Camera tracking, the simple prediction and Kalman filter approaches work well enough. For more information on occlusion, see section 2.3.1.

3.4 Chosen Features

A wide variety of features were chosen to represent the appearance properties of the objects tracked. This was done in theme with the variable weighting of features ideas this thesis presents. Without a large number of features to select, the variable weighting effects are diminished.

In order to facilitate comparison between different objects and blobs and across different camera views, each of the features is normalized. The range chosen depends on the feature in question, but most follow a range of 0 to 1, a percentage range. This also allows features to be added and subtracted easily, an important operation when combining multiple feature vectors together to build an aggregate feature vector for an object. See section 3.5.3 for more information.

In each of the following descriptions of features, d represents the difference value between features, and the subscripts x_a and x_b represent the two features to be compared. $height$ and $width$ are the x and y dimensions of the cameras image sensor.

The following sections describe each of the features used in this thesis: Location (3.4.1),

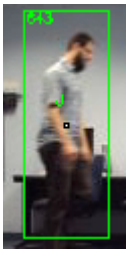

Figure 3.1: Location of an Object. Notice the small white pixel surrounded by a black border.

Velocity (3.4.2), Height, Width and Size (3.4.3), Colour Histogram (3.4.4), and finally the Zernike Moment Shape (3.4.5).

3.4.1 Location

Simple and intuitive, location is a 2D value representing where an object is located in the camera's field of view.Figure 3.1 shows an example of the location of an object. It is labeled as the small white dot near the center of the object.

An objects location could be set to one of the objects bounding box corners, or the centre of the bounding box, however this gives an unstable and unrealistic measurement of location. As an example, a person raising one of their hands, or when holding a broom horizontally. Instead, like other works[1], the location of an object or blob is its centroid:

$$\mathbf{c} = \frac{\sum_i^N \mathbf{x_i}}{N}$$

where N is the number of pixels belonging to the object and x_i is the location of each pixel.

Initially, this value is a integer pixel value for the x-axis and the y-axis, unsuitable for comparison between different cameras. To normalize, divide the pixel value by the dimensions of the camera:

$$x = \frac{x_p}{width}; y = \frac{y_p}{width}$$

where x_p and x_p are the pixel based location coordinates.

Comparison is simply the euclidean distance between values:

$$d = \frac{\sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}}{\sqrt{2}}$$

The value in the denominator is to ensure that the output of the difference value is scaled from 0 (no difference) to a maximum of 1 (most different possible).

3.4.2 Velocity

Velocity is the 2D rate at which an object moves, in pixels per frame. This value is then normalized to the size of the camera view. Unlike location, velocity cannot be directly measured at each frame of video. It is a derived feature, taken as the average difference in location over a number of previous frames:

$$x = \frac{\sum_i^N x_{pi}}{N * width}; y = \frac{\sum_i^N y_{pi}}{N * width}$$

where x_{pi} is the pixel based x-location of the object, in previous frame i, and similarly for y. This formula also normalizes the velocity to values between -1 and 1. Velocity comparison is done using the euclidean distance operation, similar to the location feature:

$$d = \frac{\sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}}{2\sqrt{2}}$$

The notable difference from the location feature is the denominator has a larger value,

to compensate for the velocity features wider range of values.

3.4.3 Height,Width, and Size

Height, Width and Size are pixel-based one-dimensional features used like other previous works. Height and Width are the averaged bounding box height and width of the object or blob, and size is the number of foreground pixels that belong to the blob or object. This was chosen over using bounding box area for it's higher accuracy.

They are crude descriptors in that they are simple to compute and can vary widely. Regardless, they are included in this thesis to provide a way to describe the overall size of an object.

Like the location and velocity features, height width and size are normalized to the size of the cameras view:

$$h = \frac{h_p}{height}; w = \frac{w_p}{width}; s = \frac{s_p}{height * width}$$

where h_p, w_p and s_p are the height, width and size of the object or blob in pixels. The difference function for these features is simply the absolute value of their difference:

$$d = |x_a - x_b|$$

where x could be any of height, width or size.

3.4.4 Colour Histogram

As noted in section 2.2.2, Colour Histogram is a good feature that describes the overall colour distribution of an object. An example showing the color histogram for an object (as

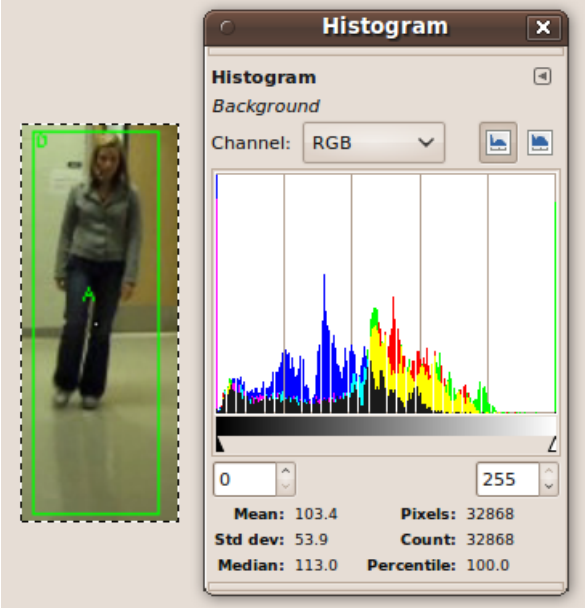

Figure 3.2: Colour Histogram of an Object

computed by image software GIMP[33]) is provided in Figure 3.2.

For this thesis, a 16-bin histogram for each of Red, Green and Blue (RGB) colour channels was used. Each of the three histograms is normalized so that the sum of the bins adds to one:

$$\sum_{i}^{16} h_i = 1$$

where h_i is the value of histogram bin i. The reason RGB was chosen over more specialized colour spaces is because of the colour transfer functions used between different cameras in this thesis (Section 3.5.2). RGB has a higher performance in the presence of these functions[8].

For comparison, the Bhattacharyya method of comparing histograms is used:

$$d_c = \sqrt{1 - \sum_{i}^{16} \frac{\sqrt{a_{ci} \cdot b_{ci}}}{\sqrt{\sum_{i}^{16} a_{ci} \cdot \sum_{i}^{16} b_{ci}}}}$$

where a and b are the two histograms being compared, i is histogram bin, and c is the colour channel.

The average of all three colour channels is used to obtain a final difference:

$$d = \frac{d_r + d_g + d_b}{3}$$

Initially the correlation, the chi-square and the intersection methods of histogram comparison were tested. The Bhattacharyya distance was found to have marginally better results than other histogram comparison methods, with the added benefit that it natively outputs normalized values.

3.4.5 Zernike Moments

During the research portion of this thesis, it was noted that most of the works in existence do not use the concept of shape very much. Humans watching a video screen can use shape as a strong discriminator. As an example, consider black and white, low-quality security camera video feeds surveilling a scene. Each of the moving objects in the scene (most likely people) can often be distinguished quite easily, yet there is no colour information. Due to the multiple camera nature, location also isn't a clue. A human can still recognize and track the objects.

Some authors have tried to capture this property and use it to help distinguish and track objects, as noted in section 2.2.2. The common technique to these works is that a large, complicated single feature is built, combining colour and shape together. In contrast, for

this thesis an independent descriptor that solely describes the objects shape was desired.

Zernike Moments were chosen to be a shape feature, based on a survey of common ways to represent and compare shapes in image processing[37]. Historically used in optics[36], Zernike Moments were first used as image moments in 1990[20]. The deciding factor in choosing Zernike Moments was a 2003 survey that compared various image moments[37]. Zernike Moments were considered best overall for a variety of shapes. They are rotation, scale, and translation invariant, good properties for comparison. Their bases are orthogonal on the unit disk, leading to no information redundancies in their values.

A Zernike Moment is a single number that represents a particular aspect of an images shape on the unit circle. Computing this number is accomplished by the integrating over the unit circle the corresponding Zernike Polynomial:

$$Z_{n,m} = \int_0^1 \int_0^{2\pi} R_{n,m}(\rho) e^{jm\phi} \quad \text{and} \quad R_{n,m}(\rho) = \sum_{k=0}^{\frac{n-m}{2}} \frac{(-1)^k (n-k)!}{k!(\frac{n+m}{2}-k)!(\frac{n-m}{2}-k)!} \rho^{n-2k}$$

where ρ is the radius and ϕ is the angle in polar coordinates. Each combination of m and n leads to a different Zernike Moment. Due to the fraction inside the factorial, only combinations that lead to integers are used.

Any number of zernike moments could be used to represent the shape of an object. The lower order moments describe the overall base shape of an object, and the higher order moments represent the detail. Too few, and there is not enough detail to truly represent the object. Too many, and the computation of these moments becomes too expensive. A median value of 49 zernike moments ($n = 0 \cdots 12$) was chosen[22]. (Table 3.1.)

An efficient Zernike Moment calculation algorithm was chosen for computing the shape feature[2]. This algorithm rearranges the calculation such that previous mid-calculation

Table 3.1: Zernike Moment Orders

Order	Moments	No.
0	$Z_{0,0}$	1
1	$Z_{1,1}$	1
2	$Z_{2,0}$, $Z_{2,2}$	2
3	$Z_{3,1}$, $Z_{3,3}$	2
4	$Z_{4,0}$, $Z_{4,2}$, $Z_{4,4}$	3
5	$Z_{5,1}$, $Z_{5,3}$, $Z_{5,5}$	3
6	$Z_{6,0}$, $Z_{6,2}$, $Z_{6,4}$, $Z_{6,6}$	4
7	$Z_{7,1}$, $Z_{7,3}$, $Z_{7,5}$, $Z_{7,7}$	4
8	$Z_{8,0}$, $Z_{8,2}$, $Z_{8,4}$, $Z_{8,6}$, $Z_{8,8}$	5
9	$Z_{9,1}$, $Z_{9,3}$, $Z_{9,5}$, $Z_{9,7}$, $Z_{9,9}$	5
10	$Z_{10,0}$, $Z_{10,2}$, $Z_{10,4}$, $Z_{10,6}$, $Z_{10,8}$, $Z_{10,10}$	6
11	$Z_{11,1}$, $Z_{11,3}$, $Z_{11,5}$, $Z_{11,7}$, $Z_{11,9}$, $Z_{11,11}$	6
12	$Z_{12,0}$, $Z_{12,2}$, $\cdots$, $Z_{12,8}$, $Z_{12,10}$, $Z_{12,12}$	7

results can be reused:

$$Z_{n,m} = \frac{n+1}{\pi} \sum_{k=\|m\|}^{n} \beta_{n,m,k} \cdot \chi_{m,k}$$

where

$$\beta_{n,m,k} = \frac{(-1)^{\frac{n-k}{2}} (\frac{n+k}{2})!}{(\frac{n-k}{2})!(\frac{k+m}{2})!(\frac{k-m}{2})!}$$

and

$$\chi_{m,k} = \sum\sum_{x^2+y^2 \leq 1} e^{-jm\theta(x,y)} \rho(x,y) f(x,y)$$

where *n*,*m* are parameters of the Zernike Moment, $f(x,y)$ is the pixel value at x and y, ρ is the distance from the centroid, and θ is the angle from the +x axis. β is independent of any pixel values, and can be computed and tabulated before the program even starts. $\chi_{m,k}$ changes for each shape, and can be computed by replacing the double sum with a single

sum over all pixels that are part of the foreground:

$$\chi_{m,k} = \sum_{i}^{M} e^{-jm\theta(i)} \rho(i)$$

where M is the number of foreground pixels for this blob/object (same as size feature), θ is the angle between pixel i and the +x axis, and ρ is the distance of pixel i from the object/blob centroid (location feature). $\chi_{m,k}$ can be tabulated as higher Zernike Moment orders are computed for the same blob/object. To facilitate normalization, the pixel with the furthest distance from the centroid is set as $\rho = 1$, and all other pixel distances are a fraction of this distance.

Zernike Moments are normally compared using euclidean distance[22]. Due to small differences between object shapes, the Bhattacharyya distance was used instead. See the previous section 3.4.4 for more information.

3.4.6 Feature Vector Algebra

Feature Vectors are used in more ways than just comparing differences. As the following sections will show, there is a need to add, subtract, and divide feature vectors like normal real-valued numbers. Though different in size and name, but still all number normalized from 0 to 1, they can be treated a just an array of floating point values. This is how entire feature vectors can be added and subtracted from one another. Of note, the subscript f_i does not refer to an individual number in this 104 element array, but rather a named feature like location or histogram.

3.5 Inter-Camera Object Tracking

Inter-Camera Object Tracking is the task of deciding ”who is who” for objects that leave and reappear between separate camera views. This goal requires the surveillance system to recognize an objects appearance, in contrast to Intra-Camera tracking where the continuous viewing of objects in the scene lends spatial cohesion. The spatial discontinuity is what makes Inter-Camera tracking a difficult task.

The same features used in Intra-Camera tracking are also used for Inter-Camera tracking. Similarly, feature vectors are compared and used to make decisions on ”who is who”. What makes the Inter-Camera tracking algorithm different is its inclusion of more than just frame-by-frame feature vectors. An object can appear many times, in different cameras with different properties, for varying lengths of time.

A core contribution in this thesis is the Inter-Camera Tracking method. Previous works might use a single and complex matching feature, or features that are simply averaged. This thesis presents a very dynamic framework where features are emphasized or diminished individually for better matching. The intuitive concepts of feature stability, more recent appearances, and reliably translated features all factor into the math that governs how objects across different cameras are matched.

This section will follow a bottom-up approach to describe the Inter-Camera Object Tracking algorithm. First, section 3.5.1 describes the weighting factors that govern which certain features or appearances are used. Section 3.5.3 describes the process of building a single feature vector for a previously tracked object. Section 3.5.2 describes the process of how feature vectors computed in one camera can be translated to feature vectors in another camera. Section 3.5.4 describes how the appearance profile feature vectors are compared, and finally section 3.5.5 ties everything together describing the overall Inter-Camera object

tracking algorithm.

3.5.1 Weighting Factors

The following section provide mathematical formulas for feature weighting ideas described previously. These values are used throughout the Inter-Camera Object Tracking process in various algorithms.

Stability Factor

Implicitly, an objects appearance doesn't change. A feature that is unstable does not represent an objects appearance very well, and should be diminished, while stable feature further emphasized. The stability of a feature could be computed using statistical variance, but this requires re-computing the mean. In this thesis, the variance of a feature is simply the average of all differences:

$$v = \frac{\sum_{i=1}^{N-1} |f_{i+1} - f_i|}{N-1} = \frac{\sum_{i=1}^{N-2} |f_{i+1} - f_i|}{N-1} + \frac{|f_N - f_{N-1}|}{N-1}$$

where f_i is the value of feature f in frame i. This allows for the variance to be incrementally built frame by frame. Though not a true variance, the measurement is close enough to serve it's purpose. The quality of a feature is built from this value with the simple formula: $q = 1 - v$.

Age Factor

Mainly application to appearance profile building, the age factor ensures an objects recent appearances are emphasized more than older ones. In addition, the age factor ensures that there is a limit to how many appearances can used.

Table 3.2: Age Factor values

N	i=1	i=2	i=3	i=4	i=5	i=6	i=7
1	1	0	0	0	0	0	...
2	0.55	0.45	0	0	0	0	...
3	0.55	0.24	0.21	0	0	0	...
4	0.55	0.24	0.11	0.10	0	0	...
5	0.55	0.24	0.11	0.06	0.04	0	...
6	0.55	0.24	0.11	0.06	0.04	0	...
7	⋮	⋮	⋮	⋮	⋮	⋮	⋱

Mathematically, this is implemented as a time function $t(i,N)$, a function of the number of frames relative to the last frame and the total number of frames. Table 3.2 shows the values for this function, ensuring that for any value of N, all of the $t(i)$ values added up to N total to 1. It also ensures that any i value beyond 5 is not used.

Camera Translation Quality

The Camera Translation Quality describes the amount of matching history between a pair of cameras. Initially, all translation qualities are set to a low value, since no objects have been matched across. After several matches, the quality improves, and asymptotically approaches one.

$$ctq = 1 - \frac{1}{Ax + B}$$

$$B = \frac{1}{1-c} \text{ and } A = \frac{10 - B}{N}$$

where x is the number of matches, c is the initial camera quality (zero matches), and N is the number of matches required to reach a 90% level.

3.5.2 Feature Vector Translation

This section describes the process of translating a feature vector built from one camera to another. The main reason for this functionality is to overcome the problems associated with tracking among differing cameras. Different placement, viewpoint, and camera hardware can all change an objects appearance.

This section follows the work of [8], with the exception that entire feature vectors are translated, not just colour. There is value in translating location for instance, to exploit the redundancy in paths taken by objects between cameras. Simple shape features like height, width, and size compensate for the scene location of cameras. And most importantly, colour histogram feature is translated to compensate for differing colour sensitivities in different cameras. Over time with many matches, the camera links will record and use this evidence to help with matching feature vectors.

Between each pair of cameras there exists a Camera Link that stores the information used to translate feature vectors. This information includes a vector of feature transfer functions along with a stored number x indicating the number of successfully matched objects contributing to these functions. The following describes how each feature is translated:

Location Location is translated to exploit the fact that many objects travel in predictable patterns, in line with what other authors have done with geometric matching [30, 28]. To compute the location translation vector, an average of the difference between exit and entry locations $\mathbf{a}$ of each object is computed:

$$Tl_{A\to B}(\mathbf{l}) = \mathbf{a} + \mathbf{l}$$

where

$$\mathbf{a} = \frac{\sum_i^N \mathbf{l}_{Bi} - \mathbf{l}_{Ai}}{N}$$

$l_A i$ is the location value for in camera A for match i. To reverse the direction $Tl_{B\to A}(\mathbf{l})$, simply negate vector **a**.

Velocity Velocity is transformed to compensate for changes in orientation between the cameras. The velocity transfer function has the same formula as for location, simply substitute v for l.

Width, Height, Size These one dimensional features are transformed to compensate for distance and viewing angle between two pairs of cameras. Camera A might observe an object very close up and large, and the next camera may observe the same object but from far away, with a very different size.

$$Tf_{A\to B}(f) = f + a$$

where

$$a = \frac{\sum_i^N f_{Bi} - f_{Ai}}{N}$$

f and f_{Ai} are one of width, height or size for camera A and match i, and a is the averaged difference value.

Histogram A 16-bin colour quantization matrix is used to translate colour histograms[8]. For demonstration purposes, a 2-bin quantization matrix is shown below:

$$\begin{bmatrix} A_{r1} & A_{r2} & A_{g1} & A_{g2} & A_{b1} & A_{b2} \end{bmatrix} * T$$

$$\simeq \begin{bmatrix} B_{r1} & B_{r2} & B_{g1} & B_{g2} & B_{b1} & B_{b2} \end{bmatrix}$$

where T is the 6x6 transformation matrix. (If the histogram were only 2-bins for each colour channel, the above matrix would be in use.) Singular Value Decomposition (SVD) is used to find the matrix T. Initially, T is set to the identity matrix I, to act as a pass through. Each time there is a match, a new T_m match matrix is computed, and then merged into the existing matrix T:

$$T = \alpha T + (1 - \alpha) T_m$$

This formula is applied for each element in T. Higher values of α update T slower, and lower values of α update T more quickly. For this thesis, a value of 0.75 is used. For the reverse direction, the inverse T^{-1} is used.

Shape Due to the translational, scale and rotational invariance of the Zernike moment, no translation function was implemented.

3.5.3 Object Appearance Profile

An objects appearance profile is the set of all previous appearances of an object. This is stored as a single feature vector for every camera appearance, the average of all feature vectors observed during the appearance (excluding location). The primary task with this data is to build a single feature vector that can be used for comparison. This feature vector is called the matching feature vector, tailored to a specific camera in order to facilitate matching to intra-camera objects currently in view.

To build a matching feature vector of an object, previous appearances in other cameras

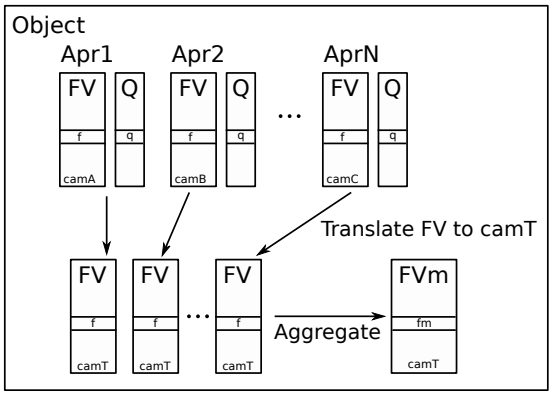

Figure 3.3: **FVm** computation

must be translated to the current camera. The translations will have varying quality (ctq), based on the number of successful object matches. There may also be a large number of existing appearances (t), and only the more recent should be taken into account. Finally, some features observed in the objects past appearances may be very unstable, and should be filtered out.

All of these principles are used in the building of matching feature vectors. Figure 3.3 shows this process.

- N is the number of appearances for this object
- $camT$ is the target camera to be translated to
- f_i is one feature of a feature vector, for appearance i
- f_m is one feature from the matching feature vector **FVm**
- **Q** is the quality vector of an appearance (feature stability)

First the feature vector of each appearance is transformed into the target camera. Then,

for each feature f_m in **FVm**:

$$f_m = \sum_i^N f_i \cdot s_i$$

where s_i is the overall weighting factor for this appearance and feature. s_i is computed by:

$$s_i = \begin{cases} 1 & \text{if } N = 1; \\ \frac{a_i}{T} & \text{otherwise.} \end{cases}$$

The reason for this added complexity is to ensure that the total of all weighting factors is summed to 1, to keep the overall **FVm** in balance. Finally:

$$a_i = t_i \cdot clw(camApr, camT) \cdot q_i \text{ and } T = \sum_i^N a_i$$

where t_i and q_i are the time factor and quality of feature j for appearance i. The function $clf(camA, camB)$ is the camera link weight for the camera link that binds the current appearances camera ($camApr$) and the target camera ($camT$).

3.5.4 Comparing Object Profiles

In the Intra-Camera tracking portion of this thesis, feature vectors are compared by simply averaging the differences between all the feature vectors. For Inter-Camera tracking however, a more complex method of weighing features based on the three weighting factors described in section refsec:weightingFactors is used.

An overall weighting vector **Wm** is used to weigh the features in comparison. (The *m* stands for matching.)

Table 3.3: Quality function for Features

Feature	Formula
Location	$qf(n) = \frac{1}{5}n$
Velocity	$qf(n) = \frac{1}{3}n$
Height, Width, Size	$qf(n) = \frac{1}{5}n$
Histogram	$qf(n) = 1$
Shape	$qf(n) = 0.5$

$$w_j = \frac{b_j}{T} \text{ and } T = \sum_j^M b_j$$

where w_j is an element from **Wm**, corresponding to feature j. This step is to ensure that the total of all w_j add up to 1. M is the number of features, T is total sum for feature j, and b_j is the unscaled weighting factor:

$$b_j = \sum_i^N t_i \cdot qf_j(clw(camApr, camT)) \cdot q_i$$

where N is the number of appearances, t_i and q_i are the time factor and quality of feature j for appearance i. Appearances in the same camera as the unmatched objects are not included in this average. In the event that there are no appearances from other cameras, a default camera link weight equal to c is used. (Section 3.5.1)

The quality function qf_j is a per-feature function that describes how reliable a feature is for a given camera link quality. Certain features can work poorly without proper camera link statistics, and some work well. This function represents this concept. Table 3.3 describes the function.

Most features are dependent on the camera quality, being weighed less when the transfer function is poor, and more as statistics are built up. The Zernike Moment is already independent and has the constant function of 0.5. The histogram feature, even though de-

pendent on the camera transfer function, is still of value in the absence of statistics and is a constant 1.

The final difference score D is computed by:

$$D = \sum_{j}^{M} \text{diff}_j(A_j, B_j) \cdot w_j$$

where A_j and B_j are feature j of the feature vectors A and B, diff_j is the difference function for feature j. Inverting this value, the similarity score is calculated as:

$$S = 1 - D$$

3.5.5 Overall Algorithm

The primary task in the Inter-Camera Object tracking algorithm is deciding, upon a new object being observed in a camera, whether it has been seen before (previously known object) or is a new object altogether.

To begin, for each previously observed object, an appearance profile is built for the camera the new object is in. (Section 3.5.3). Each appearance profile is noted as FV_m, the matching feature vector. In addition, a weighting vector W_m is built for each previously observed object. Then, a similarity score using both FV_m and W_m as described in section 3.5.4 is calculated and stored.

The highest matching score is chosen from the list of previously observed objects. If this score is lower than a threshold, (no good match), the object is considered new. If this score is higher that a threshold, it is considered a previously observed object, and a new appearance is noted.

In the Intra-camera tracking section, the decision threshold is a constant value of 0.85.

For the Inter-Camera tracking algorithm, a dynamic threshold is used:

$$t = ThresLow + a \cdot (ThresHigh - ThresLow)$$

where t is the threshold value. $ThresLow$ and $ThresHigh$ define the range of values the threshold can have, set to 0.60 and 0.85 respectively. a is the scaling factor, set to the average camera link quality between the unmatched object in a camera and the cameras of all the appearances the object has made. This enables the tracking system to start with a low threshold, and becomes more precise as statistics are built up. Apperances that are in the same camera as the unmatched object are excluded from this average.

Intuitively, the moment an object in a camera is ”new” is the first frame that object appears. The above algorithm could be applied, and a decision made as to whether or not this object is new or has been observed previously. Unfortunately this approach leads to erroneous behaviour, as the first frame of an object often doesn't represent it's true appearance. For this reason, the Inter-Camera tracking algorithm delays making a decision for a number of frames, set to 35 frames.

3.6 Conclusion

A proposed method for inter-camera object tracking has been presented. The two contributions were made in this proposed method. The first is the application of Zernike Moments has a shape feature. Zernike Moments have been used in other image recognition works as a good shape descriptor. The second contribution is the method of using many features together to build and appearance profile of an object. Weight is given to recent appearances, more stable features, and more reliably translated features from other camera views. The

next chapter shows the results of an implementation of this proposed method.

4

Implementation and Results

4.1 Implementation

A working implementation of the entire system described in chapter 3 was built to test the proposed method. Programmed in C++, the implementation is comprised of over 6000 lines of code. The Intel Open Computer Vision (OpenCV) library was used extensively in this work, taking care of video decoding and some basic image processing tasks. In addition, the code provided alongside the background subtraction survey[27] was used nearly verbatim as the background subtraction module.

A full graphical user interface (GUI) was built, using the open source gtkmm library. Figure 4.1 shows a screenshot. It shows the number of cameras loaded, the intra-camera tracked objects, and the global inter-camera tracked objects. Debug output of the background subtracted image, the blob-builder algorithm, and the object tracking can also be shown.

Programmed to be as platform-neutral as possible, the implementation has only been compiled and tested in an Ubuntu Linux environment. Though the computation engine

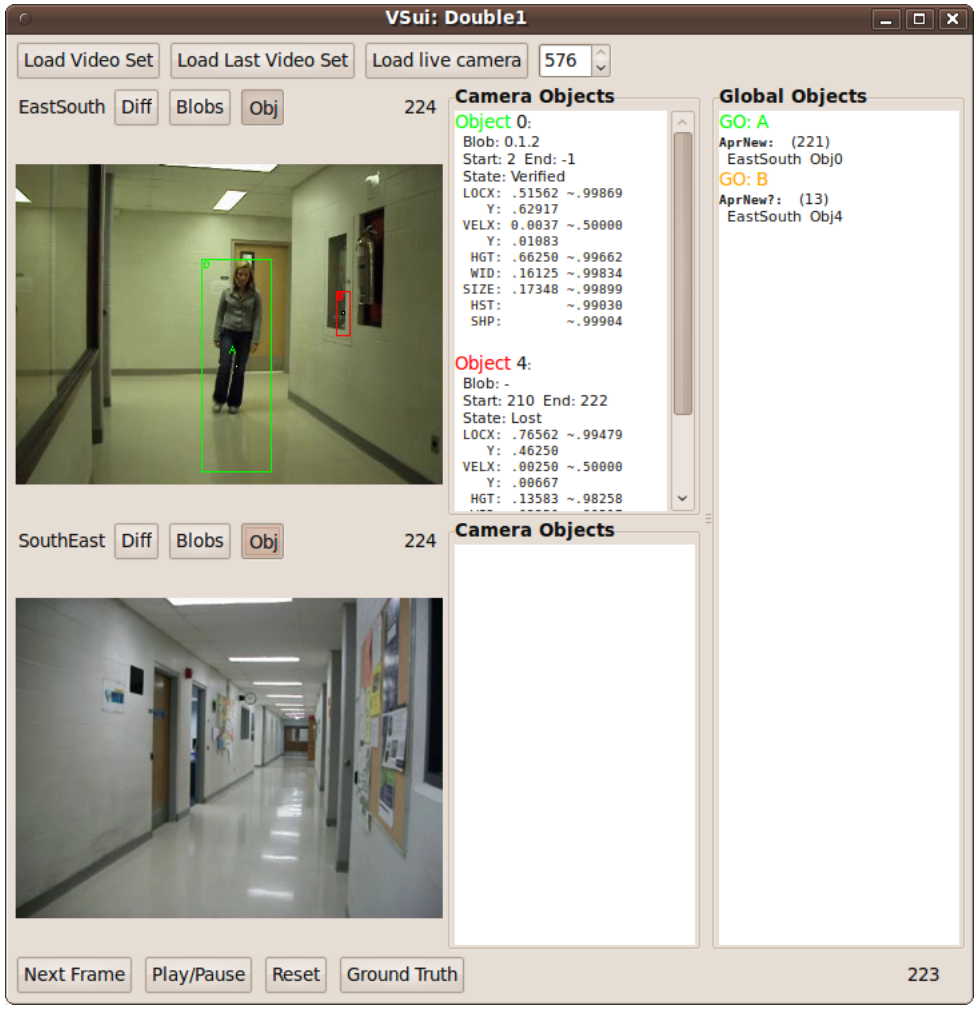

Figure 4.1: Video Surveillance GUI

part of the implementation works frame by frame, it is not real-time for several reasons. Live cameras present a challenge due to hardware access and library issues. As described in chapter 3, there is much work to be done for each frame of processing. The current implementation can process at best 40 frames per second, and at it's worst only 5 frames per second. Efficient multi-threading and code optimization can help with this problem.

4.2 Surveillance Videos

Automated video surveillance is a popular topic in Computer Vision, and as such many datasets of video surveillance recordings are available for use as test videos. This enables researchers to better compare and evaluate different algorithms and implementations.

Unfortunately during the research portion of this thesis, no suitable video surveillance datasets were found. There are many single camera videos, and many multiple camera videos with overlapping fields of view. But no multiple-camera, separate field of view video surveillance databases. In all existing works, authors recorded their own surveillance videos, which was also done for this thesis.

To demonstrate the robustness of the proposed method, two different off-the-shelf low end cameras were used to record surveillance videos, a Sony Cyber-shot DSC-S930 and a Kodak EasyShare C180. These are hand held point and shoot photograph cameras mounted on tripods, used in movie mode to record videos. After recording, the videos were down-sampled to 320×240 resolution at 25 frames per second.

The environment filmed is an indoor office setting, viewing a university building hall-way. It is uncluttered with a white background. The indoor lighting also provides few shadows, with polished floors that cause numerous shadows, providing a challenge for video surveillance software. The example pictures of each environment show how the two cam-

eras have different colour sensitivities. This serves to test the camera transfer functionality described in chapter 3.

These videos also show the limitations and assumptions of the proposed methods object tracking. In all videos, the cameras are stationary and not moving. This is because background subtraction assumes a stationary camera. In addition, background subtraction requires good lighting, with minimal reflections and shadows, properties of all four videos. Finally, there is a low incidence of occlusion in these videos. This was done on purpose due to the focus on inter-camera tracking, and not occlusion handling. Occlusion errors, and hence intra-camera errors, only serve to hinder further inter-camera object tracking measurements.

4.3 Results

This section describes the results of the proposed method working on the surveillance videos. Because this thesis's focus is on inter-camera object tracking, only the accuracy of the inter-camera transitions and object re-appearances are noted. These are the only situations when the inter-camera object tracking algorithms are invoked and used. The accuracy of intra-camera object tracking is not addressed.

4.3.1 Video One

The first test video is a simple proof of concept to test if the inter-camera object tracker is functioning. It consists of a woman walking into the first camera's view, then disappearing, then appearing later on in the second view. Figure 4.2 shows the successful tracking of object with label A. Note that while the woman was being tracked in the first camera, her

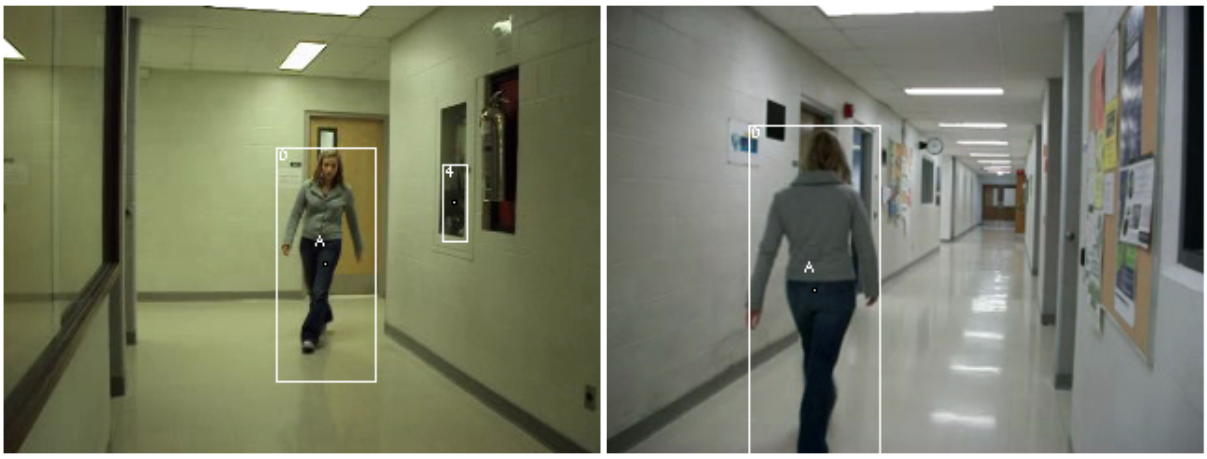

Figure 4.2: First test video

attributes were calculated and saved as explained in Section 4. When she re-appears in the second camera, her attributes are calculated again. These new attributes are compared with the previous appearance, and the matching is performed as described in section 3.5.4.

The woman was successfully tracked with a match score in the second camera of 0.81, above the 0.6 threshold. Although this might seem like an easy case, it is still a challenging problem as no matching statistics have been built up yet. As such, the histograms and other attributes are not very stable across different cameras and lighting or reflections.

4.3.2 Video Two

The next incrementally challenging test video takes place in the same environment, but with 2 people. There are 6 appearances by the people in total, giving 3 inter-camera transitions and 1 re-appearance. Figures 4.3 and 4.4 show the tracking output at frame 165 and frame 341 respectively. Again, all transitions and re-appearances were correctly tracked, although with lower scores. Table 4.1 shows the results:

Both initial thresholds and scores for the objects are -1. This is due to the fact that

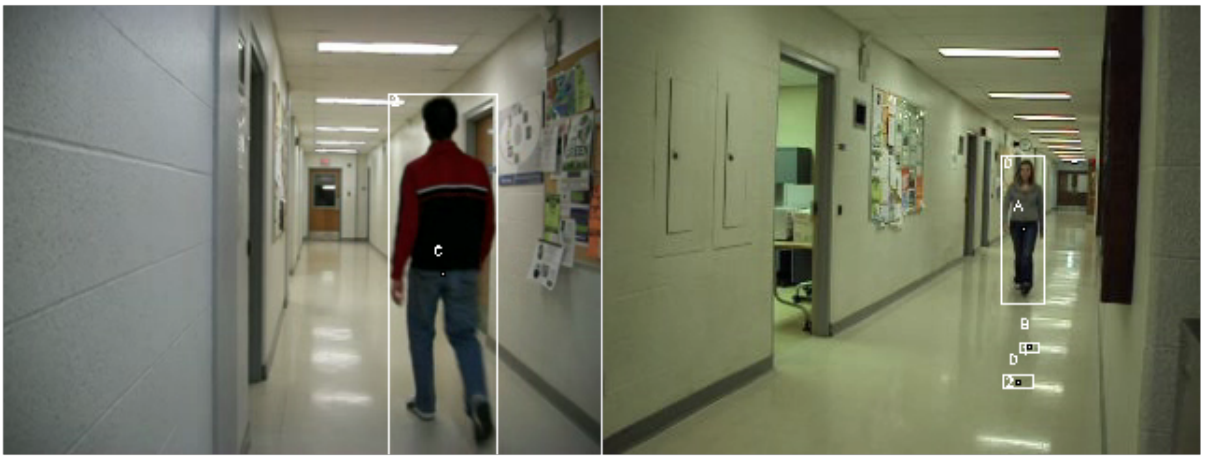

Figure 4.3: Third test video, frame 165

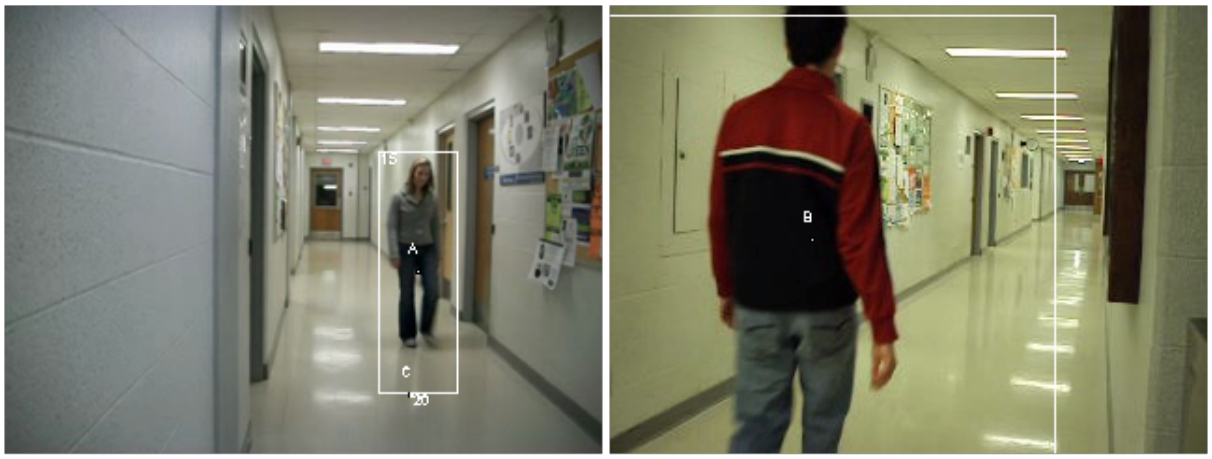

Figure 4.4: Third test video, frame 341

Appearance	Camera	Threshold	Score
1	South	-1	-1
2	North	0.60	0.80

(a) Woman

Appearance	Camera	Threshold	Score
1	South	-1	-1
2	North	0.68	0.80
3	North	0.72	0.80
4	South	0.74	0.79

(b) Man

Table 4.1: Appearances of objects in video two

no objects have been observed yet, and as such there is no object to compare to. Worthy of mention is the slowly increasing match threshold. At first, with a low value of 0.6, increasing with each successful match.

Figure 4.5 demonstrates the camera bloom effect, when the man walks close to the North camera. Nearly all of the camera's pixels suddenly become foreground. This is also noticeable in Figure 4.4 by how large the bounding box is for the walking man. The inter-camera tracking algorithm can still handle this spurious noise, as the overall decision made still matches the ground truth value.

Figure 4.5: Camera Bloom: The white is foreground, black is background.

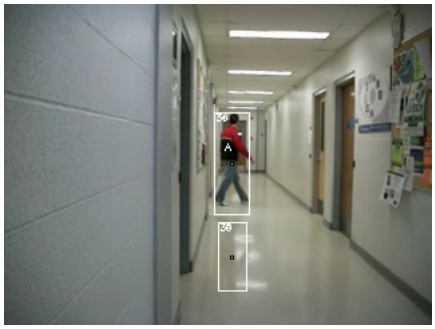

Figure 4.6: Tracking error in third video. Notice the artifacts created from reflections and shadows

4.3.3 Video Three

In the third test video, two people walk from opposite directions. They enter different cameras, leave each camera at roughly the same time, and then walk into view of the other camera again at roughly the same time. Both people are in the unobserved zone between each camera's views at the same time. This requires the inter-camera tracking algorithm to not use any location or timing cues.

The two simultaneous camera transitions are tracked correctly. There is also another re-appearance of the second object that is not successfully tracked, erroneously labelled as the first object (Figure 4.6). The correct object had a matching score of 0.85, while the wrong object had a matching score of 0.86. This slight difference can be attributed to the intra-camera tracking algorithm errors for some of the observed frames, along with the smaller view size of the object not providing as much detail as a more close up viewpoint.

4.3.4 Video Four

The fourth test video is the most complex video with three different people as objects. The video is just over 4 minutes long. Objects appear multiple times in each of the two cameras,

often at the same time. Table 4.2 shows the complete results of object tracking.

Man 1 (Red Shirt)							
Apr#	Cam	Object	Thres	Score	Type	Tracked	Notes
1	East	B	-1	-1	-	-	New Object
2	North	H	.725	.716	Inter	×	
3	North	-	-	-	Re-App	-	Intra-cam error, occlusion
4	East	B	.763	.865	Inter	×	
5	North	N	-	-	Inter	×	Intra-cam error, occlusion
6	East	B	.763	.847	Re-App	✓	From Apr 4
7	East	B	.763	.813	Re-App	✓	
8	East	B	.763	.857	Re-App	✓	
9	North	B	.784	.995	Inter	✓	Bloom present
10	North	N	.763	.774	Re-App	×	
11	East	D	-	-	Inter	-	Intra-cam error
12	North	D	.791	.996	Inter	✓	
13	North	B	-	-	Re-App	-	Intra-cam error, switch to B
14	North	N	-	-	Re-App	-	Intra-cam error
Man 2 (Green Shirt)							
Apr#	Cam	Object	Thres	Score	Type	Tracked	Notes
1	North	D	.689	.570	-	-	New Object
2	East	D	.725	.997	Inter	✓	
3	North	D	.748	.809	Inter	✓	
4	North	D	.763	.883	Re-App	✓	
5	East	D	.775	.998	Inter	✓	
6	North	H	.763	.852	Inter	×	
7	East	-	-	-	Inter	-	Intra-cam error
8	North	B	.791	.998	Inter	×	
9	East	D	.791	.997	Inter	×	
10	North	B	-	-	Inter	-	Intra-cam error
11	East	D	.801	.999	Re-App	✓	
Man 3 (Blue Shirt)							
Apr#	Cam	Object	Thres	Score	Type	Tracked	Notes
1	East	G	.762	.455	-	-	
2	North	D	.763	.793	Inter	×	Heavy Occlusion
3	North	N	-	-	Re-App	-	Intra-cam error
4	North	N	.763	.848	Re-App	✓	
5	East	D	.783	.995	Inter	×	
6	East	D	.784	.999	Re-App	✓	
7	North	B	.791	.999	Inter	×	
8	East	D	.791	.998	Inter	×	
9	North	H	.791	.999	Inter	×	
10	East	D	.763	.832	Inter	×	

Table 4.2: Appearances of objects in video five

Explaining table 4.2, column *Apr* shows the appearance number, *Cam* identifies the

camera the person appeared in, $Object$ shows the object name matched, $Thres$ and *Score* show the numerical threshold and similarity score, and finally $Type$ and $Tracked$ show the type of transition and whether it was successfully tracked. Inter-camera tracking is shown with $Inter$, and a same camera re-appearance is shown with $Re-App$. A check mark is for a correct label, an $\times$ for an incorrect label, and a dash to indicate an error elsewhere in the tracker.

The three people have 35 appearances in total. The first appearance of each person establishes the person as a tracked object, leaving 32 inter-camera transitions or same camera re-appearances. Of these 32 transitions, 11 were correctly tracked, 13 were incorrectly tracked, and 8 were intra-camera tracking errors. The summary table 4.3 shows these results.

	Total	✓	×	-
Inter	20	5	12	3
Re-App	12	6	1	5
Transitions	32	11	13	8

Table 4.3: Tracking Totals for video four

Out of 20 inter-camera transitions, only 5 were successfully tracked. This low score can be attributed to the general similarity of the three people in the video. All three have similar height, width and size, and a similar shape. They also walk with nearly the same speed through the videos. Though with different clothing, the unique shirts portion of clothing is a small fraction of the total object. In addition, background elements like shadows, reflections are also included in the object due to background subtraction imperfections. As a result, the three people have similar object profiles.

Coupling this with the location feature, and the result is that objects tend to ”stick” to the same camera. The best example of this is tracking object D in table 4.2. Out of 14

appearances, only 2 are in the North camera. The rest are in the East camera, matched to all three different people at different times. Figure 4.7 shows this example.

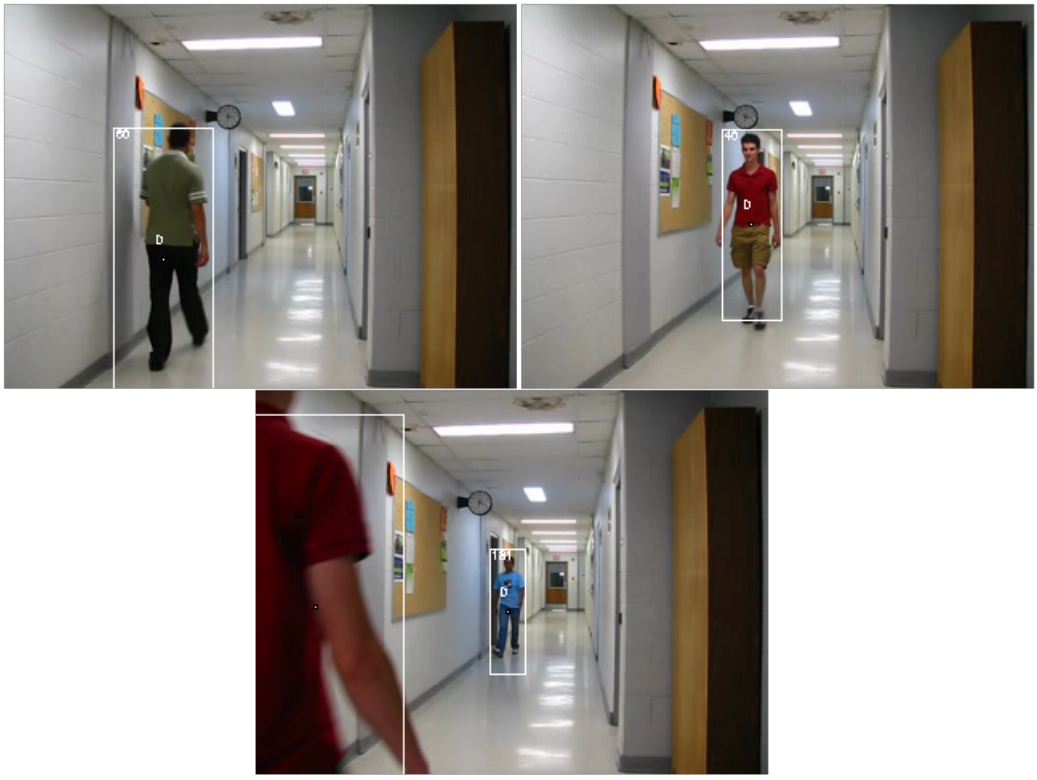

Figure 4.7: Object Sticking example

Since each object has a similar profile, the location feature matches better within the same camera, and hence the assigned label stays in the same camera. This effect increases with time, as more camera link statistics are built up and the weight of simpler features increases. A positive side effect is that objects that re-appear in the same camera are tracked very well, with 6 correct labels and 1 incorrect.

If object profiles are built with errant information, this can also further decrease the accuracy of the inter-camera object tracking algorithm. This is especially evident near the end of the video, with scores for objects being around 0.99 and yet matching to different

people. As a tracked object has feature data averaged from different real life objects, its profile becomes blurred and more apt to match with any object.

One error in tracking can also lead to another, termed cascading errors in this thesis. The inter-camera object tracker assumes that an object cannot be in two places at once. If an object is incorrectly labelled, and the true object appears after a decision has been made, the true object cannot steal the proper label. Figure 4.8 shows an example of this effect.

Figure 4.8: Cascading Error

The left image shows the person being labelled incorrectly as object B. The right image shows the person who should be labelled as B instead labelled erroneously as object D.

The 11 incorrect transitions that are indicated by a dash in table 4.3 are intra-camera tracking errors that are not included in the inter-camera tracking analysis. Most often these were caused by occlusion or faulty background segmentation.

4.4 Conclusion

A C++ implementation of the proposed method in chapter 3 was developed and presented in this chapter. It uses the OpenCV library for video file decoding and basic image processing

tasks. Background subtraction code from [27] was also used. A GUI for observing the results of the implementation was built using the gtkmm library in a Linux environment.

Due to the lack of available test surveillance videos that have multiple separate camera views, off the shelf picture cameras were used in movie mode to record local test videos. The environment filmed was an indoor office environment inside a university building. All the videos featured two cameras. Only the inter-camera object tracking results were noted because of the focus of this thesis.

The first three videos were simpler proof of concepts. The first involved one object moving from one camera to another. The second involved two people, with six appearances in total. The third also had two people, but was more difficult due to the objects being in the unobserved space between cameras at the same time.

The fourth surveillance video presented a longer, more complex video with three people and a total of 35 appearances. It is challenging due to the similarity of the objects tracked, the large number of appearances, and cascading errors from an initial error. In spite of these difficulties, the inter-camera object tracking algorithm still correctly tracks 5 inter-camera transitions out of 20.

5

Conclusion

This thesis has presented a new method for Inter-Camera Object Tracking. Less investigated than the popular application of Object Tracking, Inter-Camera Object Tracking focuses on tracking objects between multiple cameras that have separate fields of view.

A significant amount of research was undertaken and presented, in order to understand the current state of the art in Inter-Camera Object Tracking. Pioneering efforts by authors like A. Gilbert [6, 8] and O. Javed [13, 15, 14] were studied, along with basic object tracking techniques presented in [1, 31, 25].

From this research, a new method for Inter-Camera Object Tracking was presented, using the best techniques from previous works and research[27, 37]. Two main contributions were made to inter-camera object tracking; i.e; (1) the use of zernike moments as a general shape feature[22] and (2) the dynamic weighing together with the combination of several features to achieve better tracking results.

Given that inter-camera object tracking is a recent research topic, no suitable video datasets were found to test the proposed methods accuracy. Like previous works, home-made surveillance videos were used instead. The first three are simple, proof of concept

videos that only have a couple of transitions with few objects. It was shown that the proposed method works on these videos. The last and most complex video is a difficult and long one, with 3 different objects and 35 appearances. The proposed method was able to correctly label 5 out of 20 inter-camera transitions in this video.

Future work for this thesis includes building a large and varied set of surveillance videos with corresponding hand labelled ground truths. Combined with a framework to automatically compare object tracking output to a ground truth, creative ideas can be quickly implemented and tested, with more reputable tracking results. This is the standard operating procedure for other computer vision problems, an area where inter-camera object tracking should catch up.

Other feature weighting methods could be implemented and tested for merit. One idea is to apply common pattern recognition methods to the feature weighting. These methods could also discredit features that are not distinguishable between objects. This would have the effect of emphasizing unique features for particular objects. Another improvement would be to better scale feature differences to better highlight true object differences.

Code optimization is also needed in order to bring greater performance to this methods implementation. This would allow for faster results generation and the ability to run the code in true real-time. In addition, critical parts that are CPU-intensive could be written to use multiple threads and even OpenCL to take advantage of emerging GPGPU computing resources.

Bibliography

[1] M.A. Ali. Feature-based tracking of multiple people for intelligent video surveillance. In *Masters Abstracts International*, volume 45, 2006.

[2] G. Amayeh, A. Erol, G. Bebis, and M. Nicolescu. Accurate and efficient computation of high order zernike moments. *Lecture notes in computer science*, 3804:462, 2005.

[3] MD Beynon, DJ Van Hook, M. Seibert, A. Peacock, and D. Dudgeon. Detecting abandoned packages in a multi-camera video surveillance system. In *Proceedings. IEEE Conference on Advanced Video and Signal Based Surveillance, 2003.*, pages 221–228, 2003.

[4] Y. Cai, K. Huang, and T. Tan. Human Appearance Matching Across Multiple Non-overlapping Cameras. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–4, 2008.

[5] ED Cheng, C. Madden, and M. Piccardi. Mitigating the Effects of Variable Illumination for Tracking across Disjoint Camera Views. In *IEEE International Conference on Video and Signal Based Surveillance, 2006. AVSS'06*, pages 32–32, 2006.

[6] A. Gilbert and R. Bowden. Tracking objects across cameras by incrementally learning inter-camera colour calibration and patterns of activity. *Lecture Notes in Computer Science*, 3952:125, 2006.

[7] A. Gilbert and R. Bowden. Multi person tracking within crowded scenes. *Lecture Notes in Computer Science*, 4814:166, 2007.

[8] A. Gilbert and R. Bowden. Incremental, scalable tracking of objects inter camera. *Computer Vision and Image Understanding*, 2008.

[9] F. Hao, Z. Miao, P. Guo, and Z. Xu. Real Time Multiple Object Tracking Using Tracking Matrix. In *Proceedings of the 2009 International Conference on Computational Science and Engineering-Volume 02*, pages 37–41. IEEE Computer Society, 2009.

[10] M. Harville and M. Harville. Stereo person tracking with short and long term plan-view appearance models of shape and color. In *Proceedings IEEE Conference on Advanced Video and Signal Based Surveillance*, pages 522–527. Citeseer, 2005.

[11] K. Huang, L. Wang, T. Tan, and S. Maybank. A real-time object detecting and tracking system for outdoor night surveillance. *Pattern Recognition*, 41(1):432–444, 2008.

[12] J. Humphreys and A. Hunter. Multiple object tracking using a neural cost function. *Image and Vision Computing*, 27(4):417–424, 2009.

[13] O. Javed, Z. Rasheed, K. Shafique, and M. Shah. Tracking across multiple cameras with disjoint views. In *Ninth IEEE International Conference on Computer Vision, 2003. Proceedings*, pages 952–957, 2003.

[14] O. Javed, K. Shafique, Z. Rasheed, and M. Shah. Modeling inter-camera space–time and appearance relationships for tracking across non-overlapping views. *Computer Vision and Image Understanding*, 109(2):146–162, 2008.

[15] O. Javed, K. Shafique, and M. Shah. Appearance modeling for tracking in multiple non-overlapping cameras. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, page 26. Citeseer, 2005.

[16] K. Jeong and C. Jaynes. Object matching in disjoint cameras using a color transfer approach. *Machine Vision and Applications*, 19(5):443–455, 2008.

[17] L. Jiao, Y. Wu, G. Wu, E.Y. Chang, and Y.F. Wang. Anatomy of a multicamera video surveillance system. *Multimedia Systems*, 10(2):144–163, 2004.

[18] J. Kang, I. Cohen, and G. Medioni. Persistent Objects Tracking Across Multiple Non Overlapping Cameras. In *IEEE Workshop on Motion and Video Computing, 2005. WACV/MOTIONS'05 Volume 2*, volume 2, 2005.

[19] S.M. Khan and M. Shah. Tracking multiple occluding people by localizing on multiple scene planes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 505–519, 2009.

[20] A. Khotanzad and YH Hong. Invariant image recognition by Zernike moments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(5):489–497, 1990.

[21] K. Kim and L.S. Davis. Multi-camera Tracking and Segmentation of Occluded People on Ground Plane Using Search-Guded Particle Filtering. *Lecture Notes in Computer Science*, 3953:98, 2006.

[22] W.Y. Kim and Y.S. Kim. A region-based shape descriptor using Zernike moments. *Signal Processing: Image Communication*, 16(1-2):95–102, 2000.

[23] C. Madden and M. Piccardi. A framework for track matching across disjoint cameras using robust shape and appearance features. In *Proceedings of the 2007 IEEE Conference on Advanced Video and Signal Based Surveillance-Volume 00*, pages 188–193. IEEE Computer Society Washington, DC, USA, 2007.

[24] Pier Mazzeo, Paolo Spagnolo, and Tiziana DOrazio. Object Tracking by Non-overlapping Distributed Camera Network. *jaa*, 5807:516–527, 2009.

[25] S.J. McKenna, S. Jabri, Z. Duric, A. Rosenfeld, and H. Wechsler. Tracking groups of people. *Computer Vision and Image Understanding*, 80(1):42–56, 2000.

[26] A. Mittal and L.S. Davis. M 2 Tracker: a multi-view approach to segmenting and tracking people in a cluttered scene. *International Journal of Computer Vision*, 51(3):189–203, 2003.

[27] D.H. Parks and S.S. Fels. Evaluation of Background Subtraction Algorithms with Post-Processing. In *Proceedings of the 2008 IEEE Fifth International Conference on Advanced Video and Signal Based Surveillance*, pages 192–199. IEEE Computer Society, 2008.

[28] R. Pflugfelder and H. Bischof. Tracking across non-overlapping views via geometry. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–4, 2008.

[29] M. Piccardi and ED Cheng. Track matching over disjoint camera views based on an incremental major color spectrum histogram. In *IEEE Conference on Advanced Video and Signal Based Surveillance, 2005. AVSS 2005*, pages 147–152, 2005.

[30] A. Rahimi, B. Dunagan, and T. Darrell. Simultaneous calibration and tracking with a network of non-overlapping sensors. In *IEEE COMPUTER SOCIETY CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION*, volume 1. IEEE Computer Society; 1999, 2004.

[31] A. Senior. An Introduction to Automatic Video Surveillance. *Protecting Privacy in Video Surveillance*, page 1, 2009.

[32] C. Stauffer and WEL Grimson. Adaptive background mixture models for real-time tracking. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 2, 1999.

[33] The GIMP Team. Gnu image manipulation program (gimp). http://www.gimp.org/.

[34] Wikipedia. Connected component labeling. http://en.wikipedia.org/wiki/Connected_Component_Labeling.

[35] A. Yilmaz, O. Javed, and M. Shah. Object tracking: A survey. *ACM Computing Surveys (CSUR)*, 38, 2006.

[36] F. Zernike. Beugungstheorie Des Schneidenverfahrens und seiner verbesserten Form, der Phasenkontrastmethode [Diffraction theory of the knife-edge test and its improved form, the phase-contrast method]. *Physica,s Grav*, 1:689–704, 1934.

[37] D. Zhang and G. Lu. Review of shape representation and description techniques. *Pattern recognition*, 37(1):1–19, 2004.

[38] Z. Zhang, A. Scanlon, W. Yin, L. Yu, P.L. Venetianer, and O.V. Inc. Video Surveillance using a Multi-Camera Tracking and Fusion System. In *Workshop on Multi-camera and Multi-modal Sensor Fusion Algorithms and Applications-M2SFA2*, 2008.

[39] Z. Zhao, S. Yu, X. Wu, C. Wang, and Y. Xu. A multi-target tracking algorithm using texture for real-time surveillance. In *Proceedings of the 2008 IEEE International Conference on Robotics and Biomimetics*, pages 2150–2155. IEEE Computer Society, 2009.

[40] Z. Zivkovic and F. van der Heijden. Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern recognition letters*, 27(7):773–780, 2006.

Vita Auctoris

NAME:	Trevor Montcalm
PLACE OF BIRTH:	Windsor, Ontario
YEAR OF BIRTH:	1984
EDUCATION:	Hon W.C. Kennedy Collegiate Institute High School, Windsor 1998-2003 University of Windsor, Windsor, Ontario 2003-2007 B.Sc.