2008-01-01

Homography Based Multiple Camera Detection of Camouflaged Targets

Jesus Flores Guerra
University of Texas at El Paso, jfguerra@miners.utep.edu

HOMOGRAPHY BASED MULTIPLE CAMERA DETECTION OF CAMOUFLAGED TARGETS

JESUS FLORES GUERRA

Department of Electrical & Computer Engineering, ECE

APPROVED:

Jose Gerardo Rosiles, Ph.D., Chair

Sergio D. Cabrera, Ph.D.

Virgilio Gonzalez, Ph.D.

Noe Vargas Hernandez, Ph.D.

Patricia Witherspoon, Ph.D.

Dean of the Graduate School

To my crazy and beloved family,

for always believing in me, encouraging me and supporting me throughout this journey.

&

In memory of Dimpna Cruz de Guerra,

for her unconditional love. We miss you and will always keep you in our hearts.

HOMOGRAPHY BASED MULTIPLE CAMERA DETECTION OF CAMOUFLAGED TARGETS

by

JESUS FLORES GUERRA, BSEE

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of Electrical & Computer Engineering, ECE

THE UNIVERSITY OF TEXAS AT EL PASO

December 2008

Acknowledgements

I would like to express my deepest gratitude and appreciation to the following people, whose help during this research was invaluable. Many thanks go out to the three professors in the Wireless Sensor Networks Group. I am particularly grateful with Dr. Gerardo Rosiles for his kind guidance and great patience towards me during this thesis. His accepting me as a research assistant allowed me to complete my Master's degree and exposed me to travelling to conferences, trainings and gave me teaching experience as well. I also need to thank Dr. Sergio Cabrera for introducing me to the world of computer vision and inspiring me to pursue work in this area. This field has helped me admire the beauty of the human machine even more. Dr. Cabrera also provided funding for my degree through the Texas Instruments Foundation Endowed Scholarship, whose aid was critical to afford tuition. I would also like to thank Dr. Virgilio Gonzalez for all his help during my earlier years as an electrical engineer and for encouraging me to pursue higher education.

Many of my colleagues were also instrumental to the completion of this thesis. James Roeder allowed me to learn from his mathematical brilliance while keeping me company, Brandon Aguirre always reminded me of the potential of a day's work if you push yourself and Kristian Alcantar provided comical relief in stressful times and was a faithful friend. I am in debt to them and other fellow students.

Lastly, I would like to thank my friends who helped make El Paso my home. All my respect and thanks go to the Perez family, for being great friends and my family away from home. Last, but certaintly not least, I would like to thank many friends that supported me through the years: Carlos Perea, Monica Moreno, Miguel Perea, Alejandro Macias, Alba Aguilar, Said Aguilar, Nikky Aguilar, Isaac Villalpando, Erica Grijalva, Lupita Garcia, Tumi Layinka and Saili Davidoff.

Abstract

This thesis proposes an improvement on segmentation of camouflaged objects by combining information from multiple static cameras with overlapping views onto a common reference plane. Previous camouflage detection work relies on a single camera and focuses on pattern recognition, gray level co-occurrence matrix (GLCM) computation, convexity detection and operations such as connected components or morphology. These methods make real-time implementation computationally expensive. The proposed work diverges from these techniques, taking advantage of the added information from several viewpoints. Background subtraction is performed on each individual camera and then, the segmentation result is transformed onto a common plane using homography. A multiple camera view is generated by averaging the transformed views from each camera. This multiple camera view is used to achieve a better segmentation of the camouflaged object. A multi-camera surveillance system is developed in Matlab as part of this research. The surveillance system includes processes such as classification, centroid calculation, tracking using a Kalman Filter and bounding the target with a rectangle. In addition, a graphic user interface was designed to be able to choose from several cameras, viewpoints and motion detection methods.

Experiments were run to test the performance of the background subtraction method versus two other reference model methods. Tests were also run to observe the challenges faced by background subtraction under camouflaged conditions. Additional experiments compare the tracking from individual cameras to the results from the Kalman Filter and Multi-Camera Average. The computation time between the proposed method and GLCM was also studied. The results indicate that a multiple camera surveillance system provides an improved segmentation of a camouflaged target and is capable of tracking targets more consistently than individual cameras. Further results indicate that a multi-camera system can perform faster than GLCM in some circumstances.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Background information

1.1.1 What is Surveillance?

In general, the term surveillance refers to the monitoring of behavior. This behavior can be exhibited by many different sources ranging from people and automobiles to animals, plants, and diseases. In clinical surveillance, for example, the goal could be to monitor the behavior of a salmonella outbreak using blood samples.

In contrast, this thesis focuses on systems surveillance. The goal of systems surveillance is to monitor the behavior of people and objects for security or social control. Systems surveillance uses a wide variety of monitoring methods. These forms include eavesdropping, phone tapping, CCTVs, GPS tracking, reconnaissance aircraft and computer surveillance, among other forms. Currently, the most common form of surveillance is visual surveillance. The goal of automatic visual surveillance is to describe the events of a monitored area and then take appropriate action [4]. Visual surveillance has become so widespread that cameras are a common icon for surveillance.

1.1.2 History of surveillance

The earliest reports of using video surveillance in public places used closed-circuit television (CCTV) cameras and human monitoring. These reports date back to 1965. The United Kingdom was one of the countries that pioneered the use of extensive CCTV to monitor public areas. In 1975, video surveillance systems were installed in the Underground Train Stations and on major intersections to monitor traffic flow. These cameras were constantly monitored by policemen. Today, more than ever, cities and towns across the UK have cameras linked to police authorities.

The marketing of the video cassette recorder in 1977 allowed analog technology to record and use the surveillance as evidence. In the 1980s, American businesses such as banks, mini-marts and gas stations began using video surveillance to deter and catch thieves [41]. In 1990, the development of digital multiplexing enabled recording on several cameras at once. This allowed time-lapse and motion-only recording. This saved spaced on videotapes. By the mid 1990s ATMS throughout the US had video cams to record transactions.

The digital revolution brought many advantages to video surveillance. The price of digital recording dropped. There was no longer a need to rewind and change video tapes. Besides the storage benefits, images could be manipulated, zoomed, enhanced and published quickly. Deploying large numbers of surveillance cameras has become economically and technically feasible thanks to the proliferation of inexpensive cameras and high-speed wired and wireless networks [15]. This allowed the popularity of video surveillance to be extended into the household. Commercializing surveillance cameras drove them to become even smaller and improve their resolution.

Recent terrorist threats have also triggered more research into the area of surveillance. Video surveillance is increasingly being used as a way to fight terrorist threats and increase public security, due largely to the actions of United States legislators involved in security and intelligence services [15].

Currently, the most common type of sensor networks are camera networks [16]. These sensor networks are everywhere and used in real-world applications such as surveillance, intelligent environments and scientific remote monitoring [16]. Cameras are now required to do more sophisticated tasks than simply recording and storing video. Face recognition, biometrics and multi-camera fusion techniques are recent research topics of interest for camera networks.

Current and future research on video surveillance will develop on video analytics and video understanding. Cameras cannot only rely on detecting motion anymore. Computer Vision and Machine Learning are being integrated to not only detect an object, but also understand the scene's events. Analysis of traffic flow, highway accident detection, consumer demographic compilation in shopping malls and amusement parks and counting endangered species are only few of the many commercial applications for networked video surveillance [12].

1.1.3 Analog vs. Digital

In the past, most visual surveillance systems have used analog closed-circuit television (CCTV), where the picture is viewed or recorded, but not broadcasted. CCTV systems have been popularized due to their inexpensive price and ease of use. However one of their major disadvantages is that they require expensive installations. A person walking through a typical large city can expect to be video-taped hundreds of times per day, since public cameras have become ubiquitous [4]. Today, CCTV can still be found in many public places such as banks, universities, streets and even home security systems.

This introduces another drawback of CCTV: the manpower required to supervise surveillance cameras is expensive, even though surveillance cameras are cheap and ubiquitous [4]. The attention span of a trained security guard watching CCTV monitors is very limited. Many distractions may occur, especially if the guard is given secondary tasks and must monitor multiple screens. There are too few eyes to keep track of what goes on in the large number of cameras [3].

Most of the time, the recorded video is used after a crime is committed as evidence, instead of actually preventing the incident.

Another common problem with analog technology is that video tapes must be changed daily and usually wear out after months of reuse. Additionally, recording at night or in low light presented challenges.

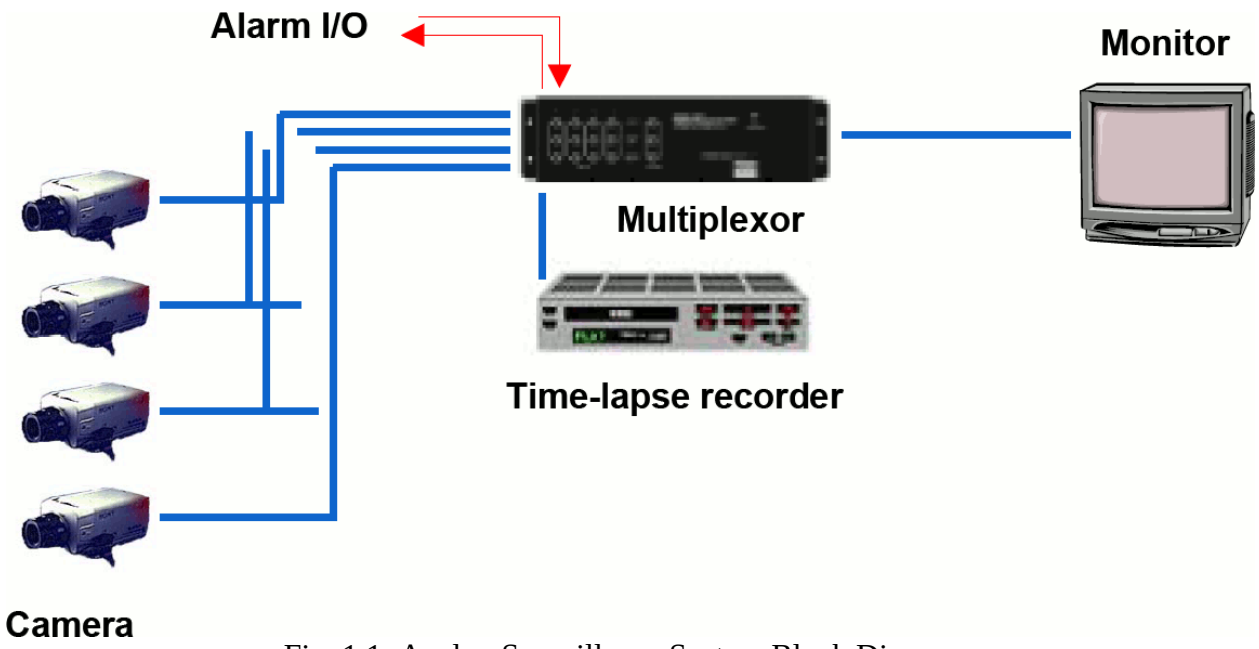


Fig. 1.1: Analog Surveillance System Block Diagram

Digital surveillance and camera networks address and solve many of the issues of analog surveillance. Wireless networked cameras avoid expensive infrastructure and are easier to install than wired CCTV. Digital cameras can store video on a server, solving the problems with video tapes. Videos can be compressed to reduce transmission bandwidth and allow for even longer recording times. Image processing and computer vision techniques can be used to enhance the video footage and make decisions about the scene in real time. Surveillance software needs to make decisions in real-time in order to be considered useful [4].

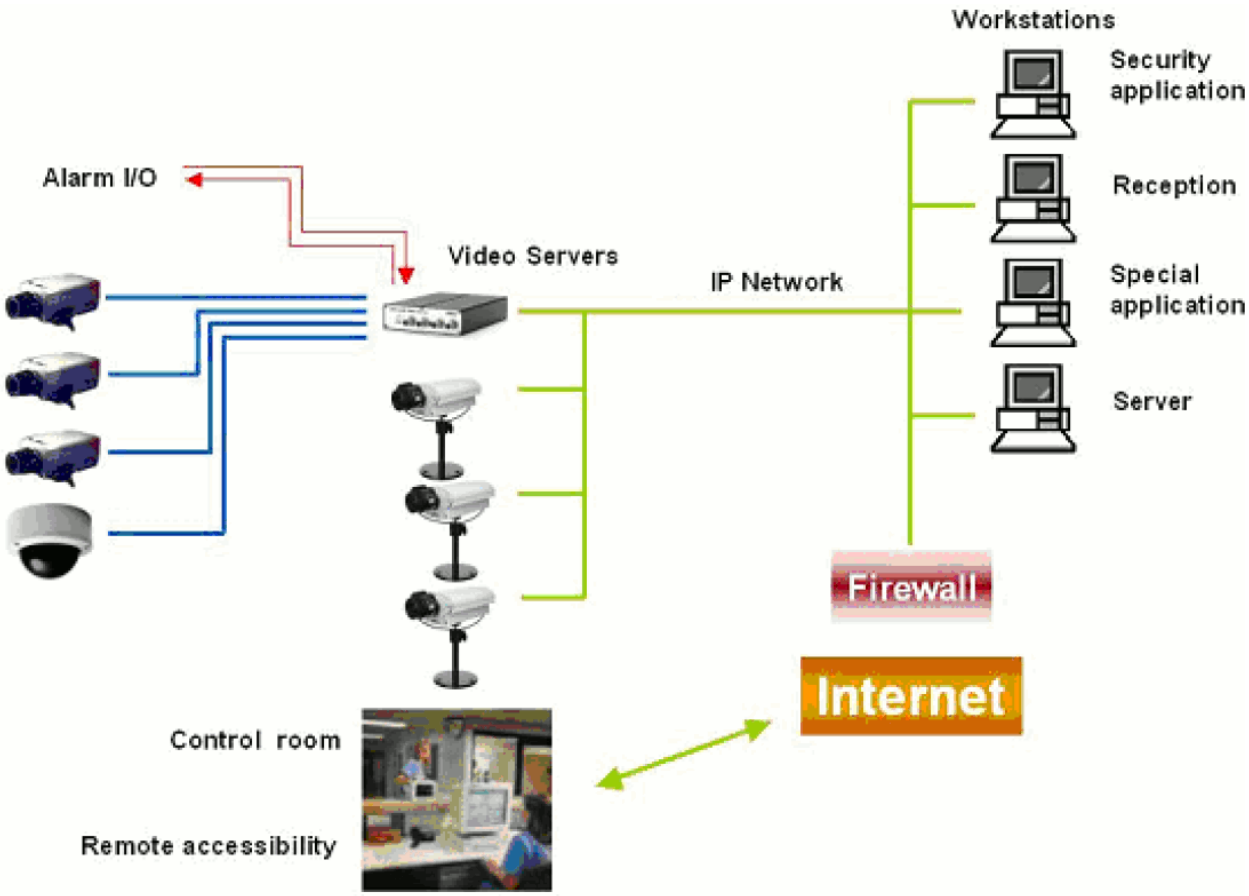


Fig. 1.2: Digital Surveillance System Block Diagram

Finally, the price of digital cameras continues to drop as they become more popular. Hendy's Law states that "The number of pixels per dollar found in digital cameras will double every year" [40]. The demand for surveillance systems is also favorable. Systems that can process the video from the rising number of CCTV cameras are "one of the top items in demand" according to a survey of hundreds of US security executives [3].

1.1.4 Challenges with Motion Detection

The problem of visual surveillance refers to using imaging sensors to monitor the activity of targets in a scene [21]. These imaging sensors can range from high-end pan-tilt-zoom (PTZ) cameras to low cost CMOS image sensors.

From a simplistic perspective, the visual surveillance problem can be divided into two major sub-problems: detecting and tracking targets [21]. Detection is a crucial step in the surveillance problem. It is impossible to track undetected targets [21]. Furthermore, errors in this detection stage are propagated to other stages of the surveillance system. Common examples of these errors are detecting objects that are not of interest or failing to detect objects of interest.

In order to detect objects, surveillance systems seek differences between a reference frame and incoming frames. This simple method works given the right conditions such as a static background and constant illumination. However, the results can vary due to factors such as illumination changes, stop and go motion, and camera resolution. The main difficulty in change detection is that even under controlled conditions, scenes are continuously subject to changes. Seemingly “static” scenes can become dynamic by the changes of the environment’s lighting, target shadows, and sensor artifacts such as auto gain correction. Additional difficulties are introduced by natural motion, such as moving clouds and uninteresting motion, such as tree branches swaying in the wind [21].

Outdoor lighting, for example, is continually varying according to the time of day and weather conditions. In the case of camouflaged targets, parts of the target will match the background and other parts will not. Fragmentation of the detected object is expected. These are only a couple of the challenges that automated motion detection faces. Additional processing is used to minimize these distractions from real objects of interest. The kind of approach varies greatly, as many of

the surveillance solutions are application specific. Before the dream of reliable automated surveillance is achieved, many major technical challenges need to be overcome. These technical challenges include not only object identification, tracking and analysis, but also practical considerations such as camera placement, network bandwidth requirements, installation costs, privacy concerns and robustness to changing illumination and weather conditions [4].

1.2 Related Work

1.2.1 Camouflage Breaking

There are several approaches to camouflage breaking. These approaches rely on a single camera [21,23-28] and only a couple of them use image sequences instead of still images [25, 27]. Boult et al. [21] present the Lehigh Omni-directional Tracking System (LOTS) which uses an omni-directional camera in combination with background modeling and quasi connected components (QCC). QCC is a novel alternative to morphological processing that takes into account pixel intensities and pixel region areas to close gaps caused by fragmentation.

Bhajantri and Nagabhushan [28] developed a method to identify camouflaged defects by calculating texture parameters from a grey level co-occurrence matrix (GLCM) and then applying edge detection. The advantages of using the Co-occurrence method, followed by Canny edge detection are that it is a simple algorithm and it creates an outline of the object. However, it does not extract the target. This method must calculate the GLCM and then calculate each of the parameters (energy, entropy, homogeneity, contrast and correlation) for each image block. The computation load that represents this intensive texture analysis does not match the goal of a real-time implementation.

Another method discussed by Tankus and Yeshurun [24] presents biological evidence which suggests that detecting convexity can break camouflage. The convexity based method assumes that most foreground objects are convex. This method can locate foreground objects of interest regardless of their color or camouflage. The advantages of this method are its robustness and precision in detecting foreground objects. However, the object is not extracted and the results highly depend on a threshold that must be determined [23]. Even with these drawbacks, camouflage breaking using convexity-based methods has been proven to be more robust and effective in many cases when compared to edge-based techniques [24].

Huang and Jiang [25] proposed a weighted region consolidation method to track camouflaged objects with a similarly colored background. The object position is refined by an active contour model that includes shape and edge map information. A threshold that determines potential motion pixels must be set. The threshold can have a significant impact on the results and convergence of the iteration process.

Nyberg and Schutte [26] studied image texture features to characterize the spatial variations in background and camouflaged targets. They found that it is possible to find a measure of similarity between camouflaged targets and the surroundings. However, selecting a suitable set of features for general scenes is a difficult task. Furthermore, using many features produces better results, but at the cost of a heavy computation load. Motion was found to be one of the most helpful features in detecting camouflage.

The proposed work combines multiple views from a network of overlapping cameras rather than calculating texture features such as convexity or GLCM.

1.2.2 Multiple Camera Surveillance

The possibility of obtaining data from multiple camera sensors has opened the door for research ranging from integrating the information to camera calibration and modeling. Many papers have been published on the topic of multiple cameras applied to the surveillance problem [11-20].

Stein et al. [18] used feature correspondences to determine the camera geometry and self-calibrate multiple cameras separated at large distances. Planar geometric constraints are applied to a moving object in the scene to align the multiple views to a ground plane. A single global coordinate frame is obtained from a set of distributed, un-calibrated cameras.

A framework for multi-camera video surveillance consisting of detection, representation and recognition is developed by Jiao et al. [15]. The framework was tested in a parking lot surveillance application, showing that the spatio-temporal fusion scheme and biased sequence-data learning method are effective in identifying suspicious activity in a scene.

Multiple camera systems have also been used extensively to track people or pedestrians in dense crowds, as is the case in many urban surveillance scenarios. Dense crowds bring forth added challenges, such as occlusion and the need for multiple trackers.

Davis and Mittal [20] present a multi camera approach to segment, detect and track multiple people in a cluttered scene. Their fully automated system uses several synchronized surveillance cameras and can use the evidence from many cameras to infer data used for detection and tracking. Partial and full occlusions caused by the density of the crowd are resolved in real-time.

A method for tracking people in a dense crowd using multiple cameras with overlapping fields of view is suggested by Eshel and Moses [36]. People's heads are tracked in order to avoid

occlusions due to people standing in front of others. Intensity correlation is applied to frames from different cameras after the head tops are detected. This method is capable of tracking up to 21 individuals in a dense crowd and robust to occlusions and lighting conditions. Khan and Shah [37] describe yet another method of tracking people in crowded scenes. Their approach uses a novel planar homography constraint to resolve occlusions and locate people's feet on the ground plane. People are accurately tracked in each view.

Multiple camera systems have also been used to handle shadows, specularities and illumination changes. Lim et al. [38] propose an alternative to detection using stereo matching. A simple subtraction of the intensity images obtained from stereo-generated conjugate pairs speeds up the detection. A detailed analysis of errors due to false and missed detections are presented, as well as algorithms to eliminate them. This system results in a fast and simple surveillance system that avoids errors due to lighting changes and shadows.

In their paper, titled "Fast Lighting Independent Background Subtraction", Ivanov et al. [39] also describe a background subtraction method that is invariant to illumination changes and can be applied in real-time. A simplified stereo algorithm is used to segment the object from a geometrically static background. This system requires two or more cameras with overlapping fields of view and needs to calculate disparity fields between the cameras.

The surveillance system proposed in this thesis is inspired by these previous studies. The main difference lies in the focus of this research towards camouflaged targets and the desire to produce a real-time surveillance system with simple algorithms that maintain computational cost to a minimum, while still performing well.

1.3 Project Overview

1.3.1 Project Scope

The problem of automated surveillance can be divided into several subtasks according to the application. In general, two major subtasks must be done in order to perform automated surveillance: target detection and tracking [17]. More subtasks may appear depending on the application requirements and algorithms being used. A surveillance system usually performs the following tasks: 1) use predefined models to identify multiple moving targets 2) track the identified targets over the monitored region 3) provide video data and concise feedback of a tracked target to the operator [17].

This thesis addresses the problem of detecting camouflaged objects by combining views from multiple cameras. For this research the surveillance task will be divided into three main subtasks: target detection, multi-camera integration, and tracking/classification. However, most of the focus will be in the first two subtasks and camouflaged object detection in particular.

Surveillance software needs to make decisions in real time in order to be useful [4]. One of the main goals of this system was to have it work in real-time. Using multiple cameras generates a lot of video data, which can slow down the implementation. Minimizing network traffic within the surveillance system should be one of the goals of the implementation. For example, one way to minimize network traffic is by transmitting and processing only frames in which activity has been detected [4]. Appropriate image resolution and algorithm selection were necessary to pick the combination that gave the best performance in real-time.

Another important goal was to be able to create the system using Commercial Off-The-Shelf systems (COTS). USB webcams were used for this purpose.

Finally, implementing highly complex computational algorithms is not the goal of this research. The main idea is to discover the minimum computational algorithm that can still perform the tasks adequately. This preference of less computationally expensive algorithms is due to the fact that part of this research will eventually be used in camera motes, where power constraints are an issue. The algorithms need to be real-time and suitable for use on low cost, low power, embedded COTS systems [21].

Some system constraints are:

- Continually and naturally varying outdoor lighting.
- Moving trees, brush and clouds.
- Targets use camouflage to blend in, so the system must be very sensitive. Target fragmentation is expected since parts of the target will often match the background. Large amounts of occlusion cause additional fragmentation.
- Targets consist primarily of persons and vehicles.
- The algorithms need to be real-time and use COTS.

1.3.2 Motivation

While testing segmentation methods on camouflaged targets, it was apparent that their performance did not provide good results. Due to similarities and disparities between the background and the camouflaged target, the segmentation appeared fragmented. After background subtraction and thresholding was applied on the image, it was not clear whether the blobs in the binary image belonged to a same larger object or if several smaller objects were being detected. The same situation was observed from different cameras. However, the segmentation information from each camera provided distinct details about the target. This

observation led to combine together the segmented data from each camera to obtain an improved segmentation of the target. In theory, this combined segmentation should be better than any single camera segmentation alone.

An added advantage of combining camera information is noise removal. Since noise from a single camera is not present in another view, noise can be reduced by averaging the different views. This also applies to image noise due to illumination changes or changing weather conditions.

1.4 Outline

In Chapter 1, general background information regarding surveillance is presented. A brief history of surveillance, including its advantages, challenges and future is discussed. An overview of the thesis project and the motivation to pursue investigation in this area are expanded. Chapter 2 focuses attention on the theoretical background necessary to understand the intrinsic details of the system. These topics range from reference subtraction models and the Kalman filter to algorithms on homography and classification. Chapter 3 gives specific details on the system physical construction and the software developed for this thesis. It discusses the mock scenario setup, image acquisition procedure, and the entire system development, including Matlab code snippets. Chapter 4 covers the different experiments attempted and presents the obtained results. Chapter 5 discusses the conclusions of the research as well as the future work and improvements that can be done to this research.

Chapter 2: Theoretical Background

This section covers several topics which are necessary to understand how the designed surveillance system works. We provide a review of reference model techniques, auto-thresholding, pinhole camera model, homography, centroid and classification. Two techniques for camouflage breaking discussed in the literary review are also expanded.

2.1 Reference Subtraction Models

In both nature and automated surveillance, motion is a key feature in detecting objects from their background. A common approach to automated object detection is achieved by observing differences between two or more frames and creating a difference map. If no differences are present in the video, nothing can be detected. Reference models operate on this very simple principle. Suppose we point a camera at a scene we wish to monitor and assume no targets are currently populating the scene. If we save this captured image as our reference image B, then any change in this scene can be detected by simply performing a pixel-by-pixel comparison between our reference image B and any new incoming image [21]. In practice, these differences can be created by the target's movement, illumination changes, background movement or camera hardware corrections. Therefore it is important to distinguish the large differences from the small differences using a threshold. It is assumed that the images that are to be differenced are the same size. The equation for a reference model is

$$d(x,y) = \begin{cases} 1, & if\ |Frame - Reference| > T \\ 0, & otherwise \end{cases}$$

where *T* is a predefined threshold and *d(x,y)* is the binary difference map.

The following three methods are techniques that were used to create reference images and detect differences in frames during this research.

2.1.1 Background subtraction

The most intuitive and easiest method for motion detection is background subtraction using a reference background image. A background image of a static scene is acquired and continuously compared to every incoming image frame, as shown in Figure 2.1. The stationary elements of the image cancel each other out. Only the nonzero entries that correspond to moving image components remain. The resulting image can then be thresholded to reveal the possible targets. The main drawback of this method is that an initial static background image is needed and is not usually available in practical applications. Furthermore, even in cases when the background image is available, changes in illumination and background objects, such as tree branches and sea waves can cause false detections. This is considered to be a low complexity method. The equation for background subtraction is

$$d(x,y) = \begin{cases} 1, & if\ |f(x,y,t_i) - f(x,y,t_0)| > T \\ 0, & otherwise \end{cases}$$

where T corresponds to a predetermined threshold, $f(x,y,t_i)$ is the current frame at time i, $f(x,y,t_0)$ is an initial static background taken at time 0 and $d(x,y)$ is the binary difference map.

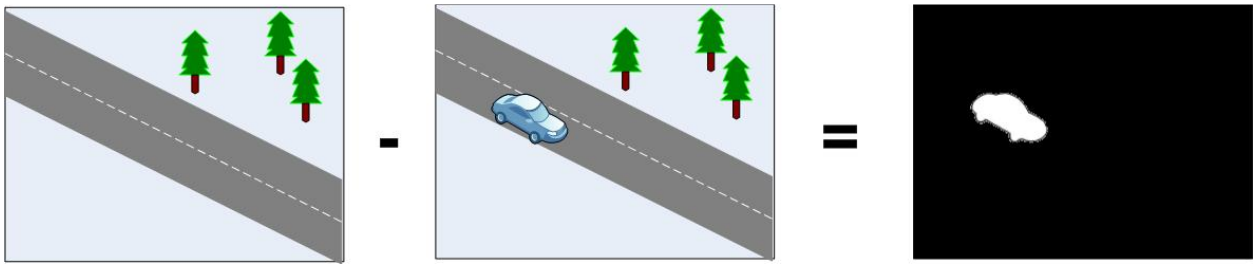


Fig. 2.1: Background Subtraction

2.1.2 Frame to Frame differencing

Changes in highly dynamic image sequences are commonly detected through frame differencing [7]. In this method, every incoming frame is differenced to its previous frame and if the difference is greater than a threshold, the pixel is considered as foreground. One advantage to this method is that no initial background image is required. This method is promising as it quickly adapts to changes in the background and has a small computational load. However, there are several drawbacks to this method. First, solid moving objects are not segmented entirely, but only their outlines, as seen in Figure 2.2. Second, an object can avoid detection if it remains still. The method only works in particular conditions of the object's speed and frame rate. Third, once the object decides to move again it will create a difference in the area it had been, called a ghost image. This ghost image leads to a temporary false detection. The mathematical expression for frame to frame differencing is

$$d(x,y) = \begin{cases} 1, & if\ |f(x,y,t_i) - f(x,y,t_{i-1})| > T \\ 0, & otherwise \end{cases}$$

where T is a predetermined threshold, $f(x,y,t_i)$ is the current frame at time i, $f(x,y,t_{i-1})$ is the frame at time $i-1$ and $d(x,y)$ corresponds to the binary difference map.

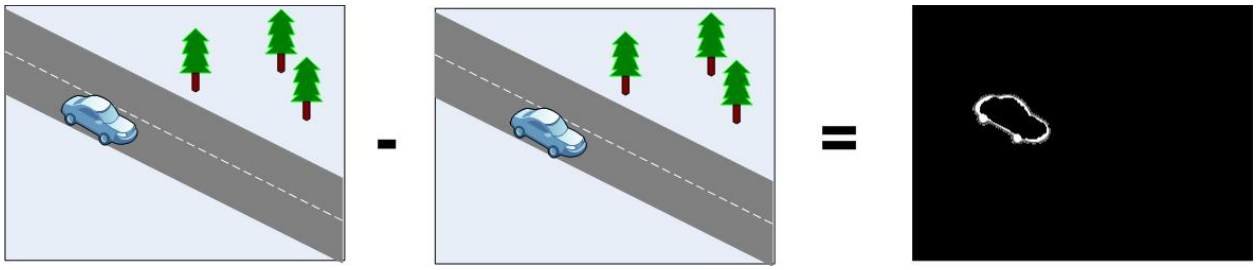


Fig. 2.2: Frame to Frame Differencing

2.1.3 Background Averaging

Another solution to the segmentation problem is creating a background image from several captured frames. In earlier background averaging methods, several frames would be stored and then averaged. This required having enough storage for the number of frames to be averaged. This storage can be avoided by using a running weighted average. Every incoming frame is averaged with the previous mean background. The constant α is used to control how fast the background evolves.

$$\bar{u}_i = \alpha * f(x, y, t_i) + (1 - \alpha)\bar{u}_{i-1}$$

where $\bar{u}_i$ is the current background average at time *i*, $\bar{u}_{i-1}$ is the previous background average at time *i-1*, $f(x,y,t_i)$ is the current frame at time *i*, and α is the learning rate for background averaging. Figure 2.3 illustrates background averaging. As in the previous methods, each frame is differenced with the background mean and then thresholded. This can be expressed mathematically as

$$d(x, y) = \begin{cases} 1, & if\ |f(x, y, t_i) - \bar{u}_i| > T \\ 0, & otherwise \end{cases}$$

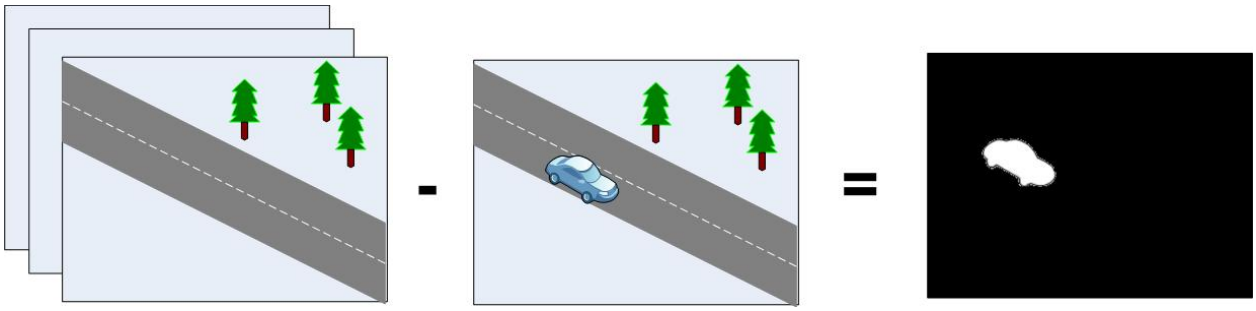


Fig. 2.3: Background Averaging. The left part of the figure depicts a couple of frames that are averaged in order to be subtracted to the current frame. The result is the segmented image.

2.2 Thresholding

The three reference subtraction models previously covered all use a threshold value *T* to convert the image map into a binary image of ones and zeros. If the difference is greater than the threshold, it is considered as a 1 in the difference map. If it is less than the threshold, it will be considered as a 0. Thresholding has been popularized due to its ease of implementation. Global thresholding refers to using a single threshold value for the whole image. The success of this method depends greatly on the image content and on how distinct the parts of the histogram are. Fixed global thresholding can be useful in cases where the threshold can be set manually by a human observer. Figure 2.4 shows the result of thresholding a grayscale image with a fixed global threshold. By observing the histogram of the grayscale image, a threshold value of 148 was chosen, since it best separates the parts of the histogram. This fixed threshold works well to segment the apples and oranges from the table.

However, in dynamic scenes, a fixed global threshold does not offer the flexibility to adapt to changes in the scene. A way to set the threshold automatically is necessary to develop a real time implementation. The following algorithm, shown in Table 2.1, details how to find the threshold value T using only the image contents without a human observer [1].

Table 2.1: Auto Threshold Algorithm

Auto Threshold Algorithm
1. Select an initial estimate for T. 2. Segment the image using T. This creates two groups G_1 and G_2. G_1 corresponds to the pixels greater than the threshold and G_2 corresponds to the pixels less than or equal to the threshold. 3. Compute the average gray level values μ_1 and μ_2 for each of the regions G_1 and G_2. 4. Compute a new threshold value: $$T = \frac{1}{2} * (\mu_1 + \mu_2)$$ 5. Repeat process starting from step 2, until the difference in T in successive iterations is smaller than an error threshold.

A more elaborate method to set the threshold automatically in grayscale images is called Otsu's method [31]. This method is un-parametrized and unsupervised and uses the zero-th and first order cumulative moments of the grayscale histogram.

2.3 Pinhole Camera Model

The pinhole camera model is considered the ideal model of a camera. It does not take into account practical considerations such as geometric distortions caused by the lenses and the fact that cameras have only discrete image coordinates. The relationship between the coordinates of a 3D point and its projection onto an image plane is described by the pinhole camera model. To simplify the pinhole camera model, the image plane is placed between the focal point of the

camera and the object, to avoid having an inverted image. Figure 2.5 shows the pinhole camera model. A point X from the 3D world is projected onto the image plane. On the right of Figure 2.5, the pinhole camera model is seen from a side view.

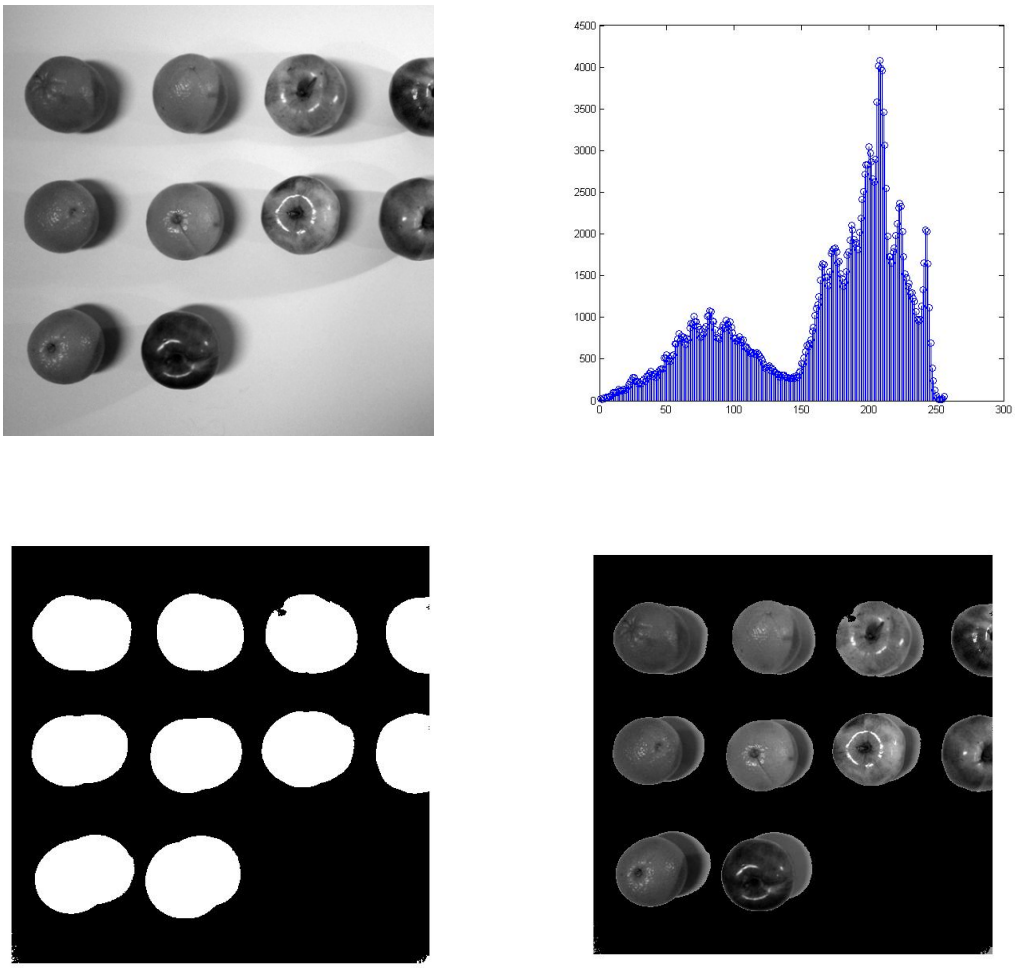


Fig. 2.4: Thresholding: a) Grayscale image (top left); b) Histogram (top right);
c) Binary Threshold Mask (bottom left); d) Image Result after Threshold (bottom right)

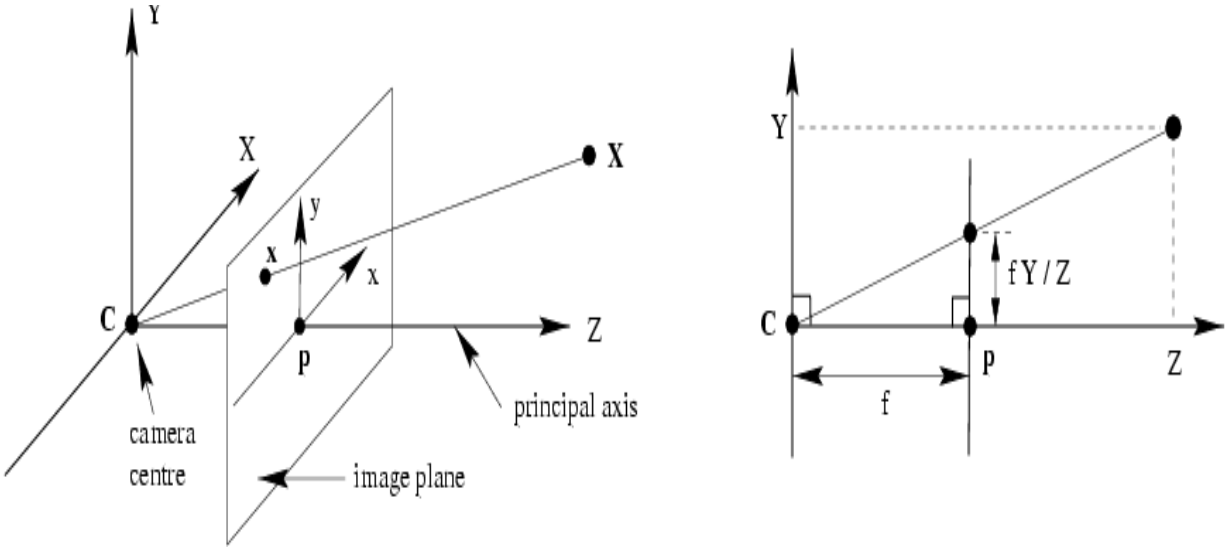


Fig. 2.5: Pinhole Camera Model

In order to find the 2D height y at which the 3D point X will be projected, it is necessary to look at the side view of the pinhole camera model. Since, the point X has a 3D height of Y and is at a distance Z from the camera center C, it can be observed that the following equation holds

–

The angle C is the same for the larger triangle as for the smaller triangle. Therefore, the following equation also holds if we look at the point x on the image plane, located at a distance *f* from the camera center

–

where y is the vertical side of the smaller triangle given by

–

The same geometrical observations can be done for the 2D horizontal location of the point x on the image plane. The 3D coordinates are mapped on the 2D image plane as follows

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \leftrightarrow \begin{bmatrix} f\frac{X}{Z} \\ f\frac{Y}{Z} \end{bmatrix}$$

2.4 Homogeneous coordinates and matrices

As an introduction to projective transformations the topic of homogeneous coordinates and matrices will first be discussed. Homogeneous matrices allow spatial transformations of images, as is discussed in the following section. A 2D column matrix such as $\begin{bmatrix} x \\ y \end{bmatrix}$ can represent either a vector between two points or the position of a particular point. In order to avoid confusion, a third component is added. This third component will be 1 for a point and 0 in the case of a vector. Thus, the point (x,y) is represented by $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ and the vector x**i**+y**j** is represented by $\begin{bmatrix} x \\ y \\ 0 \end{bmatrix}$. For a point $\begin{bmatrix} x \\ y \end{bmatrix}$, its homogeneous coordinates are $\begin{bmatrix} Kx \\ Ky \\ K \end{bmatrix}$ for any nonzero constant K. Generally, K is equal to 1. The overall scaling is not critical.

Homogeneous coordinates are used to write the action of a perspective camera as a matrix. When going from non-homogeneous to homogeneous coordinates a 1 is added as the next coordinate. When going from homogeneous to non-homogeneous coordinates, the first two coordinates are divided by the third coordinate.

Any 2D point can be represented by the vector $\begin{bmatrix} x \\ y \end{bmatrix}$and transformed with a 2x2 matrix$\begin{bmatrix} A & B \\ C & D \end{bmatrix}$. Adding an extra axis, z, the original figure can be written as $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ and transformed using a padded matrix such as

$$\begin{bmatrix} A & B & 0 \\ C & D & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

This matrix is called a homogeneous matrix and can be modified to add fractions or multiples of z to the x and y components. For a translation, the following matrix would be used

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{bmatrix},$$

where T_x is the translation in the x axis, and T_y is the translation in the y axis.

For scaling, the transformation matrix is

$$\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

where S_x is the scaling in the x axis, and S_y is the scaling in the y axis.

Shearing an image can also be performed by a matrix transformation. The matrix

$$\begin{bmatrix} 1 & Sh_y & 0 \\ Sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

allows shearing an image, where Sh_x and Sh_y are the shearing constants for x and y, respectively.

Rotation by an arbitrary angle is also possible using a matrix transformation. The matrix for this operation uses r, which is the angle of rotation

$$\begin{bmatrix} \cos(r) & \sin(r) & 0 \\ -\sin(r) & \cos(r) & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

The results of applying these transformations to an image are demonstrated in Figure 2.6.

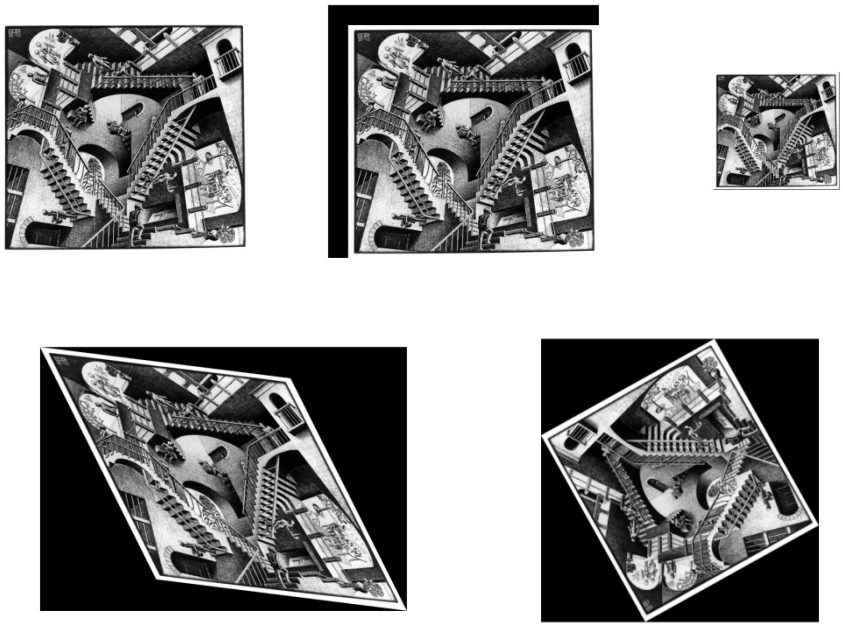


Fig. 2.6: Examples of Spatial Transformations. a) Original Image (top left); b) Translation (top center); c) Scale (top right); d) Shear (bottom left); e) Rotation (bottom right)

2.5 Homography

From the pinhole camera model, 3D image points from real world objects are projected onto a 2D image plane, according to a viewer's position and perspective. Image points from one view are related to the image points in another view by a planar homography. Homography plays a key role in multiple view geometry and can be used for calibration, image registration, image mosaicking and other vision techniques. Homographies relate planar views and do not recover 3D information.

A planar homography relates points on planes through a non-singular linear relationship. A non-singular matrix is a square matrix that has an inverse. A homography matrix can be determined by finding constraints to fix the 8 degrees of freedom in the equations. A homography can be estimated from 4 matching pairs of points from two views. Each matching pair gives two

constraints and solves for two degrees of freedom. Therefore, the minimum necessary data to calculate a homography are four corresponding pairs from two distinct camera views. A homography is also called a planar projective transformation. A projective transformation describes what happens to the positions of objects when the observer changes its location. Figure 2.7 shows how a 3D object is projected onto each camera's 2D image plane. The planes from each camera can then be related by a homography. Different camera views can be obtained by using homographies, including completely new views as the case of the top ground plane view.

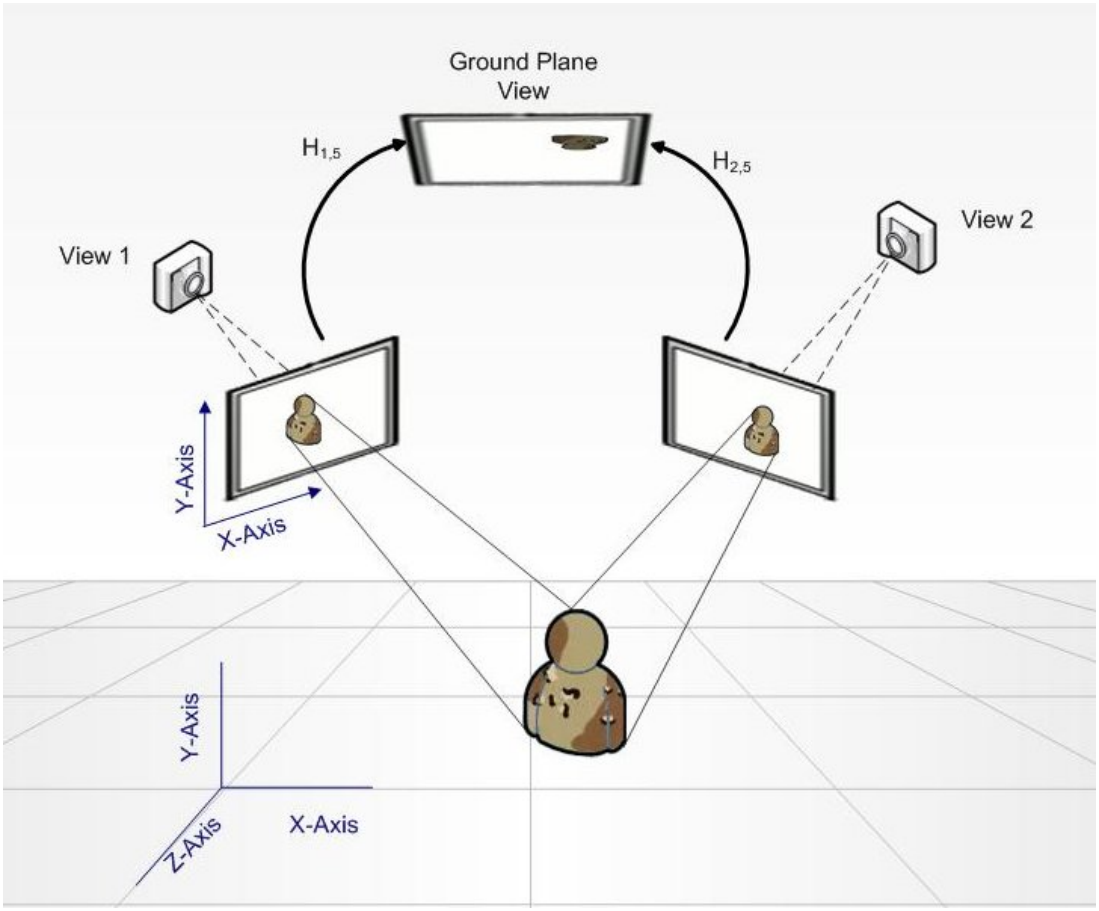


Fig. 2.7: Homographic Correspondence between Different Views

A planar projective transformation can be represented by a non-singular 3x3 matrix shown below

$$\begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

This equation can be written simply as

$$\begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \end{bmatrix} = H \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

H is a 3x3, non-singular, homogeneous matrix, which means that only the ratio of the matrix elements is significant. There are eight independent ratios amongst the nine elements of H. Because of these eight ratios, a projective transformation has eight degrees of freedom.

Let the coordinates of two matching points on an plane be (x,y) and (x',y'), the projective transformation can be written in inhomogeneous form as:

$$x' = \frac{x'_1}{x'_3} = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}}$$

$$y' = \frac{x'_2}{x'_3} = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}}$$

Each corresponding point is responsible for generating two equations for the elements of H.

$$x'(h_{31}x + h_{32}y + h_{33}) = h_{11}x + h_{12}y + h_{13}$$

$$y'(h_{31}x + h_{32}y + h_{33}) = h_{21}x + h_{22}y + h_{23}$$

Four point correspondences are sufficient to solve for H, as long as the four points are not collinear.

2.5.1 An Example of Homography Matrix Computation

Suppose the correspondences $(x, y) \leftrightarrow (x', y')$ are known for four points, then H can be determined. We can express a set of point correspondences using the following notation

$$\begin{pmatrix} x_1, y_1 \\ x_2, y_2 \\ x_3, y_3 \\ x_4, y_4 \end{pmatrix} \leftrightarrow \begin{pmatrix} x'_1, y'_1 \\ x'_2, y'_2 \\ x'_3, y'_3 \\ x'_4, y'_4 \end{pmatrix}$$

For the following example, assume four pairs of corresponding coordinates are known. For example, the coordinate (1,1) in one image corresponds to the (2,1) in the other image plane, (0,0) corresponds to (0,0) and so forth for all four pairs.

$$\begin{pmatrix} 0,0 \\ 1,0 \\ 0,1 \\ 1,1 \end{pmatrix} \leftrightarrow \begin{pmatrix} 0,0 \\ 1,0 \\ 0,1 \\ 2,1 \end{pmatrix}$$

Using the first correspondence points $(0,0) \leftrightarrow (0,0)$ on the general transformation equation

$$\lambda_1 \begin{bmatrix} x'_1 \\ y'_1 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} x_1 h_{11} + y_1 h_{12} + h_{13} \\ x_1 h_{21} + y_1 h_{22} + h_{23} \\ x_1 h_{31} + y_1 h_{32} + h_{33} \end{bmatrix}$$

can be set as

$$\lambda_1 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{13} \\ h_{23} \\ h_{33} \end{bmatrix}$$

From which the first H matrix coefficient values can be calculated using

$$\begin{bmatrix} h_{13} \\ h_{23} \\ h_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \lambda_1 \end{bmatrix}.$$

Using the second correspondence points $(1,0) \leftrightarrow (1,0)$ and $h_{13} = 0,\ h_{23} = 0$

$$\lambda_2 \begin{bmatrix} x'_2 \\ y'_2 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} x_2 h_{11} + y_2 h_{12} + h_{13} \\ x_2 h_{21} + y_2 h_{22} + h_{23} \\ x_2 h_{31} + y_2 h_{32} + h_{33} \end{bmatrix}$$

can be set as

$$\lambda_2 \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & 0 \\ h_{21} & h_{22} & 0 \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} \\ h_{21} \\ h_{31} + h_{33} \end{bmatrix}$$

$$\begin{bmatrix} h_{11} \\ h_{21} \\ h_{31} + h_{33} \end{bmatrix} = \begin{bmatrix} \lambda_2 \\ 0 \\ \lambda_2 \end{bmatrix}$$

Now, using the third correspondence points $(0,1) \leftrightarrow (0,1)$ and $h_{13} = 0,\ h_{23} = 0, h_{11} = h_{31} + h_{33}, h_{21} = 0$, we have

$$\lambda_3 \begin{bmatrix} x'_3 \\ y'_3 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_3 \\ y_3 \\ 1 \end{bmatrix} = \begin{bmatrix} x_3 h_{11} + y_3 h_{12} + h_{13} \\ x_3 h_{21} + y_3 h_{22} + h_{23} \\ x_3 h_{31} + y_3 h_{32} + h_{33} \end{bmatrix}$$

can be set as

$$\lambda_3 \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{31} + h_{33} & h_{12} & 0 \\ 0 & h_{22} & 0 \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{12} \\ h_{22} \\ h_{32} + h_{33} \end{bmatrix}$$

$$\begin{bmatrix} h_{12} \\ h_{22} \\ h_{32} + h_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ \lambda_3 \\ \lambda_3 \end{bmatrix}.$$

Finally, using the last correspondence points $(1,1) \leftrightarrow (2,1)$ and all the previously found constraints $h_{13} = 0,\ h_{23} = 0, h_{11} = h_{31} + h_{33}, h_{21} = 0, h_{22} = h_{32} + h_{33}, h_{12} = 0$ we arrive to

$$\lambda_4 \begin{bmatrix} x'_4 \\ y'_4 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_4 \\ y_4 \\ 1 \end{bmatrix} = \begin{bmatrix} x_4 h_{11} + y_4 h_{12} + h_{13} \\ x_4 h_{21} + y_4 h_{22} + h_{23} \\ x_4 h_{31} + y_4 h_{32} + h_{33} \end{bmatrix},$$

which can be rewritten as

$$\lambda_4 \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{31} + h_{33} & 0 & 0 \\ 0 & h_{32} + h_{33} & 0 \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{31} + h_{33} \\ h_{32} + h_{33} \\ h_{31} + h_{32} + h_{33} \end{bmatrix}.$$

By matrix multiplication, each row is multiplied by the column vector resulting in

$$\begin{bmatrix} h_{31} + h_{33} \\ h_{32} + h_{33} \\ h_{31} + h_{32} + h_{33} \end{bmatrix} = \begin{bmatrix} 2\lambda_4 \\ \lambda_4 \\ \lambda_4 \end{bmatrix}.$$

Taking λ_4 as a constant and solving the above for H using simultaneous equations we get

$$H = \lambda \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 2 \end{bmatrix}.$$

We can conclude that the forward transformation matrix H maps the correspondence points. In order to un-map the points, the inverse of the H matrix can be used.

2.6 The Direct Linear Transformation (DLT) using point correspondences

Points and lines are the most common features used to estimate homographies. One of the methods that uses point correspondences is the DLT. Given a set of four 2D to 2D point correspondences $x \leftrightarrow x'$, H is to be calculated. The equation $x' = Hx$ can be expressed in terms of the vector cross product as $x_i' \times (Hx_i) = 0$. Letting the jth row of the matrix H be denoted by h^{jT},

$$Hx_i = \begin{bmatrix} h^{1T}x_i \\ h^{2T}x_i \\ h^{3T}x_i \end{bmatrix}$$

and letting $\mathbf{x}_i' = (x_i', y_i', w_i')^T$, the cross product can be expressed as

$$x_i' \times Hx_i = \begin{bmatrix} y'_i h^{3T}x_i - w'_i h^{2T}x_i \\ w'_i h^{1T}x_i - x'_i h^{3T}x_i \\ x'_i h^{2T}x_i - y'_i h^{1T}x_i \end{bmatrix}.$$

The previous equation can be rewritten in the form

$$\begin{bmatrix} 0^T & -w'_i x_i{}^T & y'_i x_i{}^T \\ w'_i x_i{}^T & 0^T & -x'_i x_i{}^T \\ -y_i x_i{}^T & x'_i x_i{}^T & 0^T \end{bmatrix} \begin{bmatrix} h^1 \\ h^2 \\ h^3 \end{bmatrix} = 0.$$

We can rewrite this expression as

$$L_i h = 0$$

where *i* represents each correspondence point pair, L_i is taken to be a 3×9 matrix and $h = \begin{bmatrix} h^{1T} \\ h^{2T} \\ h^{3T} \end{bmatrix}$ corresponds to a 9x1 vector formed by vertically concatenating each row of H.

Three equations are expressed in the L_i matrix, however only two of them are linearly independent and necessary to solve the equations. Each of the four point correspondences gives two equations in the entries of H. Therefore, the equation can be reduced to

$$\begin{bmatrix} 0^T & -w'_i x_i{}^T & y'_i x_i{}^T \\ w'_i x_i{}^T & 0^T & -x'_i x_i{}^T \end{bmatrix} \begin{bmatrix} h^1 \\ h^2 \\ h^3 \end{bmatrix} = 0.$$

With the four point correspondences, a set of Lh=0 equations can be determined, where L is the result of stacking the rows of L_i from each correspondence and h is the vector of unknown entries of H.

Table 2.2: Direct Linear Transformation Algorithm

Direct Linear Transformation Algorithm
Given n>=4 2D to 2D corresponding points x↔x', the homography matrix H that will solve $x_i'=Hx_i$ is to be found. 1. For each correspondence x↔x' compute L_i. Only the first two rows are necessary. 2. Construct n 2×9 matrices L_i into a single 2n×9 matrix L. 3. Calculate the Singular Value Decomposition of L as UDV^T with D diagonal with positive diagonal entries, arranged in descending order down the diagonal, then h is last column of V. 4. Determine H from h.

2.7 Computation of Binary Image Centroids

From geometry, it is known that the centroid of a figure is the point where all straight lines that divide a planar figure into parts of equal moment intersect. In computer vision, the centroid is defined as the average of all the foreground pixels on a binary image. In layman's terms, it is the center of all the white points on a binary figure. The centroid is a useful feature that can reveal certain characteristics about an image. It can be used to determine a distinctive point on a blob and use this information to track it.

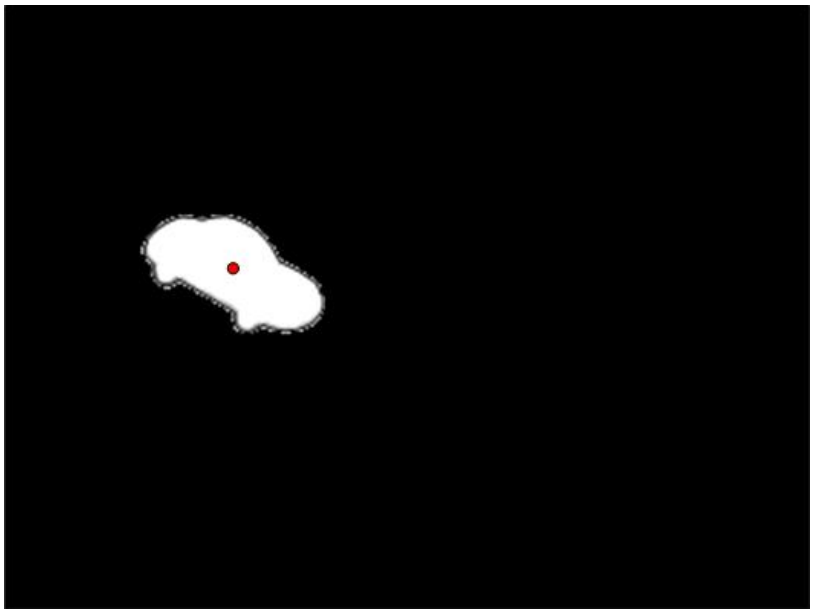


Fig. 2.8: Binary Image Centroid Result

Let the *I(x,y)* denote the intensity value in a binary image I at the pixel location (x,y),

$$I(x,y) = \begin{cases} 1, & foreground\ pixel \\ 0, & background\ pixel \end{cases}$$

The zero order moment for the foreground pixels is given by

$$M_{00} = \sum_x \sum_y I(x,y)$$

M_{00} turns out to be a sum of all the foreground pixel values across the whole image. M_{00} is called the binary area of the image. The first moments of the image, M_{01} and M_{10} are also necessary to calculate the centroid. These are found using

$$M_{10} = \sum_x \sum_y x * I(x,y)$$

$$M_{01} = \sum_x \sum_y y * I(x,y).$$

Once the area and first moments of an image are known, the centroid $(\bar{x}, \bar{y})$ can be calculated as

$$\bar{x} = \frac{\sum_x \sum_y x * I(x,y)}{M_{00}}$$

$$\bar{y} = \frac{\sum_x \sum_y y * I(x,y)}{M_{00}}$$

As an example, figure 2.8 indicates the centroid with a red dot for a segmented vehicle.

2.8 Kalman Filter

The Kalman Filter is one of the most common tools used for stochastic estimation of a signal from noisy sensor measurements. In essence, the Kalman Filter is a set of equations that give the optimal estimator that minimizes the estimated error covariance. The Kalman Filter finds this estimator by alternating through a prediction and correction process. The Kalman filter uses a form of feedback control. The current state is estimated and compared to noisy measurements, before being fed back to the filter. The two equations used by the Kalman Filter can be divided into the time update (predictor) and measurement update equations (correction). The time update equation calculates the a priori estimates for the next state using the current state and the error covariance estimates. The measurement update equation incorporates a new measurement to the a priori estimate to calculate an improved estimate a posteriori.

The Kalman Filter equations attempt to estimate the state *x* of a discrete-time controlled process given by the stochastic equation

$$x_k = Ax_{k-1} + Bu_k + w_{k-1}$$

using measurements modeled as

$$z_k = Hx_k + v_k.$$

The matrix A is a n×n matrix that relates the previous state x_{k-1} to the current state x_k. B is a n×l matrix that relates the control input u to the current state x_k. This control input is optional. In the measurement equation, the matrix H is a m×n matrix relating the current state x_k to z_k, which is the current measurement. w_k and v_k are independent random variables that represent the process and measurement noise, respectively. These noise measurements are assumed to be white and drawn from a normal distribution. The process noise w_k has a covariance Q, while the measurement noise v_k has a covariance R.

$$p(w) \sim N(0, Q)$$

$$p(v) \sim N(0, R)$$

Some of the advantages of this filter over other estimation techniques are its relative simplicity and robustness. Applications for the Kalman Filter exist in autonomous or assisted navigation and tracking in interactive computer graphics [34].

2.9 Classification

Classification refers to identifying a target after it has been segmented from the background. Such a classification could be used to distinguish the type of object or its activity. One of the simplest ways of classifying a target is by its spatial features. Once a blob has been detected, features such as its pixel height and width can be calculated to determine if the object is a vehicle or a person, for example. A simple classification algorithm is presented in Table 2.3.

Table 2.3: Classification Algorithm

Classification Algorithm
1. Locate the blob's outermost pixels on the x axis, x_{left} and x_{right}. 2. Locate the blob's outermost pixels on the y axis, y_{top} and y_{bottom}. 3. Find the blob's width by subtracting x_{right} - x_{left}. 4. Find the blob's height by subtracting y_{top} - y_{bottom}. 5. Calculate the ratio of the blob's width/height. 6. If the ratio is greater than 1, the blob is classified as a vehicle. 7. If the ratio is less than 1, the blob is classified as a person. 8. Otherwise, the blob is unidentified.

Intuitively, blobs are classified by comparing the width-height ratio, under the assumption that vehicles are more elongated than persons. Mathematically we represent this assumption as

$$Blob = \begin{cases} Vehicle & , |x_{right} - x_{left}| / |y_{top} - y_{bottom}| > 1 \\ Person & , |x_{right} - x_{left}| / |y_{top} - y_{bottom}| < 1 \\ Unidentified & , otherwise \end{cases}.$$

This classification can be done for a single camera. When dealing with a multiple camera setting, each individual camera can classify the blob independently and afterwards vote for the corresponding classification. If a camera detects a vehicle, it will increment the vehicle classification vote by one. In a similar manner, if a camera detects a person, it will increment the person classification vote by one. If the total votes for the vehicle classification exceed the total votes for the person classification, the blob will be classified as a vehicle. If the total votes for the person classification exceed the total votes for the vehicle, the blob is classified as a person.

In the event of a tie, the blob is labeled as an unidentified object. For n cameras, the voting classification is represented as

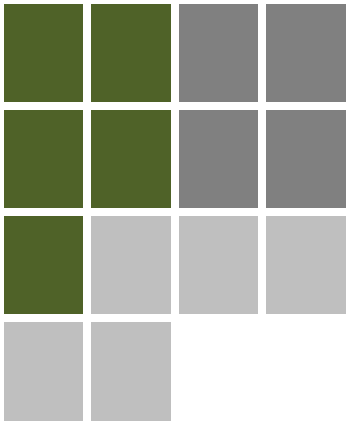
$$Blob = \begin{cases} Vehicle, & if \sum_{i=1}^{n} vehicle_i > \sum^{n} person_i \\ Person, & if \sum_{i=1}^{n} vehicle_i < \sum^{n} person_i \\ Unidentified, & if \sum_{i=1}^{n} vehicle_i = \sum_{i=1}^{n} person_i \end{cases}$$

where *n* corresponds to the total number of cameras nodes, $vehicle_i$ is the individual vote from camera *i*, if camera *i* detected a vehicle and $person_i$ is the individual vote from camera *i*, if camera *i* detected a person.

2.10 GLCM

As discussed in the literary review, the Grey Level Co-occurrence Matrix can be used to perform a texture analysis that can detect camouflaged defects as described by Bhajantri and Nagabhushan [28]. The procedure for texture analysis is to first find the GLCM of an image and then use this matrix to calculate image parameters: energy, entropy, homogeneity, contrast and correlation.

For example, the image in Figure 2.9 shows a 4x4 pixel image and its corresponding values in a matrix. Each color receives its own number in the matrix. In order to calculate the GLCM, a direction must be chosen to compare pixels across the matrix. An offset must also be set to know which pixels will be considered in the process. In this example, the direction is east and the offset is one. This means that the comparison will take place between a pixel and its immediate neighbor to the right.



0	**0**	**1**	**1**
0	**0**	**1**	**1**
0	**2**	**2**	**2**
2	**2**	**3**	**3**

Fig. 2.9: Simple example image for GLCM computation [44].

The GLCM will be created by counting how many times a pixel pair, consisting of a reference pixel and its neighbor, is repeated across the matrix. For example, the combination 0,0 is present 2 times in the image. The direction of the pair matters, so vertical relations between 0 and 0 do not count. The result of the count for each combination will be placed according to the matrix indices shown in Figure 2.10.

	Neighbor Pixel			
Ref. Pixel	0	1	2	3
0	0,0	0,1	0,2	0,3
1	1,0	1,1	1,2	1,3
2	2,0	2,1	2,2	2,3
3	3,0	3,1	3,2	3,3

Fig. 2.10: GLCM Matrix Indices [44].

The resulting GLCM matrix is shown in Figure 2.11. The (0,0) location in the GLCM has a value of 2 since the pair (0,0) occurs two times in the matrix between a pixel value and its neighbor to

the right. In the same manner, the pair (3,3) only occurs once in the image and a 1 is placed in the (3,3) location of the GLCM.

2	2	1	0
0	2	0	0
0	0	3	1
0	0	0	1

Fig. 2.11: GLCM Result [44].

As can be seen by this example, searching for pixel pairs across the image as done by the GLCM is quite computationally intensive. Increasing the number of directions that are compared and the pixel offset contributes to the computations. By itself, the GLCM does not detect camouflaged objects, but it makes the calculation of texture features easier to calculate.

2.11 Camouflage Breaking Using Quasi-Connected Components

Quasi-Connected Components is a novel alternative to morphological processing developed by Boult et al. [21]. This method detects camouflaged objects by taking into account pixel intensities and pixel region areas to close gaps caused by fragmentation. The basic procedure for QCC is shown in Figure 2.12. A high threshold T_h and a low threshold T_l are applied to the difference image. Pixels that are below T_l are rejected, while all other pixels are kept. However, from the remaining pixels, a distinction is made between the pixels greater than T_h and those between T_h and T_l. A down sampled image is created of the difference image, which helps in detecting the blobs of interest. Two conditions must be met in order to accept a region as a target. There must be a pixel greater than the high threshold and a minimum area must be met. Figure 2.12 shows three different possible targets. The yellow target is rejected for

not satisfying the area criteria. The green blob does not have any pixels greater than the high threshold and is also rejected in the final segmentation. The blue target is kept since it satisfies both conditions.

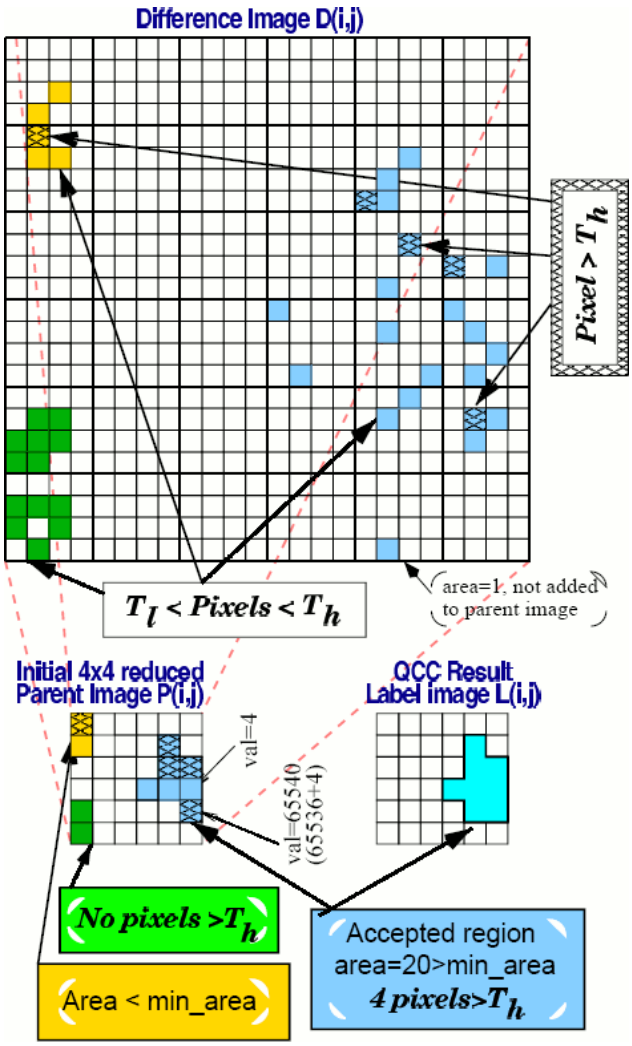


Fig. 2.12: QCC Process. Figure obtained from reference [21].

Chapter 3: Implementation of a Real-Time Multi-Camera Surveillance System

3.1 Overview

In this chapter the implementation of an experimental multi-camera surveillance system is presented. The system is capable of real-time video processing at about 10 to 15 frames per second using QVGA resolution with five cameras working simultaneously. More significantly, the system was implemented with a low cost philosophy using commercial off-the-shelf components (COTS) and the power of the MATLAB processing environment.

One of the objectives of this research is to create a multi-camera research platform that could be easily replicated by others in the future. Besides the physical components of the system, a critical component is the software necessary for its implementation. For this reason, this chapter is written using a tutorial-like format in order to describe in detail the software development. This chapter gives specific details on the system physical construction and the software developed for this thesis. It discusses the mock scenario setup, image acquisition procedure, and the entire system development, including MATLAB code snippets.

A mock scenario was setup in the lab to allow for testing. The scenario attempted to recreate an event where a camouflaged vehicle is attempting to avoid detection. Preliminary work was done using a single camera acquisition and background subtraction methods. This work was extended to multiple cameras and additional processing such as classification and tracking was performed on the video data.

3.2 Multi-Camera Network Setup

In order to create a testing environment for image acquisition, a mock scenario was set up. The scenario consisted of PVC tubing arranged in the form of a cube. Four commercial off the shelf (COTS) webcams were placed near each of the bottom corners of the structure, facing the inside of the cube. Another camera was placed at the top center face of the cube. This camera was used for calibration and to compare the results with a ground truth acquisition. The webcams used were two Logitech Quickcam Deluxe for Notebooks and three LiveCam! Notebook Pro webcams. These cameras can operate at a 640x480 resolution, however the system works with QVGA or 320x240 to improve performance in real-time. A remote-controlled car was used as the object to be tracked. Camouflaged fabric was laid on the ground and used cover the car. The texture and colors between the target and background were almost perfect matches, creating a challenging camouflage scenario. The mock scenario is represented in Figure 3.1.

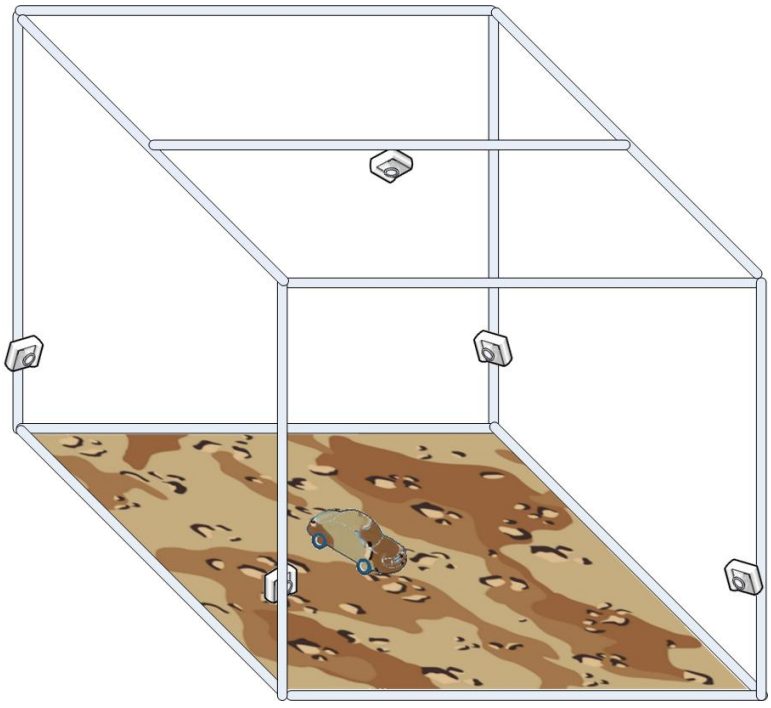


Fig. 3.1: Mock Scenario Setup

3.3 Physical Characteristics of the Multi-Camera Mock Scenario

The dimensions of the PVC structure were designed so that the cameras would have the same distance to the center of the cube and also maximized the field of view of the top camera. USB extension cables were used to connect the USB webcams to the PC. USB specifications limit the maximum length of a USB extension cable. The maximum length for a USB 2.0 cable is 5 meters (16 feet 5 inches). This also was taken into consideration during the design of the mock scenario.

Fig. 3.2: Logitech Quickcam Deluxe for Notebooks and LiveCam! Notebook Pro

The five USB cameras were connected to the ports on a PC. The PC had a 2.20 Ghz AMD Athlon 64 X2 Dual Core processor, 1.93GB of available RAM and was running Windows XP and Matlab R2007b.

USB bandwidth issues were encountered when streaming video from all of the cameras at once. The cameras with the higher USB bandwidth consumption were found to be the Logitech cameras. The two Logitech USB cameras were relocated to two dedicated USB ports on the front of the PC. The other three LiveCam! Webcams were left in the back. This split the bandwidth load between two different USB controllers and solved the issue.

3.4 Selection of MATLAB as Development Environment

Several development tools were considered at the time the research was started: LabView, OpenCV, and MATLAB. However, MATLAB was a clear choice due to previous experience integrating webcams and MATLAB. Labview was another strong candidate for a development tool, but was not capable of supporting simultaneous multiple camera acquisition using webcams [29].

For Image Acquisition using webcams two options were available: Video for MATLAB (VFM) [42] and the MATLAB Image Acquisition Toolbox [43]. Both work well for capturing frames from a USB camera and support multiple cameras. The Image Acquisition toolbox was chosen due to the fact that more reference material and support exists for the Image Acquisition toolbox than for VFM.

3.5 System Architecture and Block Diagram

The system architecture is represented by Figure 3.3. Five cameras are connected to a personal computer via the USB ports. The hardware device driver for the cameras allows the cameras to interface with the computer, while the Image Acquisition Toolbox's hardware driver adaptor interfaces the cameras with MATLAB. The Image Acquisition Toolbox also contains functions that start, configure and trigger the cameras. A video source object is created for each camera. Finally, an RGB image is returned into the MATLAB workspace by the video input object where it can be processed.

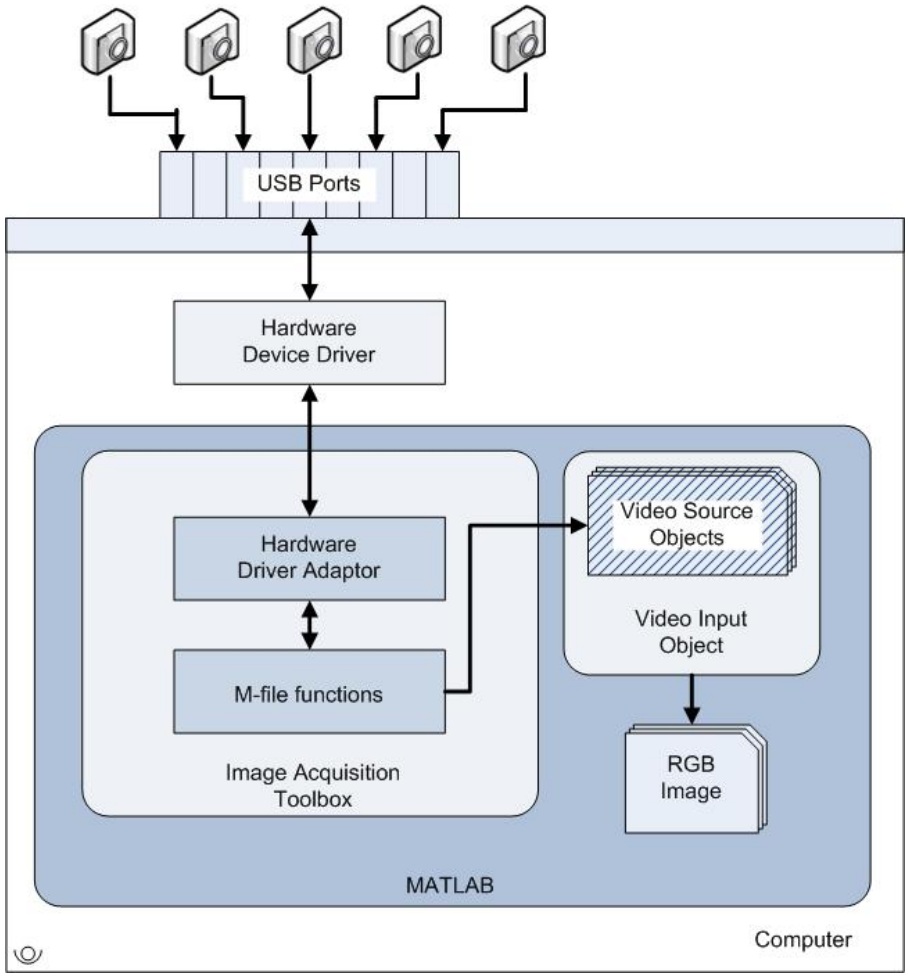


Fig. 3.3: Surveillance System Architecture

The surveillance algorithm implemented in this thesis can be broadly described by the block diagram shown in Figure 3.4. After initial calibration, each camera acquires the image. One of three different motion detection methods is used, with a corresponding threshold. The image is then transformed to the top-view or ground plane. Then, the transformed views from each camera are averaged. Finally, additional processing such as calculating the centroid, enclosing the object with in a rectangle or classifying the object is done.

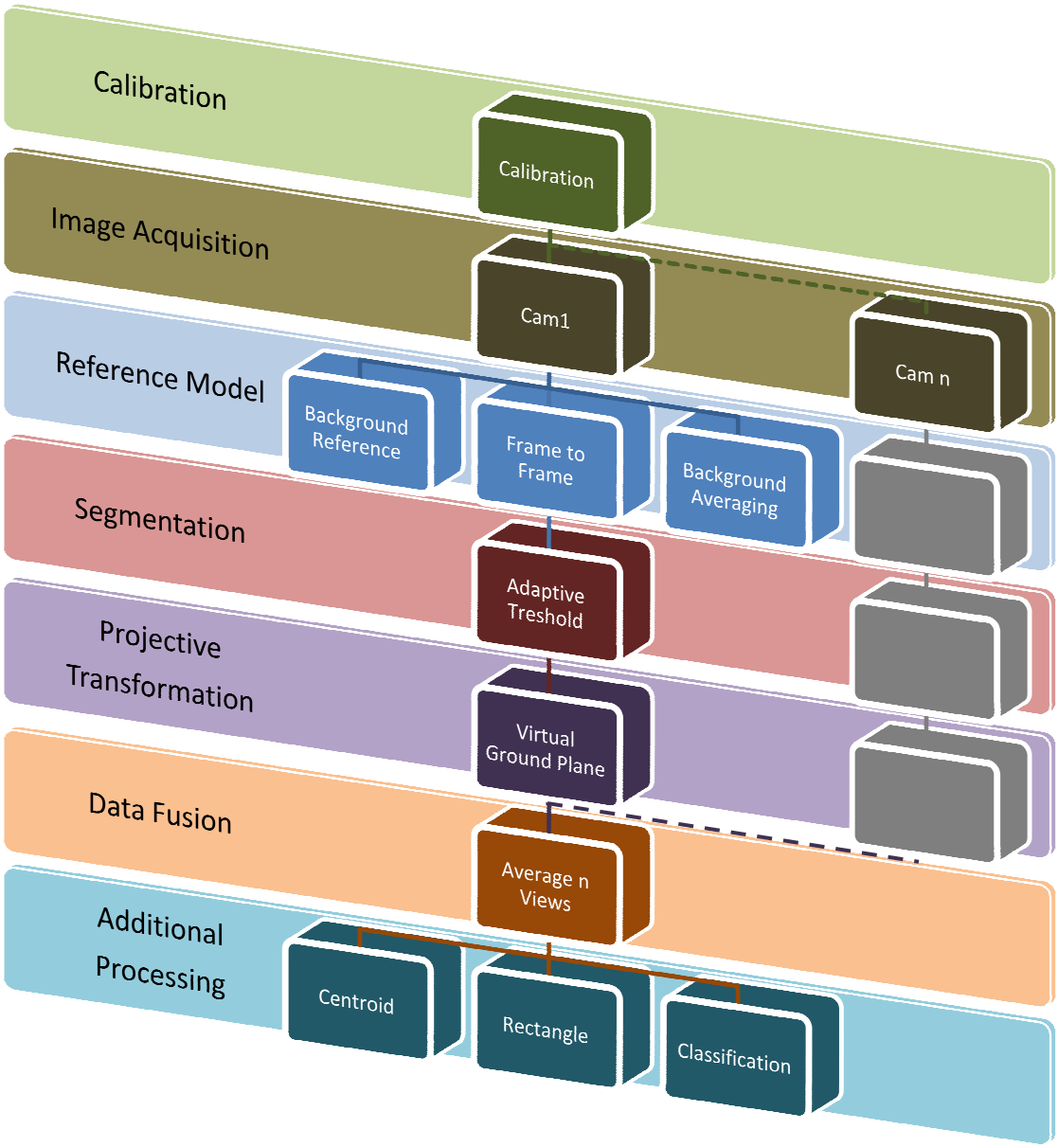


Fig. 3.4: Surveillance System Block Diagram

3.6 MATLAB Based Software Design for a Multi-Camera Surveillance System

This section describes each of the parts of the surveillance system along with implementation details. The intention of this section is to explain how the system code works in an easy to follow manner. For this purpose we have included code snippets where it becomes the most convenient and clear way to elucidate the implementation.

3.6.1 Image Acquisition

3.6.1.1 Single Camera Acquisition

The first step to develop a visual surveillance system is to capture video from a camera. The MATLAB Image Acquisition Toolbox 3 has many features that allow capturing video frames in many different ways. For example, it is possible to acquire a single snapshot, an interval of images triggered by an external event or continuously acquire images.

The basic procedure for acquiring an image using the MATLAB Image Acquisition Toolbox 3 is the following

1. **Install the Image Acquisition Device**. This involves installing software drivers required by the imaging device, connecting the device to the PC and testing that it is working properly.
2. **Retrieve Hardware Information**. Hardware information such as the adaptor name and device ID are necessary to identify the device that is going to be accessed. An adaptor is the software that will be used by the toolbox to communicate with the camera device drivers.

3. **Create a Video Input Object**. A video input object represents the connection between MATLAB and an imaging device. Many aspects of the image acquisition can be configured by modifying the video object's properties.
4. **Acquire Image Data.** First, the video object must be started. This prepares the object for data acquisition. Data acquisition will not begin until a trigger is executed. A trigger can be executed automatically or manually, depending on the device properties. Finally, the acquired frames are stored into memory or a disk file. The frames can then be brought into the MATLAB workspace and processed.
5. **Clean Up**. Once the image acquisition process is finished, the variables and video objects involved with the acquisition must be removed and cleared from the workspace to free resources.

Following the steps mentioned above, the USB camera drivers were installed. Hardware information was retrieved and before acquiring any of the images, the camera was initialized. The following code creates a video object named *vid1*, sets the image resolution to capture at QVGA (320x240) and sets the trigger to manual. A manual trigger means the image capture will not be automatic, but will be triggered by a line in the code 'trigger(vid1)'.

```
imaqhwinfo
info=imaqhwinfo('winvideo');
dev_info1=imaqhwinfo('winvideo',1);
vid1=videoinput('winvideo',1,'RGB24_320x240');
get(vid1)
triggerconfig(vid1,'manual');
set(vid1,'FramesPerTrigger',1);
set(vid1,'TriggerRepeat',Inf);
```

In order to have more control of the capturing process and avoid capturing image frames that

could not be processed, the image acquisition was triggered manually inside a while loop. The following code starts the camera and continuously captures and displays a new frame.

```
start(vid1);
while(1)
    trigger(vid1);
    image1=getdata(vid1,1);
    figure(1);
    image(image1);
end
```

As mentioned in the Clean Up step of image acquisition, to free the resources used during acquisition, the video object must be stopped and deleted when it is not needed anymore.

```
stop(vid1);
delete(vid1)
clear vid1
```

After executing the code, a variable called *image1* has been created. This variable is a matrix containing the captured images through time.

3.6.1.2 Multiple Camera Acquisition

Once single camera acquisition is working, the extension to multiple cameras is straightforward. The same code can be used to trigger acquisition from several cameras. However special considerations must be taken to minimize the time differences between the instant when each camera is triggered. The camera acquisition must be done by stages. First, all the cameras need to be started before starting the while loop. Second, when a camera is triggered, the other cameras should also be triggered in the following lines of code to minimize time differences between different cameras. Then, the results of the acquisition can be displayed for each camera. Finally, each camera needs to be stopped and their video object deleted. The following code shows a simple image acquisition loop for two cameras.

```
start(vid1); start(vid2);
while(1)
    trigger(vid1); trigger(vid2);
    image1=getdata(vid1,1);
    image2=getdata(vid2,1);
    figure(1);
    subplot(1,2,1); image(image1);
    subplot(1,2,2); image(image2);
end
```

The acquired images are stored in variables `image1` and `image2`. From this point the images can be further processed using other MATLAB tools. Once the processing is finished, the clean up stage can be performed with the code shown below to free system resources.

```
stop(vid1); stop (vid2);
delete(vid1); delete(vid2);
clear vid1
clear vid2
```

3.6.2 Processing of RGB images

When the image is captured as an RGB image, it is stored in MATLAB as a *length* x *width* x 3 matrix. The third dimension represents the different color bands, i.e. red, green and blue. For this particular case, the matrix dimensions were 320x240x3. When capturing video, there is a fourth dimension to the stored matrix. This fourth dimension represents the number of frames that have been acquired. In this case it is stored as *length* x *width* x 3 x *frames.* Figure 3.5 illustrates how a video stream is structured in MATLAB.

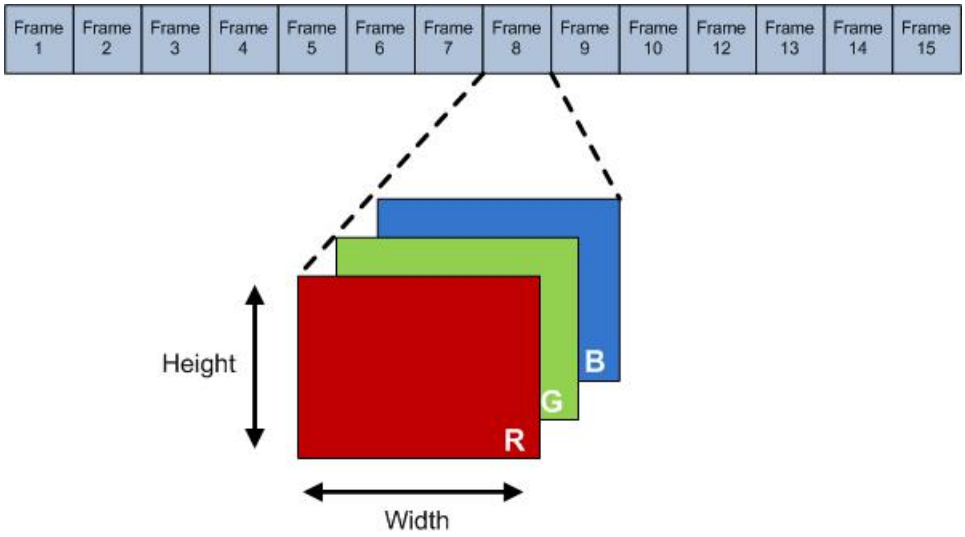


Fig. 3.5: Video stream structure

3.6.3 Calibration

Calibration is an important step to ensure the accuracy of a measuring device. In the case of cameras, calibration is important to infer the position, rotation or color range of a camera. For example, to take advantage of the different viewpoints from multiple cameras, some added information must be known about the camera's location. Calibration is therefore necessary. This system uses calibration to determine how each camera view relates to a bird's eye view of the scene. Knowing this information is important to be able to apply homographic transformations to each of the views and combine their data. Camera calibration is in itself another research problem. The focus of this work is not on calibration. The method described here is a quick way to use MATLAB to calibrate the cameras. Other calibration or image registration techniques can be used [1].

For calibration, a known object is placed in the view of all the cameras. The known object was a magazine with a unique color at each corner: yellow, green, red, blue. Then, images are captured from each camera. At least four corresponding points between each ground camera and the top

view camera are selected by the user. This must be done in a sequential order, selecting one point from a ground camera and then one point from the top view camera until at least four points have been selected for each camera. The code below inputs the image from a ground camera, `image1`, and the image from the top view camera, `image5`, to the *cpselect* tool shown in Figure 3.6. The optional parameter "wait, true" pauses the execution of the MATLAB code until the user has finished selecting the corresponding points. The *cpselect* tool returns the corresponding points in a 4x2 vector for each camera.

```
[input_points1, base_points1]=cpselect(image1,image5,'Wait', true);
```

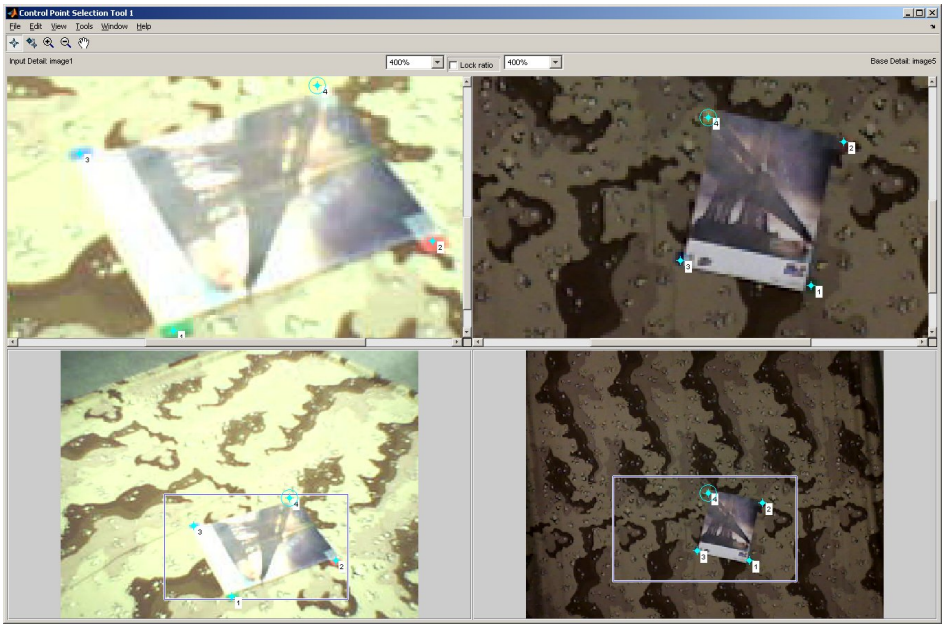


Fig. 3.6: Correspondence Point Selection for Calibration

These corresponding points are then used to obtain the homography transformation matrices using the `maketform()` function, which generates a geometric transformation structure. This structure contains the homography matrix *H*, which maps the points from one view to the top

camera view. A homography matrix *H* must be found for each camera. The MATLAB code to obtain this transformation for camera 1 is given by

```
tform1 = maketform('projective',input_points1,base_points1);
```

where the "projective" parameter specifies that the transformation is projective and not of another kind, such as an affine or a box transformation. A similar computation will be required for each camera in the network. We note that the computation of these matrices is only required during the setup of the system. Once these transformations are found, they can be stored and reused during the multi-camera processing of the video.

3.6.4 Motion Detection

Detecting motion is analogous to detecting changes in an image. These changes may occur in the spatial or temporal domain. It is known that a significant and difficult research problem in automated surveillance is change detection [7]. There are many methods for detecting moving objects. The different methods can be classified into two main groups: Motion-based and spatio-temporal. The spatio-temporal methods include edge detection and the watershed algorithm. The motion-based include optical flow and change detection.

Having a real-time application in mind restricts the allowable computational load for these methods. The thesis thus focuses its attention on change detection methods, particularly low to medium complexity background subtraction methods. Other higher complexity methods, such as representing pixels as a mixture of Gaussians, Mean Shift based Estimation and kernel density estimators [6] are not considered in this work.

3.6.4.1 Segmentation

In essence, finding targets is a change detection problem [21]. Once motion has been detected, it is necessary to determine if it was caused by an object of interest (i.e. a vehicle or person) or an uninteresting object (i.e. camera or illumination noise, tree branch movement, waves in water.) The goal of segmentation is to subdivide an image into its constituent regions or objects. The degree of subdivisions depends on the problem being solved. When the objects of interest are distinguishable, the segmentation process should stop [2]. For a simple motion detection example, the goal of segmentation is to detect the foreground objects and the background objects, given an image frame from a fixed camera.

A popular method for change detection is image differencing, followed by thresholding [9]. However, differences can be attributed to objects of interest or background dynamics. Deciding which values of the difference map are considered information (i.e. moving targets in the scene) and which values are not (i.e. camera noise) is a question that must be faced with this method. A threshold must be applied on the difference map in order to obtain a binary image [8].

One of the most difficult challenges in image processing is segmenting nontrivial images correctly. The accuracy of the segmentation many times determines the success or failure of the computerized analysis procedures [2]. The result of the analysis depends heavily on the segmentation quality, since image segmentation is usually the first step to image analysis [10].

3.6.4.2 Reference Models

Several methods such as active background subtraction and temporal differentiation exist for detecting, tracking and classifying moving targets such as humans, vehicles and wildlife [17]. Subtraction of a background model, followed by thresholding is one of the simplest and most

common types of change detection [14, 21]. The main advantage of reference model methods is their simplicity. There is no need to model the geometry or photometry of a scenario. This makes the system operation faster and allows it to operate in many different environments without having to undergo extensive calibration routines [21].

For this system, three low-complexity reference model methods are implemented. Since the system deals with static cameras, background subtraction is a particularly attractive method due to its applicability to our scenario. The following code describes the implementation of real-time background subtraction.

```
trigger(handles.vid);
yback=getdata(handles.vid,1);
yback=rgb2gray(yback);

while(get(handles.pbStop,'UserData')==0)
      trigger(handles.vid);
      y=uint8(getdata(handles.vid,1));
      yg=rgb2gray(y);
      y=im2bw(abs(yg-yback),get(handles.slider1,'Value'));
      axes(handles.cam);
      imagesc(y);
end
```

First, the camera is triggered and an image of the static background not containing any moving objects is obtained. This image is converted to grayscale and stored in the variable `yback`. A while loop continuously triggers the camera storing the current frame in grayscale to the variable `yg`. Afterwards, the absolute difference between the background and the grayscale image is calculated and a binary image is obtained from this difference using the `im2bw()` function. Finally, the result is displayed and the while loop continuously repeats the process.

Another method that was explored was temporal differencing. This method is very similar to background subtraction, but a previous frame is used as the reference to subtract. The following code describes how this is implemented in MATLAB for a single camera.

```
while(get(handles.pbStop,'UserData')==0)
      trigger(handles.vid);
      y=uint8(getdata(handles.vid,1));
      yg=rgb2gray(y);
      diff=im2bw(abs(yg-ypast),get(handles.slider1,'Value'));
      ypast=yg;
      axes(handles.cam);
      imagesc(diff);
end
```

Every time the while loop executes, it triggers the camera and acquires the new frame, storing it in `y`. Then, the image is converted from RGB to grayscale and stored in `yg`. The absolute difference between the current frame and the previous frame is calculated and the result is converted to a binary image using `im2bw()`. The result is displayed and the current frame is stored in `ypast` so it can be used for a subsequent calculation of the temporal difference. Finally, another approach to motion detection was attempted using a background average as our reference model. This method uses a weighted running average to adapt the background reference to changes in the scene caused by lighting or changing weather conditions. The following code describes how this is done.

```
while(get(handles.pbStop,'UserData')==0)
    trigger(handles.vid);
    y=uint8(getdata(handles.vid,1));
    yg=rgb2gray(y);
    alpha=.25;
    ymean=alpha*yg+(1-alpha)*ymean;
    diff=im2bw(abs(yg-ymean),get(handles.slider1,'Value'));
    axes(handles.cam);
    imagesc(diff);
end
```

As in the previous reference models, a while loop continuously captures and converts the image to grayscale. A constant `alpha` is set to .25. This means that every new frame accounts for .25 of the updated average, while the previous average accounts for .75 of the updated average. The running average is calculated, using the previous average `ymean`, the current image `yg` and the

`alpha` constant. This running average background is differenced with the incoming image and the results are displayed.

3.6.5 Automatic Global Thresholding

The attentive reader will have noticed that the `im2bw()` function used in all three reference model methods has an added parameter that has not been discussed. This parameter is the threshold or level at which the image difference will become either a 1 or a 0 in the final binary image. Determining the appropriate threshold for any of these methods is a challenge, since the threshold distinguishes significant changes. The ideal threshold value will be high enough to avoid detecting noise, yet low enough to detect targets [21].

A human observer can determine the adequate threshold value to optimally segment an object. However, it is not practical to have an observer present adjusting the threshold level constantly. The threshold value must be set automatically by the surveillance system. In this system, a global threshold is set for every incoming frame using the `graythresh()` function.

```
diff=im2bw(abs(yg-ymean), graythresh(abs(yg-yback)));
```

Critics of the thresholding technique argue that thresholding is sensitive to noise and illumination variations, and does not take into account local consistency properties of the change image [6]. This is true for a single camera. However, from the results obtained in this thesis it was concluded that averaging the result from multiple cameras helps reduce the effect of illumination and noise on the final segmentation.

3.6.6 Virtual Ground Plane

From the calibration stage, a homography matrix *H* was found for each camera. This homography matrix maps the points from each camera view to a common top camera view [19]. By applying this homography to every incoming frame, a virtual ground plane view of the scene can be created. The ground plane was selected as the plane to which project all the camera views since most outdoor surveillance scenes consist of targets moving about on the ground [19].

This virtual ground plane serves two purposes. First, having a ground plane view allows for better scene understanding. For example, tracking an object can be easier when done from an orthogonal view to the ground plane (i.e. a top view of the ground plane) than from any arbitrary viewpoint. Second, having all the cameras projected to a common plane allows the system to combine the data from each of their views and improve on single camera data. This then, sets the stage for global scene analysis [18]. The incoming frame from each camera is transformed to the ground plane view by applying the homography matrix for that specific camera. The homography matrix is applied by using the 2-D spatial transformation `imtransform()` function on the grayscale image `yg` using the following MATLAB code

```
[res1 xback1 yback1]= imtransform(yg, tform1,'Size',[240 320],'Xdata',[1
                        320],'Ydata',[1 240]);
```

Additional parameters such as '`Size`', '`Xdata`', and '`Ydata`' are added to keep the image size constant. The transformed image is returned in `res1`, which corresponds to the virtual ground plane resulting from transforming the view from camera 1 to the top camera view.

3.6.7 Multi-Camera Image Fusion

The information from each camera view exists on its own plane and cannot be combined until it is put into a common plane with another camera view with overlapping fields of view. In order to combine all four camera views, they are projected onto the ground plane using a homographic transformation, as described above. The information from each camera is then combined into a multiple camera view, by averaging the transformed contributions from each individual camera as seen in Figure 3.7. This can be expressed as follows

$$M = \frac{1}{n}\sum_{i=1}^{n} H_n(I(x,y))$$

where *n* corresponds to the number of camera nodes, *I(x,y)* is the current frame and H_n is the homography matrix that maps *I(x,y)* to the top view plane.

3.6.8 Additional Processing

After the multiple views have been combined to obtain an improved segmentation of the camouflaged target, additional processing is performed on the segmented image to assist in the object detection. Such additional processing includes calculating the centroid, creating a bounding box around the object, classifying the target as a vehicle or person and finally, tracking its trajectory using a Kalman filter.

3.6.8.1 Computaiton of Centroids Using Binary Images

The object centroid is an image feature that can prove to be very useful, as it represents the center of the target. Each centroid is calculated for each camera's individual segmentation. The centroid is also calculated for the result of the multiple camera segmentation. The centroid is

calculated every time the while loop repeats. So for every frame, there is a corresponding centroid. The code below describes how the centroid is calculated for a single camera.

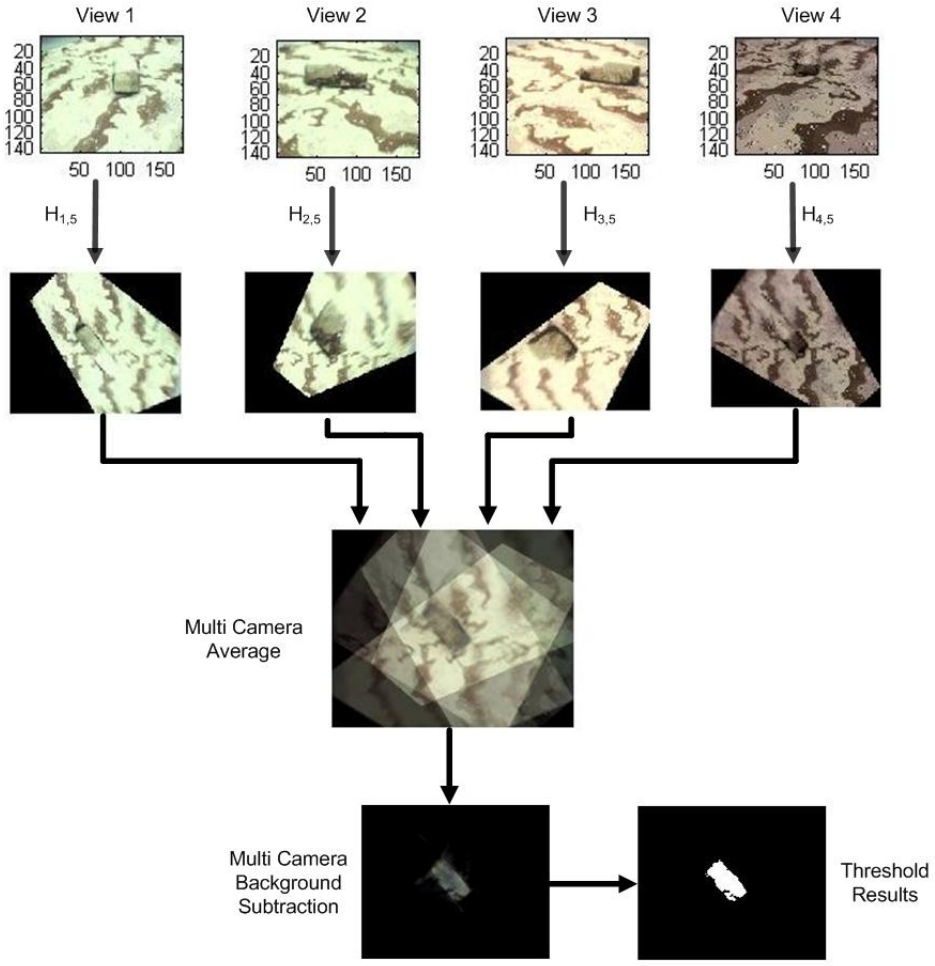


Fig. 3.7: Multiple Camera Data Averaging. The current frame from each camera view is transformed via an homography to a common plane. The views are averaged to combine their data and background subtraction is performed. After thresholding, a good segmentation of the camouflaged object is obtained.

```
t=t+1;

[rows,cols] = size(yg);

X = ones(rows,1)*[1:cols]; % Matrix with each pixel set to its x coordinate
Y = [1:rows]'*ones(1,cols); %   "     "     "    "    "  "   "  y    "

area1 = sum(sum(yg));
cx(t) = sum(sum(double(yg).*X))/area1;
cy(t) = sum(sum(double(yg).*Y))/area1;
```

First, the two matrices X and Y are created that will be necessary to compute the centroids. The size of X and Y must match the image size. X is a matrix with each pixel equal to its x coordinate. Y is a matrix with each pixel set to its y coordinate. They correspond to part of the first order moments.

The total area of the binary image is calculated by using the sum function twice. This is equivalent to its zero-th order moment. Afterwards, the centroid (x,y) values are calculated taking advantage of the previously calculated X and Y matrices. The output of the centroid calculation is shown in Figure 3.8. The vectors `cx` and `cy` contain the x and y locations of the centroid respectively for t frames. Keeping track of the centroids from each camera view will be useful when analyzing the results and using the Kalman filter.

3.6.8.2 Computation of Target Bounding Boxes

In many motion detection applications, a bounding box is drawn around the detected object. This is done to aid the human observer in detecting areas where targets are located. It can also be used to create a region of interest on which additional processing can be applied without considering the whole image. The required information to draw a bounding rectangle around the target is the top left and bottom right pixels of the result of the segmentation. The code below shows how the x and y coordinates for the top left and bottom right points of the segmented target are located.

First, the columns of the binary image are added using the sum function. Then, using the find function, the indices of the values that are greater than zero are returned. The index with the minimum value will correspond to the x coordinate of the top left point. The same is done for the y coordinate, except the image is transposed so that the rows are summed instead of the columns.

```
xleft1=min(find(sum(ygt)>0));
yleft1=min(find(sum(ygt')>0));
xright1=max(find(sum(ygt)>0));
yright1=max(find(sum(ygt')>0));
```

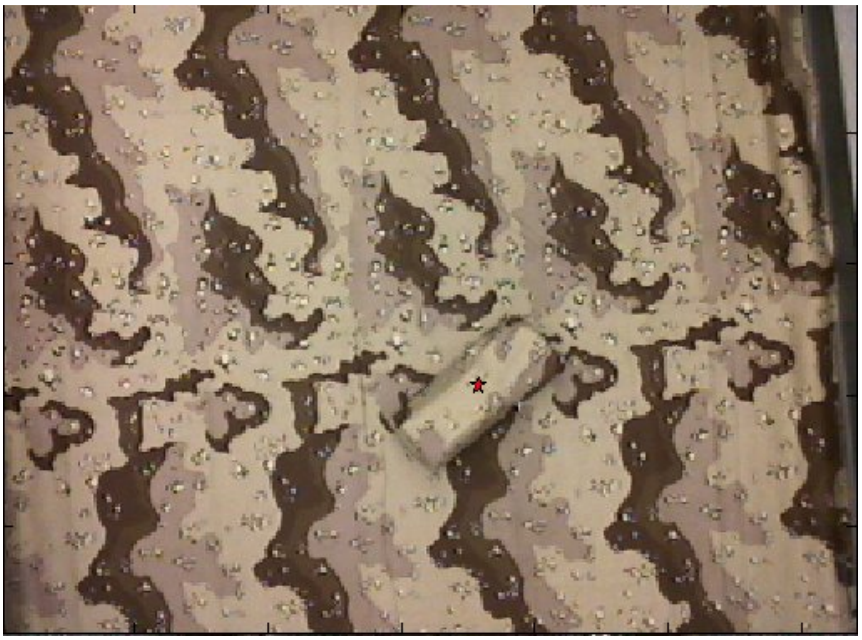


Fig. 3.8: Example of Real-Time Centroid Computation. The centroid of the target is signaled by a red star.

The process is repeated to locate the bottom right point, but the maximum indices are desired in this case. After these two points are located, the `rectangle()`function is used to plot a rectangle in a MATLAB figure using the following code.

```
if isempty(xleft1)==0
rectangle('Position',[xleft1,yleft1,xright1-xleft1+1,yright1-
yleft1+1],'LineWidth',2,'LineStyle','-','EdgeColor','r');
end
```

The '`Position`' parameter requires the top left rectangle coordinates and its height and width. An additional condition is added, so that the rectangle is only plotted if coordinates exist. An example of this operation is shown in Figure 3.9.

Fig. 3.9: Real-Time Rectangle Output

3.6.8.3 Target Classification

After the target has been segmented it can also be classified. Classification refers to identifying the target and categorizing it. For example, an automated surveillance system could trigger alerts when a vehicle other than white appears. In this system, a spatial classification algorithm is used to classify the target into three categories: person, vehicle or unidentified. The procedure is fairly simple and uses the width to height ratio of the target to determine if its shape is elongated horizontally or vertically. If the shape is elongated horizontally, it is assumed to be a car. If the

segmented area is vertically elongated, it is assumed to be a person. If neither condition applies, the target is categorized as unidentified. The code below shows how the classification is done. If the width to height ratio is greater than one, the object is elongated horizontally and the `v1car` flag is set to one. If the ratio is less than one, the `v1person` flag is set to one, since it is assumed a person is detected. If neither of the conditions apply, both flags remain at zero.

```
if (xright1-xleft1+1)/(yright1-yleft1+1)>1 && (xright1-xleft1+1)<315

                              v1car=1;
                              v1person=0;

elseif (xright1-xleft1+1)/(yright1-yleft1+1)<1 && (xright1-xleft1+1)<315

                              v1car=0;
                              v1person=1;
else

                              v1car=0;
                              v1person=0;
end
```

The classification algorithm is simple and executed by each camera individually. However, the accuracy of the results is greatly improved when the classification results from each camera are taken into account. This is done by a voting process. Each camera votes, according to its own classification whether the target is a person, a vehicle or an unidentified target. The votes from each camera are tallied for each the vehicle and the person classification. Finally, the total votes are compared to make a simple majority decision.

3.6.8.4 Kalman Filter

In some surveillance applications, it is desired to track the target's trajectory. This is done to predict the direction the target is moving in. This information can be used to ready and activate cameras that are expected to capture the moving target. Each individual camera keeps track of

the object's centroid as mentioned before. These centroids represent data points useful for tracking. However, in practice, these data points are corrupted by noise due to camera artifacts, illumination changes, or other factors. This is where the Kalman Filter comes into play. The Kalman filter can take the centroids from all the cameras and estimate the current and future direction of the target, modeling the target's trajectory. Averaging the centroids can also model the trajectory, but averaging is more susceptible to deviations due to the presence of noise in the images. The Kalman filter provides more robustness in the presence of noise than simply averaging. In order to implement the Kalman Filter, this system uses the Kalman Filter Toolbox [32]. The function was modified to take in the centroid values, instead of generating a random signal with white noise. The y values were subtracted 240 due to differences in the axis when MATLAB plots an image. The (0,0) coordinate in a MATLAB image is at the bottom left corner of the image, while the (0,0) coordinate in a MATLAB matrix is the top left corner. This processing step is described in the following code.

```
% Make a point move in the 2D plane
% State = (x y xdot ydot). We only observe (x y).
% This code was used to generate Figure 15.9 of "Artificial Intelligence: a
Modern Approach",
% Russell and Norvig, 2nd edition, Prentice Hall, 2003.
% X(t+1) = F X(t) + noise(Q)
% Y(t) = H X(t) + noise(R)

ss = 4; % state size
os = 2; % observation size
F = [1 0 1 0; 0 1 0 1; 0 0 1 0; 0 0 0 1];
H = [1 0 0 0; 0 1 0 0];
Q = .1*eye(ss);
R = 1*eye(os);
initx = [160 120 50 50]'; %(position 10,10 and velocity 1,0)
initV = 50*eye(ss);

x=cxmv;
y=240-cymv;

[xfilt, Vfilt, VVfilt, loglik] = kalman_filter(x, F, H, Q, R, initx, initV);
[yfilt, Vfilt, VVfilt, loglik] = kalman_filter(y, F, H, Q, R, initx, initV);
```

The center of the image 320 x 240 image was chosen as the initial starting point for the Kalman filter. The coordinates of this point is (160, 120) and are set in the `initx` variable.

3.6.9 Graphic User Interface (GUI)

In the early stages of the research, video from each camera was displayed in its own MATLAB figure. This was undesirable, since it created a cluttered workspace and generating all the video slowed the performance of the surveillance system. Furthermore, having video feedback from all the cameras causes a loss of attention span, making it difficult to keep track of the activity [17]. Another problem with the system was that it required different pieces of code to be run in specific sequences in order to calibrate, initialize cameras and begin processing. This caused confusion from a new user perspective. As the surveillance system continued to take form, the necessity to develop a graphic user interface (GUI) to help users interact with the system in a visual form became obvious. Besides making the system more user-friendly, the GUI also ran the code faster than previous versions, since only video from selected cameras was shown.

The GUI, shown in Figure 3.10, provides different push buttons for calibration, starting and stopping the system. It also has list boxes that allow the selection of the camera to be displayed, or the combined multi-camera view. Another list box allows the selection of three different background subtraction methods to be used and the rectangle, classification or centroid features. The rectangle feature displays a box around moving objects in the scene, while the centroid feature calculates the target's centroid. The GUI also displays the results of the classification as a string. The GUI for this thesis was created using MATLAB's graphic user interface tool (guide). There are two steps to creating the GUI. First, a .fig file must be created which contains the visual aspects of the GUI. Buttons, text, scroll bars and images can be added to the GUI layout.

Fig. 3.10: Graphic User Interface

Chapter 4: Camouflaged Target Segmentation and Tracking

Throughout the research, several scenarios and experiments were considered to compare reference model methods and test the segmentation and tracking performance of the proposed scheme against another camouflage detection method.

4.1 Evaluation and Comparison of Reference Models

In order to determine the most accurate reference model for our application, the three reference models introduced in Chapter 2 were evaluated using a single camera. Video sequences were captured and segmented using the three different reference models considered for this research. The results from each segmentation method were visually compared.

The background subtraction model proved to have more advantages than the other reference models, even though it is the simplest of the three. One of the major advantages is that even without target movement, differences were still visible making segmentation possible. In the background averaging method, if the target sits still for a couple of frames, it would be averaged with the background and disappear from the final segmentation. In the case of frame to frame differencing, if the target stopped moving it would also disappear from the segmentation, after only a couple of frames. Another problem with the frame to frame differencing is that only the bounds of the object are detected, so a complete segmentation of the object is never really achieved for uniformly colored objects.

In order to have more evidence supporting the used of background subtraction, other than the subjective comparison, the computation times for each reference model method were also computed for images of the same size. The computation times for each method were recorded and plotted in Figure 4.1. The computation times varied slightly each time the experiment was

done, so three different trials were run for each reference model. The results clearly indicate that background averaging takes the longest to compute, while frame to frame differencing and background subtraction have very close computation times. On closer inspection, it can be see that the frame to frame differencing is slightly slower. The results of this experiment and previous observation of segmentation results made background subtraction the selected method of segmentation for this application.

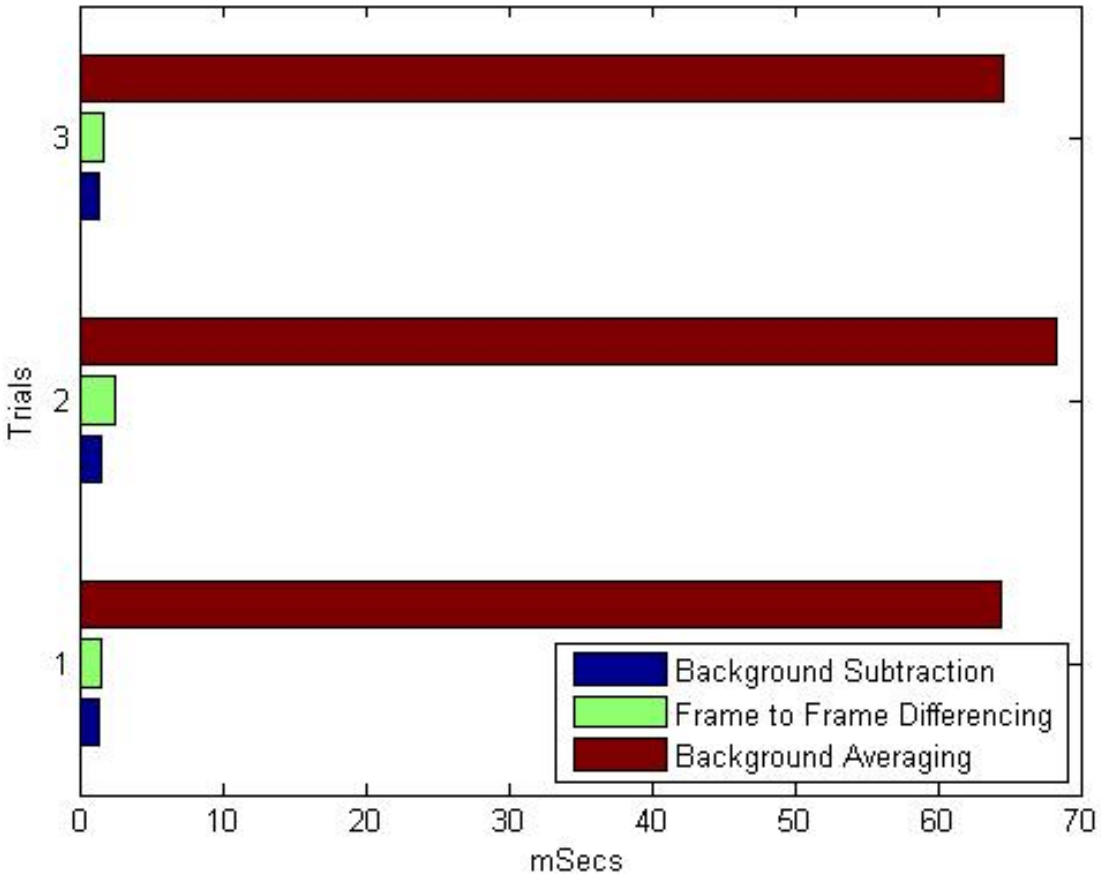


Fig. 4.1: Reference Models Computation Time. Background Subtraction is the fastest out of the three reference models considered.

4.2 Comparison of Background Subtraction under Camouflaged and Non-Camouflaged Conditions

In this section we compare the segmentation results of a camouflaged target to those of a non-camouflaged target on both a camouflaged and non-camouflaged background. Images from a single camera were acquired for each of four cases: a non-camouflaged vehicle on a non-camouflaged background, a camouflaged vehicle on a non-camouflaged background, a non-camouflaged vehicle on a camouflaged background and a camouflaged vehicle on a camouflaged background. Background subtraction was applied on each case and the results were compared.

First, the two cases with the non-camouflage background were tested. Figure 4.2 shows the image used as the static background in the reference model. Figure 4.3 depicts the camouflaged vehicle in the scene, while Figure 4.4 shows the result of the background subtraction. A fairly decent segmentation of the camouflaged vehicle results from this test. An even better segmentation results in the case of a non camouflaged object. Figure 4.5 shows a non-camouflaged vehicle on a non-camouflaged background. Again, background subtraction is performed resulting in Figure 4.6. The segmentation covers most of the vehicle, with the exception with a small fraction of the hood, due to lighting effects. Studying these images demonstrates that using a non-camouflaged background does not present a problem with segmentation.

Fig. 4.2: Background Reference Image for Non-Camouflaged Condition

Fig. 4.3: Camouflaged Vehicle on Non-Camouflaged Background

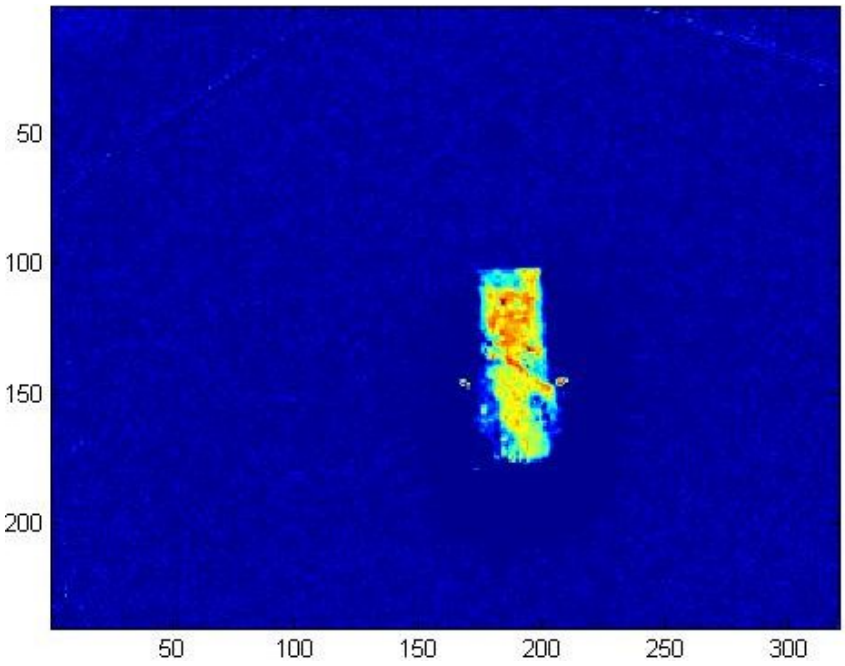


Fig. 4.4: Background Subtraction Result of a Camouflaged vehicle on Non-Camouflaged Background

Fig. 4.5: Non-Camouflaged Vehicle on Non-Camouflaged Background

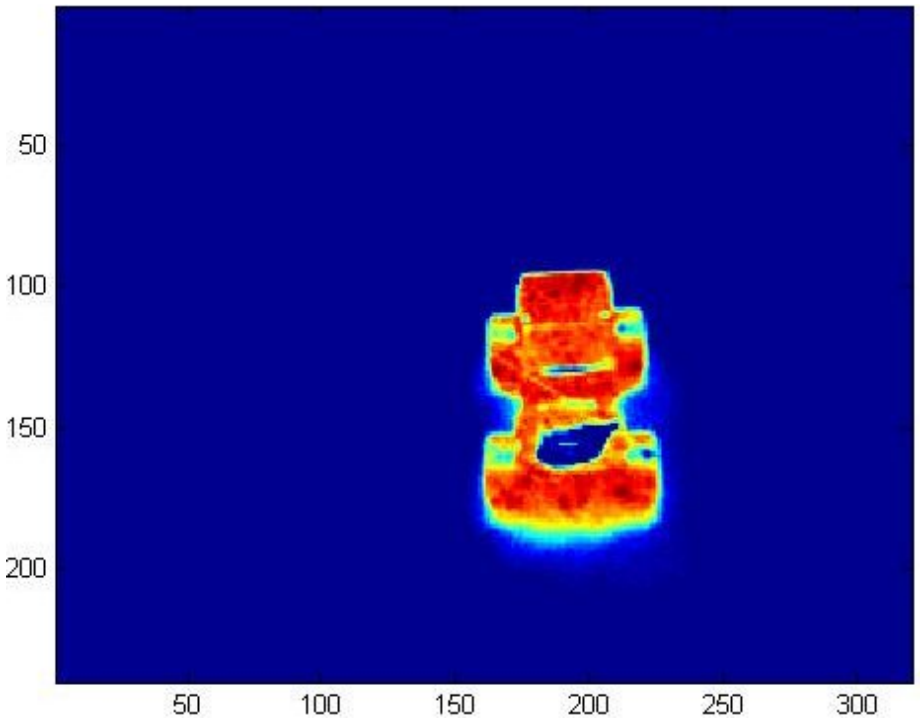


Fig. 4.6: Background Subtraction Result of a Non-Camouflaged vehicle on Non-Camouflaged Background

In the case of a camouflaged vehicle on a camouflaged background the situation differs. Figure 4.7 is the background used for the camouflaged background cases. A camouflaged vehicle was placed on the camouflaged background as seen in Figure 4.8. Background subtraction was then performed resulting in the image in Figure 4.9. This segmentation suffers from fragmentation, confirming previous knowledge of this problem. For the case of a non-camouflaged object on a camouflaged background, background subtraction resulted in a more defined segmentation of the vehicle.

Fig. 4.7: Background Reference Image for Camouflaged Condition

Fig. 4.8: Camouflaged Vehicle on Camouflaged Background

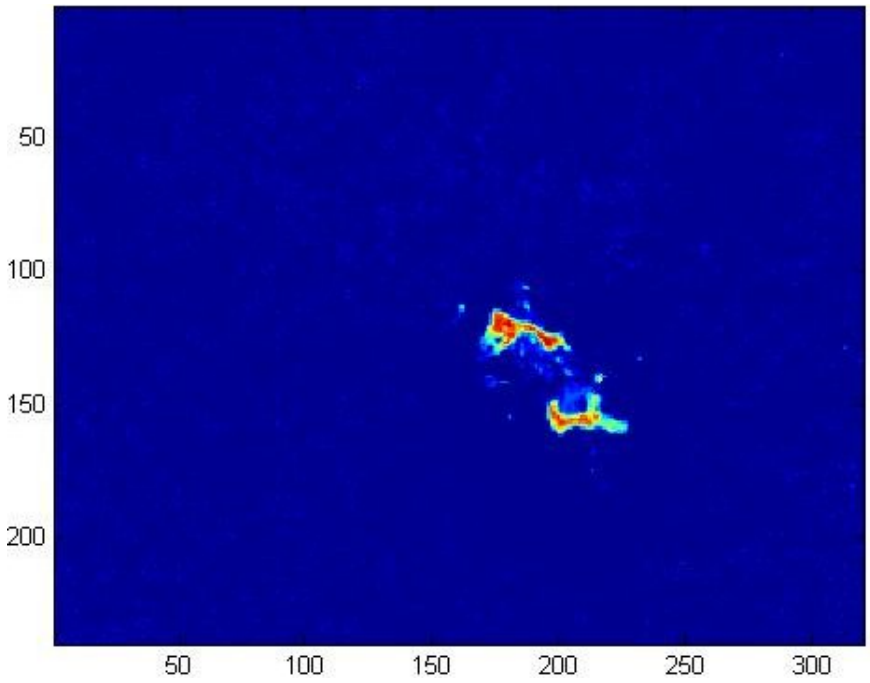


Fig. 4.9: Background Subtraction Result of a Camouflaged vehicle on Camouflaged Background

Fig. 4.10: Non-Camouflaged Vehicle on Camouflaged Background

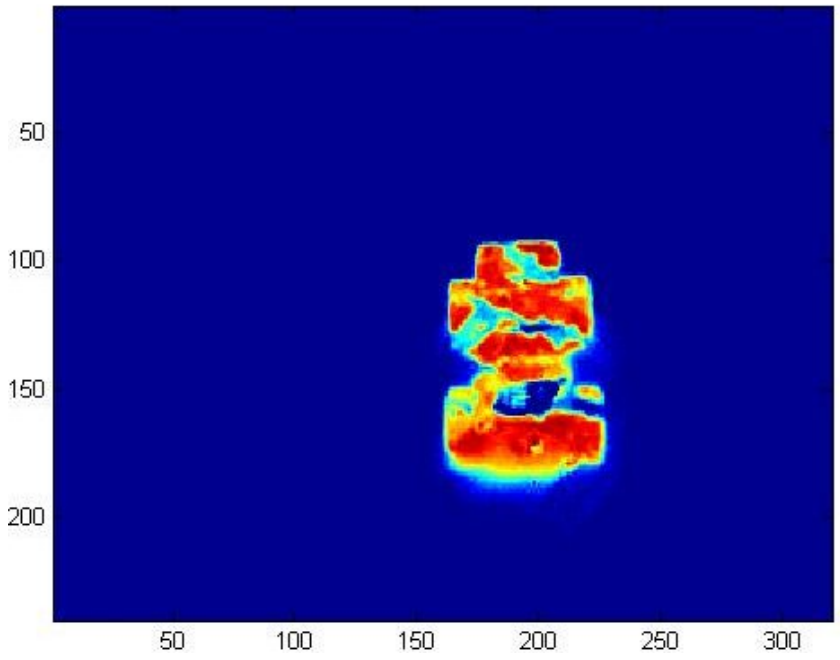


Fig. 4.11: Background Subtraction Result of a Non-Camouflaged vehicle on Camouflaged Background

4.3 Segmentation of Camouflaged Targets Using a Multi-Camera Network

In this section we present one of the main results of this thesis. In this scenario, video from each camera is segmented individually and transformed through a homography to fit the top view. The camera views are then averaged. Through this experiment the main goal of improving the segmentation of a camouflaged object by combining data from different sensors was achieved.

Figure 4.12 presents a typical set of frames obtained from the camera network for a camouflaged target. The images in Figure 4.13 show the segmentation obtained using the background subtraction algorithm on the video from the individual cameras. Judging by the images, it is unclear whether there is a single object or multiple objects. Furthermore, the shape of the object varies greatly and is difficult to determine. The segmentation using the combined data shows a

more defined shape following the form of an ellipse or rectangle, which could be identified by an operator or classification algorithm as a vehicle.

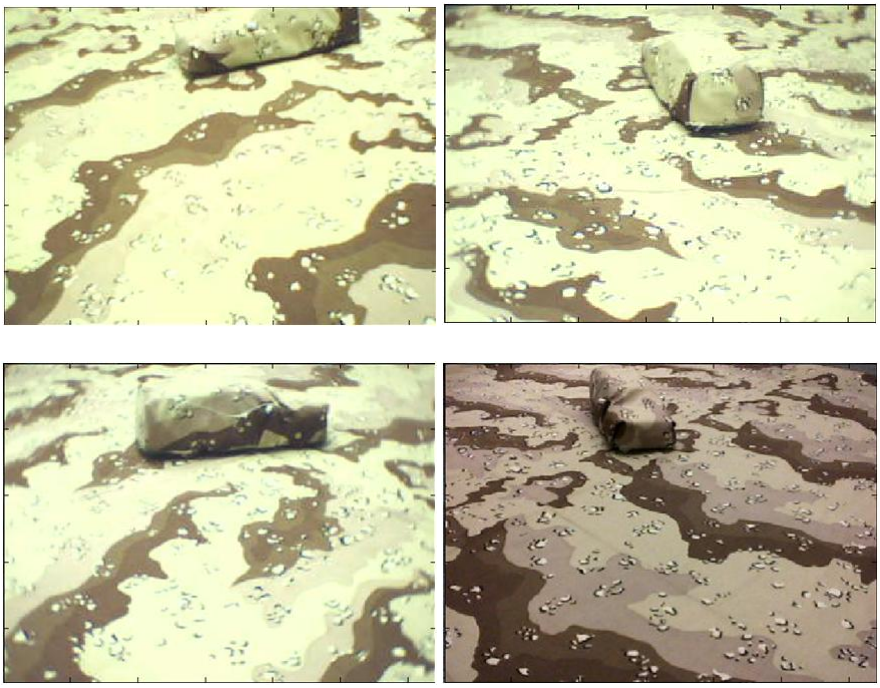


Fig. 4.12: Acquired Images from Camera1 (top left), Camera 2 (top right), Camera 3 (bottom left) and Camera 4 (bottom right)

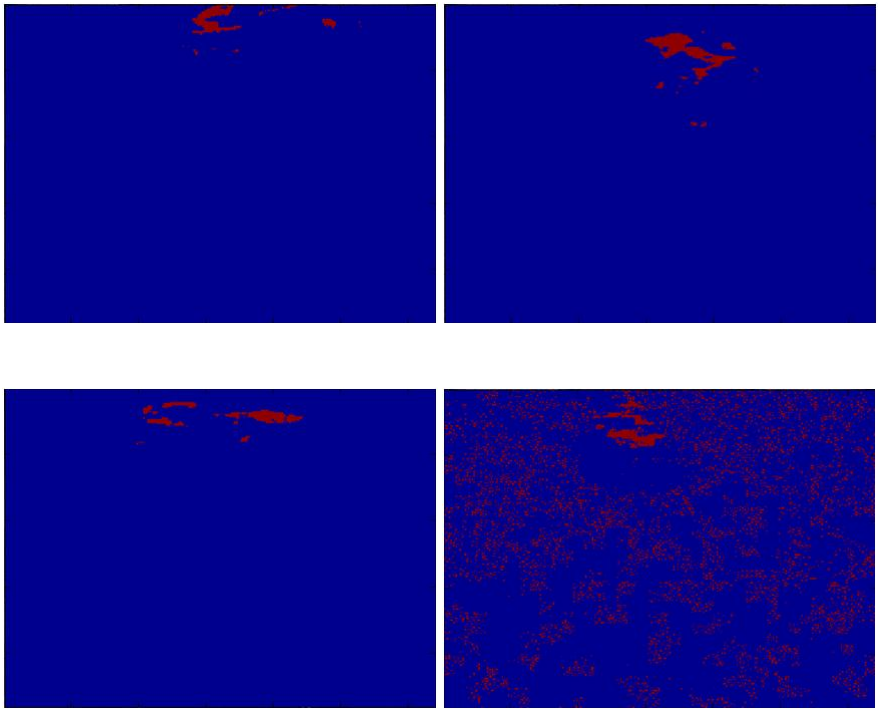


Fig. 4.13: Segmentation Results for Camera1 (top left), Camera 2 (top right), Camera 3 (bottom left) and Camera 4 (bottom right). Notice the presence of noise in Camera 4.

Another result from this experiment is that camera noise can be reduced in the final segmentation by combining the data from different cameras. As shown in the segmentation from camera 4 in Figure 4.13, camera noise can be present in the image. However, since the noise is present in only one of the views it can be reduced or even eliminated when the data from different sources is combined.

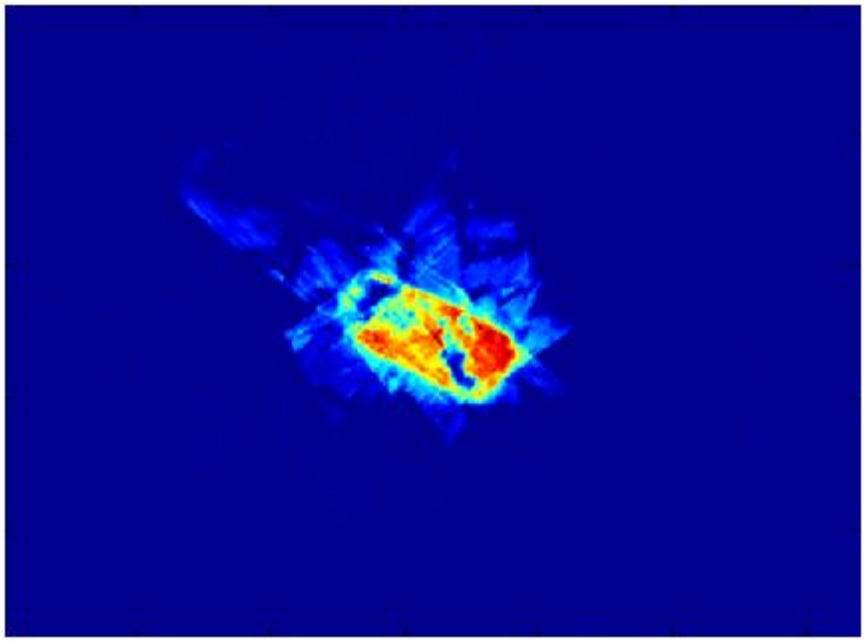


Fig. 4.14: Multi-Camera Segmentation obtained by Combining the Transformed Segmentation from Each Camera

In Figure 4.14, we present the camouflaged target segmentation obtained through the combination of the projected frames from multiple camera sensors. Each frame from Figure 4.12 was projected to the ground plane view using the homographic transformation described in Section 2.5. The ground plane was derived using the calibration procedures described in Chapter 3. The projected frames were combined using their arithmetic average. As seen in Figure 4.14, we achieve a better representation of the object than the segmentation from any of the individual cameras. This was observed in general for all sequences, where the target overlapped the field of view of all the cameras. The proposed real-time system achieved camouflage segmentation at a rate of 10 to 15 frames per second.

This experiment also proved that projective transformations are an adequate method of mapping the information from individual cameras onto a single plane. The ground plane proved to be a

good choice for this common plane, as a bird's eye view can provide information regarding the orientation and direction of the target. The result after combining the projective transformations from each individual camera is shown in Figure 4.16. This image was compared to the ground truth video data acquired from the fifth top view camera, shown in Figure 4.15. Although there are areas in the combined view that the fields of view of the individual cameras do not cover, both images are strikingly similar. This indicates that the projective transformations were successful and a virtual ground plane view was attained. As previously mentioned there is, however, one drawback to the projective transformation mappings: it is assumes that the object is planar. This assumption does not apply when the camera and the object are at the same elevation, resulting in distortions in the segmentation. Maintaining the cameras higher than the target and pointed downward towards the monitored area help decrease these distortions.

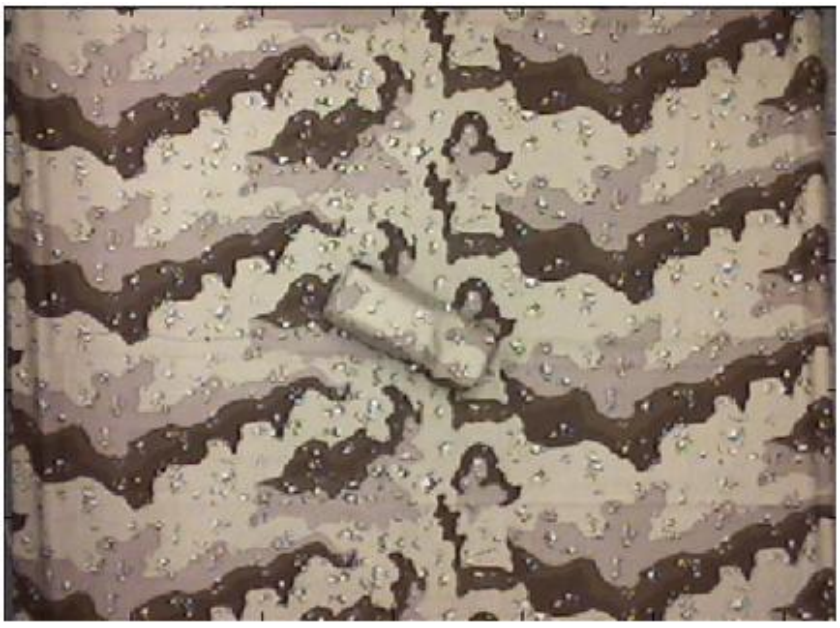


Fig. 4.15: Ground Truth Image Obtained from the Top View Camera Used For Comparison

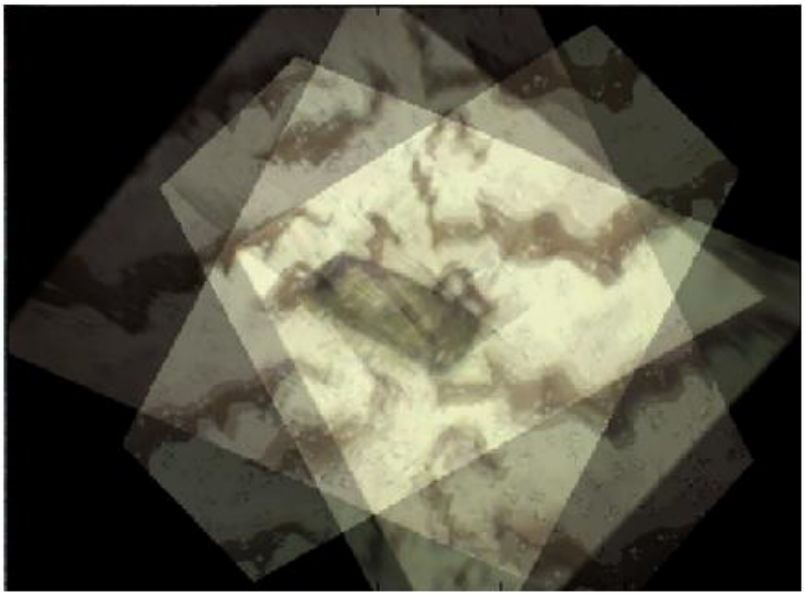


Fig. 4.16: Multi-Camera Average obtained by Combining the Transformed Views from Each Camera

4.4 Homography Using a Camera Node as a Reference Plane

Using the homography matrices from the calibration step, the views from each individual camera was transformed to a top view camera perspective. The results of this segmentation were very satisfying, as it proved the hypothesis of the research. By combining the information from several cameras, an improved segmentation was achieved for a camouflaged target, as mentioned in the previous experiment. The final result was better than any of the individual camera segmentations. The top view camera data was used as the ground truth to which the results were compared. These positive results encouraged experiments on other reference planes.

Experiments were also done to test the possibility of using a ground camera as the reference view for the homographies. Concerns were raised about the unfeasibility of having a top view camera for calibration in a real life scenario. The homographies were calculated for a ground camera.

The results were successful in terms of combining the multiple views to a ground camera view. However, having a side view of the ground plane did not prove to be as advantageous as the top view. The top camera view is more attractive in terms of presenting the results and tracking the target.

4.5 Evaluation of Different Camera Arrays

Previous experiments had only focused on using camera sensors near each of the corners of the mock scenario. Organizing the camera sensors in different ways could lead to improved results or reduced infrastructure costs. For example, setting up all cameras on one post could be less expensive than setting up a post for each camera. Experiments were therefore done using different camera arrangements. The tests involved setting up the cameras in a vertical array and a horizontal array. Some problems were observed with these configurations. In the vertical array case, there was no horizontal change of perspective, so when segmenting the camouflaged target the segmentation from each camera was similar to the others. The real advantage of combining the camera views comes when each camera can detect fragments of the camouflaged object that are unique from the other views. In the vertical configuration, this was not the case. For the horizontal array, the problem was that the cameras had a reduced overlapping field of view. The areas of the scene where more cameras overlap will have better results. In the horizontal array, only two cameras overlapped at a time near the edges of the field of view. The horizontal and vertical camera arrays are illustrated in Figure 4.17 and Figure 4.18 respectively.

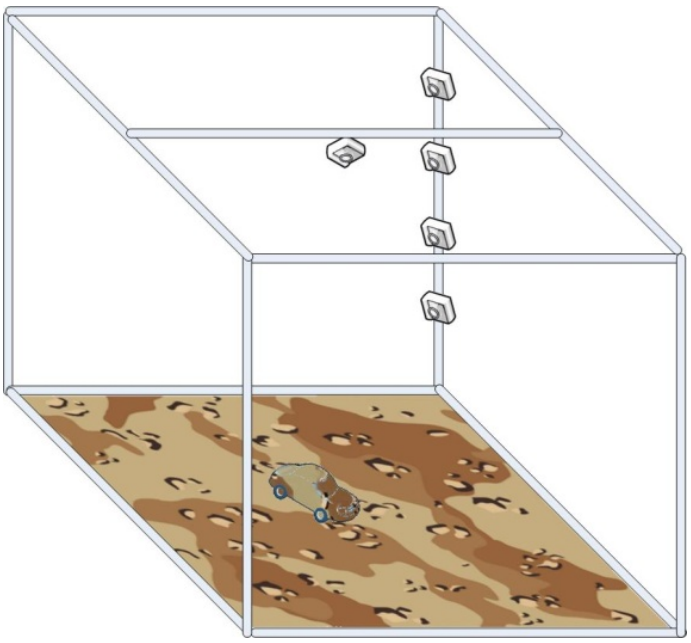


Fig. 4.17: Mock Scenario with Vertical Camera Array

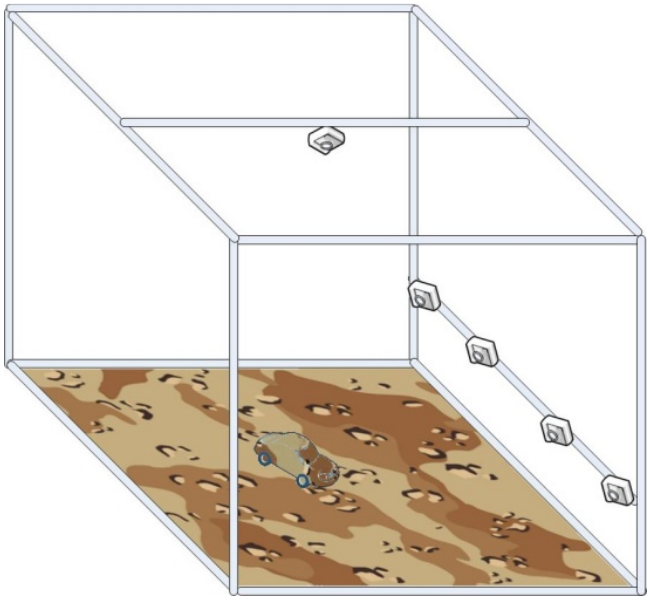


Fig. 4.18: Mock Scenario with Horizontal Camera Array

The idea of using different camera arrangements to improve background subtraction has been previously researched [38, 39]. Ivanov et al. [39] use two views to compare color differences between conjugate pixels to distinguish between background and foreground. The intensities of

the conjugate pair will change the same way if they both view background, but differently if one camera views background and the other foreground. This method is fast and simple, but results in missed and false detections generated by homogeneous foreground objects. Lim et al. [38] propose a sensor configuration that eliminates most of the false detections. The configuration consists in setting the reference camera as the lower of the two cameras in a vertical array. This configuration eliminates most of the false detections, but missed detections remain. No mention of camouflaged objects is found in the two papers. However, since intensities are being compared from conjugate pixels to detect foreground objects, it is likely that this system will also fail to detect camouflaged objects, given that pixel intensities of camouflaged objects blend in with the background.

4.6 Tracking of Camouflaged Targets

In this section, we present tracking of camouflaged targets using our multi-camera network. For this experiment, the camouflaged vehicle followed a path over the camouflaged background. We implemented a multi-camera tracking algorithm based on the Kalman filter. In this scheme, the centroids of the segmented images were calculated at each time frame for each camera. The average of the centroids was used as observations for the Kalman filter which in turn estimated the target trajectory.

This experiment was repeated three times for different trajectories. The first two trajectories followed a curve, while the third trajectory circled the observed area. Figures 4.19 through Figure 4.24 show the results of the first trajectory. The second trajectory was tracked in Figures 4.25 through 4.30. The results of the third, more challenging trajectory following a circular path, are shown in Figure 4.31 through 4.36. The top view camera was used as the ground truth to

which compare measurements. The top view camera is not immune to camera noise, but it is the only camera that covers all of the observed area and is therefore the closest measurement to the ground truth. We should emphasize that the top view camera was used as a control view, which was not included in the computation of the multi-camera trajectory estimation.

It can be observed that in some cases, individual cameras have strong deviations from the ground truth. This is likely due to the fact that the object was outside of the field of view of that particular camera. The presence of noise in the camera could also contribute to these errors. These deviations are particularly noticeable in the circular trajectory of Figures 4.31 through Figures 4.36. In all three experiments, the multi-camera average and Kalman filter appear closer to the ground truth than any of the individual cameras.

In figure 4.19 we compare the trajectory of the ground truth, the multi-camera centroid average and the Kalman filter tracking results for camera 1 only. All these trajectories are as seen from a top view. The x axis range is from 0 to 320, while the y axis is 0 to 240. These dimensions correspond to the image resolution being used by the cameras. It can be observed that camera 1 does not perform well for the first part of the trajectory. However, camera 1 improves its accuracy afterwards and follows the ground truth line more closely. For the same trajectory, camera 2 does a much better job. Camera 2 stays very close to the multi-camera centroid average. These results are shown in Figure 4.20. The tracking results for camera 3 are shown in Figure 4.21. This camera has a large error at the very beginning of the trajectory, but quickly changes to follow the path of the ground truth moderately well. Camera 4, seen in Figure 4.22, as large errors at the beginning and final of the trajectory, but also does a fairly close tracking to the ground truth in between.

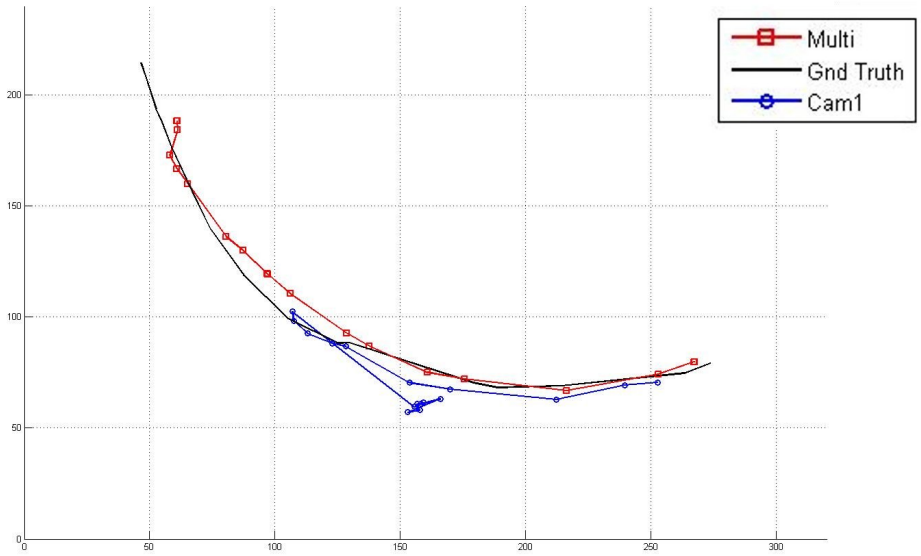


Fig. 4.19: Comparison of ground truth, multi-camera centroid average and Kalman tracking for camera 1 for the first trajectory.

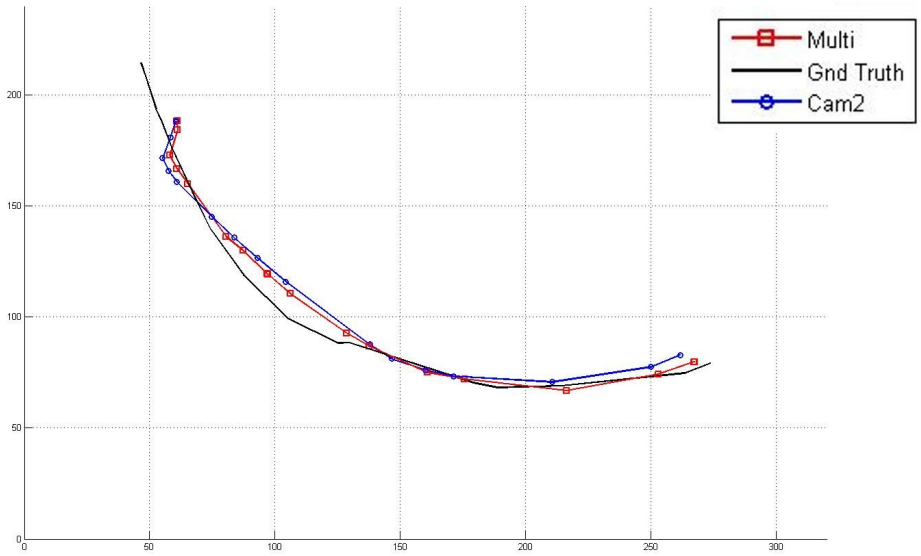


Fig. 4.20: Comparison of ground truth, multi-camera centroid average and Kalman tracking for camera 2 for the first trajectory.

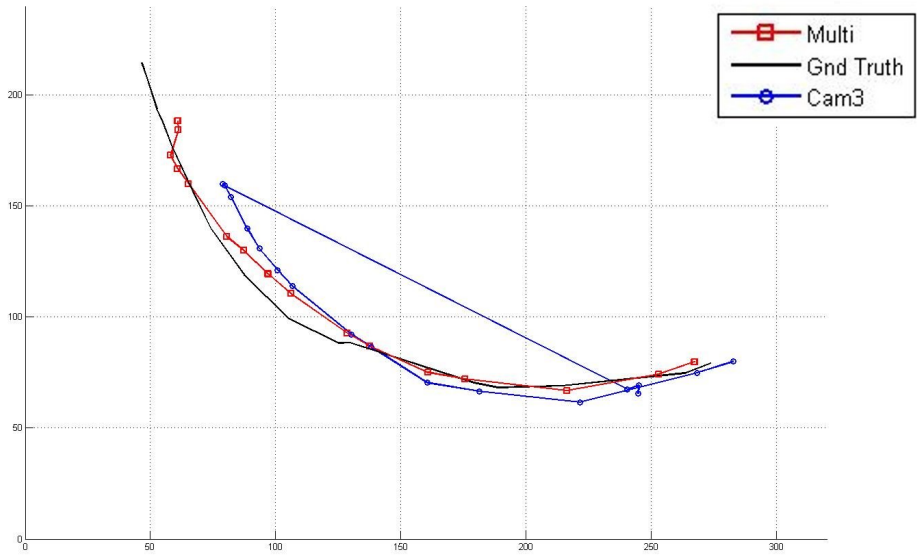


Fig. 4.21: Comparison of ground truth, multi-camera centroid average and Kalman tracking for camera 3 for the first trajectory.

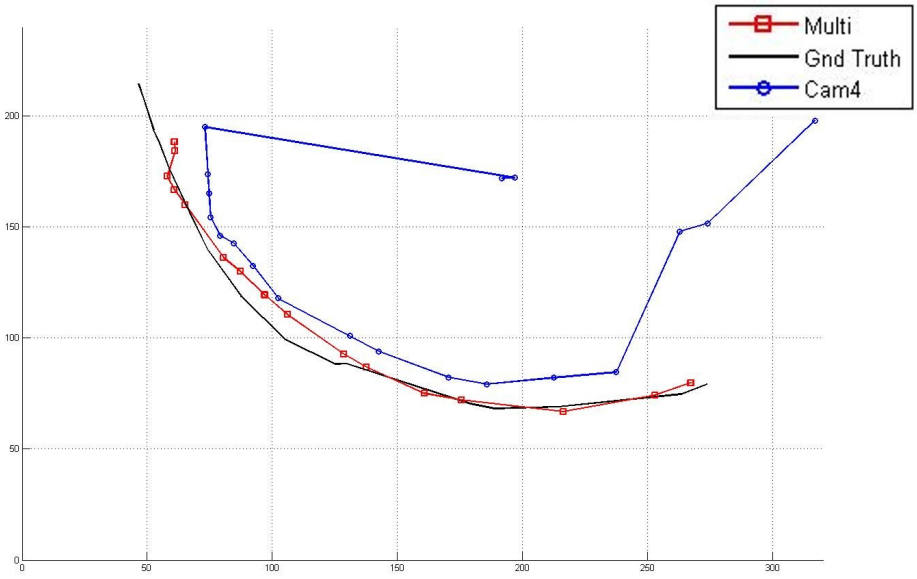


Fig. 4.22: Comparison of ground truth, multi-camera centroid average and Kalman tracking for camera 4 for the first trajectory.

Each of the previous plots show the position for the centroids for an individual camera (in blue) compared to the ground truth (in black) and the Multi-camera average (in red). A comparison between individual cameras can be done by comparing these plots, but also by looking at the following plot. Figure 4.23 shows the centroid positions for all the cameras throughout the trajectory. Each camera is represented by a color. The Multi-camera average and ground truth are also represented.

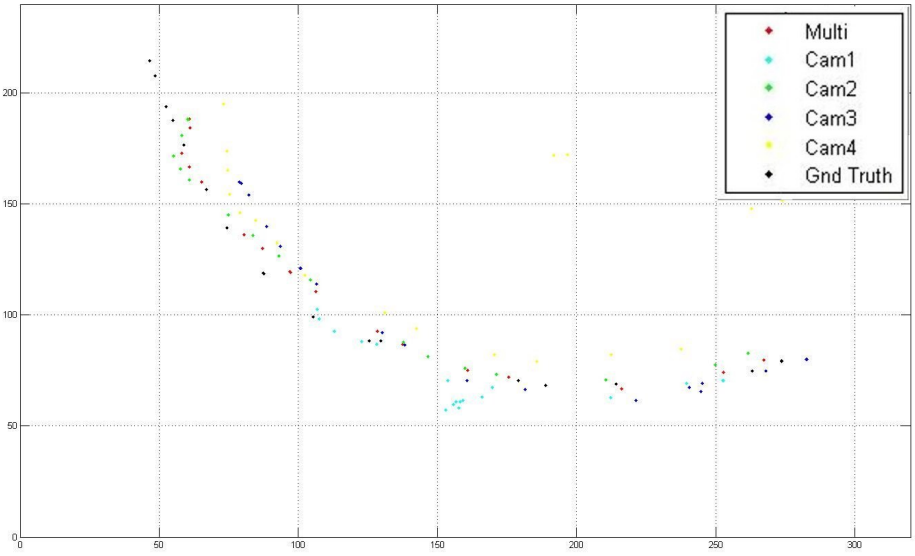


Fig. 4.23: Tracking results for all Cameras for the first trajectory

To conclude the plots for the first trajectory, the multi-camera average of the centroids from each individual camera and the result of the Kalman filter tracking are shown in Figure 4.24. The Multiple camera average and the results of the Kalman Filter are very close to the ground truth path.

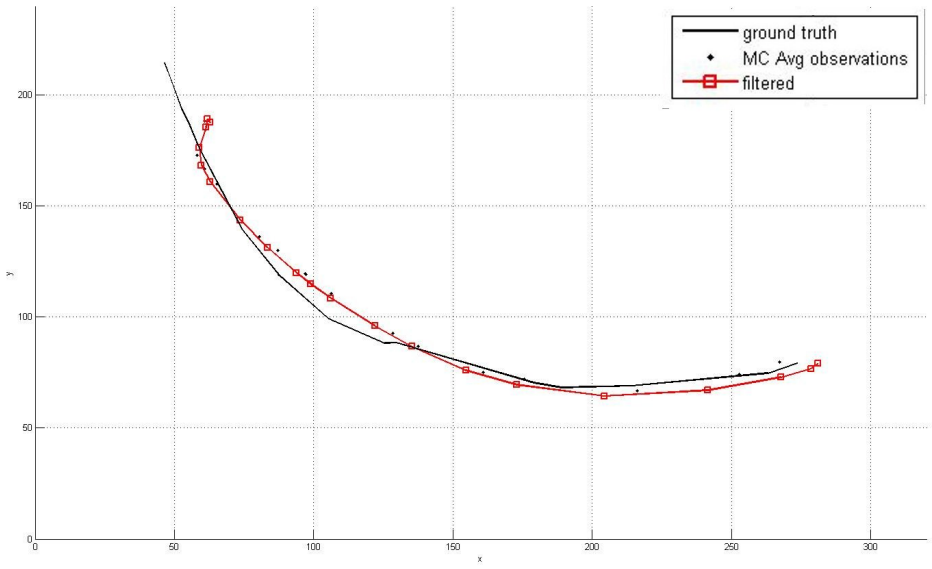


Fig. 4.24: Multi-Camera Average and Kalman Filter Comparison for the First Trajectory

A second trajectory was chosen for the camouflaged vehicle on the camouflaged background to test the tracking of the surveillance system. The results of the tracking are presented for each individual camera in the following plots in Figures 4.25 through 4.28. Individual cameras at times perform good tracking, but other times are subject to noise or errors due to their limited field of view. Camera 1 in Figure 4.25, for example, does a good job of tracking the path of the vehicle in this second trajectory as opposed to a mediocre tracking from the first trajectory, shown in Figure 4.19. Figure 4.26 shows the tracking results for camera 2. The first half of the trajectory was completely missed, while the second half is closer to the path indicated by the ground truth. For camera 3, the first and last third of the path are missed. This camera appears to have performed better during the middle section of the path. For the second trajectory, camera 4 tracks the vehicle fairly well, except for a slight deviation towards the end.

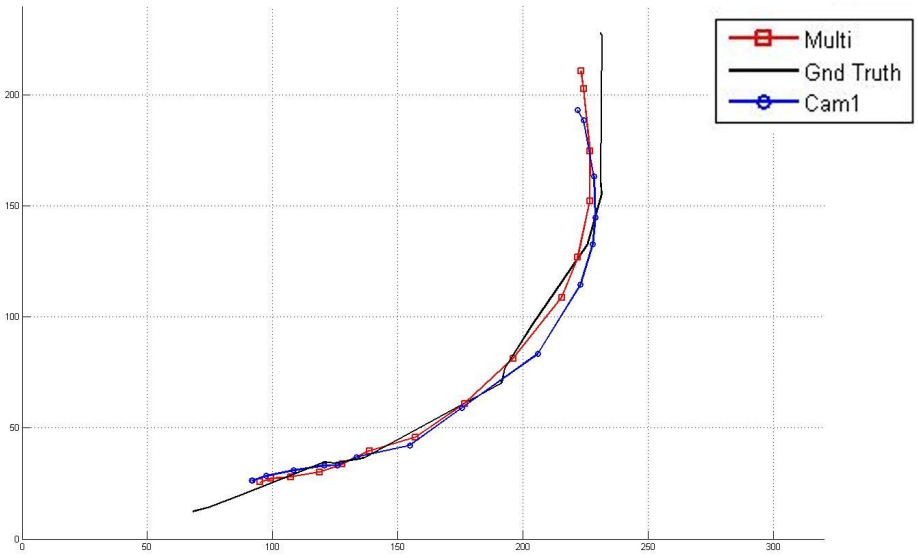


Fig. 4.25: Comparison of ground truth, multi-camera centroid average and Kalman tracking for camera 1 for the second trajectory.

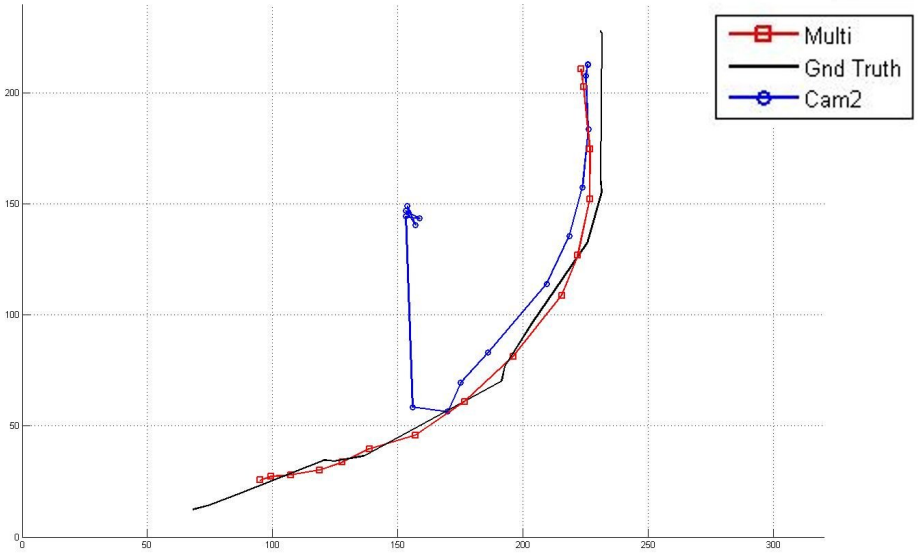


Fig. 4.26: Comparison of ground truth, multi-camera centroid average and Kalman tracking for camera 2 for the second trajectory.

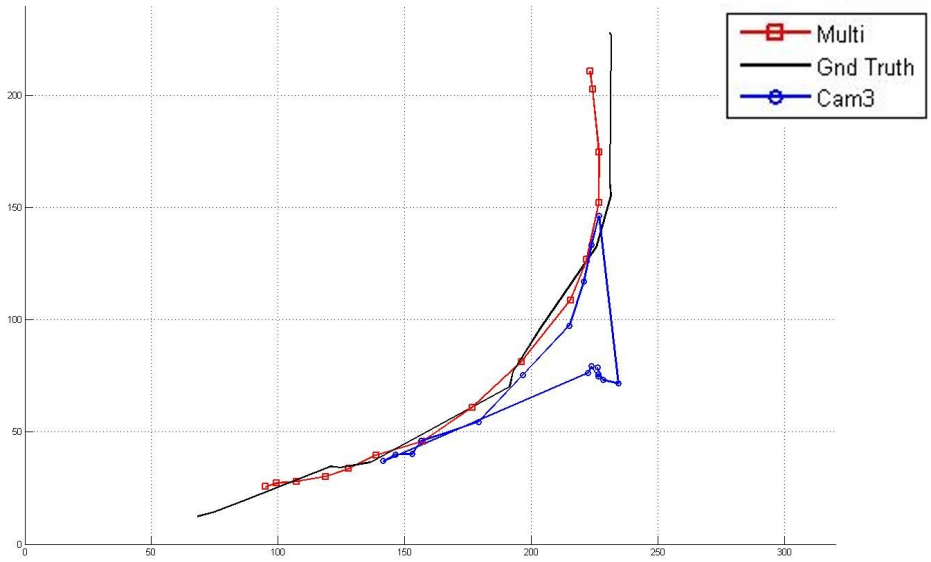


Fig. 4.27: Comparison of ground truth, multi-camera centroid average and Kalman tracking for camera 3 for the second trajectory.

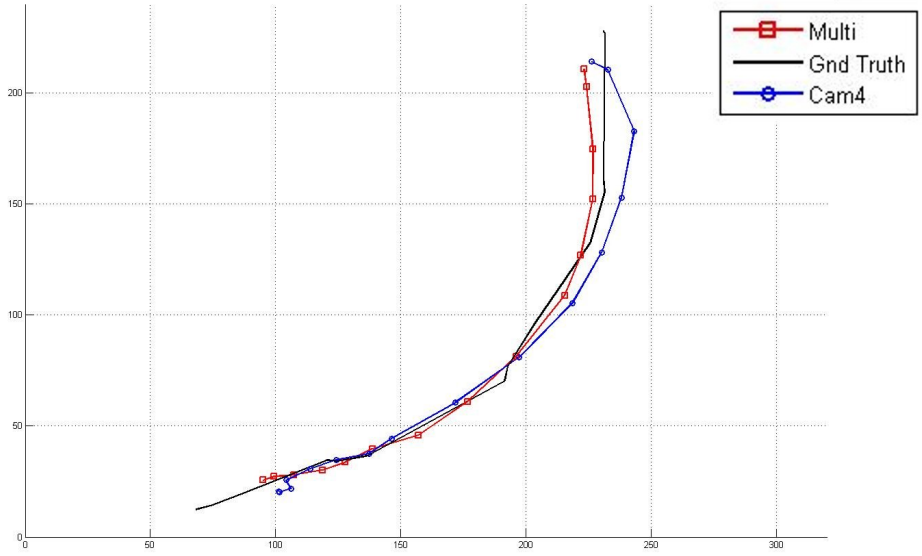


Fig. 4.28: Comparison of ground truth, multi-camera centroid average and Kalman tracking for camera 4 for the second trajectory.

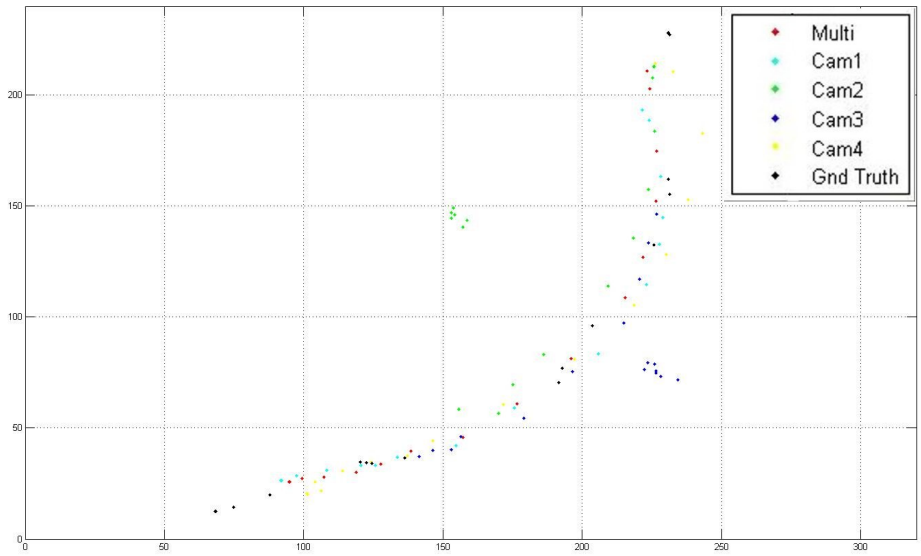


Fig. 4.29: Tracking results for all Cameras for the Second Trajectory

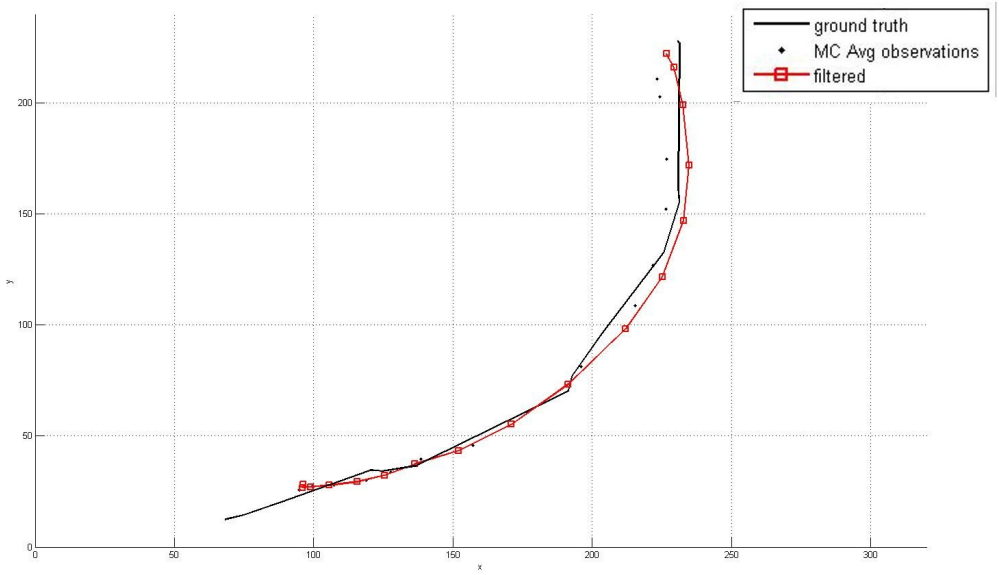


Fig. 4.30: Multi-Camera Average and Kalman Filter Comparison for the Second Trajectory

As plotted for the first trajectory, the centroids of each of the cameras were plotted for the whole trajectory. Each camera is identified by a specific color. The multi-camera average and the Kalman Filter are also included in Figure 2.29.

Finally, for the second trajectory, the multi-camera average observations were compared to those of the Kalman Filter. Judging from Figure 4.30 it is difficult to determine the better of the two, since both start off very close to the ground truth and deviate towards the second half of the trajectory.

During the previous experiments, the vehicle crossed through the center of the monitored area. The center of the monitored area is where all of the camera views overlap, so the best results for the segmentation are obtained in that area. In areas closer to the perimeter, only a couple cameras overlap or only a single camera, if any, monitors the area. Due to this, a more challenging trajectory is one that does not necessarily cross this central area and follows a counterclockwise perimetrical path. Such a path was used to test the tracking in the following figures: Figure 4.31 through Figure 4.36. Figure 4.31 shows how camera 1 fails to track the object at the beginning and end of the trajectory. The multi-camera tracking is more true to the path shown by the ground truth. Camera 2, shown in Figure 4.32, appears to do a better job of tracking the vehicle, but also has large errors at the first and third quarters of the trajectory. Camera 3 also has large errors as seen in Figure 4.33. These errors even appear larger than both previous cameras. By simply observing the tracking done by camera 3, it would be difficult to know that the trajectory was circular. Camera 4, shows better results than camera 3, but still misses a great part of the middle of the trajectory.

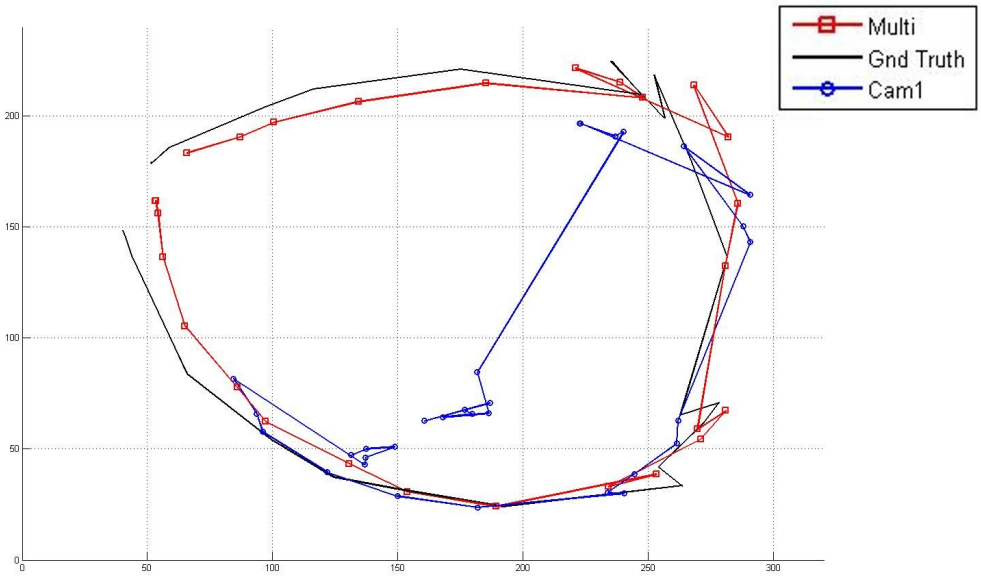


Fig. 4.31: Comparison of ground truth, multi-camera centroid average and Kalman tracking for camera 1 for the third trajectory.

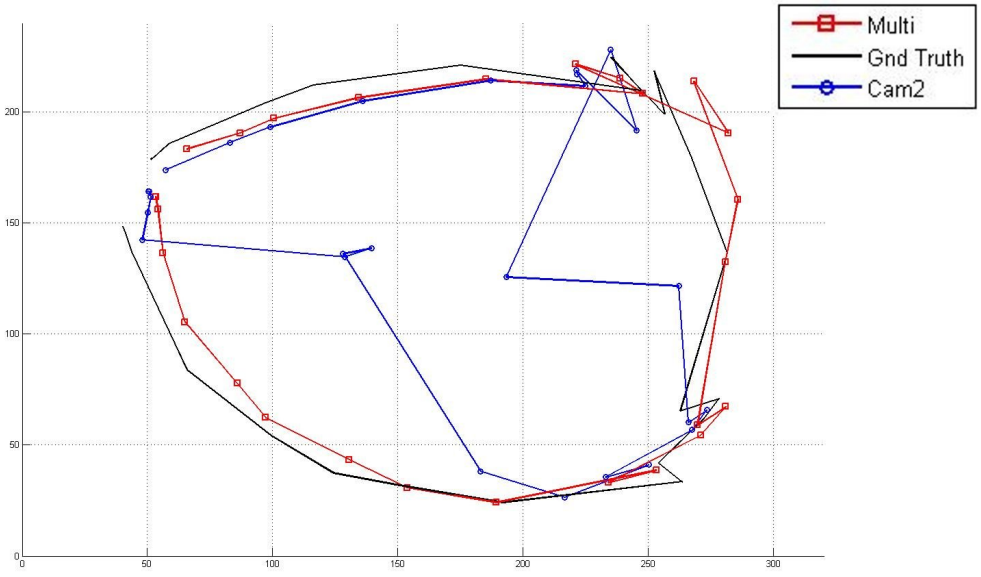


Fig. 4.32: Comparison of ground truth, multi-camera centroid average and Kalman tracking for camera 2 for the third trajectory.

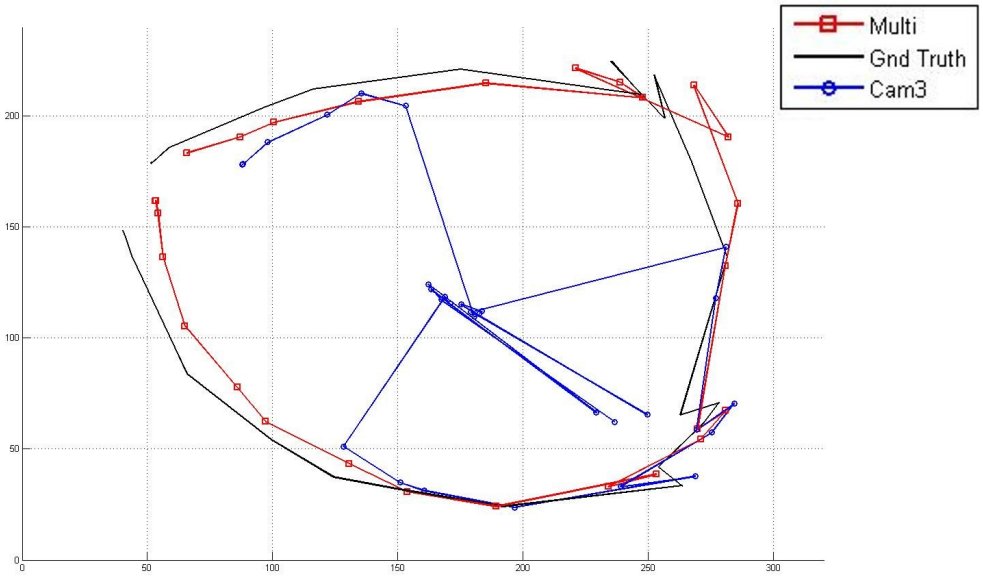


Fig. 4.33: Comparison of ground truth, multi-camera centroid average and Kalman tracking for camera 3 for the third trajectory.

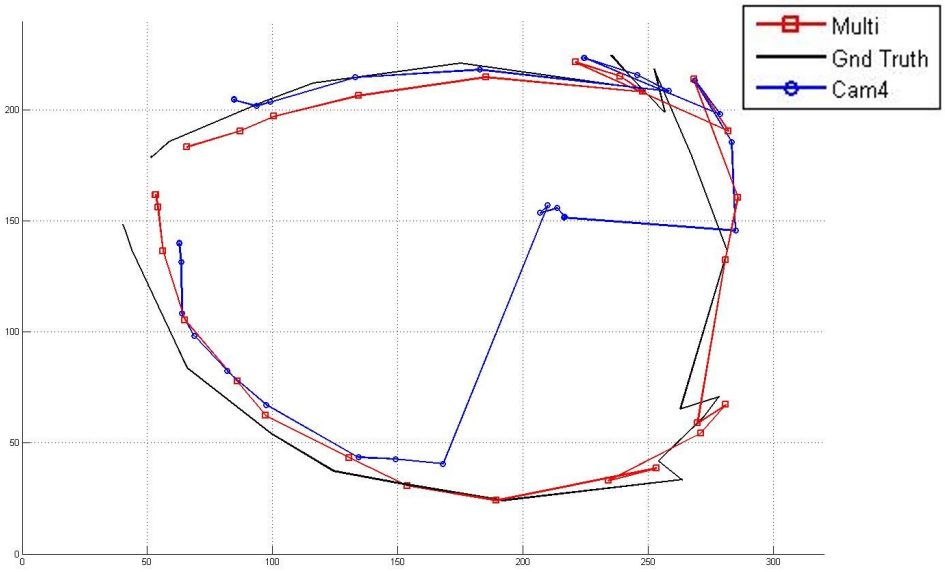


Fig. 4.34: Comparison of ground truth, multi-camera centroid average and Kalman tracking for camera 4 for the third trajectory.

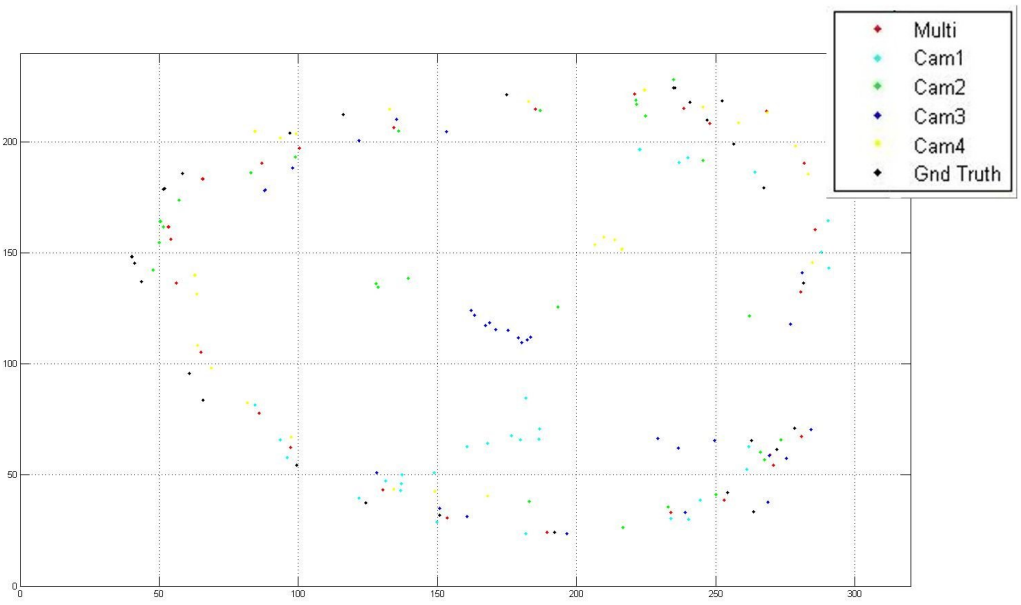


Fig. 4.35: Tracking results for all Cameras for the Third Trajectory

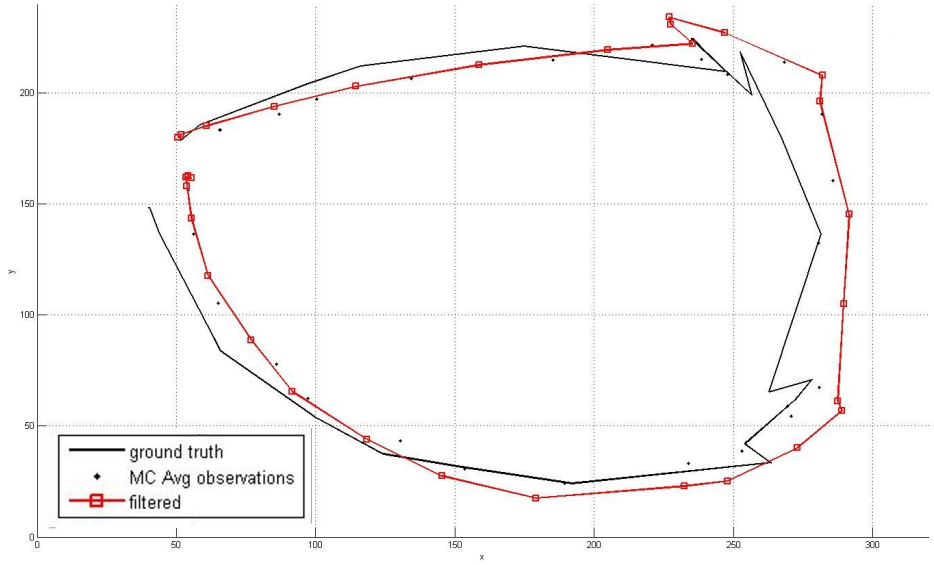


Fig. 4.36: Multi-Camera Average and Kalman Filter Comparison for the Third Trajectory

As in the previous trajectory experiments, the multiple camera average points were also compared to the Kalman filter results. Both the multi-camera average and the Kalman filter appear to perform better tracking than any of the individual cameras. There are some variations, but in general the tracked vehicle's path matches the ground truth observations.

So far, the graphs have only been compared visually. Even though the errors are evident in cases such as the circular trajectory, it is good to have a quantitative measure of the results to help with the comparisons. The mean squared error (MSE) for each individual camera was calculated with respect to the ground truth camera. The MSE for the Multi-Camera and Kalman Filter tracking was also calculated. Since, the centroid has two values per coordinate (x and y) the Euclidean distance between the points on the trajectories was calculated. The MSE for each trajectory was plotted in Figures 4.37 through 4.39.

The MSE plot of the first trajectory (Figure 4.37) reveals that the most unreliable camera is Camera 4, with Camera 1 following as second worst. Camera 2 is the best out of the individual cameras. However, the Multi-Camera Average and Kalman Filter have lower errors than Camera 2. The Multi-Camera Average is better than the Kalman Filter by a slim margin. The second trajectory MSE plot shows that now Camera 4 (most unreliable in previous trajectory) is now the best out of the individual cameras. Camera 3 is by far the worst. For this trajectory example, the Multi-Camera and Kalman Filter still produce lower mean square errors than the best individual camera, Camera 4.

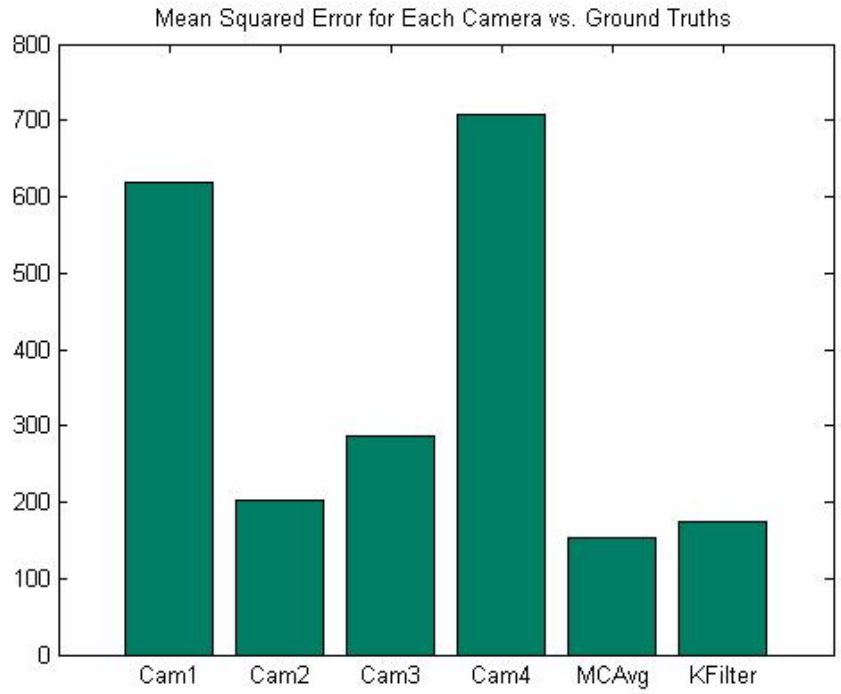


Fig. 4.37: Mean Square Errors for each Tracking Method for First Trajectory

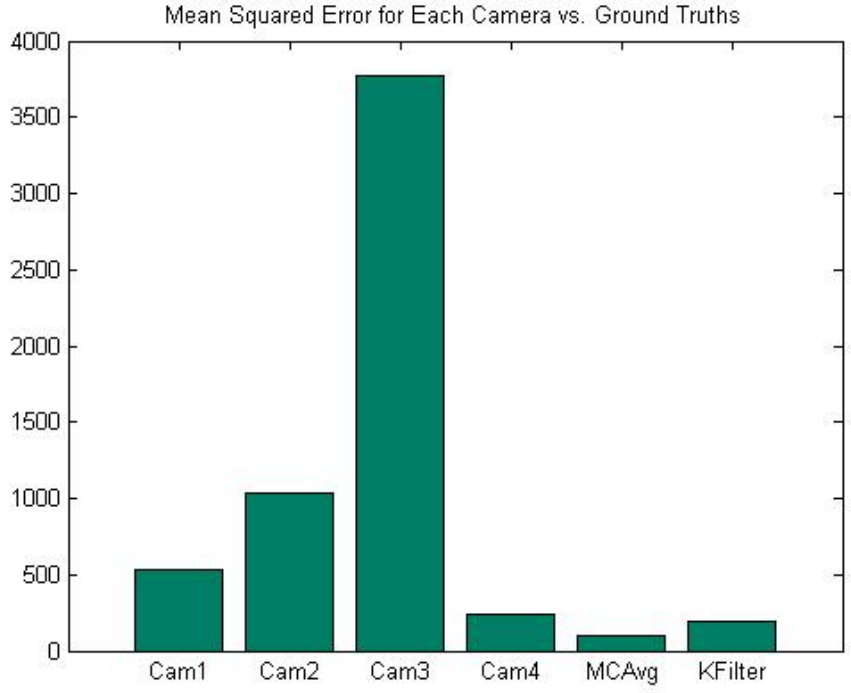


Fig. 4.38: Mean Square Errors for each Tracking Method for Second Trajectory

For the third trajectory, the results are similar. Out of the four individual cameras, Camera 3 is the worst, while Camera 2 follows closely. Camera 1 and Camera 4 have a mean squared error comparable to the Multi-Camera Average and the Kalman Filter. The Kalman Filter appears to have performed slightly better than the Multi-Camera Average on this occasion.

Comparing the different mean squared errors for each of the three trajectories showed that the individual cameras are an unreliable source for tracking. Depending on the trajectory, a single camera can have varying performance. The previous plots show that individual cameras have the highest mean squared errors. Both the Multi-Camera average and the Kalman Filter had lower MSEs for all three trajectories. The Multi-Camera average has the best results overall, while the Kalman filter is a close second.

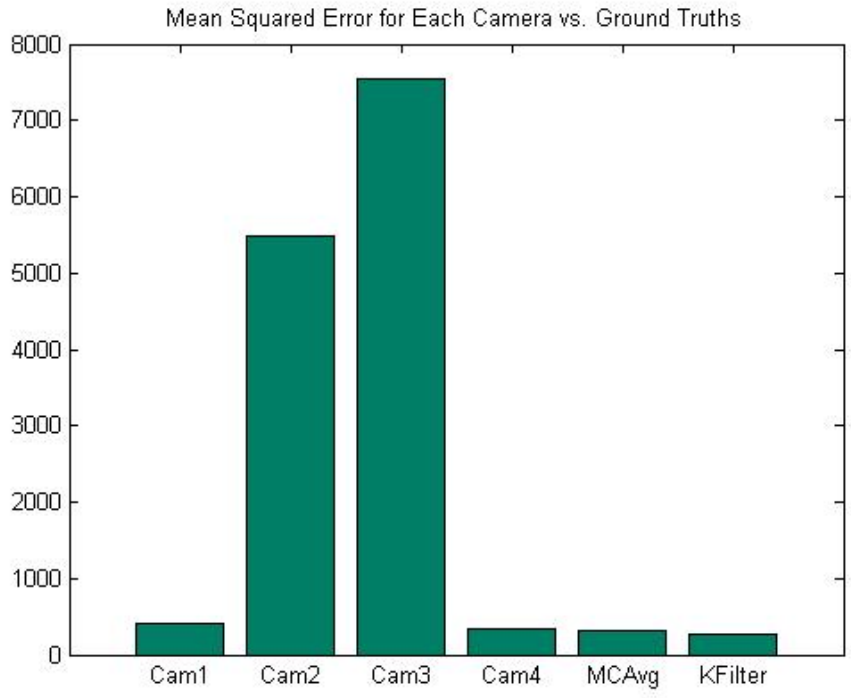


Fig. 4.39: Mean Square Errors for each Tracking Method for Third Trajectory

4.7 Complexity Comparison with Gray Level Co-Occurrence Matrix

Finally, the computation time of the GLCM and Multi-Camera camouflage detection methods were compared for three different image sizes. The GLCM was calculated with an offset of 1 and with all eight neighboring pixels. The GLCM calculates the contrast, correlation, homogeneity, and energy for a single image. The multi-camera detection executes background subtraction for four images, applies a different homography matrix to each one and averages the transformed views. It is important to mention that the calculation of the homography matrices is assumed to be known for this experiment. The homography matrix calculation is not considered for this comparison as it only needs to be done once offline. The computation time for each of these procedures was calculated using the Matlab *tictoc* function. Tests were run for three different image sizes. The image was scaled to a larger size using bi-cubic interpolation. The results of this experiment are summarized by Figure 4.40. In the case of smaller images, calculating the GLCM is faster than applying a homography to each camera. However, as the image size increases, the computational burden of the GLCM calculation becomes obvious. It appears that the GLCM calculation exponentially takes longer as images become larger, while the Multi-camera method also takes longer as the image size increases, but in a linear fashion. The results observed in this experiment encourage using the proposed method for camouflage detection over GLCM if a multiple camera network is available.

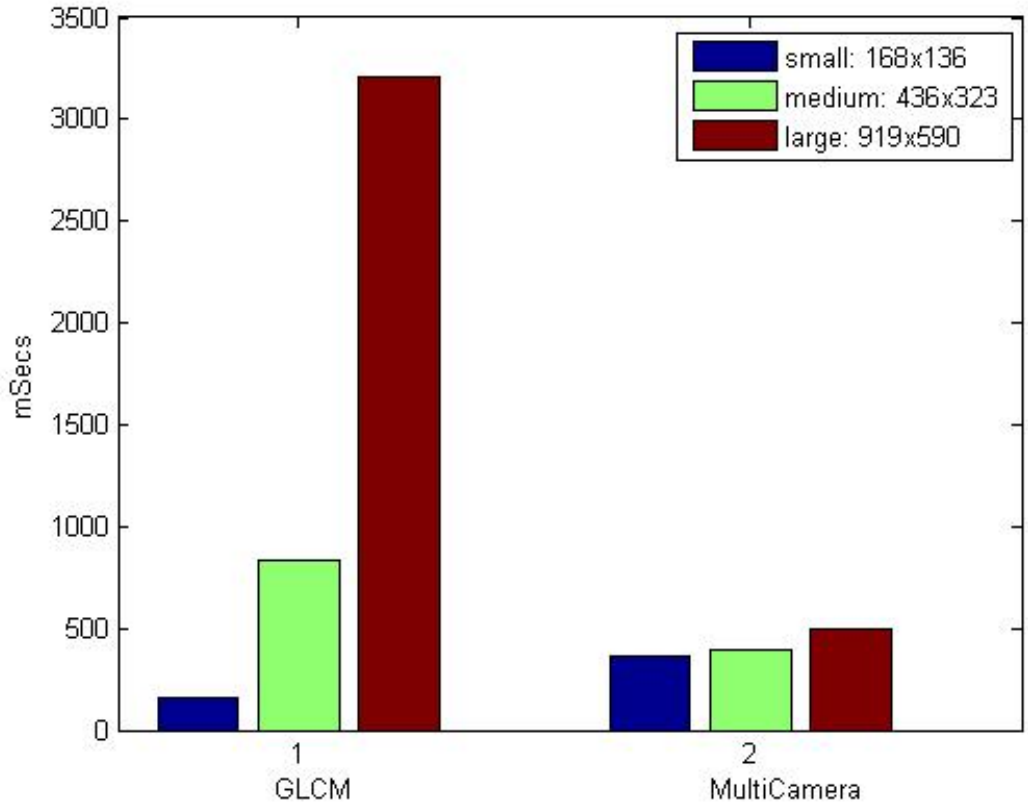


Fig. 4.40: Computation Time of GLCM and Multi-Camera Average for Three Image Sizes

Chapter 5: Conclusions and Future Work

Creating this surveillance system involved both theoretical and practical considerations. There are many positive results from this research that encourage future work in the area of camouflage breaking using camera networks.

The main conclusion and contribution of this thesis is that camouflaged targets can be detected without computationally expensive texture analysis if multiple cameras are used. By using these extra sensors and infrastructure, the algorithms can be simplified allowing them to run in a real-time application.

Another conclusion is that by using multiple cameras, the observation noise due to quick illumination changes, changing weather conditions, and shadows can be reduced. This is a noticeable advantage over a single camera system.

The results found by this research can be used to implement these algorithms and methods on a wireless vision sensor network (VSN) without much computational load. The calibration step could be done previously to avoid solving for the DLT online. Furthermore, instead of transmitting back the whole image, perhaps only certain image features such as the centroid and object variance in x or y could be transmitted. This would save on sensor power and bandwidth resources. This idea is depicted in Figure 5.1.

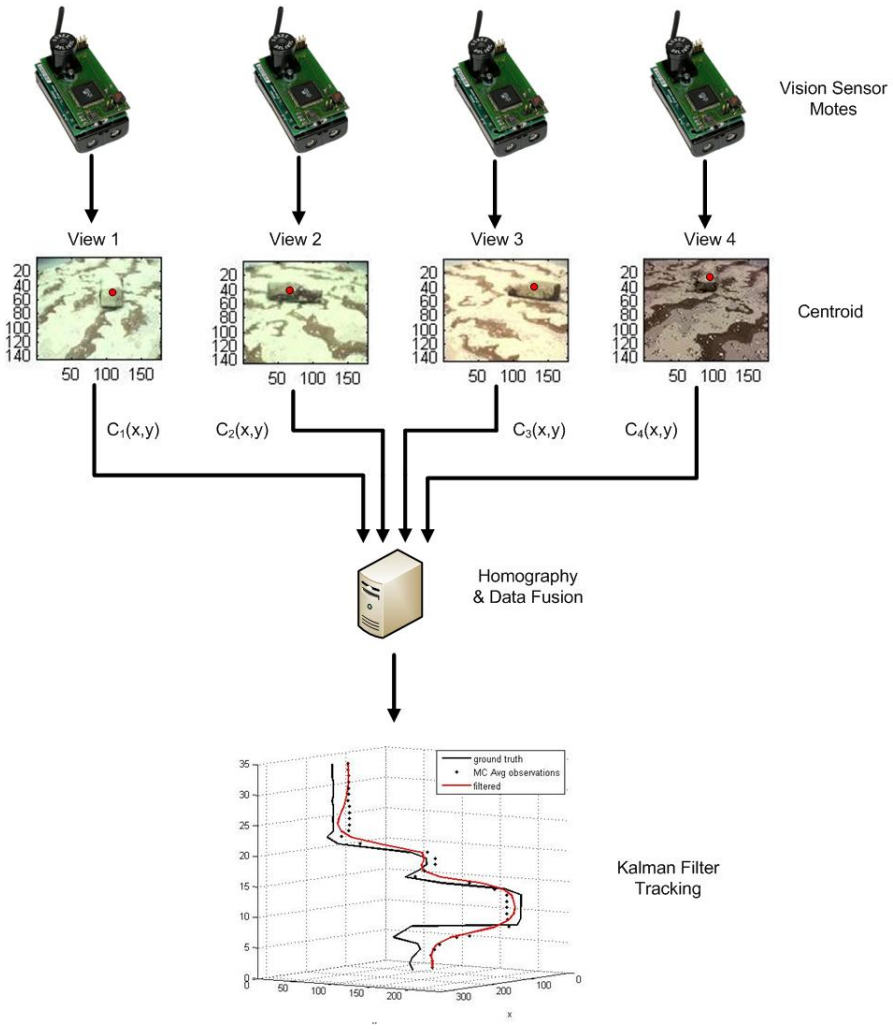


Fig. 5.1: Vision Sensor Surveillance

Other future work that could be done to improve this system is to use a different kind of cameras other than webcams. IP cameras or omni-directional cameras could allow the system to have more flexibility in terms of implementation. Using wireless IP cameras for example could make deployment easier. Omni-directional cameras could achieve larger overlapping fields of view, thus improving results.

The camera calibration method used for this system is done manually. A different camera calibration technique could be used. This could be done automatically, without the human intervention, making the system even more autonomous.

As far as tracking, the Kalman filter could be implemented in a non-centralized fashion. A central Kalman filter can be decomposed into n micro-Kalman filters achieving the same results for the estimate of the state [33].

Despite the promising results obtained by this study, further work is necessary to translate the system to a commercial application particularly to improve its robustness [5]. Eventually, a real world application would determine several constraints that the system would have to address. However, this thesis has helped to study the different tasks, challenges and outcomes that can be achieved by a multiple camera surveillance system operating in real-time.

Finally, a side contribution of this work is the creation of a low cost real-time multi-camera system that can be easily implemented by other researchers interested on doing research in this area. The mock scenario and camera arrays can be slightly modified to be used for stereo vision, 3D reconstruction or other multi camera research.

References

[1] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, March 2004.

[2] R. C. Gonzalez and R. E. Woods, *Digital Image Processing (3rd Edition)*. Prentice Hall, August 2007.

[3] New Scientist, 12 July 2003, page 4-5.

[4] A. Dick and M. Brooks. Issues in automated visual surveillance.*Proc. of the 7th Int. Con. on Digital Image Computing:Techniques and Applications*, pages 195–204, 2003.

[5] I. Pavlidis, V. Morellas, P. Tsiamyrtzia, and S. Harp. Urban surveillance systems: from the laboratory to the commercial world. Proceedings of the IEEE, 89(10):1478-1497, 2001.

[6] R. Radke, S. Andra, O. Al-Kofahi, and B. Roysam, "Image change detection algorithms: A systematic survey," *Image Processing, IEEE Transactions on*, vol. 14, no. 3, pp. 294-307, 2005.

[7] Xu, M., Niu, R., Varshney, P.K.: Detection and tracking of moving objects in image sequences with varying illumination. ICIP, Vol. 4 (2004) 2595-2598

[8] L. Snidaro and G. Foresti, "Real-time thresholding with Euler numbers," *Pattern Recognit. Lett.*, vol. 24, no. 9\–10, pp. 1533-1544, 2003.

[9] C. Su and A. Amer, "A real-time adaptive thresholding for video change detection," in *IEEE Int. Conf. On Image Processing*, Atlanta, GA, USA, Oct. 2006, pp.157–160.

[10] Yu Jin Zhang, "A review of recent evaluation methods for image segmentation," *Signal Processing and its Applications, Sixth International, Symposium on. 2001* , vol.1, no., pp.148-151 vol.1, 2001

[11] Stancil, B. A.; Zhang, C.; Chen, T., "Active Multicamera Networks: From Rendering to Surveillance," *Selected Topics in Signal Processing, IEEE Journal of* , vol.2, no.4, pp.597-605, Aug. 2008.

[12] Collins, R.T.; Lipton, A.J.; Kanade, T., "Introduction to the special section on video surveillance," *Pattern Analysis and Machine Intelligence, IEEE Transactions on* , vol.22, no.8, pp.745-746, Aug 2000.

[13] May, A.; Teh, J.; Hobson, P.; Ziliani, F.; Reichel, J., "Scalable video requirements for surveillance applications," *Intelligent Distributed Surveilliance Systems, IEE* , vol., no., pp. 17-20, 23 Feb. 2004.

[14] James Black and Tim Ellis, *Multi Camera Image Tracking*, 2nd IEEE International Workshop on Performance Evaluation of Tracking and Surveillance, PETS2001, Hawaii, USA, December 2001.

[15] L. Jiao, G. Wu, Y. Wu, E. Y. Chang, and Y. F. Wang, "The anatomy of a multi-camera video surveillance system."

[16] S. Funiak, C. Guestrin, M. Paskin, and R. Sukthankar, "Distributed localization of networked cameras," 2006, pp. 34-42.

[17] Amit Goradia, Zhiwei Cen, Clayton Haffner, Ning Xi, Matt W. Mutka: Design, Implementation and Performance Analysis of Pervasive Surveillance Networks. FLAIRS Conference 2006: 472-477.

[18] L. Lee, R. Romano, and G. Stein, "Monitoring activities from multiple video streams: establishing a common coordinate frame," *Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 758-767, 2000.

[19] B. Bose and E. Grimson, "Ground plane rectification by tracking moving objects."

[20] Mittal, A., and Davis, L.S., M2 Tracker: A multi-view approach to segmenting and tracking people in a cluttered scene, International Journal of Computer Vision, 2005.

[21] T. E. Boult, R. J. Micheals, X. Gao, and M. Eckmann, "Into the woods: visual surveillance of noncooperative and camouflaged targets in complex outdoor settings," *Proceedings of the IEEE*, vol. 89, no. 10, pp. 1382-1402, 2001.

[22] B. Flinchbaugh and T. Olson, "Autonomous video surveillance", in 25th AIPR Workshop: Emerging Applications of Computer Vision, May 1996.

[23] Whicker, A. “Camouflage Breaking: A Review of Contemporary Techniques”.
[24] Ariel Tankus, Yehezkel Yeshurun, "Convexity-Based Camouflage Breaking," icpr,pp.1454, 15th International Conference on Pattern Recognition (ICPR'00) - Volume 1, 2000.
[25] Huang, Z.Q. and Jiang, Z.: Tracking Camouflaged Objects with Weighted Region Consolidation, Proceedings of Digital Image Computing: Techniques and Application, (2005) 161-168.
[26] S. Nyberg and L. Bohman, "Assessing camouflage methods using textural features," Opt. Eng. 40(9) (2001).
[27] J. F. Camapum Wanderley and M. H. Fisher, “Spatial-Feature Parametric Clustering Applied to Motion-Based Segmentation in Camouflage”, *Computer Vision and Image Understanding*, no. 84, pp. 1-14, December 2001.
[28] Bhajantri, N.U.; Nagabhushan, P., "Camouflage Defect Identification: A Novel Approach," *Information Technology, 2006. ICIT '06. 9th International Conference on* , vol., no., pp.145-148, 18-21 Dec. 2006.
[29] National Instruments. “Can I Acquire from Two USB Cameras Simultaneously with NI-IMAQ for USB Cameras 1.0?”. NI Support Knowledge Base . April 2008. http://digital.ni.com/public.nsf/allkb/6985219C76DB128C862571DC005CB1CC
[30] Hussain, S. and Samad, T., “Graphical User Interface (GUI) Development for Object Tracking System in Video Sequences”, World Applied Sciences Journal 4 (2):244-249, 2008.
[31] Otsu, N., "A Threshold Selection Method from Gray-Level Histograms," IEEE Transactions on Systems, Man, and Cybernetics,Vol. 9, No. 1, 1979, pp. 62-66.
[32] Kevin Murphy, “Kalman Filter Toolbox for Matlab”. 1998. The University of British Columbia. June 2004. http://www.cs.ubc.ca/~murphyk/Software/Kalman/kalman.html
[33] Olfati-Saber, R., "Distributed Kalman Filter with Embedded Consensus Filters," *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05. 44th IEEE Conference on* , vol., no., pp. 8179-8184, 12-15 Dec. 2005
[34] G. Welch and G. Bishop, "An introduction to the kalman filter." March 2007. University of North Carolina at Chapel Hill http://www.cs.unc.edu/~welch/kalman/kalmanIntro.html
[35] A. Agarwal, C. V. Jawahar, and P. J. Narayanan. A survey of planar homography estimation techniques. Technical report, IIT-Hyderabad, 2005.
[36] Eshel, R.; Moses, Y., "Homography based multiple camera detection and tracking of people in a dense crowd," *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on* , vol., no., pp.1-8, 23-28 June 2008
[37] S.M. Khan, M. Shah, A multiview approach to tracking people in crowded scenes using a planar homography constraint, in: European Conference on Computer Vision, Graz, Austria, May 7–13, 2006.
[38] S.N. Lim, A. Mittal, L.S. Davis, N. Paragios, Fast illumination-invariant background subtraction using two views: error analysis, sensor placement and applications, in: Computer Vision and Pattern Recognition, San Diego, CA, June 20–25, 2005.
[39] Ivanov, Y.; Bobick, A.; Liu, J., "Fast lighting independent background subtraction," *Visual Surveillance, 1998. Proceedings., 1998 IEEE Workshop on* , vol., no., pp.49-55, 2 Jan 1998
[40] http://en.wikipedia.org/wiki/Moore's_law
[41] http://www.video-surveillance-guide.com/history-of-video-surveillance.htm
[42] http://vips.sci.univr.it/tools/vfm/
[43] http://www.mathworks.com/products/imaq/
[44] http://www.fp.ucalgary.ca/mhallbey/tutorial.htm

Curriculum Vitae

Jesus Flores Guerra was born on July 18th, 1981 in Torreon, Coahuila, Mexico. He is the son of Jesus Vicente Flores Morfin, Ph.D. and Dimpna Guerra de Flores. He is the middle child of the family. He has an older sister, Dimpna Flores Guerra, MSIT and a younger sister Jacqueline. His pursuit of higher education began in Mexico at the Universidad Iberoamericana de la Laguna. After a year and a half of studies, he relocated to El Paso, Texas to continue his studies in the company of his older sister. During his time at the University of Texas at El Paso, he worked as a support technician at the Undergraduate Learning Center. Afterwards he worked for the Information Technology Department at the University of Texas at El Paso. His bachelor's concentration was Communications. For his senior year project, he and his teammates Brandon Aguirre and Yassir Granollo, built an ultrasonic loudspeaker capable of directing sound like a beam. This project earned the distinction of Outstanding Senior Project for Fall of 2006. He graduated cum laude from the University of Texas at El Paso in Fall of 2006. He continued his studies as a graduate student and was accepted into the Wireless Sensor Networks Group, by his advisor Dr. Gerardo Rosiles. During this time, he worked as a teaching assistant for Digitals Design I and as a research assistant working on multi-camera surveillance. He obtained his Master's degree from the University of Texas at El Paso in Fall 2008. Jesus is a member of IEEE, MAES, Tau Beta Pi, Golden Key National Honor Society and Eta Kappa Nu. His research interests have been and continue to be digital signal processing, in particular audio and video, vision sensor networks, video analytics & computer vision, data compression, human computer interaction and their real-time implementations.