An Improved Gaussian Mixture Model Algorithm for Background Subtraction

by

Nikhil Sadarangani

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2002

© Nikhil Sadarangani, MMII. All rights reserved.

The author hereby grants to MIT permission to reproduce and distribute publicly paper and electronic copies of this thesis document in whole or in part.
An Improved Gaussian Mixture Model Algorithm for Background Subtraction

by

Nikhil Sadarangani

Submitted to the Department of Electrical Engineering and Computer Science on May 24, 2002, in partial fulfillment of the requirements for the degree of Master of Engineering in Electrical Engineering and Computer Science

Abstract

Background subtraction algorithms are the standard method for detecting moving objects from video scenes; they work by subtracting a model of the background image from the video frames. Previous methods developed by Stauffer and Grimson[12] used adaptive Gaussian mixture models to represent the background model. This thesis presents improvements to this approach. Specifically, it examines the use of the HSV color space to detect and avoid shadows, which is a significant problem in the original algorithm. It also describes adaptive algorithms for per-pixel learning constants and background thresholds. Performance results for the modified tracker, as well as those for the original algorithm, are presented and discussed.

Thesis Supervisor: W. Eric L. Grimson
Title: Bernard Gordon Professor Of Medical Engineering
Acknowledgments

First and foremost, I would like to thank my advisor, Professor Eric Grimson, for not only giving me the opportunity to conduct this research but also allowing me the flexibility to guide its direction. I am also deeply indebted to Chris Stauffer, without whom I would have surely led myself astray.

Words cannot express my gratitude to my family for all their love and support during these past 22.84 years. But it wouldn't be very nice if I didn't write something. To my parents, thank you for having the faith to let me make my own mistakes and find my own path through adolescence. Although I was sometimes closer to ending up in the Massachusetts State Penitentiary than the Massachusetts Institute of Technology, I eventually straightened myself out. Sapna, thanks for all the free food and the "sightseeing" in the Leverett dining hall. Knowing you were always a few minutes down the road made Boston feel a little more like home. The all-you-can-eat dinners didn't hurt either. Tina, you really didn't do much but you know I still love ya.

To my friends, I will always cherish the many episodes of drunken debauchery we enjoyed over the past five years. Well, at least the ones I remember. To Rooz, Emelio, Ashok, and even Jimmy, with whom I lived, partied, and even started companies, you've become the multi-ethnic, adopted brothers I never had. I'm sure we'll all cross paths again soon. Abhishek, you've really been the closest thing to a brother I've had these past few years. If you make my little sister's dreams come true and marry her, we can make it official. Jason, your unceasing optimism and support helped get my through this last month. If not for your tales of the wonders in store for me in California, I might not have had the motivation to complete my thesis on time.

Lastly, much thanks goes to the wonderful people at the Friendly Eating Place, without whom I might have starved before completing my thesis.

In memory of my grandfather, Harbhajan Sandhu (1932 – 2002).
3 Evaluation of Improvements 47
  3.1 Testing Methodology ................................................. 47
  3.2 Results ..................................................................... 48
    3.2.1 HSV-Based Algorithms ........................................... 49
    3.2.2 Adaptive Learning Constants ................................. 51
    3.2.3 Adaptive Background Threshold ............................... 54

4 Discussion 57
  4.1 HSV-Based Algorithms ............................................... 57
    4.1.1 General Performance .............................................. 57
    4.1.2 Shadow Detection ................................................... 59
    4.1.3 Modified HSV Color Space ...................................... 60
    4.1.4 Pixel Smoothing ..................................................... 61
  4.2 Adaptive Learning Constants ....................................... 62
    4.2.1 Performance Results ............................................... 62
    4.2.2 Trade-off Between Sleeping and Waking Objects .......... 65
  4.3 Adaptive Background Threshold .................................... 65

5 Conclusions 67
  5.1 Synopsis of Results .................................................. 67
  5.2 Future Work ............................................................. 68
    5.2.1 Performance Improvements ...................................... 69
    5.2.2 Hybrid RGB/HSV Color Spaces ................................. 69
List of Figures

2-1 The HSV hexacone color model .................................................. 23
2-2 Plot of the PDF of max(r, g, b) against a univariate Gaussian PDF . 27
2-3 Plot of the PDF of min(r, g, b) against a univariate Gaussian PDF . 28
2-4 Expected saturation noise for the original and modified color spaces . 34
2-5 Transfer function for α .............................................................. 41
2-6 Decision matrix for adjusting background threshold .................... 44
2-7 Sample transitions for adaptive thresholding algorithm ............... 45

3-1 Sample frame of stalled cars at an intersection .......................... 54

4-1 Foreground detection of stalled objects ................................. 64
### List of Tables

3.1 Comparison of general outdoor tracking performance for gray vs. non-gray objects ................................ 50
3.2 Video footage for HSV-based shadow detection ....................... 51
3.3 Detection rates for uniform black background ........................ 52
3.4 Detection rates for uniform gray background ......................... 52
3.5 Detection rates for uniform red background .......................... 53
3.6 Detection rates for uniform teal background .......................... 53
3.7 Detection rates for uniform cyan background with soft, visible shadows 54
3.8 Detection rates for stalled objects over a five-second window .... 55
3.9 Detection rates for adaptive thresholding algorithm ................. 56
Chapter 1

Introduction

This thesis presents a number of improvements on the Gaussian mixture model-based background subtraction algorithm developed by Stauffer and Grimson [12]. As its name might suggest, a “background subtraction algorithm” is responsible for separating objects of interest from the background of a scene. The pixels which compose these objects are called “foreground pixels.” The remaining pixels, which make up the background scene, are referred to as “background pixels.”

Chapter One discusses the motivation for developing background subtraction algorithms and previous work in the field. Also, it sets forth the goals for this project. Chapter Two describes the design of the original Gaussian mixture model algorithm developed by Stauffer and Grimson. It then presents each of the improvements researched during the course of this project. Chapter Three presents the methodology and experiments used to test the improvements, as well as the results themselves. Chapter Four provides an analysis of the result data, and describes the strengths and weaknesses of the improved algorithm in comparison to the original. Chapter Five offers concluding thoughts on the results of this research and describes possible future work.
1.1 Motivation for Object Tracking Systems

Once relegated only to academia, object tracking and recognition systems have now found their way into a number of different applications and industries. Automated video surveillance is the most common application of the technology, and for good reason. It enables areas to be monitored continuously and with little human assistance, making it both more effective and less costly over extended periods. Video cameras can be placed in almost any location, from small offices and residences to expansive parking lots and even football stadiums. The captured image streams can then be analyzed in real-time by image-tracking algorithms to detect and identify people, objects, or events of interest. Security is the most obvious application of such technology. In a much-publicized example, state and federal police set up an array of hidden cameras to monitor crowds during Super Bowl XXXV. The video feeds were then processed in real-time by computers, which extracted faces from the scenes and tried to match them against a database of known criminals—at a rate of one million images per minute. Smaller automated surveillance systems have begun to appear in banks and other institutions that require additional security [11].

For surveillance and facial recognition systems, accurate detection and robustness are essential. These systems are generally used to protect high-value items or locations, and they must perform reliably under a wide range of conditions. Improvements in object tracking technology have a clear monetary value to their users.

1.2 Background Subtraction Algorithms

In order to track moving objects in video data, one must first be able to identify them. Background subtraction algorithms attempt to detect objects by “subtracting” the background scene from video images; the remaining pixels can then be connected into groups to represent foreground objects. To correctly subtract the background scene, these algorithms need a good representation of the background itself. The background model may need to adapt over time as scene elements, light sources, and
environmental conditions change. Various approaches to creating this background model will be discussed later in this section.

1.2.1 Difficulties of Background Subtraction

Although object detection and tracking is a well-researched topic in artificial intelligence, current systems are far from perfect. Background subtraction is a problem fraught with challenges, and no one system can successfully solve them all. The tradition set of “canonical” problems are listed below [13]:

Moved Objects. Objects once part of the background scene can be displaced. These objects should eventually be incorporated back into the background model.

Time of Day. The appearance of the background may change due to gradual lighting changes during the course of the day.

Light Switch. The appearance of the background may change due to sudden lighting changes in the scene, such as turning on a light.

Waving Trees. Backgrounds may have multiple modal states; the corresponding background model must be able to represent disjoint sets of pixel values.

Camouflage. A foreground object’s pixel characteristics may be subsumed by the modeled background.

Bootstrapping. An algorithm should be able to properly initialize itself without depending on the availability of a training period.

Foreground Aperture. When a homogeneously colored object moves, change in the interior pixels cannot be detected. Thus, the entire object may not appear as foreground.

Sleeping Person. A foreground object that becomes motionless cannot be distinguished from a background object that moves and then becomes motionless.
**Walking Person.** When an object initially in the background moves, both it and the newly revealed parts of the background appear to change.

**Shadows.** Foreground objects often cast shadows which appear different from the modeled background.

The majority of the research presented in this thesis focuses on the last four problems: foreground aperture, sleeping person, waking person, and shadows.

### 1.2.2 Previous Work

Early attempts at background subtraction employed non-adaptive algorithms to detect foreground pixels. These systems required a human operator to manually initialize the background layer, usually via a training sequence of images that did not contain any foreground objects. This approach is problematic at best. Training sequences are not always available in real-world systems, and without periodic re-initialization, errors will propagate over time.

Other foreground detection algorithms systems avoid the use of a background model at all. In the pixel differencing method\cite{7}, for example, foreground pixels are detected by taking the difference of two frames in the video sequence; if this difference exceeds a given threshold, the pixel is considered to be foreground. Pixel differencing requires no initialization but suffers from a number of problems, most notably that stationary foreground objects will not be detectable. This limitation generally also applies to the broader class of “optical flow” based algorithm.

Most researchers have deserted such approaches in favor of adaptive systems. Usually, these systems maintain a statistical model of the background pixels that is continuously updated over time, thereby eliminating the need for initialization. The simplest method of background estimation is to average the color values of pixels over time. Any new pixel value that differs the background estimate by some threshold can be considered part of a foreground object. In an environment with many objects (especially slow-moving ones) or a non-static background, these systems generally
fail. Averaging-and-thresholding algorithms are also susceptible to changes in lighting conditions, which can affect pixel values across the entire image or on just a subset[12].

Newer approaches use more advanced background maintenance techniques. While they are more effective than simple averaging, they are not without their limitations. Oliver et al.[9] designed an “eigenbackground” algorithm that modeled the background using eigenvectors; their system was more responsive to lighting changes, but it did not deal well with the problem of moved objects. Ivanov[5] used disparity verification to extract foreground objects, but the system required a computationally-intensive initialization before it could be used on-line. Ridder et al.[10] employed Kalman-filtering to model each pixel. While this system was more adept at handling changes in lighting, it had a slow recovery time and could not handle bimodal backgrounds. Elgammal et al.[2] modeled each pixel using a kernel estimator based on a sliding window of recently observed pixel values. The Wallflower system[13] modeled each background pixel using a one-step Wiener filter; this was augmented by a region-based algorithm to overcome some of the limitations of a purely pixel-based approach, specifically foreground aperture problems.

The idea of using Gaussian distributions to model background pixels is a popular approach to adaptive background maintenance. The Pfinder system[14] uses a multi-class statistical model for tracked objects and models each background pixel as a Gaussian distribution, updated using a simple adaptive filter. Any observed pixel values that do not lie within some distance from their respective means are considered to be foreground. This method adapts well to slow scene changes, but performs poorly in scenes with multimodal backgrounds. This approach was extended by Friedman et al.[3] to support three Gaussian distributions per pixel. However, the system suffered the same problems with multimodal scenes, as only one pixel was used to model the background. Stauffer and Grimson[12] generalized the approach to a mixture of $K$ weighted Gaussians; this algorithm forms the foundation for the improvements in this thesis.

---

1 Friedman’s system was designed specifically for tracking road scenes. The three Gaussian distributions were used to model the pixel values of the road, car, and shadow. Thus, only one Gaussian (that of the road) was being used to model the background.
Others have also attempted to improve upon some of the limitations of the Gaussian mixture model (GMM) approach. In [6], a chromatic distortion index was used in conjunction with observed RGB pixel values to separate shadowed pixels from foreground pixels. Harville et al.[4] used stereo cameras operating in the YUV color space to separate shadowed pixels, and also introduced per-pixel, adaptive learning constants. The adaptive algorithm will be discussed in more detail in Section 2.3.2.

1.3 Project Goals

The aim of this research project is to improve upon the overall performance and detection rates of the original GMM algorithm in [12]. The selection of this algorithm as the basis for this research project is based on a number of factors. First, the algorithm provides a solid foundation for improvement; it has already proven to be robust under a range of scene conditions and environmental factors. It can successfully adapt to slow lighting changes, repetitive motions of scene elements, and multimodal backgrounds, and can track well even in cluttered or heavily trafficked regions. Second, the algorithm’s key limitations are few in number and quite specific: it detects shadows as foreground pixels, and fast lighting changes can corrupt large regions or the entire scene. Also, tracking suffers for stalled foreground objects (i.e., the “sleeping person” problem mentioned in Section 1.2.1). Ideally, the improved algorithm presented in this thesis should offer better performance in these areas without sacrificing quality in other aspects.
Chapter 2

Base Algorithm and Improvements

This chapter provides a detailed explanation of the original Gaussian mixture model algorithm in [12]. Then, it presents the improvements researched in this project.

2.1 Gaussian Mixture Model Algorithm

In the original algorithm presented by Stauffer et al., each pixel’s observed values are modeled as a mixture of adaptive Gaussian random variables. The rationale for this is as follows. Each pixel’s observed value can be interpreted as the result of light reflected off the object closest to the camera that intersects the pixel’s optical ray. In a completely static scene under constant lighting conditions, a single Gaussian would be sufficient to model this value and the noise introduced by the camera. However, scenes are never completely static; as objects move within the scene, some subset of the observed pixel values in the scene will change due to the presence of a different surface reflecting light. Each pixel will require separate Gaussians for the surfaces that pass through it. Furthermore, lighting conditions in most scenes are not usually static; in outdoor scenes, for example, the position of the sun relative to the camera changes over time. The Gaussians in each pixel’s mixture must be able to adapt over time to correct for the changes in lighting.

Every pixel has its own background model, which is determined by the variance and persistence of the Gaussians in its mixture. The backgrounding algorithm is
dependent on two key parameters: $\alpha$, the learning constant which determines how quickly the model adapts, and $T$, the proportion of observed pixel values that should be accounted for by the background model.

At a given time $t$, the recent history of observed values for each pixel, $\{X_0, X_1, \ldots, X_{t-1}\}$, is modeled by the pixel's Gaussian mixture. The probability of observing the current pixel value is

$$P(X_t) = \sum_{i=1}^{K} [\omega_{i,t} \times \eta(X_t, \mu_{i,t}, \Sigma_{i,t})]$$

where $\omega_{i,t}$, $\mu_{i,t}$, and $\Sigma_{i,t}$ are the weight, mean, and covariance matrix of the $i^{th}$ Gaussian in the mixture at time $t$, respectively. The prior weights are normalized so that they sum to one. $K$ is a parameter in the algorithm specifying the number of Gaussians in each pixel's mixture, and $\eta$ is the multivariate Gaussian probability distribution function

$$\eta(X_t, \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(X_t-\mu)^T \Sigma^{-1} (X_t-\mu)} .$$

$K$ is limited by the computational resources available; generally it is kept between 3 and 5. Also, to avoid the computational expense of matrix multiplication and inversion—at the expense of some accuracy in the model—the color components of each pixel's observed values are assumed to have the same variance:

$$\Sigma_{i,t} = \sigma_i^2 I .$$

We would like to adjust our model to maximize the likelihood of the observed data. Rather than using an exact expectation maximization algorithm on the recent data window, which would be computationally expensive, an on-line $K$-means approximation is used. Every new pixel value $X_t$ is matched against the $K$ Gaussians in the existing mixture (sorted by likelihood) until a match is found. A successful match is defined as a pixel value which lies within $2.5^1$ standard deviations from the mean of a particular distribution.

$^1$This value can be perturbed with little effect on the overall performance of the algorithm.
If none of the $K$ Gaussians matches the observed value, then the “least likely”
distribution is replaced by a new Gaussian with a large variance, low prior weight,
and mean equal to the newly observed value. The likelihood of a particular Gaussian
is based on its score, which is defined as $\omega/\sigma$. This way, Gaussians with greater
supporting evidence (i.e., higher $\omega$) and lower variance will receive high scores and
considered more likely to be part of the background model.

The prior weight of each of the $K$ distributions at time $t$ is adjusted by a standard
learning algorithm:

$$\omega_{k,t} = (1 - \alpha)\omega_{k,t-1} + \alpha(M_{k,t})$$

(2.4)

where $\alpha$ is the learning rate parameter and $M_{k,t}$ is 1 if the $k$th model matches and 0
otherwise. The term $1/\alpha$ may be viewed as the time constant which dictates the rate
at which the model is updated. Upon each update, the weights are renormalized.

For unmatched Gaussians, the parameters $\mu$ and $\sigma$ are left unchanged. However,
for a matched Gaussian, they are updated in the same fashion as $\omega_{k,t}$:

$$\mu_{k,t} = (1 - \rho)\mu_{k,t-1} + \rho X_t$$

(2.5)

$$\sigma^2_{k,t} = (1 - \rho)\sigma^2_{k,t-1} + \rho(X_t - \mu_{k,t})^T(X_t - \mu_t)$$

(2.6)

where

$$\rho = \alpha \eta(X_t, \mu_{k,t}, \sigma_{k,t}).$$

(2.7)

After all the parameters for each Gaussian have been updated, the Gaussians are
sorted according to their score, which as mentioned above is equal to $\omega_{k,t}/\sigma_{k,t}$. The
background model is then defined as the first $B$ distributions in the sorted list, where

$$B = \text{argmin}_b \left( \sum_{k=1}^b \omega_k > T \right).$$

(2.8)

$T$ is the background threshold parameter that specifies the minimum percentage of
the observed data that should be explained by the background model. For small
The values of $T$, the background model is generally unimodal. For pixels with multimodal backgrounds, generally caused by repetitive background motion (e.g., waving branches on a tree), a high background threshold could cause more than one distribution (and hence more than one color) to be included in the background.

2.2 Improvement 1: Using HSV Color Space

The original GMM algorithm was intended to be used in the Red-Green-Blue (RGB) color space. RGB color is the *de facto* color model for computer graphics, primarily because it lends itself well to most display and recording devices, such as video monitors and CCD cameras. By no means is RGB the only color space at our disposal; numerous other color spaces exist. In fact, the shadow detection algorithm to be presented in this section uses the Hue-Saturation-Value (HSV) color space. Before discussing the algorithm itself, we shall briefly review the HSV color model.

2.2.1 HSV Color Space Defined

The HSV color space is commonly used by graphics artists to represent color because the space is natural; the individual color components are more easily perceivable by a human user. The *hue*, commonly referred to as “tint,” is the gradation of color being perceived, such as “reddish” or “greenish.” The *saturation* is the chromatic purity of the color (*i.e.*, the degree of difference from the achromatic light-source color of the same brightness). The final component is the *value*, often referred to as “brightness.” It corresponds to the intensity of the light being emitted by the pixel. Mathematically, these three components form a cylindrical coordinate space; the range of colors can be visualized by the hexacone model in figure 2-1.

The HSV color space offers us two distinct advantages over RGB that we can exploit when designing our background subtraction algorithm. First, the HSV color space is nearly *perceptually uniform*. By this, we mean that the proximity of colors in the space is indicative of their similarity. Second, since the color components are mutually orthogonal, a decrease in pixel brightness caused by a shadow should not
affect the values of the hue and saturation. This property is crucial to our ability to detect and avoid shadows.

**Color Conversion Formulas**

Because the video input captured by our CCD camera is in RGB format, we must be able to convert pixels from RGB-space to HSV-space before processing them. Unlike many other pairs of color models, conversion between RGB and HSV is non-linear; however, the transformation algorithm is both computationally simple and reversible. In both color spaces, it is assumed that all of the color components are continuous and lie between 0 and 1, inclusive. The conversion algorithm is presented below[1].

**RGB to HSV conversion.** Let \( \bar{u}(r, g, b) \) be a point in RGB space, \( \bar{w}(h, s, v) \) be its corresponding point in HSV space, and \( T \) be our transformation algorithm such that \( T(\bar{u}) = \bar{w} \). Then for \( r, g, b \in [0, 1] \) \( T \) will produce \( h, s, v \in [0, 1] \) as follows:
where $\Delta = \max(r, g, b) - \min(r, g, b)$.

Note that in the degenerate case where $r, g,$ and $b$ are all equal to 0, then $s$ is undefined. Also, if $r = g = b$, as is the case for purely gray pixels, the hue is undefined as well.

**HSV to RGB conversion.** Using the same notation as before, let $\vec{u}(r, g, b)$ be a point in RGB space, $\vec{w}(h, s, v)$ be its corresponding point in HSV space, and $\mathcal{T}^{-1}$ be our inverse transformation algorithm such that $\mathcal{T}^{-1}(\vec{w}) = \vec{u}$. Then for $h, s, v \in [0, 1]$ $\mathcal{T}$ will produce $r, g, b \in [0, 1]$ as follows:

$$
\begin{align*}
    r &= \begin{cases} 
        v & \text{if } [6h] = 0 \text{ or } [6h] = 5 \\
        v \times (1 - s) & \text{if } [6h] = 2 \text{ or } [6h] = 3 \\
        v \times (1 - s \times \delta) & \text{if } [6h] = 1 \\
        v \times (1 - s \times (1 - \delta)) & \text{if } [6h] = 4 
    \end{cases} \\
    g &= \begin{cases} 
        v & \text{if } [6h] = 1 \text{ or } [6h] = 2 \\
        v \times (1 - s) & \text{if } [6h] = 4 \text{ or } [6h] = 5 \\
        v \times (1 - s \times \delta) & \text{if } [6h] = 3 \\
        v \times (1 - s \times (1 - \delta)) & \text{if } [6h] = 0 
    \end{cases}
\end{align*}
$$

(2.12)  (2.13)
\[
\begin{align*}
    b &= \\
    &\begin{cases}
        v & \text{if } \lfloor 6h \rfloor = 3 \text{ or } \lfloor 6h \rfloor = 4 \\
        v \times (1 - s) & \text{if } \lfloor 6h \rfloor = 0 \text{ or } \lfloor 6h \rfloor = 1 \\
        v \times (1 - s \times \delta) & \text{if } \lfloor 6h \rfloor = 5 \\
        v \times (1 - s \times (1 - \delta)) & \text{if } \lfloor 6h \rfloor = 2
    \end{cases}
\end{align*}
\]

where \( \delta = 6h - \lfloor 6h \rfloor \).

\[2.2.2 \text{ The Problems of RGB to HSV Conversion}\]

Although converting video data from the RGB color space to the HSV color space may seem trivial, it introduces new complications which we must deal with before we can build an effective foreground separation algorithm. The main problem is that our camera does not record data directly as HSV pixels; we must convert from the camera’s format—in our case this is RGB—before we can begin processing. As we shall see, this conversion invalidates some of the assumptions the original GMM algorithm made about the nature of the data. Our system must compensate for these issues before it can be effective.

\[\text{The Nature of Camera Noise in HSV Space}\]

By definition, Gaussian mixture model algorithms assume that the observed pixel values are not perfect; they contain noise introduced by the camera itself during the recording process. Each pixel’s noise is assumed to be independent of every other pixel and approximately Gaussian in distribution. In practice, this approximation is quite reasonable. However, when we transform the pixel values into HSV space, we can no longer assume the per-pixel noise is still normally distributed. In order to gain a clearer understanding of the distribution of the noise in HSV space, we must examine the effect the conversion algorithm has on the color components.

Looking back at Equations 2.9, 2.10, and 2.11, a few issues must be examined. First, assuming \( r, g, \) and \( b \) are normally distributed random variables, we must determine the distribution of \( \max(r, g, b) \) and \( \min(r, g, b) \). Next, we can examine the distribution of the fraction terms in \( h \) and \( s \). Finally, we must understand the piece-
wise nature of the hue definition and how it affects its distribution. We shall now study these questions in greater detail.

**Minimums and Maximums of Multiple Gaussian RV’s.** Let us first look at $\max(r, g, b)$, and try to derive its cumulative distribution function $\Phi_{\max}(x)$. We can then take its derivative to obtain the probability distribution function $\phi_{\max}(x)$. Begin by noting that

$$\Phi_{\max}(z) = P(\max(r, g, b) \leq z) \equiv P(r \leq z) \land P(g \leq z) \land P(b \leq z) \quad (2.15)$$

Since we assume that the noise in each color component is independent, we can reduce this to

$$\Phi_{\max}(z) = P(r \leq z) \times P(g \leq z) \times P(b \leq z) \quad (2.16)$$

Simplifying the problem slightly, we shall assume the distributions of $r, g,$ and $b$ are normally distributed with $\mu = 0$ and $\sigma = 1$. In this case, our color components have their PDF and CDF equal to

$$\phi(x) = \frac{1}{\sqrt{2\pi}} \exp -\frac{x^2}{2} \quad (2.17)$$

$$\Phi(z) = \int_{-\infty}^{z} \phi(x)dx = \frac{1}{2} \left( 1 + \text{erf} \left( \frac{z}{\sqrt{2}} \right) \right) \quad (2.18)$$

where erf is the error function.

Substituting Equation 2.18 into Equation 2.16, we see that

$$\Phi_{\max}(z) = (\Phi(z))^3 = \frac{1}{8} \left( 1 + \text{erf} \left( \frac{z}{\sqrt{2}} \right) \right)^3 \quad (2.19)$$

Differentiation with respect to $z$ gives us the PDF of $\max(r, g, b)$.

$$\phi_{\max}(x) = \frac{d}{dx} \Phi_{\max}(x) = \frac{3}{4\sqrt{2\pi}} e^{-\frac{x^2}{2}} \left( 1 + \text{erf} \left( \frac{x}{\sqrt{2}} \right) \right)^2 \quad (2.20)$$

Upon plotting this PDF, we can clearly see that the distribution of $\max(r, g, b)$ is
Figure 2-2: Plot of the PDF of $\max(r, g, b)$ against a univariate Gaussian PDF

approximately Gaussian. Figure 2-2 shows a plot comparing our derived PDF versus the standard Gaussian distribution with $\mu = 0.82$ and $\sigma = 0.74$.

We can now perform a similar analysis for $\min(r, g, b)$, albeit in a more roundabout fashion. Our CDF is slightly more complicated in this case:

$$
\Phi_{\min}(z) = P(\min(r, g, b) \leq z) \equiv 1 - [(1 - P(r \leq z)) \times (1 - P(g \leq z)) \times (1 - P(b \leq z))]
$$

(2.21)

Following the same steps as before, we arrive at our PDF.

$$
\phi_{\min}(x) = \frac{3}{\sqrt{2\pi}} e^{-x^2/2} \left(1 - \frac{1}{2} \left(1 + \text{erf} \left(\frac{x}{\sqrt{2}}\right)\right)\right)^2
$$

(2.22)

Again, we can visually convince ourselves from Figure 2-3 that this distribution is approximately Gaussian as well.

**Ratios of Gaussian Random Variables.** With a sufficient understanding of the distribution of the minimum and maximum functions, we must now focus our attention on the fractional terms in the conversion formulas. Using the fact that the difference of two Gaussian RV’s is also a Gaussian RV, we can see that each fractional term is a ratio of two Gaussian RV’s. It is a well-known fact that the ratio of two zero-mean, independent Gaussian RV’s follows a Cauchy distribution. However, it is
exceedingly unlikely that we will encounter pixels whose color components have $\mu = 0$. Thus, assuming that pixels follow a Cauchy distribution is not a viable solution.

Marsaglia\cite{8} derived a closed form distribution for the case of non-zero means. For two normal random variables, $x$ and $y$, and two constants, $a$ and $b$, the ratio

$$w = \frac{x + a}{y + b}$$

has the probability distribution function $f(t)$

$$f(t) = \frac{e^{-\frac{1}{2}(a^2 + b^2)}}{\pi (1 + t^2)} \left[ 1 + \frac{q}{\phi(q)} \int_0^q \phi(y)dy \right]$$

$$q = \frac{b + at}{\sqrt{1 + t^2}}$$

(2.23)

where $\phi(x)$ is the standard normal PDF seen in Equation 2.17.

Trying to develop an implementation that models our pixels using this formula would be both computationally difficult and unintuitive. Fortunately, we can make a simplifying assumption. Marsaglia pointed out that when $b$, the mean of the denominator, is sufficiently large, we can simplify the CDF to

$$P \left( \frac{x + a}{y + b} \leq t \right) \approx \int_{-\infty}^{\frac{b - a}{\sqrt{1 + t^2}}} \phi(x)dx$$

(2.24)

Thus, when $b$ is large, the ratio—or at least some function of it—is approximately
normally distributed.

**Piecewise Nature of the Hue.** One final issue we must examine is the piecewise definition of the hue color component and its effect on the probability distribution. Quantitatively, this becomes quite difficult to analyze. We can make an approximation—albeit a simplistic one—that if the minimum and maximum RGB color components are sufficiently close to each other, the hue’s probability density function becomes uniform. In essence, this means that if we are sufficiently close to the grayscale axis, we can just ignore the hue component when trying to match distributions. Detection rates in this case depend on the nature of the foreground objects. If the objects have color values that are sufficiently different in saturation and value than the background, then the accuracy of the algorithm is comparable to the original GMM algorithm. However, if the objects are not sufficiently different with respect to these two color components, then detection will suffer. This issue is discussed in greater detail in Section 4.1.1.

**Our Assumptions About Noise in HSV-space.** Combining our analytical results, we can make the following assumptions about the distribution of the individual color components:

- The value component is approximately normally distributed, regardless of its original RGB values.
- The saturation component is also approximately normally distributed, unless the maximum RGB color component is low (i.e., the pixel is dark).
- The hue component can be approximated as a normal distribution unless the difference between the minimum and maximum RGB color components is small. This occurs in grayscale colors, from black to white, along the vertical axis of the HSV color cone in 2-1.

For most pixel regions, the Gaussian assumption works sufficiently well. In the regions where the approximation fails, however, the algorithm performs additional
processing to compensate. Most of the focus will be directed at dark pixels, since this region creates the most problems. The hue-related problems on the gray line occur much less frequently because the narrow band of colors are less likely to appear in a scene than dark pixels. Even in these cases, the tracker can still function using only the saturation and value components for foreground pixel detection. This issue will be discussed in greater detail in Section 4.1.1.

2.2.3 HSV-Based Background Subtraction Algorithm

The HSV-based algorithm about to be presented is similar to the original GMM algorithm, but with key modifications and the addition of a number of techniques which we will now begin to discuss. Not all of these techniques are necessarily successful; some have negative side-effects which adversely affect the performance of the tracker. The test results for each improvement are presented in Chapter 3, and an in-depth discussion is provided in Chapter 4.

Our discussion will first examine a number of small modifications to the original algorithm necessary for it to function in HSV space. Then we will focus in detail on the key improvements we have researched.

Miscellaneous Modifications

As seen in Equation 2.3, the original GMM algorithm assumed that the distribution of each of the color components had identical variance. Because the color space conversion affects each of the color components’ distributions differently, we should no longer keep this assumption. Instead, we presume that the variances of the color components are independent of each other, but no longer identical. The resultant covariance matrix becomes a diagonal matrix:

\[
\Sigma_{i,t} = \begin{bmatrix}
\sigma_h^2 & 0 & 0 \\
0 & \sigma_s^2 & 0 \\
0 & 0 & \sigma_v^2 \\
\end{bmatrix}
\]  

(2.25)
In practice, the properties of diagonal matrices make calculating inverses and performing multiplication faster than for a full matrix with non-zero covariances. At the same time, we can better model the variances of the individual color components in each distribution than if we had used a scalar value instead.

The next modification is necessary because our color model is now in cylindrical coordinates, not Cartesian. Since the hue component is angular in nature, we must be careful to take the *angular* difference when subtracting two hues. This difference is defined for $h \in [0, 1]$ as

\[
h_1 - h_2 = \begin{cases} 
  h_1 - h_2 + 1, & \text{if } h_1 - h_2 < -\frac{1}{2} \\
  h_1 - h_2 - 1, & \text{if } h_1 - h_2 \geq \frac{1}{2} \\
  h_1 - h_2, & \text{otherwise}
\end{cases}
\]  

(2.26)

Finally, the $\rho$ term in our update equations for $\mu$ and $\sigma$, which was defined in Equation 2.7 to be $\alpha \eta(X_t, \mu_k, \Sigma_k)$, is now simply

\[
\rho = \alpha
\]  

(2.27)

Two factors necessitate this modification. First, the multivariate Gaussian probability density function tends to exhibit rapid falloff in high-dimensional spaces. Combining this with the limited precision of floating-point arithmetic, we notice that when observed pixel values deviate from the mean of their respective matching Gaussians by a fair amount, the matched Gaussian will not update at all because $\rho$ gets rounded down to zero. By removing $\eta(X_t, \mu, \Sigma)$ from our update equations, we are essentially assuming that the distribution within the matched region of each Gaussian is uniform. Testing suggests that this approximation is a reasonable one. There is some performance improvement as well, since we no longer have to compute this probability density for each matched Gaussian in our model.
Shadow Detection

The HSV color model’s primary advantage over the RGB color space is its usefulness in dealing with shadows and lighting changes. Very soft shadows are naturally ignored by the HSV-based algorithm; the perceptual uniformity of the space makes the color components of the shadowed pixels similar enough to their unshadowed counterparts that they tend to fit in each other’s Gaussian models. For harder shadows, however, we need an explicit shadow detection algorithm, which we shall now present.

Just as in the original algorithm, we use an on-line $K$-means approximation. Each new pixel value $X_t$ is checked against the existing $K$ Gaussian distributions—sorted by score—until a match is found. However, our definition of a match is slightly different from our predecessor. Again, we first try to match each color component to within 2.5 standard deviations of its mean. As mentioned earlier, since the hue is angular, we must take the angular difference of the observed and historical values. If every color component matches, we can consider the Gaussian to be a match. If the color components do not lie within $2.5\sigma$, we then try and match only the hue and saturation components, but this time against a much tighter bound of $\sigma$. If these two components match, then we consider the pixel to be a match and assume that a lighting change (shadow or otherwise) has occurred.

This algorithm is simple but effective. It exploits the fact that the color components are mutually orthogonal, so that changes in light intensity do not affect the hue or saturation of the observed value. We could have ignored the value entirely and matched only the hue and saturation to within $2.5\sigma$, but doing so would needlessly discard valuable color information contained therein. Only when strong evidence exists that the pixel has undergone a lighting change do we ignore the value component of the observed pixel. Also note that under general circumstances, when the background model has stabilized and no shadows or sudden lighting changes are present, the algorithm functions identically to the original matching function. Section 3.2.1 describes the performance of this algorithm under various conditions.

As discussed in Section 2.2.2, in dark pixels—where the maximum RGB color
component is small—our assumption of Gaussian noise in the hue and saturation fails. There are two general approaches to solving this problem: we can either modify our algorithm to conform to the peculiarities of the HSV color space, or we can warp the color space itself to make it more amenable to our algorithm. We shall explore both.

**Altering the Color Space**

If we redefine the color space itself, we can make the noise more isotropic. Because of the piecewise nature of the hue conversion formula, it is very difficult to modify it without fundamentally altering the concept of the hue itself. The saturation component, on the other hand, lends itself well to this sort of approach. Recall from Equation 2.10 that the saturation is defined as

\[ s = 1 - \frac{\min(r, g, b)}{\max(r, g, b)} \]

Barring edge cases, the expected range of saturation values, given a maximum camera noise of \( \delta \), is

\[ 1 - \frac{\min(r, g, b) + \delta}{\max(r, g, b) - \delta} \leq s \leq 1 - \frac{\min(r, g, b) - \delta}{\max(r, g, b) + \delta} \]

For dark pixels, where the \( \max(r, g, b) \) term in the denominator is small, the effect of the noise term \( \delta \) will be much more pronounced. To remedy this, we can insert a small constant \( \epsilon \) into both the numerator and denominator, as shown below:

\[ s' = 1 - \frac{\min(r, g, b) + \epsilon}{\max(r, g, b) + \epsilon} \]

(2.28)

As a result, the expected variation in saturation due to noise decreases:

\[ 1 - \frac{\min(r, g, b) + \epsilon + \delta}{\max(r, g, b) + \epsilon - \delta} \leq s \leq 1 - \frac{\min(r, g, b) + \epsilon - \delta}{\max(r, g, b) + \epsilon + \delta} \]

(2.29)

Figure 2.2.3 offers a visual comparison of the expected saturation noise for the original and modified color spaces. For dark pixels, the effect is substantial. In the darkest
20% of the color space, the average expected noise decreases by more than 32%. In the remainder of the space, however, the noise is comparable. Note that in the modified color space, the possible range of saturation values extends from 0 to $\frac{1}{1+\epsilon}$ rather than from 0 to 1.

At this point, it should also be mentioned that for all its advantages, modifying the color space comes at a price. The performance of the shadow detection algorithm presented in Section 2.2.3 performs worse in the HS'V space than in original HSV. Presumably, this is because the modified saturation component of the space is no longer orthogonal to the value component. In this color space, changes in brightness will affect the measurement of the saturation as well, causing our algorithm to be less effective. The performance results of shadow detection in the two color spaces is presented in Section 3.2.1.

As a brief aside, one might have wondered earlier if using noise estimation alone to give darker pixels a wider variance would improve the noise problem. Unfortunately, the answer is no. Even with a wider initial variance, the model would tighten around the mean, since the majority of the pixels do land within $2.5\sigma$. When the variance becomes sufficiently small, infrequent but severe jumps in the hue and saturation components would push the observed values outside this boundary. To make matters worse, because these outlying points do not match the intended Gaussian distribution, it cannot correct itself properly by increasing its variance.
Pixel Smoothing

The other general method of solving the dark pixel problem is to adapt our algorithm to work around the limitations of the color space. This approach is also more compatible with the shadow detection algorithm mentioned earlier. One approach to doing this is *pixel smoothing*. Essentially, we can average some subset of observed pixel values together to obtain more stable, Gaussian-like values. This approach is computationally efficient and works well for background models that remain relatively stationary. If the model is non-stationary, however, the performance of this approach becomes worse.

**Spatial vs. Temporal Smoothing.** We have two general options for pixel smoothing: spatial and temporal. With spatial smoothing, at a given time $t$ we would average an observed pixel value with those of its neighboring pixels. Conversely, we can employ temporal smoothing, in which we maintain a historical window of observed values for each pixel and average the results. The former approach assumes that pixel values exhibit spatial locality. This is generally true for most scenes. However, special care must be taken when smoothing pixels on the boundary of two different pixel regions. With temporal smoothing, no edge detection issues exist. However, temporal smoothing imposes an additional memory cost to maintain the sliding history window for each pixel.

In our implementation, the background subtraction algorithm only utilizes a temporal smoothing algorithm. Spatial averaging produces negligible improvement on our testing platform, as evidenced by the results in Section 3.2.1. We believe that this is a result of the spatial compression algorithm used to compress the video footage. This causes the pixel noise itself to exhibit spatial correlation, thereby rendering any attempt at spatial smoothing pointless.

**Temporal Smoothing Algorithm.** In our algorithm, each pixel maintains a buffer $B$ containing historical observed pixel values. The number of elements in the buffer, $N$, is kept to at most $N_{\text{max}}$; when the buffer is full, values are replaced in the buffer
on a first-in, first-out (FIFO) basis. This allows us to keep only recent values in our history. The smoothed value of the observed pixel at time $t$ is

$$\hat{p}_t = \frac{1}{N} \sum_{n=1}^{N} B_t[n]$$  \hspace{1cm} (2.30)

An observed pixel value $p_t$ is only added to the buffer if both the background model and the observed value are considered dark. This is necessary for two reasons. First, pixel smoothing is only necessary for dark regions; we do not want to apply it arbitrarily to the entire scene. We can assume the pixel is dark if the most likely background Gaussian corresponds to a dark value. Second, if light-colored foreground objects occlude the background, we do not want these observed values to corrupt our buffer history.

In this discussion, we have been using the term “dark pixel” rather loosely; in HSV-space, however, this is quite easy to quantify. We can define a “dark” value as one whose value component $p_{t,v}$ is less than some threshold ($T_v$). Although the threshold is arbitrary, a value of approximately 0.2 seems to work quite well. In RGB-space, the corresponding definition of “dark” would be any pixel whose maximum color component is below some threshold. In practice, it makes no difference whether we smooth the pixel values in RGB-space and then convert to HSV, or convert the values to HSV and then perform the smoothing.

Mathematically, we can summarize our buffer update as follows:

$$\text{insert}(B_t, p_t) \text{ iff } (\mu_{1,t} \leq T_h) \text{ and } (p_{t,v} \leq T_v)$$  \hspace{1cm} (2.31)

The performance of this algorithm is discussed in Section 4.1.4.

**Smoothing Gray Pixels.** As mentioned earlier, the hue component cannot be approximated as a Gaussian for grayscale pixels. One might wonder if the pixel smoothing algorithms discussed in this section can be effectively applied to gray pixels to create a more Gaussian-like distribution. Unfortunately, this is not possible.
The key difference between ordinary\textsuperscript{2} dark and grayscale backgrounds is that for dark pixels, HSV color components deviate from a well-defined average due to RGB noise. With grayscale pixels, the hue component’s natural average is undefined; smoothing these pixels will not be effective.

### 2.3 Improvement 2: Adaptive, Per-Pixel Learning Constants

Recall from our discussion in Section 2.1 that the original Gaussian mixture model algorithm relied on two parameters: the learning constant (\(\alpha\)), which controls how fast new observations get included into the background model, and the background threshold (\(T\)), which sets the minimum proportion of the observed data that should be explained by the model. These terms are constant; every pixel uses the same values for \(\alpha\) and \(T\). Within a scene, however, different pixels experience different activity patterns, and we do not necessarily want the same value of the two constants for all of them. In this section, we will present a method for adapting the learning constant; in Section 2.4, we will examine an adaptive algorithm for the background threshold parameter.

#### 2.3.1 Choosing an Appropriate Value for \(\alpha\)

In pixel regions with high foreground activity, the learning constant \(\alpha\) should be lower so that the background model does not get corrupted as quickly. In low-traffic areas, however, we want a higher value of \(\alpha\) so that the background model adapts more quickly to changes in the background. Before discussing our algorithm for adapting \(\alpha\), we will derive approximate values for how high and how low \(\alpha\) should be in different settings.

We can quantify appropriate values for the learning constant by computing an upper bound on the time it takes a foreground object to become part of the background.

\textsuperscript{2}Since grayscale values can be “dark” as well, we use the term “ordinary” to distinguish between the two.
model. Assume for simplicity that the pixel’s background is unimodal; in the absence of any traffic, there will only be one Gaussian distribution in our background model being matched to observed pixel values. Further assume that this pixel has been devoid of foreground activity for a sufficiently long period of time. In this scenario, the weight of the background Gaussian will be approximately unity:

\[ \omega_{0,t} \approx 1. \quad (2.32) \]

Now suppose that a foreground object occludes this particular pixel’s background. From this frame onward, the weight of the background Gaussian will decrease according to the learning rules specified in Section 2.1:

\[ \omega_{0,t} = \omega_{0,t-1} \times (1 - \alpha). \quad (2.33) \]

Assuming the foreground object arrived at time \( t = 0 \), we can simplify this recursive expression for times \( t = \{1, 2, \ldots\} \) to

\[ \omega_{0,t} = (1 - \alpha)^t. \quad (2.34) \]

The foreground object will become part of the background model when the weight of actual background Gaussian drops below the threshold \( T \):

\[ \omega_{0,t} = (1 - \alpha)^t \leq T \quad (2.35) \]

Taking the logarithm of both sides and rearranging, an upper bound on the number of frames before the background is corrupted is

\[ t \geq \left\lceil \frac{\log(T)}{\log(1 - \alpha)} \right\rceil \quad (2.36) \]

Depending on the initial value for the background Gaussian’s prior weight \( (\omega_{0,0}) \), this number could be much lower.

Before moving on, let us briefly work through an example. Assume we are ob-
serving a road scene near an intersection with a traffic light. When the light is green, automotive traffic moves quickly, but when the light turns red the cars may remain stationary for up to 60 seconds. We would like $\alpha$ to be low enough near the intersection that the stalled cars do not incorporate into the background. If our video data rate is 15 frames per second and our background threshold is 75%, then we would want a value of $\alpha$ of at least

$$15 \times 60 = \left\lceil \log(0.75) \over \log(1 - \alpha) \right\rceil$$

$$\alpha = 1 - \exp \left( \log(0.75) \over 15 \times 60 \right) \approx 0.0003$$

In practice, it would be advantageous to choose a value of $\alpha$ some factor lower, both to ensure that the background does not become corrupted and to keep the prior weights of the Gaussians from drifting too much.

### 2.3.2 Previous Work

Harville et al.[4] proposed a method of updating the learning constant through the use of an *activity index*, $A_{x,y,t}$. The index is initially set at 0; on each frame, the index is updated according to the following equation:

$$A_{x,y,t} = (1 - \lambda)A_{x,y,t-1} + \lambda|Y_{x,y,t} - Y_{x,y,t-1}|$$ \hspace{1cm} (2.37)

$Y$ is the luminance component in the YUV color model their algorithm uses. For each frame that $A_{x,y,t}$ exceeds some threshold $H$, $\alpha_{x,y}$ is reduced by some factor $\xi$. In practice, $\xi$ has a non-zero lower bound so that the pixel can still update, albeit very slowly.

This algorithm is effective, especially with regard to noise insensitivity. However, it has two key deficiencies. First, the heuristic for updating the activity index is based on the time difference of intensity values. When foreground objects stop moving (the “sleeping person” problem mentioned in 1.2.1), the activity index will not increase.
This problem also occurs for slow-moving, homogeneously colored objects, although this is a less likely occurrence. Second, the algorithm is one-way; \( \alpha \) can only decrease. As activity patterns within the scene change over the course of time, the learning constant cannot be increased to compensate. Also, the one-way nature causes the prior weights of the Gaussians to recover more slowly than normal after periods of heavy foreground activity.

### 2.3.3 Proposed Solution

We propose a different algorithm, which improves upon both deficiencies while still maintaining the benefit of noise insensitivity. In this method, each pixel maintains a historical window \( F \) of its last \( N \) states. In state \( j \), \( F[j] \) takes the value 1 if the pixel at time \((t - N + j + 1)\) was labeled as foreground or 0 if it was labeled as background. The value of \( \alpha \) at time \( t \) is

\[
\alpha = S \left( \sum_{i=1}^{N} F[i] \right)
\]  

(2.38)

where the transfer function \( S(n) \) is essentially an inverted sigmoid, appropriately scaled and translated:

\[
S(n) = \alpha_{\max} - \frac{\alpha_{\max} - \alpha_{\min}}{1 + e^{-(n-M)}}
\]  

(2.39)

The transfer function is bounded between \( \alpha_{\min} \) and \( \alpha_{\max} \); its inflection point occurs at \( n = M \). A plot of \( S(n) \) can be seen in Figure 2-5.

A sigmoid-like function was chosen as the transfer curve primarily for the effect of noise insensitivity. If the overall foreground activity at a particular pixel is very low, then occasional noise (in the form of erroneous detection as a foreground pixel) will have little effect on \( \alpha \). Only when sufficient activity has occurred does the value of \( \alpha \) begin to respond. Using this type of transfer curve essentially limits behavior to the two extreme values, as it is unlikely for \( \alpha \) to stay near the inflection point (where \( n = M \)). For most scenarios, this is ideal. Generally, we would like to maintain \( \alpha \) at a certain value for normal tracking. Only when the activity level becomes too high do we wish to change it to a much lower value. When activity levels return to normal, \( \alpha \) reverts to its original value.
One last issue regarding this adaptive-α algorithm regards stability. Since we are using the foreground pixel history to update α, which in turn affects the determination of foreground pixels, there is a potential for unexpected effects. In actuality, the only side-effect is that the background model takes longer to stabilize. This problem is mostly limited to the first few frames, when erroneous foreground detection is at its peak. To solve this problem, we simply delay activation of the algorithm until the frame count has exceeded some number of frames.

Section 3.2.2 compares the performance of this algorithm against both a constant α and Harville's algorithm.

2.4 Improvement 3: Adaptive, Per-Pixel Background Thresholds

This section presents an algorithm for a per-pixel background threshold parameter $T$, based on the pixel's activity pattern.
2.4.1 Choosing an Appropriate Value for $T$

Unlike the learning constant, there is no quantitative method of estimating appropriate values for the background threshold $T$. Ideally, we would want the threshold to be higher when the background is multimodal, so that repetitive background elements are not detected as foreground. When the background is primarily unimodal, however, a lower threshold would be more appropriate so that heavy foreground traffic is less likely to be included into our background model.

This logic might inspire us to create a formula based on the prior weights of the Gaussians in the mixture model. For example, in unimodal scenes the drop-off between the primary Gaussian and the remainders would be very high, whereas for multimodal backgrounds the ratio between the highest few Gaussians (the ones comprising the background) would be much lower. Unfortunately, such an approach can be problematic in certain situations. For example, in a scene with a region containing waving trees, where a pixel might take on the color of a branch 70% of the time and the sky 30% of the time, a high background threshold would be appropriate. However, in a different region, we might have a traffic intersection with a stopped car that has remained stationary for long enough that its Gaussian model now has a prior weight of 30%. In this case, we would want a lower background threshold to prevent the car from being absorbed into the background.

Rather than using the ratio of Gaussians as our heuristic for determining $T$, we might consider using the rate of change at which the Gaussians in our mixture are being matched. Consider again the two scenarios mentioned above. For pixels in the region with waving branches, the index of the matched Gaussian would exhibit a high rate of change, oscillating frequently between the distribution of the branch and that of the sky. With the idle car, however, the average rate of change would be very low, changing once only when the car first occludes the background. Although this approach seems promising, it too can fail. Consider the similarities between a car stopped at a traffic light and the light on the traffic signal itself. As we just mentioned, the pixels corresponding to the car would exhibit low oscillation between
matched Gaussians—it changes only once when the car occludes the background—and a low ratio between the weights of its highest Gaussians, assuming the car has been idle for long enough. On the other hand, the pixel values on the red traffic light are bright red when the light is illuminated, and a very dark red when the yellow or green lights are illuminated. These pixels can be characterized as having a model consisting of two large Gaussians of approximately equal weight, and thus a low ratio between the weights of its highest two Gaussians. The rate of change between matched Gaussians is low since the light oscillates very slowly—it may switch colors once every 30 seconds. Based on the ratios of prior weights and the rate of change between matched models, it is impossible to discern the difference between a sleeping object and a slowly changing multimodal background.

2.4.2 Proposed Solution

Although each of the two approaches we have just discussed fails in certain situations, we can combine them to reliably make some conclusions as to when to increase or decrease our background threshold parameter. For example, if the rate of change in the index of the matched model is high and the ratio of the prior weights of the highest Gaussians is low, we can safely assume that the observed pixel values represent a multimodal background with low foreground traffic. In this case, a high threshold would be appropriate. If both the oscillation and the ratio are high, then we can assume a unimodal scene with heavy foreground activity. Here, a lower threshold would be appropriate so that the background model is less likely to be corrupted. When the ratio is high and the oscillation is low, the activity pattern is most likely that of a unimodal background with light foreground traffic. Here, we can keep our normal threshold. In the last case, when both the ratio and the rate of change is low, we cannot make any reliable assumptions about the activity pattern of the pixel; we should not attempt to modify the value of $T$. These results are summarized by the decision matrix in Figure 2-6.

We shall now propose an algorithm for adapting the background threshold parameter based on this decision model. As we shall see, the algorithm is very similar
Figure 2-6: Decision matrix for adjusting background threshold to the adaptive learning constant algorithm described in Section 2.3.

We shall maintain a historical window $M$ corresponding to its last $N$ states. The value of the buffer at index $j$ is

$$M[j] = \begin{cases} 
1, & \text{if } i_{t-N+j+1} \neq i_{t-N+j} \\
0, & \text{if } i_{t-N+j+1} = i_{t-N+j}
\end{cases}$$

(2.40)

where $i_t$ is the index of the Gaussian distribution which matched the observed pixel value at time $t$. Essentially, $M[j]$ equals 1 when the observed pixel value at that time matches a different Gaussian distribution than the previously observed value did; otherwise, it equals 0. We can calculate the average magnitude of the rate of change between matched models as

$$\tilde{m}_t = \frac{1}{N} \int_{t-N+1}^{t} \left| \frac{di_u}{du} \right| du = \frac{1}{N} \sum_{j=0}^{N-1} M[j]$$

(2.41)

We can then label the rate of change as “low” if $\tilde{m}$ less than some constant $K_m$ and “high” otherwise.
Next, we compute the ratio of the prior weights for the primary two Gaussians, which we shall call $r$:

$$r_t = \frac{\omega_{0,t}}{\omega_{1,t}}$$  \hspace{1cm} (2.42)

Again, we can label the ratio as "low" if $r_t$ is less than some constant $K_r$ and "high" otherwise.

At each frame time $t$, we use a simple algorithm to update the value of $T$:

$$T_t = T_{t-1} + \frac{T_t - T_{t-1}}{2}$$  \hspace{1cm} (2.43)

where

$$\tilde{T}_t = \begin{cases} 
T_{t-1}, & \text{if } (r_t < K_r) \text{ and } (\hat{m}_t < K_t) \\
T_H, & \text{if } (r_t < K_r) \text{ and } (\hat{m}_t \geq K_t) \\
T_N, & \text{if } (r_t \geq K_r) \text{ and } (\hat{m}_t < K_t) \\
T_L, & \text{if } (r_t \geq K_r) \text{ and } (\hat{m}_t \geq K_t)
\end{cases}$$  \hspace{1cm} (2.44)

where $T_H$, $T_N$, and $T_L$ are constants corresponding to the highest, normal, and lowest threshold values, respectively. The rate at which the threshold approaches its intended value is geometric, as seen in Figure 2-7. In theory, $T$ takes an infinite time to reach any of these values, but due to limited floating-point accuracy, this time is finite. The performance results of this algorithm will be presented in Section 3.2.3.
2.5 Research Summary

Before concluding this chapter, we shall briefly summarize the scope of the research presented here. The first improvement discussed modifying the algorithm to use the HSV color space. Many of the assumptions the original algorithm made about the nature of noise in the RGB color space are no longer valid in HSV due to the color conversion process, and we must compensate accordingly. We looked at modifying the color space to make the noise more isotropic, as well as smoothing observed values in dark pixel regions to reduce noise. We also discussed a shadow detection algorithm which can be used in this color space.

This section also proposed new algorithms for per-pixel, adaptive learning constants and background thresholds. In the case of the former, we also discussed previous work and contrasted it to the algorithms presented here.
Chapter 3

Evaluation of Improvements

This chapter presents numerical results describing the performance of the various improvements described in this thesis. This chapter only gives a brief description of each experiment and its results. A more detailed discussion of the results is available in Chapter 4.

3.1 Testing Methodology

The performance of each of the algorithms examined in this paper was tested by running each of them on a suite of video clips which attempt to capture some of the canonical problems of object tracking listed in Section 1.2.1. All of the video footage was captured using a Sony DCR-VX2000 digital video camera and recorded to the Audio-Video Interleave (AVI) video format. Each video sequence was compressed with the Intel Indeo Video 5.10 compression algorithm to meet file space limitations. The video sequences are 320 pixels in width by 240 pixels in height, and have a frame rate of 15. The bitmaps were then parsed and converted into the log output format used by the tracking application. The various algorithms were then run on the video clips, and each output file was compared against the hand-analyzed reference files.
**Accuracy Limitations.** When reading the results in this section, one must keep in mind that there are inherent limitations to the accuracy of the data. Since the video footage is first analyzed by hand to identify the reference set of foreground pixels and shadows, human error may affect the selection of pixels and reduce the accuracy of the reference data. Even without the presence of error, it is very difficult to identify the exact boundaries of shadows and foreground objects when the edges are “soft.” Fortunately, these errors are generally limited in number compared to the overall number of pixels being processed. Also, since all the algorithms are being compared to the same human-generated result set, errors will affect each of them equally.

### 3.2 Results

This section presents an explanation of each of the test experiments performed in this research project, as well as a listing of the results. A detailed discussion and interpretation of these results can be found in Chapter 4.

Before presenting the results, it would be helpful to review the terminology used in the tables.

**Foreground pixel detection rate.** The percentage of pixels marked as “foreground” in the hand-selected, reference dataset that were correctly identified as foreground.

**Type I error.** Commonly referred to as a “false positive,” a Type I error occurs when a background pixel is incorrectly marked as “foreground.”

**Type II error.** Also known as a “missed negative,” a Type II error occurs when a foreground pixel is incorrectly marked as “background.” Generally, these are far less common than Type I errors.

**Shadow pixel detection rate.** For shadows, this corresponds to the percentage of pixels marked as “shadows” in the reference dataset that were correctly identified as shadows by the tracking algorithm.
Shadow pixel avoidance rate. The percentage of pixels labeled as “shadows” in the reference dataset that were marked as “background” by the tracking algorithm. For softer shadows, many pixels are avoided rather than detected because they are sufficiently close to the mean of one of the background Gaussians.

Shadow pixel miss rate. The percentage of pixels labeled as “shadows” in the reference dataset that were marked as “foreground” by the tracking algorithm. These represent the worst-case scenario, where shadowed pixels are neither successfully detected nor avoided.

Bearing these definitions in mind, we may now examine the experiments and their results.

3.2.1 HSV-Based Algorithms

We will now examine the performance of the HSV color space-based tracking algorithms. First, we shall examine the general performance of the algorithms compared to the original GMM algorithm, in order to assess any potential limitations. Then, we shall examine, in detail, the shadow detection and avoidance capabilities of the new algorithms. In Section 4.1, we will discuss potential trade-offs between tracking performance and shadow detection.

General Performance

The first set of tests examines the performance of the algorithms under “normal” conditions. By this, we mean situations devoid of hard shadows, lighting changes, stalled objects, or unusually heavy foreground activity. This gives us an opportunity to assess the general performance of the improved tracking algorithms versus the original. The footage used for these tests was selected to replicate the most common setting for object tracking software: outdoor surveillance. The scene itself is an overhead view of a street adjacent to the MIT Artificial Intelligence Laboratory, filmed under slightly overcast conditions to minimize the presence of shadows. Segments of
<table>
<thead>
<tr>
<th>Test</th>
<th>Algorithm</th>
<th>Color Space</th>
<th>Pixel Smoothing</th>
<th>Object Color</th>
<th>Foreground Pixels</th>
<th>Shadows</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Detection Rate</td>
<td>Type I</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Avoidance Rate</td>
<td>Rate</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Miss Rate</td>
<td>Rate</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>False Positives</td>
<td>Rate</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>per Frame</td>
<td>Rate</td>
</tr>
<tr>
<td>1</td>
<td>Original</td>
<td>RGB</td>
<td>N/A</td>
<td>Gray</td>
<td>84.46%</td>
<td>243.54</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.00%</td>
<td>18.17</td>
</tr>
<tr>
<td>2</td>
<td>Original</td>
<td>RGB</td>
<td>N/A</td>
<td>Non-Gray</td>
<td>98.23%</td>
<td>249.91</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.00%</td>
<td>23.34</td>
</tr>
<tr>
<td>3</td>
<td>Improved</td>
<td>HSV</td>
<td>None</td>
<td>Gray</td>
<td>43.25%</td>
<td>179.27</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.77%</td>
<td>69.30</td>
</tr>
<tr>
<td>4</td>
<td>Improved</td>
<td>HSV</td>
<td>None</td>
<td>Non-Gray</td>
<td>85.96%</td>
<td>192.91</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.22%</td>
<td>77.16</td>
</tr>
<tr>
<td>5</td>
<td>Improved</td>
<td>HSV</td>
<td>Spatial</td>
<td>Gray</td>
<td>43.25%</td>
<td>181.82</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.77%</td>
<td>69.30</td>
</tr>
<tr>
<td>6</td>
<td>Improved</td>
<td>HSV</td>
<td>Spatial</td>
<td>Non-Gray</td>
<td>85.96%</td>
<td>171.73</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.22%</td>
<td>77.16</td>
</tr>
<tr>
<td>7</td>
<td>Improved</td>
<td>HSV</td>
<td>Temporal</td>
<td>Gray</td>
<td>43.25%</td>
<td>123.27</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.77%</td>
<td>69.30</td>
</tr>
<tr>
<td>8</td>
<td>Improved</td>
<td>HSV</td>
<td>Temporal</td>
<td>Non-Gray</td>
<td>85.96%</td>
<td>125.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.22%</td>
<td>77.16</td>
</tr>
<tr>
<td>9</td>
<td>Improved</td>
<td>HSV</td>
<td>None</td>
<td>Gray</td>
<td>64.48%</td>
<td>174.16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.46%</td>
<td>51.28</td>
</tr>
<tr>
<td>10</td>
<td>Improved</td>
<td>HSV</td>
<td>None</td>
<td>Non-Gray</td>
<td>91.13%</td>
<td>165.09</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.51%</td>
<td>74.05</td>
</tr>
<tr>
<td>11</td>
<td>Improved</td>
<td>HSV</td>
<td>Spatial</td>
<td>Gray</td>
<td>64.48%</td>
<td>178.27</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.46%</td>
<td>51.28</td>
</tr>
<tr>
<td>12</td>
<td>Improved</td>
<td>HSV</td>
<td>Spatial</td>
<td>Non-Gray</td>
<td>91.13%</td>
<td>162.18</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.51%</td>
<td>74.05</td>
</tr>
<tr>
<td>13</td>
<td>Improved</td>
<td>HSV</td>
<td>Temporal</td>
<td>Gray</td>
<td>64.48%</td>
<td>144.91</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.46%</td>
<td>51.28</td>
</tr>
<tr>
<td>14</td>
<td>Improved</td>
<td>HSV</td>
<td>Temporal</td>
<td>Non-Gray</td>
<td>91.13%</td>
<td>126.73</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.51%</td>
<td>74.05</td>
</tr>
</tbody>
</table>

Table 3.1: Comparison of general outdoor tracking performance for gray vs. non-gray objects

The surveillance footage which meet the above criteria were selected and analyzed. The results of the tests are listed in Table 3.1.

**Shadow Detection**

To test the shadow detection and avoidance capabilities of the HSV-based algorithm, we compared its performance under different lighting conditions and backgrounds. A camera was positioned in front of uniform backgrounds of various colors. First, an initialization window containing only this background was captured, in order to allow the model to converge to its correct state. Then, a foreground object easily distinguishable from the background was passed through the scene. Two video sequences were taken for each background: one under fluorescent lighting typical of what one would find indoors, and another with the same lighting plus a halogen flood light directed at the background. The normal lighting footage serves as a control, testing the accuracy of the tracker when no shadows, or only very soft shadows, are present. Conversely, the footage taken when the flood light is illuminated contains...
dark, well-defined shadows on the background; this tests the shadow detection and avoidance capabilities of the tracker under more difficult lighting conditions.

These tests represent opposite ends of the spectrum of potential shadowing conditions. To test the algorithm’s performance under more normal situation, we analyzed an additional video segment that was quite similar to the others, depicting an easily discernible foreground object passing in front of a bright, uniform background (in this case cyan) after a sufficient initialization period. However, the scene lighting was altered, this time bright enough to create visible shadow areas on the background, but not nearly as hard or crisp as those created by the flood light.

Each of the aforementioned clips was processed by the original Gaussian mixture model described in Section 2.1, as well as the HSV-algorithm with different improvements enabled and disabled. Table 3.2 summarizes the video footage used. The results, displayed in Tables 3.3 – 3.7, provide statistics on the performance of the algorithm for various combinations of algorithms and footage.

### 3.2.2 Adaptive Learning Constants

To understand the performance of the proposed adaptive learning constant algorithm, we recorded footage of cars stalled at an intersection. A sample frame from this video is seen in Figure 3-1. Each of the cars remained motionless for a different length of time, depending on when it arrived at the stoplight; this ranged from as little as 5 to
### Black Background Tests

<table>
<thead>
<tr>
<th>Test</th>
<th>Algorithm</th>
<th>Color Space</th>
<th>Pixel Smoothing</th>
<th>Shadows</th>
<th>Foreground Pixels</th>
<th>Shadows</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Original</td>
<td>RGB</td>
<td>N/A</td>
<td>Soft</td>
<td>100.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>2</td>
<td>Original</td>
<td>RGB</td>
<td>N/A</td>
<td>Hard</td>
<td>100.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>3</td>
<td>Improved</td>
<td>HSV</td>
<td>None</td>
<td>Soft</td>
<td>94.06%</td>
<td>8.48%</td>
</tr>
<tr>
<td>4</td>
<td>Improved</td>
<td>HSV</td>
<td>None</td>
<td>Hard</td>
<td>95.91%</td>
<td>0.23%</td>
</tr>
<tr>
<td>5</td>
<td>Improved</td>
<td>HSV</td>
<td>Spatial</td>
<td>Soft</td>
<td>95.08%</td>
<td>0.04%</td>
</tr>
<tr>
<td>6</td>
<td>Improved</td>
<td>HSV</td>
<td>Spatial</td>
<td>Hard</td>
<td>95.91%</td>
<td>0.00%</td>
</tr>
<tr>
<td>7</td>
<td>Improved</td>
<td>HSV</td>
<td>Temporal</td>
<td>Soft</td>
<td>96.88%</td>
<td>0.46%</td>
</tr>
<tr>
<td>8</td>
<td>Improved</td>
<td>HSV</td>
<td>Temporal</td>
<td>Hard</td>
<td>96.10%</td>
<td>0.19%</td>
</tr>
<tr>
<td>9</td>
<td>Improved</td>
<td>HSV</td>
<td>None</td>
<td>Soft</td>
<td>91.74%</td>
<td>22.41%</td>
</tr>
<tr>
<td>10</td>
<td>Improved</td>
<td>HSV</td>
<td>None</td>
<td>Hard</td>
<td>99.01%</td>
<td>5.39%</td>
</tr>
<tr>
<td>11</td>
<td>Improved</td>
<td>HSV</td>
<td>Spatial</td>
<td>Soft</td>
<td>91.74%</td>
<td>22.41%</td>
</tr>
<tr>
<td>12</td>
<td>Improved</td>
<td>HSV</td>
<td>Spatial</td>
<td>Hard</td>
<td>99.90%</td>
<td>0.85%</td>
</tr>
<tr>
<td>13</td>
<td>Improved</td>
<td>HSV</td>
<td>Temporal</td>
<td>Soft</td>
<td>95.30%</td>
<td>4.00%</td>
</tr>
<tr>
<td>14</td>
<td>Improved</td>
<td>HSV</td>
<td>Temporal</td>
<td>Hard</td>
<td>95.50%</td>
<td>1.72%</td>
</tr>
</tbody>
</table>

*For black backgrounds, it was impossible to accurately discern shadowed regions, even in bright lighting. Thus, all shadow data will be ignored.

Table 3.3: Detection rates for uniform black background

### Gray Background Tests

<table>
<thead>
<tr>
<th>Test</th>
<th>Algorithm</th>
<th>Color Space</th>
<th>Pixel Smoothing</th>
<th>Shadows</th>
<th>Foreground Pixels</th>
<th>Shadows</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Original</td>
<td>RGB</td>
<td>N/A</td>
<td>Soft</td>
<td>98.83%</td>
<td>10.25%</td>
</tr>
<tr>
<td>2</td>
<td>Original</td>
<td>RGB</td>
<td>N/A</td>
<td>Hard</td>
<td>99.99%</td>
<td>0.00%</td>
</tr>
<tr>
<td>3</td>
<td>Improved</td>
<td>HSV</td>
<td>None</td>
<td>Soft</td>
<td>97.38%</td>
<td>19.32%</td>
</tr>
<tr>
<td>4</td>
<td>Improved</td>
<td>HSV</td>
<td>None</td>
<td>Hard</td>
<td>99.34%</td>
<td>0.39%</td>
</tr>
<tr>
<td>5</td>
<td>Improved</td>
<td>HSV</td>
<td>Spatial</td>
<td>Soft</td>
<td>97.18%</td>
<td>20.18%</td>
</tr>
<tr>
<td>6</td>
<td>Improved</td>
<td>HSV</td>
<td>Spatial</td>
<td>Hard</td>
<td>99.34%</td>
<td>0.39%</td>
</tr>
<tr>
<td>7</td>
<td>Improved</td>
<td>HSV</td>
<td>Temporal</td>
<td>Soft</td>
<td>97.60%</td>
<td>17.98%</td>
</tr>
<tr>
<td>8</td>
<td>Improved</td>
<td>HSV</td>
<td>Temporal</td>
<td>Hard</td>
<td>99.34%</td>
<td>0.39%</td>
</tr>
<tr>
<td>9</td>
<td>Improved</td>
<td>HSV</td>
<td>None</td>
<td>Soft</td>
<td>83.01%</td>
<td>124.82%</td>
</tr>
<tr>
<td>10</td>
<td>Improved</td>
<td>HSV</td>
<td>None</td>
<td>Hard</td>
<td>97.34%</td>
<td>0.67%</td>
</tr>
<tr>
<td>11</td>
<td>Improved</td>
<td>HSV</td>
<td>Spatial</td>
<td>Soft</td>
<td>82.91%</td>
<td>41.06%</td>
</tr>
<tr>
<td>12</td>
<td>Improved</td>
<td>HSV</td>
<td>Spatial</td>
<td>Hard</td>
<td>97.34%</td>
<td>0.67%</td>
</tr>
<tr>
<td>13</td>
<td>Improved</td>
<td>HSV</td>
<td>Temporal</td>
<td>Soft</td>
<td>83.88%</td>
<td>39.47%</td>
</tr>
<tr>
<td>14</td>
<td>Improved</td>
<td>HSV</td>
<td>Temporal</td>
<td>Hard</td>
<td>97.34%</td>
<td>0.67%</td>
</tr>
</tbody>
</table>

Table 3.4: Detection rates for uniform gray background
### Red Background Tests

<table>
<thead>
<tr>
<th>Test</th>
<th>Algorithm</th>
<th>Color Space</th>
<th>Pixel Smoothing</th>
<th>Shadows</th>
<th>Detection Rate</th>
<th>Type I</th>
<th>Type II</th>
<th>Avoidance Rate</th>
<th>Miss Rate</th>
<th>False Positives per Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Original</td>
<td>RGB</td>
<td>N/A</td>
<td>Soft</td>
<td>99.99%</td>
<td>694.77</td>
<td>0.06</td>
<td>N/A</td>
<td>N/A</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>Original</td>
<td>RGB</td>
<td>N/A</td>
<td>Hard</td>
<td>99.99%</td>
<td>947.04</td>
<td>0.03</td>
<td>0.00%</td>
<td>13.09%</td>
<td>89.91%</td>
</tr>
<tr>
<td>3</td>
<td>Improved</td>
<td>HSV</td>
<td>None</td>
<td>Soft</td>
<td>99.98%</td>
<td>338.44</td>
<td>0.14</td>
<td>N/A</td>
<td>N/A</td>
<td>2.42</td>
</tr>
<tr>
<td>4</td>
<td>Improved</td>
<td>HSV</td>
<td>None</td>
<td>Hard</td>
<td>99.94%</td>
<td>416.67</td>
<td>0.40</td>
<td>13.85%</td>
<td>24.42%</td>
<td>61.73%</td>
</tr>
<tr>
<td>5</td>
<td>Improved</td>
<td>HSV</td>
<td>Spatial</td>
<td>Soft</td>
<td>99.98%</td>
<td>338.44</td>
<td>0.14</td>
<td>N/A</td>
<td>N/A</td>
<td>2.42</td>
</tr>
<tr>
<td>6</td>
<td>Improved</td>
<td>HSV</td>
<td>Spatial</td>
<td>Hard</td>
<td>99.94%</td>
<td>416.67</td>
<td>0.40</td>
<td>13.85%</td>
<td>24.42%</td>
<td>61.73%</td>
</tr>
<tr>
<td>7</td>
<td>Improved</td>
<td>HSV</td>
<td>Temporal</td>
<td>Soft</td>
<td>99.98%</td>
<td>338.44</td>
<td>0.14</td>
<td>N/A</td>
<td>N/A</td>
<td>2.42</td>
</tr>
<tr>
<td>8</td>
<td>Improved</td>
<td>HSV</td>
<td>Temporal</td>
<td>Hard</td>
<td>99.94%</td>
<td>416.67</td>
<td>0.40</td>
<td>13.85%</td>
<td>24.42%</td>
<td>61.73%</td>
</tr>
<tr>
<td>9</td>
<td>Improved</td>
<td>HSV</td>
<td>None</td>
<td>Soft</td>
<td>99.96%</td>
<td>382.01</td>
<td>0.26</td>
<td>14.71%</td>
<td>27.22%</td>
<td>58.07%</td>
</tr>
<tr>
<td>10</td>
<td>Improved</td>
<td>HSV</td>
<td>None</td>
<td>Hard</td>
<td>99.96%</td>
<td>382.01</td>
<td>0.26</td>
<td>14.71%</td>
<td>27.22%</td>
<td>58.07%</td>
</tr>
<tr>
<td>11</td>
<td>Improved</td>
<td>HSV</td>
<td>Spatial</td>
<td>Soft</td>
<td>99.96%</td>
<td>382.01</td>
<td>0.26</td>
<td>14.71%</td>
<td>27.22%</td>
<td>58.07%</td>
</tr>
<tr>
<td>12</td>
<td>Improved</td>
<td>HSV</td>
<td>Spatial</td>
<td>Hard</td>
<td>99.96%</td>
<td>382.01</td>
<td>0.26</td>
<td>14.71%</td>
<td>27.22%</td>
<td>58.07%</td>
</tr>
<tr>
<td>13</td>
<td>Improved</td>
<td>HSV</td>
<td>Temporal</td>
<td>Soft</td>
<td>99.96%</td>
<td>382.01</td>
<td>0.26</td>
<td>14.71%</td>
<td>27.22%</td>
<td>58.07%</td>
</tr>
<tr>
<td>14</td>
<td>Improved</td>
<td>HSV</td>
<td>Temporal</td>
<td>Hard</td>
<td>99.96%</td>
<td>382.01</td>
<td>0.26</td>
<td>14.71%</td>
<td>27.22%</td>
<td>58.07%</td>
</tr>
</tbody>
</table>

Table 3.5: Detection rates for uniform red background

### Teal Background Tests

<table>
<thead>
<tr>
<th>Test</th>
<th>Algorithm</th>
<th>Color Space</th>
<th>Pixel Smoothing</th>
<th>Shadows</th>
<th>Detection Rate</th>
<th>Type I</th>
<th>Type II</th>
<th>Avoidance Rate</th>
<th>Miss Rate</th>
<th>False Positives per Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Original</td>
<td>RGB</td>
<td>N/A</td>
<td>Soft</td>
<td>99.98%</td>
<td>1,357.69</td>
<td>0.07</td>
<td>N/A</td>
<td>N/A</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>Original</td>
<td>RGB</td>
<td>N/A</td>
<td>Hard</td>
<td>100.00%</td>
<td>934.85</td>
<td>0.00</td>
<td>0.00%</td>
<td>0.71%</td>
<td>99.23%</td>
</tr>
<tr>
<td>3</td>
<td>Improved</td>
<td>HSV</td>
<td>None</td>
<td>Soft</td>
<td>100.00%</td>
<td>488.73</td>
<td>0.01</td>
<td>N/A</td>
<td>N/A</td>
<td>0.24</td>
</tr>
<tr>
<td>4</td>
<td>Improved</td>
<td>HSV</td>
<td>None</td>
<td>Hard</td>
<td>100.00%</td>
<td>422.05</td>
<td>0.00</td>
<td>21.12%</td>
<td>3.20%</td>
<td>75.68%</td>
</tr>
<tr>
<td>5</td>
<td>Improved</td>
<td>HSV</td>
<td>Spatial</td>
<td>Soft</td>
<td>100.00%</td>
<td>488.73</td>
<td>0.01</td>
<td>N/A</td>
<td>N/A</td>
<td>0.24</td>
</tr>
<tr>
<td>6</td>
<td>Improved</td>
<td>HSV</td>
<td>Spatial</td>
<td>Hard</td>
<td>100.00%</td>
<td>422.05</td>
<td>0.00</td>
<td>21.12%</td>
<td>3.20%</td>
<td>75.68%</td>
</tr>
<tr>
<td>7</td>
<td>Improved</td>
<td>HSV</td>
<td>Temporal</td>
<td>Soft</td>
<td>100.00%</td>
<td>488.73</td>
<td>0.01</td>
<td>N/A</td>
<td>N/A</td>
<td>0.24</td>
</tr>
<tr>
<td>8</td>
<td>Improved</td>
<td>HSV</td>
<td>Temporal</td>
<td>Hard</td>
<td>100.00%</td>
<td>422.05</td>
<td>0.00</td>
<td>21.12%</td>
<td>3.20%</td>
<td>75.68%</td>
</tr>
<tr>
<td>9</td>
<td>Improved</td>
<td>HSV</td>
<td>None</td>
<td>Soft</td>
<td>99.99%</td>
<td>379.36</td>
<td>0.03</td>
<td>N/A</td>
<td>N/A</td>
<td>0.34</td>
</tr>
<tr>
<td>10</td>
<td>Improved</td>
<td>HSV</td>
<td>None</td>
<td>Hard</td>
<td>100.00%</td>
<td>390.40</td>
<td>0.00</td>
<td>12.46%</td>
<td>2.74%</td>
<td>84.80%</td>
</tr>
<tr>
<td>11</td>
<td>Improved</td>
<td>HSV</td>
<td>Spatial</td>
<td>Soft</td>
<td>99.99%</td>
<td>379.36</td>
<td>0.03</td>
<td>N/A</td>
<td>N/A</td>
<td>0.34</td>
</tr>
<tr>
<td>12</td>
<td>Improved</td>
<td>HSV</td>
<td>Spatial</td>
<td>Hard</td>
<td>100.00%</td>
<td>390.40</td>
<td>0.00</td>
<td>12.46%</td>
<td>2.74%</td>
<td>84.80%</td>
</tr>
<tr>
<td>13</td>
<td>Improved</td>
<td>HSV</td>
<td>Temporal</td>
<td>Soft</td>
<td>99.99%</td>
<td>379.36</td>
<td>0.03</td>
<td>N/A</td>
<td>N/A</td>
<td>0.34</td>
</tr>
<tr>
<td>14</td>
<td>Improved</td>
<td>HSV</td>
<td>Temporal</td>
<td>Hard</td>
<td>100.00%</td>
<td>390.40</td>
<td>0.00</td>
<td>12.46%</td>
<td>2.74%</td>
<td>84.80%</td>
</tr>
</tbody>
</table>

Table 3.6: Detection rates for uniform teal background
Table 3.7: Detection rates for uniform cyan background with soft, visible shadows as long as 60 seconds. The video was then analyzed with four separate settings for the learning constant: a constant value of $\alpha$ suitable for normal conditions, another much lower value of $\alpha$, an adaptive $\alpha$ based on the Harville algorithm, and finally the adaptive algorithm presented in this paper. The range of possible output values for the two adaptive algorithms was bounded between the normal and low values previously mentioned. Detection and error rates were examined for a five-second window beginning when the last car in the line became motionless. The results of this analysis are presented in Table 3.8.

Figure 3-1: Sample frame of stalled cars at an intersection

3.2.3 Adaptive Background Threshold

In order to test the effectiveness of the adaptive thresholding algorithm presented in Section 2.4, we shall examine its performance on video footage fitting the categories
### Stalled Car Detection Rate Over a Five-Second Window

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time (seconds)</th>
<th>Detection Rate</th>
<th>Type I</th>
<th>Type II</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed value (Normal Alpha)</td>
<td>1</td>
<td>13.28%</td>
<td>90</td>
<td>2,794</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1.89%</td>
<td>63</td>
<td>3,161</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.12%</td>
<td>27</td>
<td>3,218</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.19%</td>
<td>21</td>
<td>3,216</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.06%</td>
<td>60</td>
<td>3,220</td>
</tr>
<tr>
<td>Fixed value (Low Alpha)</td>
<td>1</td>
<td>90.38%</td>
<td>675</td>
<td>310</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>90.53%</td>
<td>640</td>
<td>305</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>90.10%</td>
<td>598</td>
<td>319</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>90.01%</td>
<td>588</td>
<td>322</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>89.73%</td>
<td>610</td>
<td>331</td>
</tr>
<tr>
<td>Harville Algorithm</td>
<td>1</td>
<td>33.92%</td>
<td>188</td>
<td>2,129</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>25.82%</td>
<td>141</td>
<td>2,390</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>20.98%</td>
<td>82</td>
<td>2,546</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>17.44%</td>
<td>74</td>
<td>2,660</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>14.90%</td>
<td>138</td>
<td>2,742</td>
</tr>
<tr>
<td>Proposed Algorithm</td>
<td>1</td>
<td>80.97%</td>
<td>522</td>
<td>613</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>80.60%</td>
<td>496</td>
<td>625</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>80.48%</td>
<td>451</td>
<td>629</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>80.13%</td>
<td>454</td>
<td>640</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>79.73%</td>
<td>506</td>
<td>653</td>
</tr>
</tbody>
</table>

Table 3.8: Detection rates for stalled objects over a five-second window
## Adaptive Threshold Algorithm Performance

<table>
<thead>
<tr>
<th>Scene Description</th>
<th>Performance (Standard Threshold)</th>
<th>Performance (Adaptive Threshold)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Errors Per Frame</td>
<td>Errors Per Frame</td>
</tr>
<tr>
<td>Gaussian Weight Ratio</td>
<td>Matched Model Oscillation</td>
<td>Description</td>
</tr>
<tr>
<td>Low</td>
<td>Low</td>
<td>Stalled cars</td>
</tr>
<tr>
<td>Low</td>
<td>High</td>
<td>Waving trees</td>
</tr>
<tr>
<td>High</td>
<td>Low</td>
<td>Asphalt road, low traffic</td>
</tr>
<tr>
<td>High</td>
<td>High</td>
<td>Asphalt road, high traffic</td>
</tr>
</tbody>
</table>

*Note: The video footage of the waving trees contained no foreground objects.*

Table 3.9: Detection rates for adaptive thresholding algorithm described in the decision matrix in Figure 2-6. This consists of a four scenes: one with a stalled object, another with a multimodal background, a scene with low foreground activity against a unimodal background, and a scene with high foreground activity against a unimodal background. Each video clip was analyzed with the original GMM algorithm, first using a fixed threshold parameter and then using the adaptive algorithm. The results of these tests are presented in Table 3.9.
Chapter 4

Discussion

4.1 HSV-Based Algorithms

This section examines the performance of various aspects of the HSV-based background subtraction algorithms. First, we shall assess general performance compared to the original GMM algorithm. Then, we shall discuss the effect each specific improvement has.

4.1.1 General Performance

In most situations, the foreground pixel detection performance of the modified algorithm is very close to the original GMM algorithm. This is especially true against well-saturated backgrounds, where the individual color components of the background Gaussians are least susceptible to noise. For example, looking at Tables 3.5 and 3.6, we can see that the detection rates for the RGB and HSV-based algorithms are virtually identical, differing by 0.05% at most. In some cases, however, the limitations of the HSV color space result in poorer detection. As one might expect from the analysis given in Section 2.2.2, the algorithm’s performance does suffer in regions with dark and gray backgrounds. We shall examine the two separately.

Dark backgrounds are the less troublesome of the two; the problems that occur are due to the pixel smoothing algorithm, which is necessary to avoid spurious fore-
ground pixel detection. Smoothing is only performed when both the background model and the detected pixel are sufficiently dark. In general, both the background model and the foreground objects that pass over it will not both be dark, so smoothing only occurs when the background is dark and not occluded. In the case when both the foreground objects and the background are different but sufficiently dark, the algorithm erroneously smoothes the observed value and inserts it into the history window. This not only tends to cause an otherwise-foreground pixel to be detected as background, but it also corrupts the history window. Thankfully, such situations are quite uncommon in most tracking scenes. When tracking lighter-colored objects on a dark background, there should be no performance loss compared to the original GMM algorithm. This is corroborated by the data in Table 3.3, which shows that the detection rate for the algorithm using the unmodified HSV color space is only 1.5% less than that of its RGB counterpart.

The problem of gray pixels is more severe, and unfortunately irreconcilable. As discussed in Section 2.2.2, when the background color values are sufficiently close to the grayscale axis, we assume that the distribution of hue values is essentially uniform. Although we can still extract foreground pixels based on the saturation and value components alone, the detection can become less accurate than normal. In the case of the gray background data in Table 3.4, the detection rate for the unmodified HSV space is approximately equal to that of the RGB-based algorithm, because the saturation and value color components of the foreground object are sufficiently different than the background. However, this is not always the case. In Table 3.1, we can see that while the best-case tracking performance for the HSV-based algorithms is comparable to the original algorithm for non-gray foreground objects (91% vs. 98%), it is a full 20% lower for gray ones (64.5% vs. 84.5%). The occurrence of grayscale background pixels is common in most tracking scenes (e.g., on floors, walls, roads, etc.) although not all pixels are sufficiently close to the grayscale axis to be problematic.
4.1.2 Shadow Detection

We can summarize the performance of the shadow detection and avoidance mechanisms by the following maxim: *the harder and more pronounced the shadow is, the more difficult it is to properly detect.* For very soft shadows, the observed color values are close enough to the unshadowed background model that no shadow detection is necessary. This is due to the fact that unlike RGB, HSV is nearly perceptually uniform, so softly shadowed pixels tend to have color component values quite close to their original, unshadowed values. In Table 3.4, for example, we see that against a gray background in ambient lighting, almost 47% of the shadow pixels are avoided using the normal HSV space, and almost 85% are avoided using the modified space. Even in slightly brighter lighting and against a brighter background—where shadows should be somewhat more pronounced—the avoidance rate can reach over 90%, as seen in Table 3.7. For both cases, the rate of actual shadow detection is rather low, at almost 0% for the gray background and about 4% for the brighter cyan background. This is most likely due to the fact that most pixels that meet the shadow detection criteria are close enough to the background model that they are detected as background pixels first.

For harder shadows, shadow detection and avoidance become less successful. This is most likely due to the fact that as the shadows become darker (i.e., the value component drops), the hue and saturation become more likely to deviate from their original values due to greater noise in dark pixel regions. In Tables 3.5 and 3.6, we can see that overall shadow detection and avoidance under hard lighting is approximately 24.3% and 38.3% for the two brighter backgrounds; compare this to the 93.4% detection rate for soft shadows against the bright cyan background. Furthermore, the rate of false shadow pixels rises by as much as tenfold, although even this higher value is still small compared to the overall number of shadow pixels per frame.

The worst shadow detection performance occurs in grayscale pixel regions. As previously mentioned, in this region we assume that the hue distribution is approximately uniform and we simply ignore it when matching observed values to background
models. For detecting foreground pixels this is less of a problem because we can still use the other two color components. With shadow detection, however, we are essentially matching only the saturation component of the observed pixel against the background model. Although the matching criteria is to a much tighter bound, this is still quite prone to errors. In general, it is probably better to avoid attempting shadow detection in grayscale regions.

4.1.3 Modified HSV Color Space

The modified HS'V color space described in Section 2.2.3 was designed to reduce the expected pixel noise, primarily in dark regions. From the results in Chapter 3, we can see that in this regard, the modification does function as expected. Table 3.3 shows that for soft lighting, the rate of Type I errors (false foreground pixel detection) drops by 26.40% for unsmoothed and spatially smoothed pixels, and almost 54% under temporal smoothing. Likewise under hard lighting, the error rate drops by 12.68% and 17.60%, respectively. This noise reduction is also evident in the results for the gray, red, and teal backgrounds, averaging 20.17% under soft lighting and 8.56% under hard lighting.

A closer inspection of the data also suggests that the effects of this modification extend well beyond the scope of noise reduction. We can see that the rate of Type II errors (undetected foreground pixels) rises considerably; against black backgrounds, it increases by a factor of over 2300% for bright lighting, and approximately 900% for soft lighting. The error rate increases for other backgrounds as well, although by a significantly smaller amount. Compounding this problem is the fact that in general, the detection rate for foreground pixels is lower in the modified space than in the original. Against brighter, more saturated backgrounds, this difference is negligible; however, in other cases it is much more pronounced. In black pixel regions, detection accuracy drops by almost 8% in hard lighting and 2.5% in soft lighting. In gray regions, the performance is even worse, dropping by nearly 15% under soft lighting.

Shadow detection is also impacted by the use of HS'V space. In general, the detection rate is higher in the original color space than in the modified version. Against
gray backgrounds, shadow detection is 3.5 times better in the original color space, while against teal backgrounds the difference is a factor of 1.7. Against red backgrounds, the detection rate actually increases slightly in the modified color space, although the difference is negligible. The shadow avoidance rate (in which they are detected neither as shadows nor as foreground pixels) is higher in the modified color space than in the original, by almost 40% in the case of gray background pixels under soft lighting. In more saturated backgrounds, this difference is less pronounced. Finally, the rate of falsely detected shadow pixels is actually higher in the HSL space than in the original, by as much as a factor of 9 in soft gray backgrounds. This is rather surprising, considering that the detection rate for actual shadow pixels is lower in the modified space.

We can conclude by these results that by modifying the color space in the manner in which we did, we effectively reduce the sensitivity of our tracking algorithm. On one hand, we benefit by avoiding more shadow pixels and detecting fewer false foreground pixels; on the other, our detection rates for shadow and foreground pixels tend to decrease as well, and the number of missed foreground pixels increases. If accurate detection of foreground pixels is the primary goal for a background subtraction algorithm, then using this modified color space should be avoided. However, if it is more important to avoid detecting spurious foreground pixels due to shadows, lighting changes, or acquisition noise, then this modification may prove useful.

4.1.4 Pixel Smoothing

Section 2.2.3 discussed two separate approaches for smoothing dark pixels to reduce noise in HSV space: spatial smoothing and temporal smoothing. From the data in Chapter 3, it seems that the latter offers better performance. In Table 3.3, we can see that for both soft and hard lighting, temporal smoothing yields a higher average detection rate (96.5% vs 95.5%) and a lower Type I error rate (523 pixels/frame vs. 758 pixels/frame). The Type II error rate is higher for temporal smoothing than for spatial smoothing; however, both rates are still quite small—0.46 and 0.04 pixels/frame, respectively. Since pixel smoothing is used only for dark pixels, the
choice of algorithms does not have any effect on the brighter backgrounds. It does have some effect on the gray background in soft lighting, because its pixel brightness is sufficiently low for the algorithm to activate. In that case, temporal smoothing offers not only better detection rates, but also lower Type I and Type II error rates than spatial smoothing.

A key difficulty for using spatial smoothing is that for pixels on the border between a dark and a non-dark region, the algorithm must be careful to smooth dark pixels only with neighboring pixels within its region. Smoothing across a border yields incorrect pixel values and tends to blur the boundary itself. Accurate boundary detection is not always possible; this is the most likely cause for the lower performance of the spatial smoothing algorithm compared to its temporal smoothing counterpart.

4.2 Adaptive Learning Constants

In this section, we will examine the performance of the adaptive-α algorithm presented in this thesis against both the original GMM algorithm in [12] and the algorithm developed by Harville et al., as described in Section 2.3.2. Then we shall examine the trade-off between accurately tracking “sleeping” objects and avoiding “waking” objects.

4.2.1 Performance Results

The data in Table 3.8 yields a number of conclusions regarding the various adaptive and non-adaptive settings for the learning constant. First, a value of α suitable for normal tracking conditions will not perform well when foreground objects stop moving for extended periods of time. By the end of the five-second window, the detection rate for this setting drops to 0.06%, meaning that almost all of the foreground objects been incorporated into the background model. Setting the learning constant at a very low value—low enough to successfully track objects for more than 60 seconds—gives much better results. Throughout the five-second window, the detection rate remains at approximately 90%.
The two adaptive algorithms show considerably different detection rates from each other. The Harville algorithm’s detection rate is approximately 34% at the start of the window, but drops to about 15% by the end. Presumably, the algorithm adapts the learning constant somewhat as the cars enter the scene, but does not continue when the cars become motionless. This is due to the fact that the algorithm’s heuristic for adaptation is based on the intensity difference of observed pixels over time; when objects stop moving, this difference becomes zero. The algorithm presented in Section 2.3 does much better, staying at approximately 80% over the five seconds. The reason the detection rate is somewhat lower than that of the low-α setting is that during the time when the foreground object first arrives—when the learning constant transitions from the normal to low values—the background model can still adapt quickly enough that some foreground pixels may enter into the background model and escape detection.

As a visualization of the difference between the four settings, Figure 4-1 shows the detected foreground pixels (in green) superimposed over the actual scene at one time-point during the window. Clearly, the normal setting (a) is only sufficient to detect the most recently arrived car; the remaining objects have already been incorporated into the background. Harville’s algorithm (c) adapts α enough to accurately track the last two cars, and detect some parts of the other cars. The proposed algorithm (d) clearly comes much closer to tracking all of the stalled objects.

One might wonder why so much effort is made to design an effective adaptive algorithm when we could just set the value of α to a low value and still achieve better detection. The reason is that this value is much lower than optimal for normal, low-traffic conditions. Higher values of α allow the background model to adapt more quickly to changes in the background scene, such as gradual lighting changes. Ideally, when there is no foreground activity at all, the learning constant should be set at 1, so that the background model exactly follows the observed values.
Figure 4-1: Foreground detection of stalled objects
4.2.2 Trade-off Between Sleeping and Waking Objects

Our research has allowed us to draw a fundamental conclusion regarding adaptive-\(\alpha\) algorithms: any attempt to improve the detection of “sleeping” objects will result in poorer performance for “waking” objects. To better understand this issue, consider the following scenario.

Assume we are using an adaptive-\(\alpha\) algorithm in conjunction with our background subtraction system. The precise design of the adaptive algorithm is not important, as long as it decreases a pixel’s learning constant during periods of high foreground activity, and perhaps increase it during low activity periods. In a scene where a foreground object enters and becomes stationary, an ideal algorithm would decrease \(\alpha\) for each affected pixel, allowing the object to remain labeled foreground for as long as possible. If the object is not stationary for too long, then upon its departure the pixels will once again label the original background correctly. However, if the object remains motionless for a long enough time (e.g., a parked car), the prior weight of its matching Gaussian distribution will become large enough to explain the background model by itself. If the object were now to “wake up” and move away, the re-exposed background would be treated as a foreground object because it no longer falls within the background threshold. The learning constant algorithm would recognize this region as foreground and again try and to keep it in the foreground for as long as possible, prolonging the tracking error. Fortunately, in most scenarios there are far more incidences of short-term sleeping objects than there are long-term ones, so this trade-off is acceptable.

4.3 Adaptive Background Threshold

The proposed adaptive background thresholding algorithm has a markedly smaller effect on overall detection than the adaptive learning constant. Looking at Table 3.9, we can see that the only noticeable effect occurs on the multimodal background video clip (labeled “waving trees”). This is not unexpected. For unimodal backgrounds with low foreground activity (“asphalt road, low traffic”), the algorithm tries to maintain
the per-pixel threshold at the normal value, and thus performs no differently from a static threshold. Similarly, for stationary objects ("stalled cars") the algorithm does nothing because, as mentioned in Section 2.4, the desired threshold is not discernable using only the ratios of the prior weights and the oscillation rate of matched models. The lack of a non-negligible effect for unimodal backgrounds with high foreground traffic is somewhat peculiar but also explainable. Although the algorithm performs correctly and the background threshold is lowered, this does not significantly effect the detection rate. Even with the original threshold, the prior weights of the Gaussians corresponding to foreground pixels almost never increase high enough that they enter into the background model.

The adaptive thresholds do make a noticeable impact for scenes with pixel regions containing multimodal backgrounds. In this case, the low prior weight ratio and high matched model oscillation causes the algorithm to increase the background threshold parameter, thereby reducing the number of spurious foreground pixels. In the test footage containing the multimodal pixel regions (labeled "waving trees"), the rate of false foreground pixels dropped by 21%. The higher threshold may reduce actual foreground pixel detection in this regions, but this hypothesis could not be tested because the video footage did not contain any foreground objects.
Chapter 5

Conclusions

This chapter presents a brief summary of the methods and results presented in this thesis. It also examines potential future improvements to Gaussian mixture model-based algorithms.

5.1 Synopsis of Results

This section provides brief summaries for each of the modifications presented in this thesis. For a more detailed analysis, please refer to Chapter 4.

- In general, the HSV color space-based algorithm performs comparably to the original GMM algorithm.

- Detection suffers primarily in dark and gray pixel regions, due to increased noise in the former and the loss of a color component in the latter.

- Shadow detection and avoidance is better for softer shadows than for harder ones.

- Using the modified color space reduces overall shadow and foreground pixel detection, but also increases shadow avoidance and reduces the number of erroneous foreground pixels.
• Temporal smoothing outperforms spatial smoothing in dark regions, offering better noise reduction and a higher detection rate at the same time. Spatial smoothing requires that pixels on the boundary of dark and non-dark regions be properly segmented, so that observed values are not averaged across the border. This can be difficult at times, and most likely accounts for the disparity when compared to temporal smoothing.

• The adaptive learning constant algorithm presented in this paper offers better detection than the Harville algorithm in regions containing heavy foreground traffic or stalled foreground objections. It is also more robust than using a fixed constant, in which case one must decide whether it is more important to accurately track stalled objects or to be more responsive to changes in the background. Unlike the Harville algorithm, it is more flexible to changes in scene activity patterns, since the value of $\alpha$ can also increase. However, like any other adaptive-$\alpha$ algorithm, the algorithm shows poorer performance in avoiding waking objects than the unmodified GMM algorithm.

• Adaptive background thresholds have a considerably smaller effect on detection rates than adaptive learning constants. The effect is most noticeable for pixel regions with multimodal backgrounds. In these cases, the algorithm generally increases the background threshold parameter, leading to fewer false positives. Under normal tracking conditions and against unimodal backgrounds, however, the effect of this algorithm is negligible.

5.2 Future Work

There are numerous potential improvements to the general class of Gaussian mixture model-based algorithms that were not discussed in this thesis. Other researchers have already begun to investigate alternate color spaces, the use of stereo cameras, and other extensions to the original algorithm. Below are two more areas of potential research: performance improvements and hybrid color spaces.
5.2.1 Performance Improvements

In order for a background subtraction algorithm to be of use in a tracking system, it must be able to run quickly on commodity hardware. The algorithms presented in this paper were not designed or implemented with performance as a primary concern. Although the algorithms could be optimized to increase performance, it would be beneficial to research and implement algorithms that are designed to scale to higher resolutions and frame rates than the ones used in this paper. From an implementation perspective, it would be very useful to develop parallelizable algorithms that could scale by running on clusters of low-cost machines, thereby making tracking systems more cost-effective to deploy.

5.2.2 Hybrid RGB/HSV Color Spaces

The RGB and HSV color spaces have different strengths and weaknesses when applied to the problem of object tracking. Rather than choosing between one or the other, it could be advantageous to develop hybrid algorithms that process all or part of the scene using both color spaces. For example, one could conceivably build an algorithm that uses RGB-color to perform background subtraction, but also maintains an HSV model to invalidate foreground pixels that are detected as the byproduct of shadows or lighting changes. Although the computational resources necessary to perform these functions in real-time would be high, the system would be inherently parallelizable and could be run on separate processors or machines.
Bibliography


