The Hebrew University of Jerusalem האוניברסיטה העברית בירושלים
The Hebrew University of Jerusalem
Faculty of Mathematics and Sciences
School of Computer Science and Engineering

Master of Science Thesis

Persistent Particle Filters For Background Subtraction

by

Yair Movshovitz-Attias

Supervisor: Shmuel Peleg

Jerusalem, 2010

To Dana. This work might not have come to light without your continuous support. Our weekly drives out of town, our *family seminar*, were the catalysts for so many ideas that I lost count after a while. I am lucky to have you.

Abstract

Moving objects are usually detected by measuring the appearance change from a background model. The background model should adapt to slow changes such as illumination, but detect faster changes caused by moving objects. Particle filters do an excellent task in modeling non parametric distributions as needed for a background model, but may adapt too quickly to the foreground objects.

A persistent particle filter is proposed, following bacterial persistence. Bacterial persistence is linked to the random switch of bacteria between two states: A normal growing cell and a dormant but persistent cell. The dormant cells can survive stress such as antibiotics. When a dormant cell switches to a normal status after the stress is over, bacterial growth continues.

Similar to bacteria, particles will switch between dormant and active states, where dormant particles will not adapt to the changing environment. A further modification of particle filters allows discontinuous jumps into new parameters enabling foreground objects to join the background when they stop moving. This can also quickly build multi-modal distributions.

Contents

Chapter 1

Introduction

As the number of video surveillance cameras increases, the need arises for sophisticated methods to manipulate, analyze, and display the data captured in them. One of the most fundamental building blocks of any such system is the ability to detect foreground objects in the scene and separate them from the background. This ability is a prerequisite for many applications, such as object tracking and identification, activity monitoring, and video summarization.

Generally, the system is made up of a stationary video surveillance camera monitoring a scene in which foreground objects move about in the frame. The goal is to extract the interesting foreground objects (people, cars, etc) from the rest of the frame. While foreground objects are characterized by movement, the system needs to be robust to movement that is caused by other factors such as camera shake or dynamic background (trees in the wind, flags, or water). Robustness should be achieved for other changes in the scene such as ones caused by change in illumination.

Background subtraction is a common approach to extract moving objects from a video sequence. It is comprised of two modules, Background Modeling, and Foreground detection. A background model is a representation of the background scene. Background models can be as simple as a single color for each pixel, or can include a representation of the color distribution of the background.

Foreground detection is the task of segmenting the current frame into moving objects (foreground) and stationary regions (background). Each pixel in a new frame is compared against the background model, and is recognized as a foreground object if it differs significantly from the model. The results of the segmentation process can be used as one of the inputs for the background modeling stage in a recursive manner.

1.1 Thesis Overview

The remainder of this thesis is organized as follows. A review of related works is presented in Chapter 2. In Chapter 3 the *Particle Filter* algorithm is described along with the needed Control Theory background. A naive implementation of Particle Filtering for background subtraction is detailed in Chapter 4. Following this description, Chapter 5 analyzes the naive technique's deficiencies and presents the notation of a *Persistent Particle Filter*. It outlines the *Bacteria*

Filters algorithm, which can overcome many of the shortcomings of the naive algorithm. Experiments and results are shown in Chapter 6, and the thesis concludes with a discussion at Chapter 8

Chapter 2

Motivation And Related Work

As Background Subtraction is a fundamental tool in computer vision it has been extensively researched in the last decade. Over the years many algorithms have been presented with increasing complexity and success. However, the problem of accurately identifying foreground regions remains a challenge and so research continues in this field. In this chapter we describe a number of noteworthy techniques for background subtraction, and analyze their properties.

Unimodal Background

Simple background models assume that the color at each pixel over time can be modeled by a unimodal distribution. Wren *et al.* [Wren et al., 1997] describe a method in which each pixel is modeled as a Gaussian and is estimated using a mean value μ_t and a covariance Matrix K_t. In each time step the parameters are adapted to account for lighting changes. The unimodal assumption fails when the scene contains background motion, such as trees moving in the wind or ripples on water, and when the camera is shaking. It also introduces difficulties when trying to model lighting changes and objects that are added or removed from the background.

Mixture of Gaussians

Non stationary backgrounds have been estimated by the Mixture of Gaussians (MoG) technique [Stauffer and Grimson, 1999; Lee et al., 2003; Javed et al., 2002; KaewTraKulPong and Bowden, 2001]. Each pixel is modeled by a distribution composed of K Gaussians (usually $K = 3$ to 5). The probability that the current pixel value I_t belongs to the background is given by

$$P(I_t) = \sum_{i=1}^{K} \omega_{i,t} \cdot \mathcal{N}(I_t, \mu_{i,t}, \Sigma_{i,t}) \tag{2.1}$$

where $\omega_{i,t}$ is the weight given to the i'th Gaussian at time t (what portion of the data is accounted for by the Gaussian), $\mathcal{N}$ is the Normal distribution Probability Density Function with $\mu_{i,t}$ and $\Sigma_{i,t}$ being the mean and covariance

matrix respectively. Usually, for computational reasons, the covariance matrix is assumed to be of the form

$$\Sigma_{i,t} = \sigma_i^2 \mathbf{I} \tag{2.2}$$

This assumes that the red, green, and blue channels are independent and have the same variance. The Gaussians' parameters are modified at each time step to reflect the changes in the scene. Every new value I_t is checked against the pixel's current K components until a match is found. A match is defined if the pixel value lies within 2.5 times the standard deviation of the Gaussian. If no match is found, the Gaussian with the lowest weight is discarded, and a new one is created with a mean equal to the pixel value, a large initial variance, and a low weight. The weights of all components are adjusted by

$$\omega_{i,t} = (1 - \alpha)\omega_{i,t-1} + \alpha M_{i,t} \tag{2.3}$$

where α is the learning rate that determines how quickly the model adapts. $M_{i,t} = 1$ if there was a match for the i'th Gaussian and $M_{i,t} = 0$ otherwise. The weights are then re-normalized to 1. The Gaussian parameters are left unchanged for components that did not have a match. For those that match the new observation the update is as follows

$$\mu_t = (1 - \rho)\mu_{t-1} + \rho I_t \tag{2.4}$$

$$\sigma_t^2 = (1 - \rho)\sigma_{t-1}^2 + \rho(I_t - \mu_t)^T(I_t\mu_t) \tag{2.5}$$

where the learning rate ρ is computed using

$$\rho = \alpha \cdot \mathcal{N}(I_t|\mu_t, \sigma_t) \tag{2.6}$$

The MoG technique may fail to accurately model fast variation in the scene with just a small number of Gaussians. If the adaptation of the Gaussian parameters to the changes in the scene is fast, slow objects can be absorbed into the background model. When the adaptation is slow the model has trouble detecting fast changes to the background such as those produced by illumination change [Toyama et al., 1999].

Histograms

Modeling the background distribution with histograms is a simple, yet robust, technique. Each pixel holds a histogram of the colors that were present at its location over some time period. Spatial information can also be integrated by adding neighborhood pixel values into the histogram [Ko et al., 2008]. The histogram can be estimated recursively by updating the prior histogram, weighted by $\alpha < 1$, with the current color weighted by $(1 - \alpha)$. Pixel histograms are non parametric and can account for any type of multi-modal distribution. However, the number of bins l in the histogram is fixed and this quantization can greatly influence the results depending on the video being processed. Moreover, this technique does not scale gracefully. To model color videos the histogram will need to contain l^3 bins in RGB space, which produces histograms that are not only very large but are also sparse. Addressing this problem by decoupling the channels and using 3 separate histograms reduces the memory required to $3l$ bins but can cause classification errors as illustrated by the toy example in Figure 2.1.

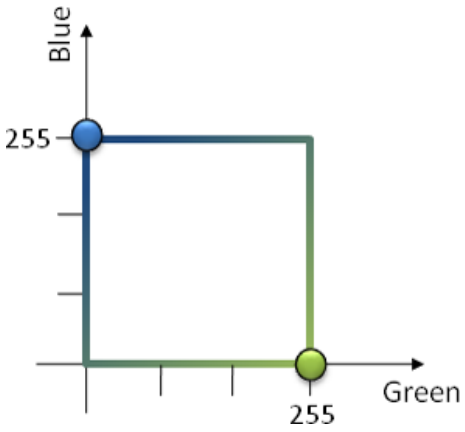

Figure 2.1: The shortcomings of treating the color channels independently can be understood by examining a toy example, displaying repetitive background motions between a tree leaf moving in the wind and the sky that is visible behind it. Let the distribution of a certain pixel be such that half the time its color is blue (sky), and half the time it is green (leaf). We omit the Red channel for simplicity. We would expect the distribution to be such that green/blue would get a 50% probability of belonging to the background, and all other colors would have probability (close to) 0. However, when using information from only one color channel at a time, any color having a green or blue component (e.g. cyan: green + blue) will have high probability.

Codebook

Kim *et al.* [Kim et al., 2005] outline a non parametric, compressed representation of the distribution using a codebook model. Samples of the distribution at each pixel are clustered into a set of code words. Every pixel has a codebook $C = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_L\}$. The length L of the codebook for each pixel is decided according to its sample variance. A code word $\mathbf{c}_i$ is made up of an RGB vector $\mathbf{v}_i = (R_i, G_i, B_i)$ and an auxiliary 6-tuple $\mathbf{aux}_i =< \hat{I}_i, \check{I}_i, f_i, \lambda_i, p_i, q_i >$ where:

$\hat{I}, \check{I}$: the *max* and *min* brightness that the code word accepted.
f : the frequency of the repetitions of the codeword.
λ : the longest interval during the training period that the code word has not recurred.
p, q :the *first* and *last* frame number in which the codeword has occurred.

When a new value $\mathbf{x} = (R, G, B)$ is encountered it is compared with all the codewords $\mathbf{c}_i$ of the pixel and a match is defined if

- $colordist(\mathbf{x}, \mathbf{v}_i) < \epsilon$
- $\|\mathbf{x}\| \in [I_{low}, I_{hi}]$

where

$$I_{low} = \alpha \hat{I}_i \tag{2.7}$$

$$I_{hi} = min\{\beta \hat{I}, \frac{\check{I}}{\alpha}\} \tag{2.8}$$

$$p^2 = \|\mathbf{x}\|^2 \cos^2 \theta = \frac{< \mathbf{x}, \mathbf{v}_i >^2}{\|\mathbf{v}_i\|^2} \tag{2.9}$$

$$colordist(\mathbf{x}, \mathbf{v}_i) = \sqrt{\|\mathbf{x}\|^2 - p^2} \tag{2.10}$$

And $\alpha < 1$, usually 0.4, $\beta > 1$, usually 1.1. If a match is found with any of the codewords the new value is marked as background, else it is marked as foreground. The model is built using a long training period where it is assumed that no objects are in the scene.

By using a Persistent Particle Filter for each pixel we can address these issues. The method has the ability to model arbitrary multi-modal distributions, thus can handle repetitive background movements and illumination changes; It scales easily to the $3D$ color space in terms of memory, without making independence assumptions that sacrifice accuracy; And it does not require a training period without moving objects.

Chapter 3

Particle Filters - Overview

3.1 Recursive State Estimation

State Estimation is the process of estimating quantities that are not directly observable, but can be inferred, using data from other, observable, quantities called *measurements*. The group of Probabilistic State Estimation techniques, to which Particle Filtering belongs, computes probability distributions over possible states.

A state x_t is a description, at time t, of a system (e.g. robot location, laptop battery level, the weather), for which it is not possible to get direct information on. A measurement z_t is data that can be acquired directly from the environment (sonar range finder, volts in the battery, humidity in the air).

The state x_t is stochastically generated from the state x_{t-1}. The evolution of the state can be expressed using the probability distribution

$$p(x_t|x_{0:t-1}, z_{0:t-1}) \tag{3.1}$$

which is called the State Transition Probability. A common assumption is that the state x is *complete*, i.e., it holds sufficient information on previous states, or more precisely, given the state at time $t-1$ the state at time t is independent of the measurements

$$p(x_t|x_{0:t-1}, z_{0:t-1}) = p(x_t|x_t - 1) \tag{3.2}$$

Note that this entails that the system is a *Markov Model*. As the states are hidden and need to be estimated from the measurements, this entails that the system is in fact a *Hidden Markov Model*.

Another process which can be modeled using a probability distribution is the generation of measurements

$$p(z_t|x_{0:t}, z_{0:t-1}) = p(z_t|x_t) \tag{3.3}$$

Which is called the Measurement Probability and, again, the equation holds if x is a complete state.

This model is known as the *Dynamic System Model*. Alternatively, the dynamic system can be represented as a set of two equations: the State Transition Equation

$$x_t = f(x_{t-1}, \mu_{t-1}) \tag{3.4}$$

where f is the evolution function and μ_{t-1} is added noise called the State Noise, and the Measurement Equation

$$z_t = g(x_t, \epsilon_t) \tag{3.5}$$

where ϵ_t is added noise called the Measurement Noise.

Recursive Filters are techniques which estimate the state using a recursive strategy, they are composed of (at least) two stages:

Predict In the prediction stage the next state is predicted

$$p(x_{t-1}|z_{0:t-1}) \rightarrow p(x_t|z_{0:t-1}) \tag{3.6}$$

Update In the update stage the new measurements are incorporated.

$$p(x_t|z_{0:t-1}) \rightarrow p(x_t|z_{0:t}) \tag{3.7}$$

These steps are performed in alternating fashion.

3.2 Bayes Filter

The most general recursive filter is the Bayes Filter. The Bayes Filter calculates the state estimate directly from the measurements and previous state. Its prediction and update stages are as follows:

Prediction

$$p(x_t|z_{0:t-1}) = \int p(x_t|x_{t-1})p(x_{t-1}|z_{0:t-1})dx_{t-1} \tag{3.8}$$

This equation is also known as the Chapman-Kolmogoroff equation. It is the prior of x_t at time t without knowledge of z_t

Update In the update step the posterior probability density function (*pdf*) is computed from the predicted *pdf* and the new measurement

$$\begin{aligned}
p(x_t|z_{0:t}) &= \frac{p(z_{0:t}|x_t)p(x_t)}{p(z_{0:t})} && (3.9)\\
&= \frac{p(z_t, z_{0:t-1}|x_t)p(x_t)}{p(z_t, z_{0:t-1})} && (3.10)\\
&= \frac{p(z_t|z_{0:t-1}, x_t)p(z_{0:t-1}|x_t)p(x_t)}{p(z_t|z_{0:t-1})p(z_{0:t-1})} && (3.11)\\
&= \frac{p(z_t|z_{0:t-1}, x_t)p(x_t|z_{0:t-1})p(z_{0:t-1})p(x_t)}{p(z_t|z_{0:t-1})p(z_{0:t-1})p(x_t)} && (3.12)\\
&= \frac{p(z_t|x_t)p(x_t|z_{0:t-1})}{p(z_t|z_{0:t-1})} && (3.13)
\end{aligned}$$

The structure of the update equation can be analyzed as

$$\text{posterior} = \frac{\text{likelihood} \cdot \text{prior}}{\text{evidence}} \tag{3.14}$$

where the likelihood is given by the measurement model, the prior is known from the prediction stage, and the evidence is a normalizing constant that can be calculated by

$$p(z_t|z_{0:t-1}) = \int p(z_t|x_t)p(x_t|z_{0:t-1})dx_t \tag{3.15}$$

While it seems that the Bayes Filter can provide an optimal solution, in the sense of computing the posterior *pdf*, in fact the integrals are usually not tractable and so in all but the simplest cases the Bayes Filter is not practical. In the following section we describe the Particle Filter algorithm which employs a sampling technique in order to approximate the optimal Bayesian solution.

3.3 Particle Filtering

Particle Filtering is a technique for estimating the hidden state X_t of a dynamical system at time t, conditioned on sensor measurements. Note the slight change of notation from x_t to X_t. In particle filtering x is reserved for representing a particle. The goal of the Particle Filter is to estimate the posterior probability density over the state space by using a set of samples (particles), x_t^i, of the state. A particle is an hypothesis as to what the true state is at time t. Each sample is associated with a weight w_t^i that indicates the quality, or importance, of that sample. The set of particles is described as

$$\mathcal{X}_t = \{< x_t^i, w_t^i > | i = 1, \ldots, N\} \tag{3.16}$$

The intuition is to approximate the posterior density by the set of particles. Ideally the likelihood of an hypothesis x_t^i to be part of the particle set would be proportional to its Bayes Filter posterior. The more densely a sub region of the state space is populated by particles, the more likely it is for the true state to be in that region.

The Particle Filter adds a third, re-sampling, step to the usual prediction-update steps of the recursive filters:

Prediction A predicted particle $\hat{x}_{t+1}$ is created by modifying the particle x_t using a transition function f that depends on the current state estimate, with an addition of random noise to simulate the effects of noise on the true state.

$$\hat{x}_{t+1} = f(x_t) + \mu_t \tag{3.17}$$

where μ_t is the state noise as described in (3.4).

Update The associated weight of each particle is evaluated using the new measurement z_t.

$$w_{t+1} = g(\hat{x}_{t+1}, z_t) + \epsilon_t \tag{3.18}$$

where ϵ_t is the measurement noise as in (3.5). The update function g measures the similarity between the measurement z_t and the particle value $\hat{x}_{t+1}$.

The filter could have consisted of only these two steps (with a normalization of weights $\sum_{i=1}^{N} w_i = 1$ after the Update stage). The problem with this

approach is that after a relatively small number of iterations, most particles will have negligible weights (most of the weight will concentrate on only a few particles). In effect these low weight particles are not used, though ideally they could have been used to represent with greater resolution the space adjacent to the true state. This is accomplished by adding a third step to the technique.

Re-sample A new set of particles $\mathcal{X}_{t+1}$ is created by sampling N times, with repetition, from the set of predicted particles $\hat{\mathcal{X}}_{t+1}$ according to the weights w^i_{t+1}.

The re-sampling step shifts the particle set to areas of high probability. Particles which had large weights are expected to produce more copies of themselves than low weight particles in the re-sampling process, thus focusing all computational resources in areas which have greater probability of being the correct state.

Doucet *et al.* [Doucet et al., 2001, 2000] show that this procedure approximates the posterior density. The Particle Filter pseudo code is summarized in Algorithm 1. Rows $2-6$ build the predicted particle set $\hat{\mathcal{X}}_{t+1}$, where the prediction step is performed in line 4, and the update step at line 5. Lines 7 to 9 perform the re-sampling stage, which produces the new particle set $\mathcal{X}_{t+1}$.

The power of Particle Filters lays in its ability to represent arbitrary probability densities, while spreading its computational resources according to the observed distribution. Similar to MoG, it can represent multi modal distributions, but it is superior to MoG due to the fact that it is not parametric. MoG not only assumes the distribution is of a Gaussian Mixture form, it uses a predetermined number of Gaussian components. If the modeled distribution contains more components the technique will inevitably mis-represent it. While Histograms are a great tool for modeling distributions in a non parametric fashion, they spread their computational resources evenly between all possible hypotheses. Particle Filters focus more resources on likely hypotheses.

Algorithm 1 Particle Filter Algorithm

PARTICLEFILTER($\mathcal{X}_t, z_{t+1}$)

1 $\hat{\mathcal{X}}_{t+1} = \mathcal{X}_{t+1} = \emptyset$
2 **for** $i \leftarrow 1$ **to** N
3 **do**
4 $\quad \hat{x}^i_{t+1} = f(x^i_t) + \mu_t$
5 $\quad w^i_{t+1} = g(\hat{x}^i_{t+1}, z_t) + \epsilon_t$
6 $\quad \hat{\mathcal{X}}_{t+1} = \hat{\mathcal{X}}_{t+1} + < \hat{x}^i_{t+1}, w^i_{t+1} >$
7 **for** $n \leftarrow 1$ **to** N
8 **do** draw particle i with probability $\propto w^i_{t+1}$
9 $\quad \mathcal{X}_{t+1} = \mathcal{X}_{t+1} + < x^i_{t+1}, \frac{1}{N} >$
10 **return** $\mathcal{X}_{t+1}$

Chapter 4

Particle Filters for Background Modeling

In this chapter we describe a naive use of Particle Filters for background modeling. in Chapter 5 we analyze its drawbacks and describe improvements that make Particle Filters more useful for background modeling.

To model the background we use Particle Filtering in a per-pixel manner. Our particle filter background model was designed to estimate the color distribution of the pixel, without making any parametric assumptions. The state of pixel r at time t is the set $\mathcal{X}_t(r)$ as in Eq. (3.16) where the particles are vectors over the RGB space.

4.1 Prediction

For a static camera and a relatively static background, we define a simple transition function

$$\hat{x}^i_{t+1}(r) = x^i_t(r) + \mathcal{N}(0, \sigma) \tag{4.1}$$

where the noise is sampled from a Normal distribution $\mathcal{N}$ with mean zero and standard deviation σ. This allows the model to adapt to slow changes in the background, and adds robustness for poor video quality.

4.2 Update

In the update stage the weight of the i'th particle $x^i_t(r)$ is set according to the particle's agreement with the current pixel value. We use the expression

$$w^i_{t+1}(r) = \exp\left(\frac{-\|\hat{x}^i_{t+1}(r) - I_{t+1}(r)\|^2_\infty}{2\sigma^2_t(r)}\right) \tag{4.2}$$

where $\|\cdot\|_\infty$ is the infinity norm, using the largest difference among the color channels, $I_t(r)$ is the value of pixel r in frame t, and $\sigma_t(r)$ is the standard deviation of the pixel calculated by an exponential decaying window

$$\mu_t(r) = \alpha \cdot \mu_{t-1}(r) + (1-\alpha) \cdot I_t(r) \tag{4.3}$$

$$\sigma^2_t(r) = \alpha \cdot \sigma^2_{t-1}(r) + (1-\alpha) \cdot (I_t(r) - \mu_t(r))^2 \tag{4.4}$$

α determines the time window in which pixels can influence the value of σ and μ. We use $\alpha = 0.99$. The statistics are initialized using the first 100 frames of the sequence.

4.3 Re-Sample

Re-sampling is done as described in Chapter 3. For added speed a low variance sampler [Thrun et al., 2005] can be used. The low variance sampler uses N equally spread samples whose phase is determined by a single random draw. In addition to saving time, this can also reduce the effects of the problem known as *sample variance* in which the sampled distribution differs from the hidden, real, distribution. However, in our experiments, switching between sampling techniques did not have a significant effect on the results and so the results presented here were achieved using the simple sampling algorithm.

4.4 Probability map and segmentation

The background probability of each pixel can be computed from its particle set using an average of the strongest particles. First, the particles are sorted according to their weights in descending order. Averaging the best particles is done by

$$P_b(r) = \frac{1}{K} \sum_{i=1}^{K} w^i(r) \tag{4.5}$$

where $w^i(r)$ are the sorted particles with weights as defined in Eq. (4.2), and $K = 0.75N$.

Foreground segmentation is achieved by thresholding the probability map P_b by a predefined value P

$$\text{Foreground}(r) = \begin{cases} 0 & P_b(r) < P \\ 1 & \text{otherwise} \end{cases} \tag{4.6}$$

We used $P = 0.3$ in all our experiments.

Chapter 5

Persistent Particles

In Chapter 4 we demonstrated how Particle Filters can be used to model background in color videos. Particle Filters usually require a large number of particles in order to accurately model complex distributions. For a background modeling algorithm to be practical, the amount of memory used should be minimal, and thus we need to limit the size of the particle set. In the following we outline how to overcome the difficulties introduced by using a small amount of particles.

5.1 Dormant Particles

A typical scene for surveillance cameras can involve a number of challenges that any background subtraction algorithm needs to cope with. These include dynamic background areas such as moving trees, shaking cameras, and foreground objects that may fuse into the background, like a car pulling into a parking space and staying there. The algorithm should be able to differentiate the latter from another scenario in which the car only pauses for a while (for instance while waiting at a traffic light) and then moves on. These types of scenarios emphasize the multi-modal nature of the background scene. While Particle Filters can represent arbitrary distributions this can require a large number of samples. When using only a small particle set, the re-sampling stage can cause one good particle (a particle whose color is close to the current color of the pixel) to drastically change the entire set, collapsing it into a unimodal representation. This is known as the Loss of Diversity problem. To address these issues we introduce *Persistent Particles*, particles which keep their estimation even when the rest of the sample population has shifted to a different region of the state space.

The persistent particles are inspired by the behavior of microbial population. The bacteria switch randomly between two states, a state of normal growth and a dormant state in which they are less vulnerable to the effects of stress such as antibiotics, and can overcome the stress condition when active bacteria do not survive [Balaban et al., 2004].

We propose to divide the particles into two subsets: *Active* particles and dormant or *Persistent* particles. Particles switch from active to persistent and vice versa when certain conditions are met. A particle is considered active if its current estimate of the background color (its RGB-vector) is close to the

current pixel value. Otherwise it is considered persistent.

$$\text{Persist}(x(r)_t^i) = \begin{cases} 1 & \exp\left(\frac{-\|x_t^i(r)-I_t(r)\|_\infty^2}{2\sigma_t^2(r)}\right) < P \\ 0 & else \end{cases} \tag{5.1}$$

where P is the probability threshold as used in Eq. (4.6).

Active particles behave in the way described in Chapter 4. Persistent particles have a different prediction step, and don't participate in the re-sampling stage. For a persistent particle $x(r)_t^i$ the prediction process is

$$\hat{x}(r)_{t+1}^i = \begin{cases} I(r)_{t+1} & p < T_1 \\ x(r)_t^i & \text{else} \end{cases} \tag{5.2}$$

where p is a uniform random number in the range $[0, 1]$ and T_1 is the *persistence threshold* which indicates the duration in which particles remain in the dormant state. In our experiments we used $T_1 = 1/1000$, implying that each particle has a $1/1000$ probability to switch to the observed color.

Examining the parking car example we can now understand the effect of these persistent particles - when a car enters the scene, the particles, tracking the background color, will switch into the dormant phase and thus will not track the color of the car even when it stops at the traffic light. If however the car parks, the particles will gradually switch into active state and will start following its color distribution and eventually merge it into the background. The amount of time for a stopped foreground object to merge into the background is determined by the value of T_1.

5.2 Color Jump For Scene Change

While the persistent particles ensure the multi-modal behavior of the sample population, it can take a significant amount of time, in the order of $1/T_1$, for the population to achieve a good representation of the pixel's histogram. When faced with a continuous, high frequency, change in the background, such as moving trees, we would like the filter to adapt quickly to the dynamic nature of the pixel. To handle such cases we give a pixel the ability to switch one particle to the changing scene. But such changes must be separated in time by enough frames. A particle is selected randomly and its value is switched to the current color if

$$\|I_t(r) - I_{t-1}(r)\|_\infty > T_2 \tag{5.3}$$
$$\text{and time from last switch} > J$$

where in our experiments we use $J = 10$, and $T_2 = 30$. This allows for quick adaptation of the filter for dynamic textures without contaminating the particle set by passing objects. Due to the particles' jump ability, the particle set can adapt quickly to such changes and hold the various colors that repeat in the background. Fig. 5.1 shows a scene in which trees are moving in the wind. It shows how the particle set successfully reproduces the background distribution.

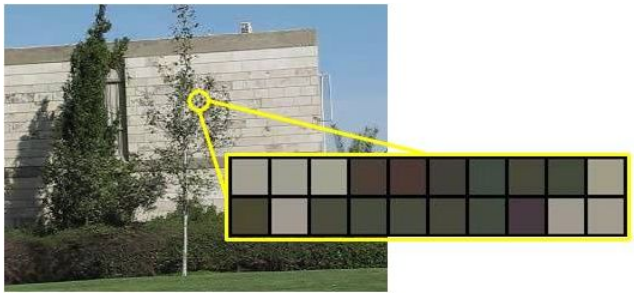

Figure 5.1: Dynamic backgrounds, such as trees moving in the wind, create difficulties to accurately estimate the background distribution. Using the particle's jump ability, the particle set is able to successfully represent the various background colors. The pixel marked by a circle displays both wall and tree colors, as captured by its particle set.

It should be noted that in this section we address significant changes between successive frames, while in the handling of the dormant particles we examine differences between a particle and the latest frame. The Bacteria-Filter pseudo code is summarized in Algorithm 2

Algorithm 2 Bacteria Filter Algorithm

BacteriaFilter($\mathcal{X}(r)_t, I_{t+1}(r)$)

$\mathcal{D}(r)_{t+1} = \mathcal{A}(r)_{t+1} = \mathcal{X}(r)_{t+1} = \emptyset$
for each Particle $x(r)_t^i$ **in** $\mathcal{X}(r)_t$
do
  if $\exp\left(\frac{-\|x_t^i(r) - I_{t+1}(r)\|_\infty^2}{2\sigma_t^2(r)}\right) < P$
  then
    $r =$ rand ($[0, 1]$)
    if $r > T_1$
    then
      $x(r)_{t+1}^i = x(r)_t^i$
      $\mathcal{D}(r)_{t+1} = \mathcal{D}(r)_{t+1} + i$
    else
      $x(r)_{t+1}^i = I_{t+1}(r)$
  else
    $x(r)_{t+1}^i = x(r)_t^i + \mathcal{N}(0, \sigma)$
    $\mathcal{A}(r)_{t+1} = \mathcal{A}(r)_{t+1} + i$

$w(r)_{t+1}^i = \exp\left(\frac{-\|x_t^i(r) - I_{t+1}(r)\|_\infty^2}{2\sigma_t^2(r)}\right)$

Sort $w(r)_{t+1}^i$ in descending order
$K = \frac{3}{4}N$
$P_b(r) = \frac{1}{K}\sum_{i=1}^{K} w(r)_{t+1}^i$
$N_{\mathcal{A}} =$ size of $\mathcal{A}$
for $n = 1$ **to** $N_{\mathcal{A}}$
do
  draw particle i with probability $\propto w_{t+1}^i$
  $\mathcal{X}_{t+1} = \mathcal{X}_{t+1} + < x_{t+1}^i, \frac{1}{N_{\mathcal{A}}} >$
$\mathcal{X}_{t+1} = \mathcal{X}_{t+1} + \mathcal{D}(r)_{t+1}$
return $\mathcal{X}_{t+1}$

Chapter 6

Experiments

Our algorithm was tested on numerous video sequences. In all our tests we assigned $T_1 = 0.001$ (Eq. 5.2), $T_2 = 30$, $J = 10$ (Eq. 5.3), $P = 0.3$ (Eq. 4.6), and $\alpha = 0.99$ (Eq. 4.3). The particle sets contained 20 particles, which were initialized by the result of averaging the first 100 frames of the sequence. Each particle is assigned the average of values witnessed for that pixel, with an addition of random noise. We did not require that these frames be without moving objects. For color videos of size 320×240 it runs at approximately 2 fps on an Intel core 2 Duo 2.53 GHz CPU.

We compare our results with the MoG algorithm [KaewTraKulPong and Bowden, 2001] and the codebook technique [Kim et al., 2005], both as implemented in OpenCV [Bradski and Kaehler, 2008]. For the Codebook algorithm we used a 100 frame training period. For the MoG we used 5 Gaussian kernels. We also implemented the histograms method [Ko et al., 2008]. To reduce computation we used three color histograms quantified to 32 colors in each channel, and did not use the spatial neighbors of a pixel. We used an α value of 0.999 for the histogram method. This is equivalent to our T_1 value which controls the speed at which foreground objects can be assimilated into the background model.

In our comparisons we did not use any post processing techniques, such as morphological operations, in order to make it easier to judge the capabilities of each algorithm. Fig. 6.1 shows frames from a number of different video sequences, and the results of the four algorithms on them. Bacteria-Filters seems to perform best, and the histograms method second best. It is worth noting that Bacteria-Filters and histograms produce segmentations which are considerably cleaner than the other two methods. A closer examination of the histograms method shows that it sometimes segments only partial foreground objects. Bacteria-Filters, on the other hand, creates objects which are fuller and have more precise outline. Note how in the sequence of the man with the dog, the foreground mask of the man is full and yet the background between the legs is not extracted as foreground.

In typical surveillance scenarios the scene can be constantly occupied by moving foreground objects. Fig. 6.2 depicts an example from a highway monitoring camera sequence in which there is no training frame without moving objects. The segmentation results are of high accuracy without traces of the moving objects that were present in the training period.

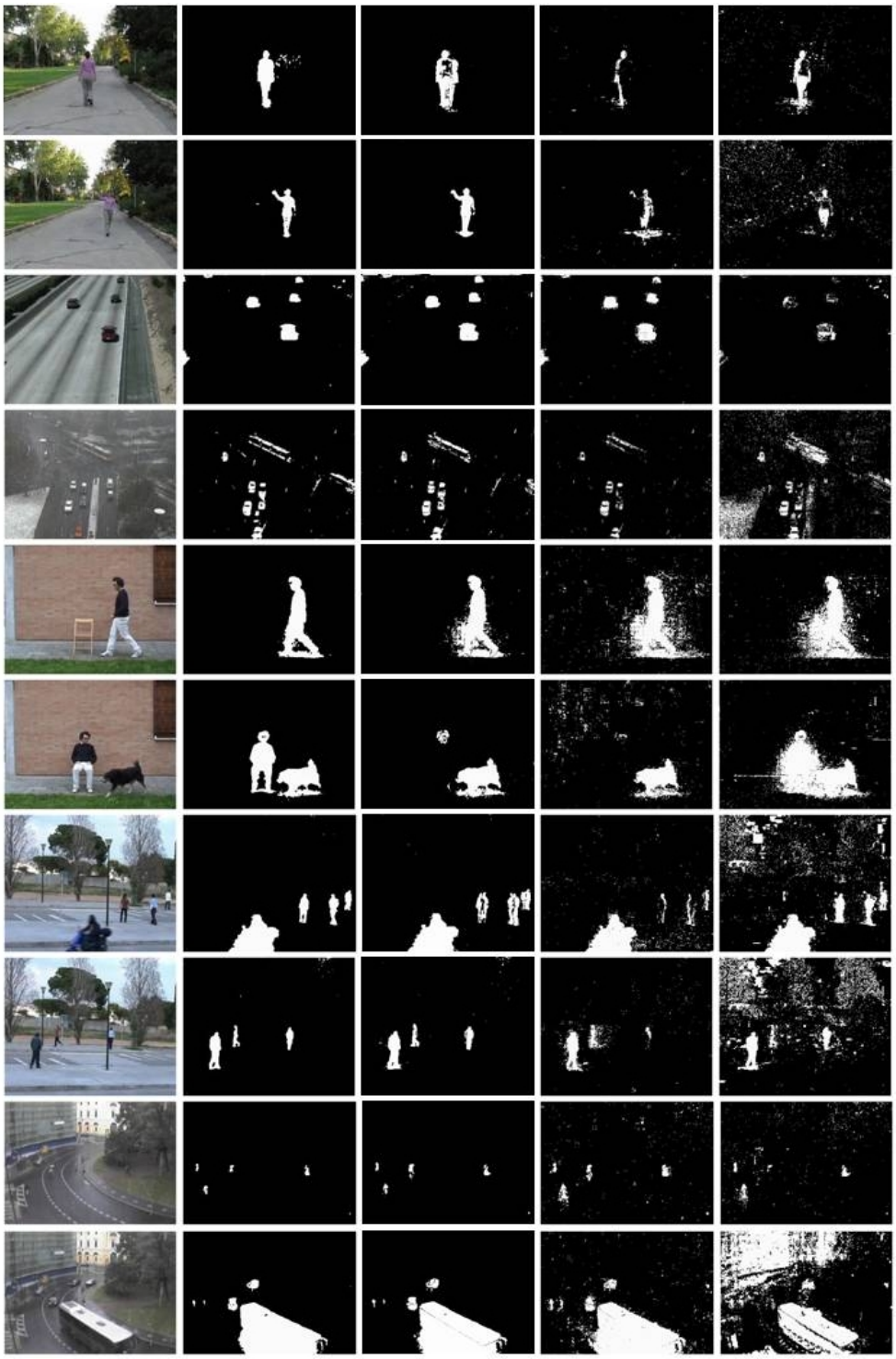

Figure 6.1: Examples of foreground segmentation. From left to right: Input frame, Bacteria-Filters, Histograms, Mixture of Gaussians, Codebook.

Figure 6.2: Results from a 500 frame sequence in which there is no training period without moving objects. A - input frame. B - probability map as calculated by Eq. (4.5), white means high probability to be in the background. C - Segmentation.

Foreground objects that stop and later on resume motion can pose a problem for many background modeling algorithms. Ideally the algorithm should retain knowledge of the background colors that are occluded by the object so that when the object continues its movement it won't mis-classify the background as a new object. If however the foreground object remains unmoving eventually the algorithm should assimilate it into the background model. Fig. 6.3 illustrates how the particles' ability to switch into the persistent form solves this problem. It shows a car pulling into a parking space, wait for a while, and pull out. As shown, the particle set gradually integrates the car's colors into the background model. When the car exits, the set holds information on both the car and the road, and so is able to correctly segment the scene. As the sequence continues more particles shift into tracking the car colors, and so if it had stayed it would have been merged into the background. Some of the other methods, encounter difficulties retaining the background distribution when faced with a static foreground objects that contains colors which are relatively similar to the background. The MoG technique blends the car into the background very quickly, and the histogram method creates a ghost segmentation when the car leaves. While the codebook algorithm maintains good representation of the car during the scene, its foreground mask contain a great degree of noise.

Some videos may contain low contrast or be in poor quality. The background model should be sensitive enough to detect all foreground objects in the scene, yet discriminative as to not include too much false positive detections. Fig. 6.4 shows detection results on a video in which visibility is reduced due to rain and fog. In this scenario the difference in quality between the methods is clear. Bacteria-Filters and histograms produce segmentations which are very clean, and yet they both manage to detect even the small shapes of pedestrians at the back of the image. The other two methods produce a great amount of noise in this low contrast video. The size and shape of the noise in the foreground mask is similar some of the actual foreground objects in the image. This makes it very hard for any post processing technique to clean it and still retain the desirable objects. As noticed before, Bacteria-Filters create fuller foreground objects than histograms.

The authors would like to thank the ViSOR repository[1], and Institut für

[1] http://www.openvisor.org

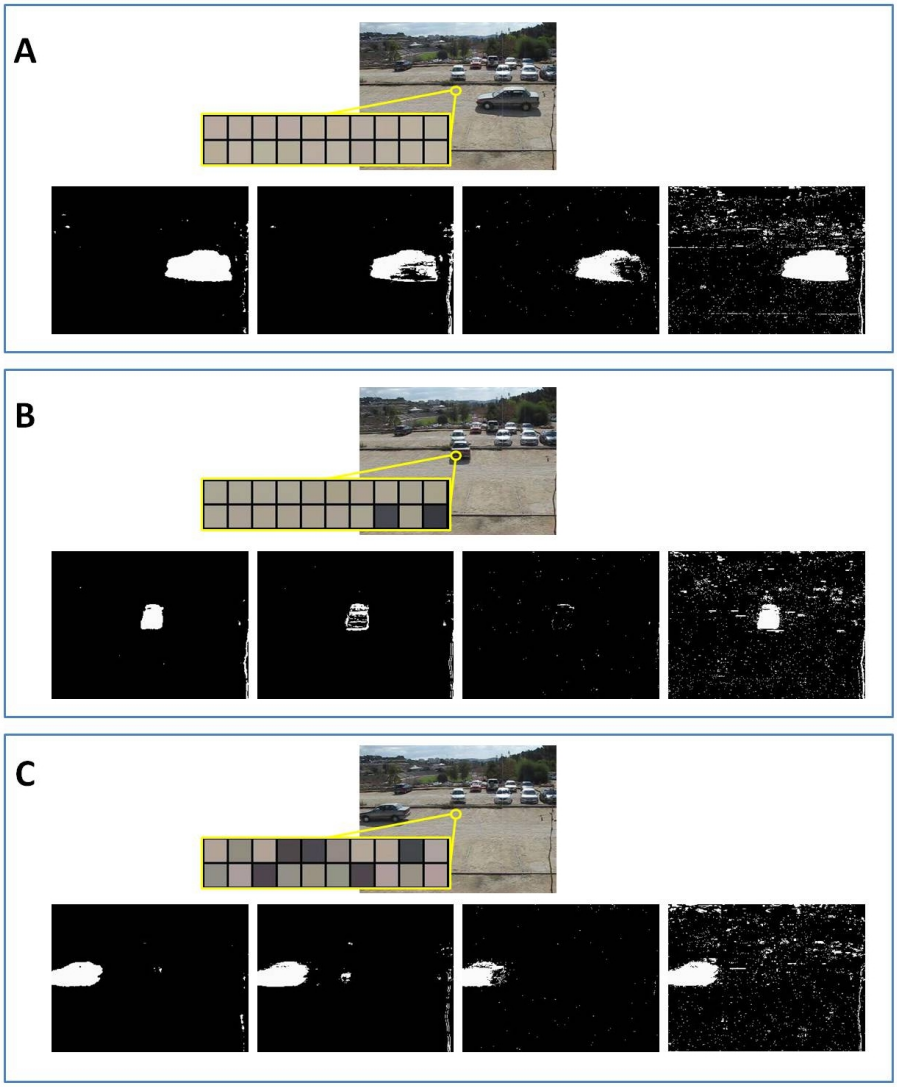

Figure 6.3: The particle set gradually adjusts to the new colors of the stopped foreground object allowing for accurate segmentation. Each block displays the results on one frame, with the top section illustrating the particle set of a pixel in the circled area. The columns are ordered from left to right: Bacteria-Filters, Histograms, Mixture of Gaussians, Codebook. A - shows the scene as the cars enters. The particle set holds only road colors. B - displays the segmentation when the car has just pulled into the parking space, note how the MoG technique quickly assimilates it into the background. C - shows the car as it exits the scene. The particle set holds both car and road colors.

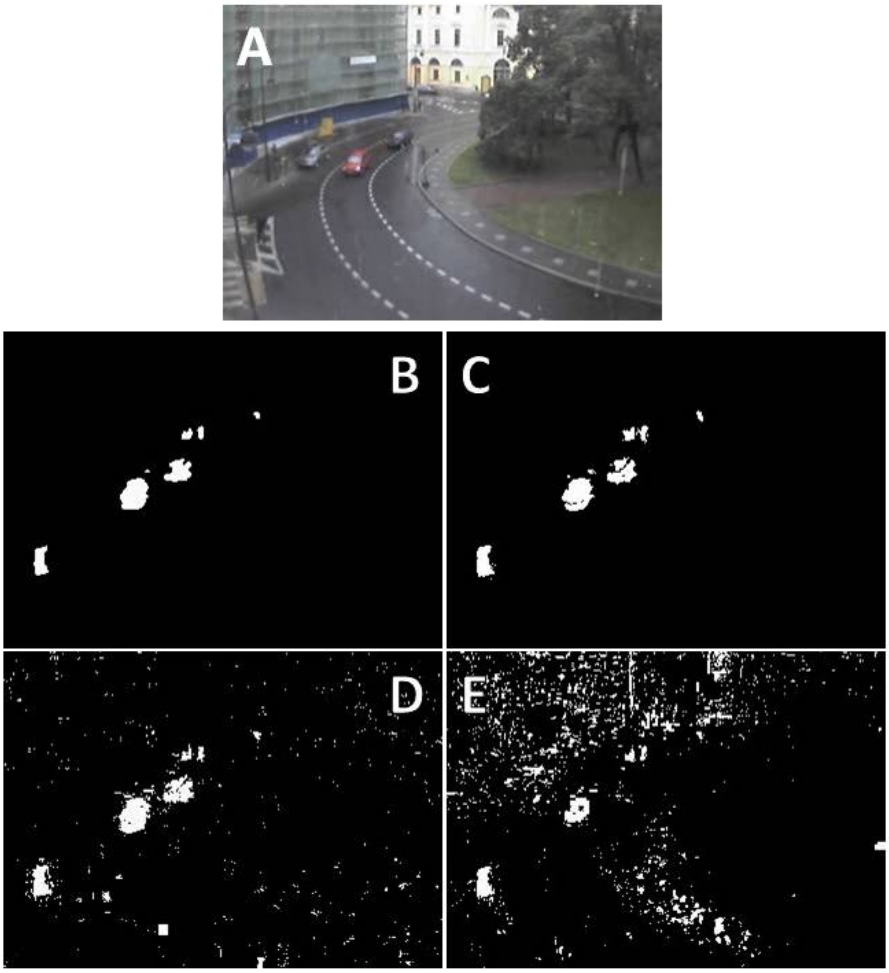

Figure 6.4: Rain and fog can create poor visibility in which foreground detection becomes more difficult. A - input, B - Bacteria-Filters, C - Histograms, D - Mixture of Gaussians, E - Codebook. In B and C, even the small detected object at the top corresponds to a real person.

Algorithmen und Kognitive Systeme[2] for providing some of the videos presented.

[2]http://i21www.ira.uka.de

Chapter 7

In Practice

Foreground detection is a basic building block in many computer vision applications. For an algorithm to be useful in practice a number of requirements must be met, and some modifications should be implemented. In this chapter we address the changes between a background subtraction algorithm, and a full foreground detection system.

7.1 Cleaning the foreground mask

The foreground mask which is the result of (4.6) may mis-classify pixels. There are two types of errors: false positives - in which pixels are marked as foreground when they belong to the background, and false negatives, in which some foreground pixels are missed. Usually some post processing is done to clean the foreground mask which, in some cases, remove false positive predictions. In other cases processing techniques such as Morphological operations can correct false negative results by closing holes in foreground areas. Recently there is growing popularity for using the min-cut algorithm to segment the image, instead of using a single threshold as in (4.6). The benefits of using this technique is that it takes into account not only the probability value of the pixel, but also its similarity to its neighbors. This allows the technique to address both problems simultaneously.

Min-Cut Segmentation

The foreground extraction task can also be addressed as a graph labeling procedure, where the nodes are the pixels of the input image, and each pixel r should be given a label L_r as either *foreground* ($L_r = 1$) or *background* ($L_r = 0$). In [Boykov and Jolly, 2001] it was shown that the labeling problem can be solved using min-cut if it is stated as an energy minimization problem.

$$E(L) = \delta \cdot \sum_{r \in I} E_d(L_r) + \sum_{(r,q) \in \nu} E_s(L_r, L_q) \tag{7.1}$$

where E_d is a data term providing external constraints (the probability of r belonging to the background), E_s is a smoothness term defined over neighboring pixels ν, and δ is a user defined weight, balancing the two terms.

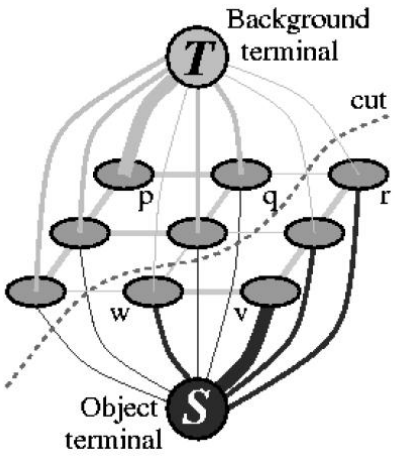

Figure 7.1: The graph induced from an image and the resulting minimal cut on it.

$$E_d(L_r) = \begin{cases} P_b(r) & L_r = 1 \\ 1 - P_b(r) & otherwise \end{cases} \tag{7.2}$$

The data term is the cost of assigning r with the label L_r.

$$E_s(L_r, L_q) = |L_r - L_q| \cdot e^{-\beta \cdot d_{r,q}} \tag{7.3}$$

where

$$d_{r,q} = \|I(r) - I(q)\|_2^2 \tag{7.4}$$

$$\beta = (2 \cdot \boldsymbol{E}[\|I(r) - I(q)\|_2^2])^{-}1 \tag{7.5}$$

The smoothness term is the cost of assigning pixel r and q different labels. The desired labeling is then achieved by minimizing the energy function using the implementation of the min-cut algorithm described in [Boykov and Kolmogorov, 2004]. A graph is induced from the image by creating a node for each pixel. Neighboring pixels are joined by edges with weights according to the smoothness term. Each node is also connected to a foreground terminal and an object terminal, with weights according to the data term. Figure 7.1 (Taken from Boykov and Jolly [2001]) shows an illustration of a graph induced from an image, and the resulting cut.

Using this technique can provide cleaner segmentation result as the algorithm takes into account the smoothness of the region around the segmentation border. Figure 7.2 shows the added accuracy gained by applying min-cut to the segmentation procedure. Note how in the bottom row using min-cut has cleaned pixels which have been wrongly classified as foreground, while in the top row it created full objects that are without holes.

Figure 7.2: By using the min-cut algorithm instead of a single threshold on the probability image better segmentations can be achieved. From left to right: input frame, background probability, foreground segmentation using the single threshold in (4.6), foreground segmentation using min cut. In the top row the foreground objects created by using min-cut are without holes which are found in the single threshold technique. In the bottom two rows pixels which are wrongly classified as foreground when using a threshold are not present when using min-cut.

7.2 Speed And Scalability

A foreground detection algorithm should consume minimal computational resources, and work close to real time. It is favorable if it can be easily scaled to a multi processor environment allowing for an increase of running speed proportional to the increase in computational power.

The algorithm we have introduced takes relatively little memory. It requires less than the histograms method, significantly so if the color channels are not assumed to be independent. While it takes more memory than the MoG technique, our method allows for non parametric distributions. As each pixel is considered independently, and in every pixel each particle is handled independently of its peers, our algorithm is extremely easy to implement for a multi processor (or even, for the case of GPGPU, massive multi processor) environments.

Chapter 8

Discussion

In this thesis, a novel background modeling technique has been introduced. Our algorithm is based on the well known Particle Filter framework which has been shown to be a robust and efficient technique for many applications. The Particle Filter algorithm is very successful in estimating a variety of probability density functions. Its popularity can be attributed to both its versatility and to the fact that it can be easily implemented.

We have modified the Particle Filter framework, allowing particles to become persistent, or dormant, and thus avoid being replaced due to a few outlier measurements, and a jump ability which enables particles to adapt quickly to multi modal distributions. These two changes help make the framework practical for background subtraction. The dormant particles allow for a small particle set to be used, in the order of 20 particles. This is due to their ability to retain their parameters even when the current frame contains significantly different colors. The jump ability allows for fast changes to the particle set if the scene contains high frequency temporal color changes.

It was shown that this technique can accurately model the background color distribution even when faced with repetitive background movement, multi modal backgrounds and illumination changes.

Bibliography

N. Q. Balaban, J. Merrin, R. Chait, L. Kowalik, and S. Leibler. Bacterial Persistence as a Phenotypic Switch. *Science*, 305(5690):1622–1625, 2004. doi: 10.1126/science.1099390. URL http://www.sciencemag.org/cgi/content/abstract/305/5690/1622. [cited on p. 13]

Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1124–1137, 2004. [cited on p. 24]

Y. Y. Boykov and M.-P. Jolly. Interactive graph cuts for optimal boundary region segmentation of objects in n-d images. In *Proc. Eighth IEEE International Conference on Computer Vision ICCV 2001*, volume 1, pages 105–112, July 7–14, 2001. doi: 10.1109/ICCV.2001.937505. [cited on p. 23, 24]

G. Bradski and A. Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly, Cambridge, MA, 2008. [cited on p. 17]

A. Doucet, S. Godsill, and C. Andrieu. On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and computing*, 10(3):197–208, 2000. [cited on p. 10]

A. Doucet, N. De Freitas, and N. Gordon. *Sequential Monte Carlo methods in practice*. Springer Verlag, 2001. [cited on p. 10]

O. Javed, K. Shafique, and M. Shah. A hierarchical approach to robust background subtraction using color and gradient information. In *IEEE Workshop on Motion and Video Computing*, volume 75, 2002. [cited on p. 3]

P. KaewTraKulPong and R. Bowden. An improved adaptive background mixture model for real-time tracking with shadow detection. In *Proc. European Workshop Advanced Video Based Surveillance Systems*, volume 1. Citeseer, 2001. [cited on p. 3, 17]

K. Kim, T. H. Chalidabhongse, D. Harwood, and L. Davis. Real-time foreground-background segmentation using codebook model. *Real-Time Imaging*, 11(3):172 – 185, 2005. ISSN 1077-2014. doi: DOI:10.1016/j.rti.2004.12.004. URL http://www.sciencedirect.com/science/article/B6WPR-4FV362T-1/2/64a99673b255f07c51631846435c3ba5. Special Issue on Video Object Processing. [cited on p. 5, 17]

T. Ko, S. Soatto, and D. Estrin. Background subtraction on distributions. In *Proceedings of the 10th European Conference on Computer Vision: Part III*, page 289. Springer, 2008. [cited on p. 4, 17]

D. Lee, J. Hull, and B. Erol. A Bayesian framework for Gaussian mixture background modeling. 3:973–976, 2003. [cited on p. 3]

C. Stauffer and W. Grimson. Adaptive background mixture models for real-time tracking. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 246–252, 1999. [cited on p. 3]

S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN 0262201623. [cited on p. 12]

K. Toyama, J. Krumm, B. Brumitt, and B. Meyers. Wallflower: Principles and practice of background maintenance. In *International Conference on Computer Vision*, volume 1, page 29. Kerkyra, Greece, 1999. [cited on p. 4]

C. Wren, A. Azarbayejani, T. Darrell, and A. Pentland. Pfinder: Real-time tracking of the human body. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):780–785, 1997. [cited on p. 3]

List of Figures