Foreground-Background Segmentation of Video Sequences

Yi Wang
Mentors: Pietro Perona, Claudio Fanti

Abstract

Automatic segmentation of foreground from background in video sequences has attracted lots of attention in computer vision. Two different implementations have been completed during the summer. The first one is known as "background subtraction". The foreground is segmented from the background by directly subtracting a mean background image B from each frame, and retaining those parts of the frame that differ the most from B.

The second method is derived from "Bilayer Segmentation of Live Video" by Criminisi et al, where a complex energy function depending on both segmentation labels and training image data is setup. The energy encodes terms that enforce coherence of segmentation in space-time as well as with image data. Some hand-labeling of a few sequences is necessary to build color and motion models. The segmentation is then estimated by minimizing this energy function.

Both approaches need to be trained before being applied to a sequence. While the first one is automatically trained, it performs rather poorly with even slight changes in luminosity. The second one, instead, requires a rather large amount of hand-labeling but performs a lot better even when the background undergoes some changes in color/light and in presence of small motions.

1. Introduction

This paper presents two implementations that are capable of separating a foreground layer from video sequences. The first one is a traditional method known as "background subtraction", and the other one is derived from "Bilayer Segmentation of Live Video" by Criminisi et al. They could be used as building blocks for two of the ongoing research projects in the Vision Lab, namely, the "fruit-fly" and the "human activity" projects.

2. Background Subtraction

This section describes the background subtraction algorithm, and presents the results of my implementation.

The basic method of background subtraction is to compare |frame – background| with a pre-defined threshold Th. If the difference of a pixel is larger than Th, then classify it as foreground; otherwise, claim that it is background.

2.1 The Implementation and Its Results

This algorithm was implemented and tested on one of the fly sequences. Because flies are very small compared with the scene and they are constantly moving around, the image of the scene's static background B can be automatically estimated by taking the average of a relatively small subset of the frames. At the same time, the standard deviation σ can be calculated for each pixel from that same subset of the frames. The assumption behind this algorithm is that the foreground is constantly moving while the background remains static.

The background image B is subtracted from every frame. The differences of background pixels should be near 0 and the differences of foreground pixels should be significantly larger than 0. The differences are compared with a pre-defined threshold Th (typically around 3σ). The values of the pixels whose differences are larger than Th are retained while the others are reset to 0. After this, most of the foreground pixels are segmented out.

Two C++ functions, meannstd.cpp and bgremoval.cpp (based on ProcessAVI.cc), and a Matlab function RemoveBG.m were written to implement this algorithm.

meannstd.cpp computes a mean frame and a standard deviation frame out of a video sequence. It takes an AVI file and returns its mean frame and its standard deviation frame. The syntax to call the function is [mean, std] = meannstd(videofile.avi, no_of_frames), where the variable mean is the output mean frame, std is the output standard deviation frame, videofile.avi is the name of the input AVI file, and no_of_frames is the number of frames the user want to use in the video to calculate the mean and the standard deviation. With larger no_of_frames, we can get more accurate results, but it takes more time. The frames used are evenly taken out of the video. For example, if we have a 1000-frame video file and set no_of_frames to be 10, then the 1st, 101st, 201st, 301st… 901st frames will be used to calculated the mean and standard deviation. The function can be compiled in Matlab and run as a Matlab function. Here are some performances of the function:

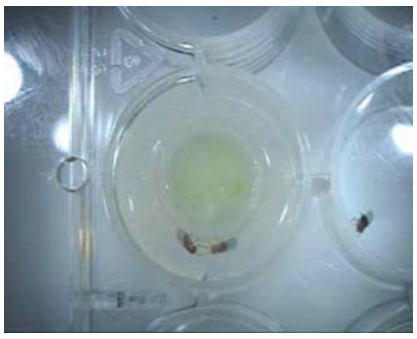

Figure 1. An image taken directly from a video sequence.

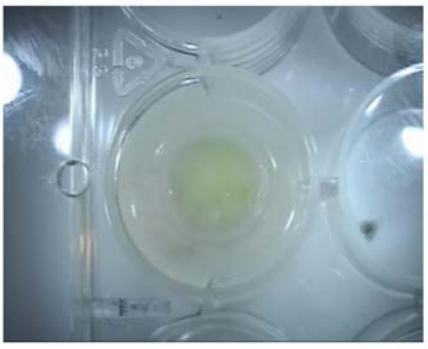

Figure 2. The mean frame out of 100 Frames

Figure 3. The standard deviation frame out of 100 frames

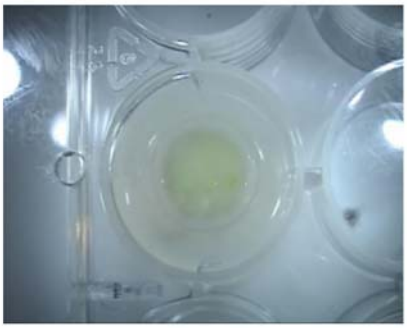

Figure 4. The mean frame out of all frames

bgremoval.cpp reads a video file, removes its background, and outputs the result in AVI or JPEG files. The background's mean and standard deviation are not calculated inside the function, but need to be passed to the function. They can be either the output of the meannstd function or computed from a specific background. The performance can be tuned by changing the values of certain variables. It gets better results when having better background information. Here is an example of how well it works:

Figure 6. A background removed frame

RemoveBG.m is just an interface function to meannstd.cpp and bgremoval.cpp to remove the background of a given video. It simply sets appropriate values of the variables and then calls meannstd function and bgremoval function.

2.2 Discussion of This Implementation

This sequence has about 1000 frames in total (roughly half a minute) and has lots of noise inside. As implied in the results, this implementation of the algorithm does not work very well and is sensitive to the threshold. One of the reasons is that in such a short sequence, the flies do not move very much, so the averaging procedure is unable to get a clean background. Besides, the threshold is not very well tuned. However, better

performance could be achieved by better estimation of the background and smoothing the frames before subtraction (takes longer time).

3. "Bilayer Segmentation of Live Video" Method

This method is derived from [1]. A complex energy function which depends on both the segmentation labels and the training image data is setup. The energy encodes terms that enforce coherence of segmentation in space (nearby pixels) and time (consecutive frames) as well as coherence of segmentation with image data (segmentation between foreground and background should hardly ever split a uniform region of the image; instead it should coincide with a high contrast area that arises were the color or the luminosity changes abruptly). Some hand-labeling of a few sequences is necessary in training in order to estimate models for how differently the foreground is colored and moves (with respect to the background). The segmentation is then computed by minimizing this energy function with respect to the segmentation labels. The minimization is accomplished by a binary graph cut package available online [3].

3.1 Notation and Image Observables

z: frames of a given input sequence in YUV color space. It is an array function of time t.

$\dot{\mathbf{z}}$: temporal derivatives of a given input sequence. $\dot{z}^t$ is the absolute difference between two Gaussian smoothed frames (Y color space channel only) at t and t-1.

g: spatial gradients of a given input sequence. They are computed by convolving the frames (Y color space channel only) with first-order derivative of Gaussian kernels.

m: motion observables $\mathbf{m} = (\mathbf{g}, \dot{\mathbf{z}})$.

The above 4 variables are observed directly from the input video.

α: segmentation labels of a given input sequence. α is a binary value, either F or B. Estimating **α** is the whole aim of the method.

3.2 The Energy Terms

In this method, the foreground/background segmentation is done by energy minimization. A complex energy function is setup, depending on the segmentation labels, the training image data, and the target video. The smaller the energy, the more correct the estimate. The energy function is the sum of four terms:

$\mathbf{V}^{\mathbf{T}}$ is the temporal prior term, which enforces the coherence of segmentation labels in time (3 consecutive frames).

$\mathbf{V}^{\mathbf{S}}$ is the spatial prior term, which imposes the coherence of segmentations labels in space (nearby pixels).

$\mathbf{U}^{\mathbf{C}}$ is the color likelihood term, which estimates the pixel labels based on color distributions in the foreground and background.

$\mathbf{U}^{\mathbf{M}}$ is the motion likelihood term, which estimates the pixel labels based on observations of motion observables in the ground-truth data.

3.3 Energy Minimization

One way to estimate $\boldsymbol{\alpha}$ is:

$$\hat{\boldsymbol{\alpha}}^t = \arg\min \quad \mathcal{E}_{\boldsymbol{\alpha}^{t-1}|\hat{\boldsymbol{\alpha}}^{t-1}} E(\boldsymbol{\alpha}^t, \boldsymbol{\alpha}^{t-1}, \hat{\boldsymbol{\alpha}}^{t-2}, \mathbf{z}^t, \mathbf{m}^t).$$

where

$$p(\boldsymbol{\alpha}^{t-1}|\hat{\boldsymbol{\alpha}}^{t-1}) = \prod_n p(\alpha_n^{t-1}|\hat{\alpha}_n^{t-1}),$$

and

$$p(\alpha^{t-1}|\hat{\alpha}^{t-1}) = \nu + (1-\nu)\delta(\alpha^{t-1}, \hat{\alpha}^{t-1}),$$

and ν typically is 0.1.

3.4 The Implementation and Its Results

Besides the four energy terms described in the paper, another spatial energy term is added to the total energy in order to get better results (at the price of more computation). This term is very similar to $\mathbf{V}^{\mathbf{S}}$, except that $\mathbf{z}$ is replaced by $\dot{\mathbf{z}}$. $\dot{\mathbf{z}}$ provides the information of the contours of the foreground in two consecutive frames and eliminates the contours of the background automatically. Though there are two contours in $\dot{\mathbf{z}}$, the right one will be selected with the help of other energy terms.

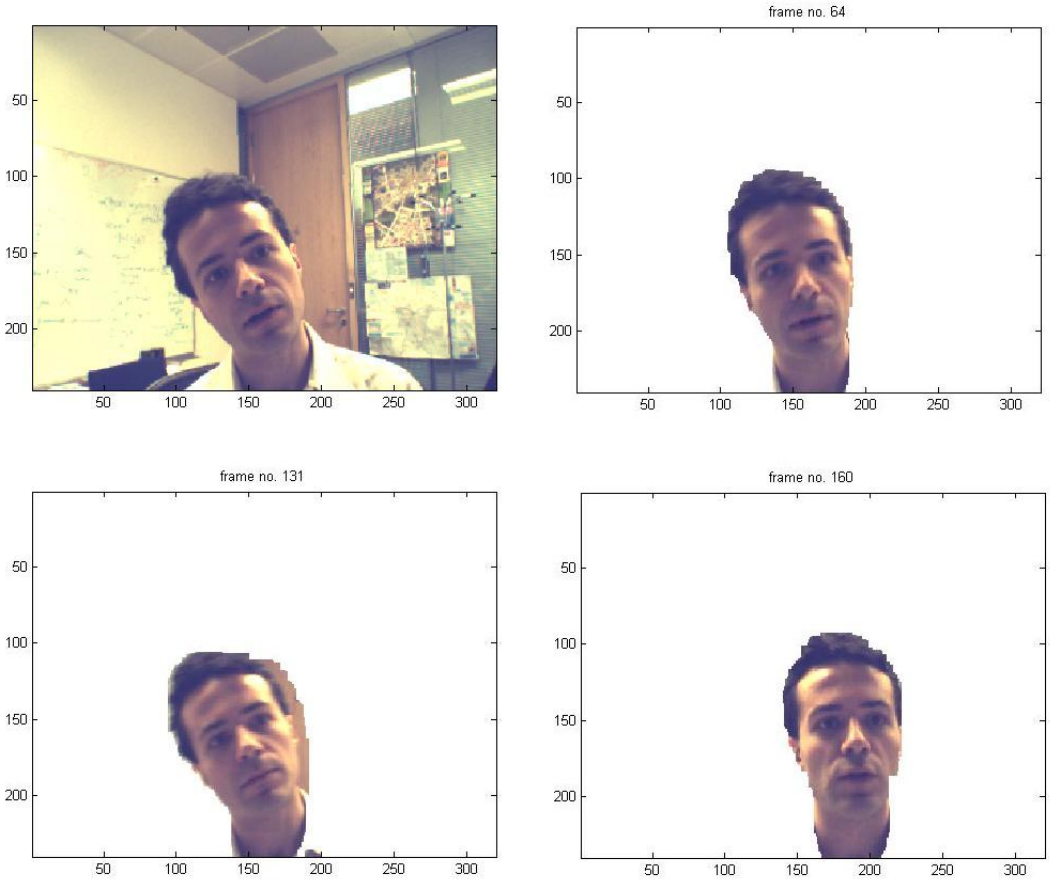

Figure 7. An example of automatic foreground/background segmentation in the monocular video sequence AC (available at [2]). The person is extracted from the sequence despite some segmentation error. Better performance could be achieved by further tuning the parameters.

3.5 Discussion of This Implementation

The implementation of this algorithm is capable of automatically segmenting foreground and background in monocular sequences, even if the background undergoes some changes in color/luminosity and in presence of small motions. Unlike background subtraction, it does not require prior knowledge of the background, so it works well even if the background cannot be obtained by averaging frames. However, the algorithm needs to be trained by lots of hand-labeled segmentation data to build the temporal prior, color, and motion models. Besides, there are no optimal values for the weights of the four energy terms – they need to be tuned for different sequences. This implementation does not work well if the background has similar color as the foreground (the foreground cannot be identified by the color model) and there is little motion in the foreground (a

pixel could be considered to be foreground even if there is little/no motion presented, according to the trained motion model).

4. Conclusions

Both implementations are capable of automatic foreground/background segmentation. The implementation of background subtraction is very easy and fast. However, it works only if the static background is known or can be estimated, which is the situation of the "fruit-fly" project. The implementation of "Bilayer Segmentation of Live Video" works well and does not require a known background. However, it needs to be trained by ground-truth extensively for different environments which takes lots of time. Besides, it is much slower than the other implementation (about 2fps, which may be OK for video conferencing but definitely not for real-time tracking of flies).

Acknowledgements

The author acknowledges helpful discussions with P. Perona, C. Fanti, and A. Criminisi.

References

[1] A. Criminisi, G. Cross, A. Blake, and V. Kolmogorov. Bilayer Segmentation of Live Video. CVPR 2006.

[2] http://research.microsoft.com/vision/cambridge/i2i/DSWeb.htm.

[3] http://www.adastral.ucl.ac.uk/~vladkolm/software.html