Change detection in videos*

Thomas B. Moeslund Rasmus R. Paulsen

Fall 2022

1 Introduction

In this note, we give a basic introduction to *change detection in videos*. The first goal of change detection is to detect if something has change *enough* in a video stream so further actions should be taken. For example the system could raise an alarm if there is movement in a storage-house. It could also trigger a second set of algorithms that can identify and track specific types of objects like cars, humans or animals. We will focus on the first step, namely to just detect if there is a change in a video stream.

A video sequence is in principle a sequence of images. Therefore, image analysis algorithms working on single images apply equally well to a video sequence. We simply process one image at a time. There is, however, two differences between a video sequence and an image. First, working with video allows us to consider temporal information and hence identify and track objects based on their motion. Secondly, video acquisition and image acquisition may not be the same, and that can have some consequences. Below this is discussed.

2 Video Acquisition

A video camera is said to have a certain *frame rate*. The frame rate is a measure for how many images the camera can capture per second and is measured in Hertz (Hz). The frame rate depends on the number of pixels (and the number of bits per pixel) and the electronics of the camera. Usually the frame rate is geared towards a certain transmission standard like USB, Firewire, Camera Link, etc. Each of these standards has a certain *bandwidth*, which is the amount of data that can be transmitted per second. With a fixed bandwidth we are left with a choice between high resolution of the image and a high framerate. When one goes up the other one goes down. In the end the desired frame rate and resolution will always depend on the concrete application.

Say we have a system including a camera with a frame rate of $20Hz$. This means that a new image is captured every 50 millisecond(ms). But it also means that the image processing algorithms can spend a maximum of 50ms per image. To underline this we often talk about two frame rates; one for the camera and one for the image processing algorithms. The overall frame rate of a system is the smallest of the two frame rates.

Another important factor in video acquisition is compression. Very often the captured video sequence needs to be compressed in order to insure a reasonable framerate/resolution. The more the video is

*This note is based on the book: Thomas B. Moeslund, *Introduction to Image and Video Processing - Building Real Systems and Applications*. Springer 2012

compressed the higher the frame rate/resolution, but the worse the quality of the decompressed video. The question is of course if the quality loss associated with video compression is a problem in a particular application or not? This topic is beyond the scope of this note, but should be considered for real world applications.

A video sequence may contain *motion blur* due to motion in the scene. A similar problem is that the depth-of-field may not cover the entire field-of-view and hence moving objects may be blurred due to an incorrect focus. Processing video containing blur will possibly affect the results and should therefore be avoided is possible. Processing solutions also exist but are also out of the scope of this note.

In conclusion, there are more factors to be aware of when acquiring and processing video sequences than still images taken under very controlled situations.

3 Detecting Changes in the Video

In many systems we are interested in detecting what has changed in the scene, i.e., a new object enters the scene or an object is moving in the scene. For such purposes we can for example use image subtraction to compare the current image with a previous image or a reference image. If they differ, the difference defines the object or movement we are looking for. In the rest of this chapter we will elaborate on this idea and present an approach for detecting changes in video data.

3.1 Change detection overview

The algorithm for detecting changes in a video sequence consists of a number of steps:

1. Estimate and save a reference image
2. Capture and pre-process the current image
3. Perform image subtraction or compare the current image with the reference in another way
4. Thresholding
5. Filter noise
6. Take a decision based on the difference image

The algorithm can be performed in several different ways depending on the goal and assumptions. In the following, we will explain the above steps.

3.1.1 Reference image estimation

Static background If the background in the scene can be assumed to be static then every new object entering the scene can in theory be identified by subtracting an image of the background from the current image. This process is denoted *background subtraction* and illustrated in figure 1. The reference image of the background is captured as on of the first images when the system commences.

Changing background Another way the algorithm can be performed is when the assumption of a static background breaks down. For example if the light in the scene changes significantly then an incoming image will be very different from the background even though no changes occurred in the scene. In such situations, the reference image can be the previous image. The rationale is that the background in two consecutive images from a video sequence is probably very similar and the only difference is the new/moving object, see figure 2. Such methods are denoted *image differencing*.

If the background is only slowly changing due to the movement of the sun for example, an alternative approach is to have a background image that is slowly changed based on the incoming frames. One approach can be considered as a time-wise low pass filtering of the incoming frames. This is explained later in this note.

The difference between the ways the algorithm can be performed results in two different types of reference images as listed above.

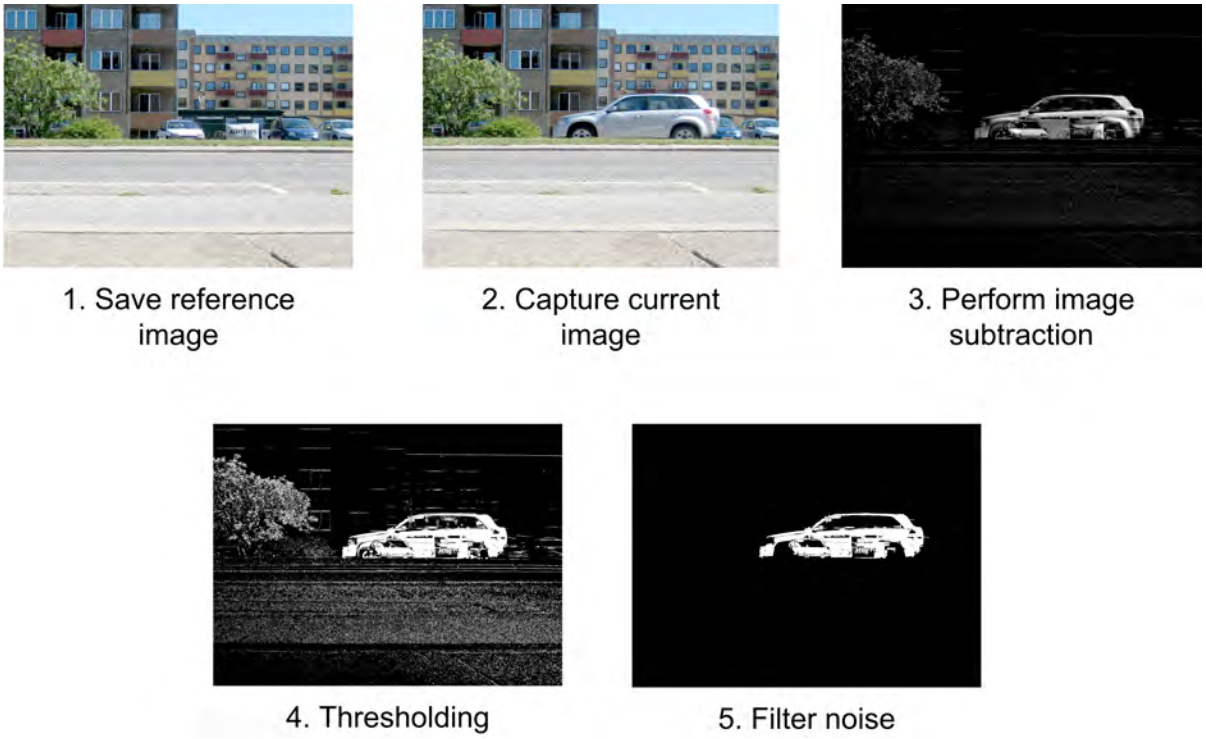


Figure 1: The steps of processing video data using background subtraction.

3.1.2 Image acquisition

When a new image is acquired from the image stream from the camera, some simple pre-processing can be applied. It can be converted from color to grayscale if the color information is not necessary. Sometime it can also be necessary to down-scale the image to save processing speed.

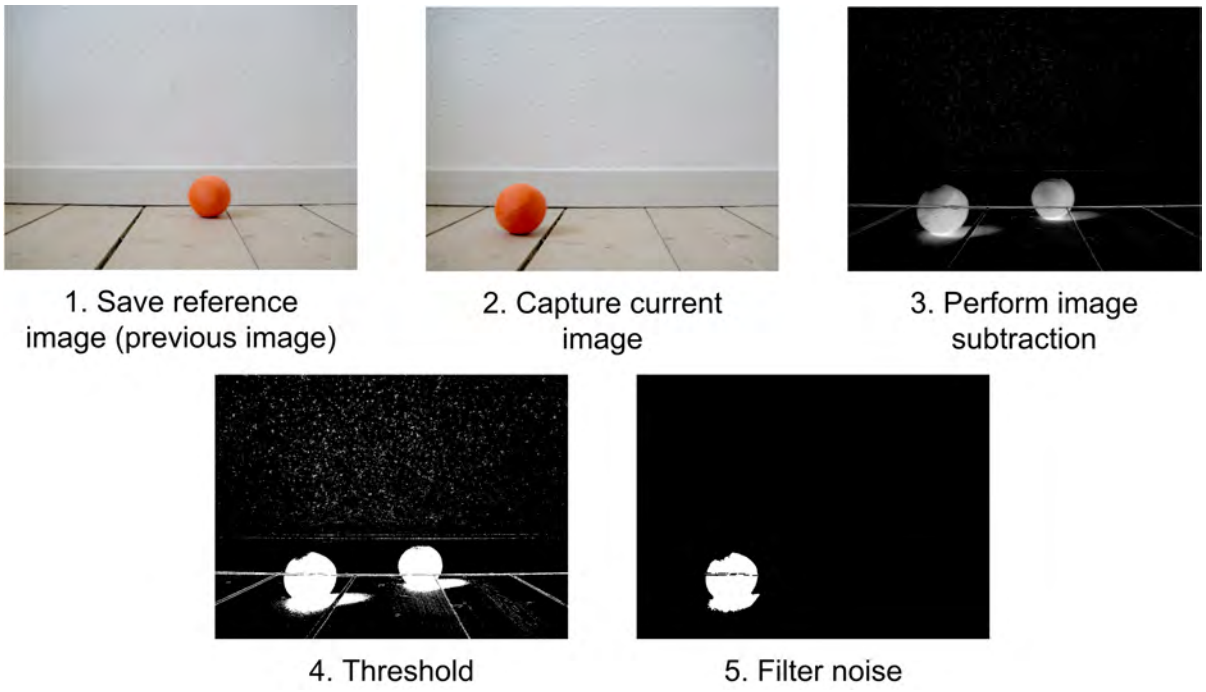


Figure 2: The steps of processing video data using image differencing.

3.1.3 Comparison with reference image

In this step, the newly acquired image is compared to the reference image. The simplest comparison is to subtract the reference image from the current image. Let us denote the reference image $r(x, y)$ and the current image $f(x, y)$. The resulting image, $g(x, y)$, is then given as:

$$g(x, y) = f(x, y) - r(x, y). \tag{1}$$

In figure 1, this step is shown. The car stands out in the resulting image since the pixel values of the car are different from the pixel values of the reference image. However, at some locations where a pixel in the reference image has a similar value to the pixel at the current image the resulting pixel value will be close to zero and hence not stand out. This can for example be seen around the wheels of the car.

As a designer you therefore need to introduce a background which is as different as possible from the object you intend to identify. For example, choosing a black background when identifying white balls is a very good idea, whereas a white background is obviously not.

Another issue regarding image subtraction, is that negative values are very likely to appear in the resulting image $g(x, y)$. Say that your task is to identify objects when they pass by your camera. The objects are black and white, meaning that they have pixel values which are either black, $f(x, y) \cong 0$, or white $f(x, y) \cong 255$. You then design a gray background which has intensity values around 100, i.e., $r(x, y) \cong 100$, and perform image subtraction:

$$g(x, y) \cong \begin{cases} 0, & \text{where the object is not present;} \\ 155, & \text{where the object is white;} \\ -155, & \text{where the object is black.} \end{cases} \tag{2}$$

A common error is to set a negative pixel to zero. If this is done then only the white parts of the object is detected. Note that whether $g(x, y) = 155$ or $g(x, y) = -155$ is equally important. The correct solution is to apply the absolute value of $g(x, y)$, $\text{Abs}(g(x, y))$.

3.1.4 Thresholding the difference image

The next step of the algorithm is simply a matter of binarizing the difference image $\text{Abs}(g(x, y))$ by comparing each pixel with a threshold value, T:

$$\text{Binary image} = \begin{cases} 0, & \text{if Abs}(g(x, y)) < T \\ 255, & \text{otherwise} \end{cases} \tag{3}$$

3.1.5 Remove noisy detections

The threshold in the previous step will, like any other threshold operation, produce noise due to an imperfect camera sensor, small fluctuations in the lighting, the object being similar to the background, etc. The noise will be in the form of missing pixels inside the silhouette of the object (false negatives and silhouette-pixels outside the actual silhouette (false positives). See figure 1 or 2 for examples.

The noise will in general have a negative influence on the quality of the results and we therefore need to remove the noise (if possible) using some kind of filtering. Small isolated silhouette-pixels outside the actual silhouette can often be removed using either a median filter or a morphologic opening operation. The holes inside the silhouette can often be removed using a morphologic closing operation. Which method to apply obviously depends on the concrete application.

3.1.6 Deciding if an action should be taken

The previous steps results in a binary image, where the number of foreground pixels indicate the change in the current image. Deciding which action should be taken depends on the application. It could for example be:

- Raise an alarm if a certain percentage of pixel has changed (for warehouse surveillance for example)
- Start or continue a vehicle tracking algorithm if the foreground pixels are approximately car sized and shaped (for vehicle tracking and counting).
- Start a face-detection algorithm if the changed pixels are approximately face sized and shaped (for human recognition).
- Start or continue a human-pose tracking algorithm (for sports or activity tracking).

3.2 Further approaches

The algorithm described above is quite simple and will break down in many situations. A lot of research has been done in making robust approaches working in different environments. Below is a short description of a few approaches.

3.2.1 Background Subtraction

Background subtraction is a simple and yet efficient method of extracting an object in a scene. This is especially true if the background can be designed to be uniform. In indoor and controlled setups this is indeed realistic, but for more complicated scenarios, other methods might be necessary. Even in the case of a controlled setup several issues must be considered:

1. Is the background really constant?
2. How to define the threshold value, which is used to binarize the difference image?
3. When should an action be taken based on the information in the binary image?

When you point a video camera at a static scene, for example a wall, the images seem the same. Very often, however, they are not. The primary reason being that artificial lighting seldom produces a constant illumination. Furthermore, if sunlight enters the scene, this will also contribute to the non-constant illumination due to the randomness associated with the incoming light rays. The effect of this is illustrated in figure 3. To the left an image from a static scene is shown. To the right two histograms are shown. The first histogram is based on the pixel values at position #1 for a few seconds and similar for the second histogram. If the images are actually the same, the histograms would contain only one non-zero bin. As can be seen this is not the case and in general no such thing as a static background exists.

Say that the pixel at position #2 in the first image of the video sequence has a value of 80 (not very likely according to the histogram, but nevertheless possible). If the first image is used as the reference image, then typical background images (around 100 according to the histogram) will result in a difference around 20. Depending on the threshold value, this could actually be interpreted as an object in the scene, since it seems different from the reference image. This is obviously not desirable and each pixel in the reference image should therefore be calculated as the *mean* of the first N images. The reference image at this particular position will then be around 100, which is much more appropriate according to the histogram. So to make the background subtraction more robust the first few seconds of processing should therefore be spent on calculating a good reference image.

Sometimes the background changes during processing. For example due to the changing position of the sun during the day or due to changes in the illumination sources, e.g., they are accidentally moved. In such situations a new reference image should be calculated. But how do we detect that this has happened? One way is, of course, if we can see that the performance of the system degrades. An automatic way is to gradually change the value of each pixel in the reference image in the following way:

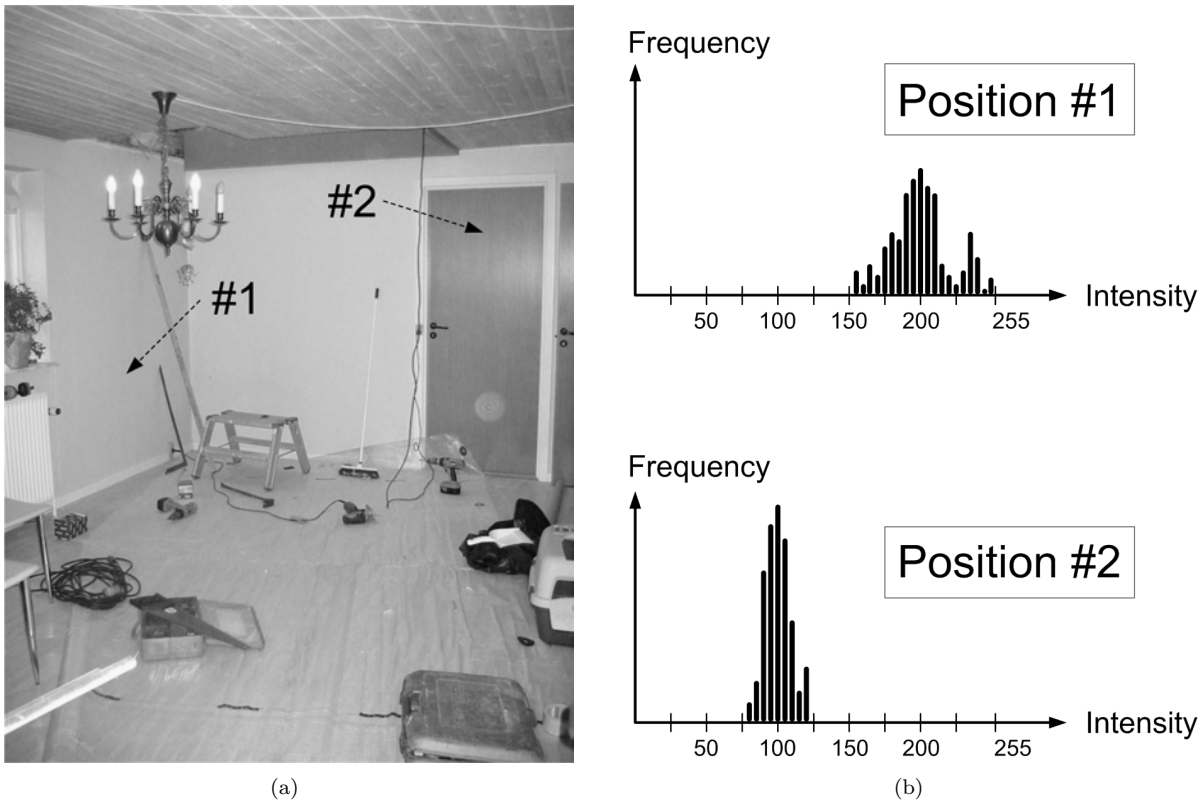
$$r_{new}(x, y) = \alpha \cdot r_{old}(x, y) + (1 - \alpha) \cdot f(x, y) \tag{4}$$

where $r(x, y)$ is the reference image, $f(x, y)$ is the current image, and α is a weighting factor that defines how fast the reference image is updated. The value of α depends on the application, but a typical value is $\alpha = 0.95$.

3.2.2 Defining the Threshold Value

As for any other threshold operation, defining the actual threshold value is a trade-off between false positives and false negatives which is application-dependent.

It is important to notice that equation 3 is actually based on the assumption that the histograms for different pixel positions are similar and only differ in their mean values. That is, it is assumed that the



(a) (b)

Figure 3: a) A static background image. The two arrows indicate the position of two pixels. b) Histograms of the pixel values at the two positions. The data comes from a sequence of images.

variation in the histograms is similar. In order to understand the implications of this assumption let us have a closer look at the bottom histogram in figure 3 together with equation 3. Say we define the threshold value to 25. This means that an object in an image needs to have a value below 75 or above 125 in order to be identify as an object pixel and not a background pixel. This seems fine. But then have a look at the top histogram in figure 3. Clearly this histogram has a larger variation and applying a threshold of 25 will result in incorrect identification of pixel values in the intervals: $[150, 175]$ and $[225, 255]$.

In many situations different histograms will occur simply because the different parts of the scene are exposed to different illumination conditions, which yields histograms with different variations. For example, you could have some parts of the background which move slightly (due to a draft for instance) and this will create a larger variation. So to sum up the above, the problem is that each position in the image is a associated with the same *global threshold* value.

The solution to this potential problem is to have a unique threshold value for each pixel position! Finding these manually, is not realistic simply due to the number of pixels and the threshold values are therefore found automatically by the use of the standard deviation for each pixel position. So when the mean of each pixel is calculated, so is the standard deviation. Equation 3 is therefore reformulated as:

$$\text{Binary image} = \begin{cases} 0, & \text{if } Abs(g(x,y)) < \beta \cdot \sigma(x,y) \\ 255, & \text{otherwise} \end{cases} \tag{5}$$

where β is a scaling factor and $\sigma(x, y)$ is the standard deviation at the position (x, y). Since β is the same for every position, we have no more parameters to define than above, but now the thresholding is done with respect to the actual data, hence a *local threshold*. This approach is similar to the statistical test called the *t-test*.

3.2.3 Image Differencing

If the assumption of a static background is violated significantly then background subtraction will produce incorrect results. In such situations we can apply image differencing to detect changes in a scene. As stated above, image differencing operates as background subtraction, except for the fact that the reference image is now a previous image.

Image differencing is simple and can efficiently measure changes in the image. Unfortunately the method has two problems. The first is a lack of detecting new objects which are not in motion. Say a new object enters a scene. As long as the object moves, image differencing detects this in the image subtracting process, but if the object stops moving, the reference image will be equal to the current image and hence nothing is detected. This is a clear weakness compared to background subtraction, which is indifferent to whether the new object is moving or not, as long as the appearance of the object is different from the background.

The other problem associated with image differencing is the notion of *ghost objects* illustrated in figure 4. The figure contains artificial images from a sequence where an object is moving horizontally through a scene. To make it simple, the object is a square with uniform gray-scale value. What can be seen is that the image differencing produces two segments (smaller objects). One originates from the current object and the other one from the object in the reference image - where the object was. This latter segment type is denoted a ghost object, since no object is present. A ghost object can also be seen in figure 2.

If the goal is only to obtain the coarse motion in the image, then this does not matter. If, however, we are only interested in the position of the object in the current image, then we need to remove ghost objects.

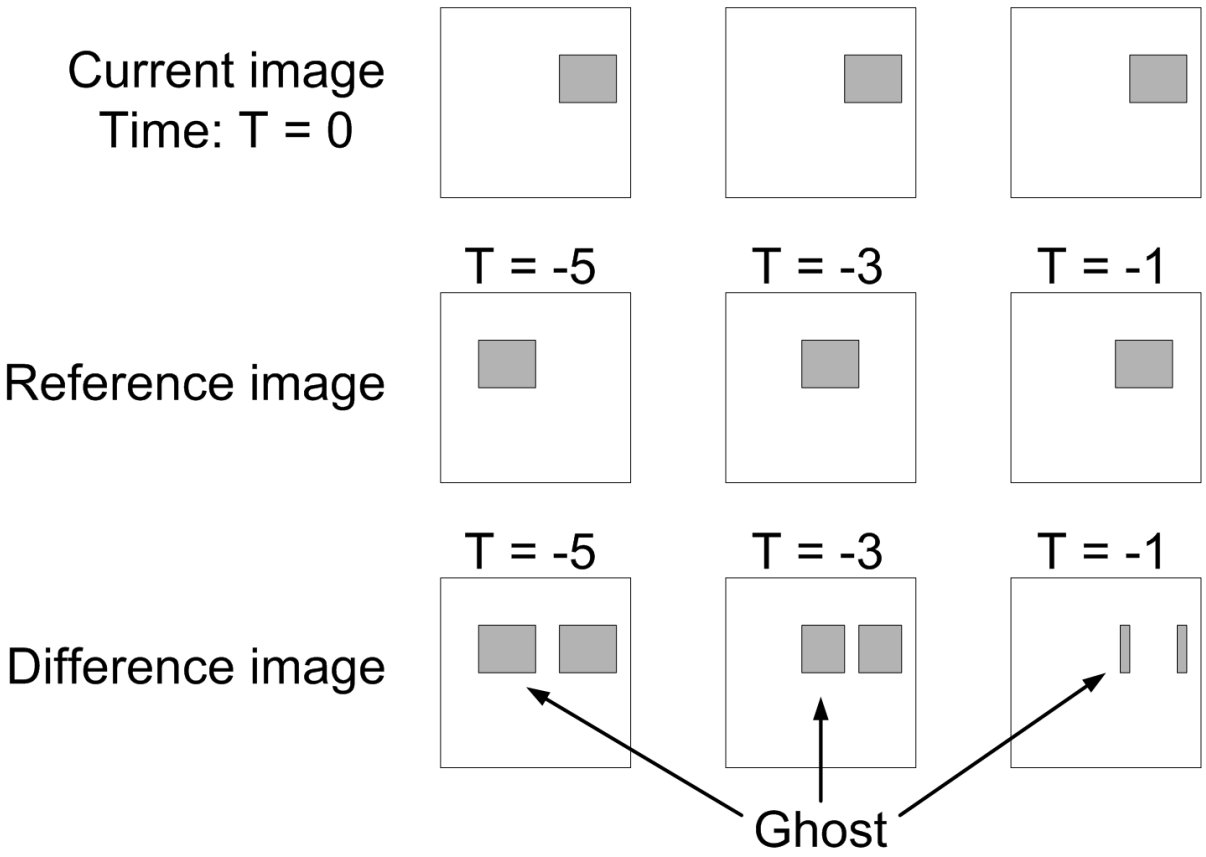


Figure 4: Image differencing. The effect of changing the reference image.

One approach for doing so is if we know the moving direction of the object. We can then infer which is the object and which is the ghost. Another approach is if we know that the object is always brighter than the background. Then the pixels belonging to the ghost will have negative values after the image subtraction.

It should also be noticed that when the object is overlapping in the reference and current image, then we only detect a part of the object, as seen in figure 4. If we know the size and speed of the object we can calculate how long time there should be between the reference image and the current image to avoid overlap. Or in other words, the reference image need not be the previous image $T = -1$, it can also be for example $T = -5$, see figure 4.

3.2.4 Advanced reference image estimation

Background subtraction can be a powerful allied when it comes to identifying objects in a scene. The method however has some build-in limitations that are exposed especially when processing video of outdoor scenes. First of all, the method requires the background to be empty when learning the background model for each pixel. This can be a challenge in a natural scene where moving objects may always be present.

One solution is to median filter all training samples for each pixel. This will eliminated pixels where an object is moving through the scene and the resulting model of the pixel will be a true background pixel. An extension is to first order all training pixels (as done in the median filter) and then calculate the average of the pixels closest to the median. This will provide both a mean and variance per pixel. Such approaches assume that each pixel is covered by objects less than half the time in the training period.

Another problem that is especially apparent when processing outdoor video is the fact that a pixel may cover more than one background. Say we have a background pixel from a gray road. Imagine now that the wind sometime blows so a leaf covers the same pixel. This will result in two very different backgrounds for this pixel; a greenish color and a grayish color. If we find the mean for this pixel we will end up with something in between green and gray with a huge variance. This will render a poor identification of this pixel during background subtraction. A better approach is therefore to define two different background models for this pixel; one for the leaf and one for the road. Recent approach defines several statistical models per pixel adaptively, to be able to dynamically create a flexible background estimation. This is beyond the scope of this note.

Yet another problem in outdoor video is shadows due to from strong sunlight. Such shadow pixels can easily appear different from the learnt background model and hence be incorrectly classified as object pixels. Different approaches can be followed in order to avoid such misclassifications. First of all, a background pixel in shadow tends to have the same color as when not in shadow - just darker. A more detailed version of this idea is based on the notion that when a pixel is in shadow it often means that it is not exposed to direct sunlight, but rather illuminated by the sky. And since the sky tends to me more bluish, the color of a background pixel in shadow can be expected to be more blueish too. Secondly, one can group neighboring object pixels together and analyze the layout of the edges within that region. If that layout is similar to the layout of the edges in the background model, then the region is likely to be a shadow and not an object.