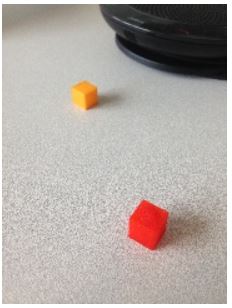
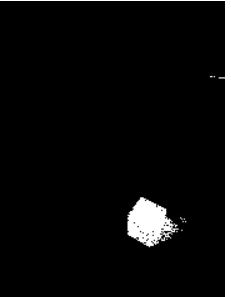
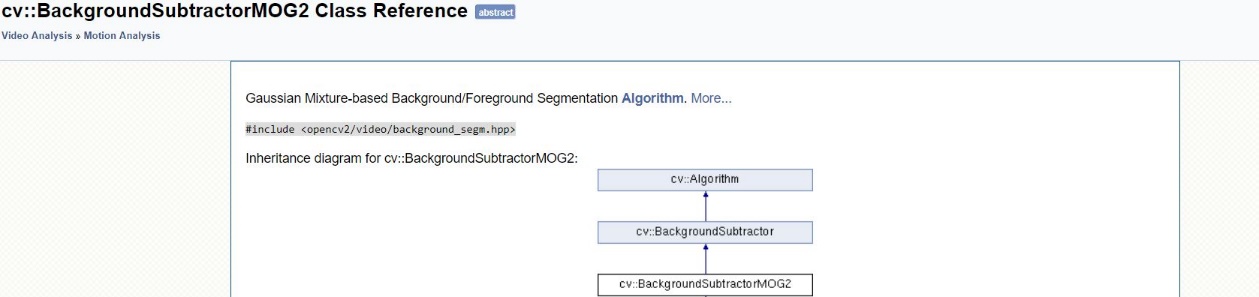
cv2.createBackgroundSubtractorMOG2(Motion Analysis)

In this step we are going to detect shadow motion based on background subtraction. OpenCV has few implementations of Background Segmentation. We will be looking at one of those. The class is called BackgroundSubtractorMOG2. It is a Gaussian Mixture-based Background Segmentation Algorithm. This algorithm takes the background pixels and assigns a Gaussian distribution to each one. The weight of this distribution is the amount of time the colors stay in the scene. Basically, the algorithm tries to identify the background by the information from the Gaussian mixture. The idea is that the longer the color stays the higher the probability that it is a part of the background. The Gaussian distribution helps this method to adapt to variance in illumination. This class supports parallel computing.

**[259]**

Zoran Zivkovic and Ferdinand van der Heijden. Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern recognition letters*, 27(7):773–780, 2006.

**[260]**

Zoran Zivkovic. Improved adaptive gaussian mixture model for background subtraction. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 2, pages 28–31. IEEE, 2004.

Foreground detection

**Foreground detection** is one of the major tasks in the field of [computer vision](https://en.wikipedia.org/wiki/Machine_vision) and [image processing](https://en.wikipedia.org/wiki/Image_processing) whose aim is to detect changes in image sequences. **Background subtraction** is any technique which allows an image's foreground to be extracted for further processing (object recognition etc.).

All detection techniques are based on modelling the background of the image, i.e. set the background and detect which changes occur. Defining the background can be very difficult when it contains shapes, shadows, and moving objects. In defining the background it is assumed that the stationary objects could vary in color and intensity over time.

Scenarios where these techniques apply tend to be very diverse. There can be highly variable sequences, such as images with very different lighting, interiors, exteriors, quality, and noise. In addition to processing in real time, systems need to be able to adapt to these changes.

A very good foreground detection system should be able to:

* Develop a background (estimate) model.
* Be robust to lighting changes, repetitive movements (leaves, waves, shadows), and long-term changes.

Background Subtraction

Background subtraction is a widely used approach for detecting moving objects in videos from static cameras. The rationale in the approach is that of detecting the moving objects from the difference between the current frame and a reference frame, often called "background image", or "background model". Background subtraction is mostly done if the image in question is a part of a video stream. Background subtraction provides important cues for numerous applications in computer vision, for example surveillance tracking or human poses estimation.

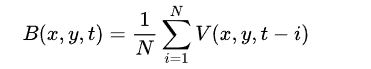
Background subtraction is generally based on a static background hypothesis which is often not applicable in real environments. With indoor scenes, reflections or animated images on screens lead to background changes. Similarly, due to wind, rain or illumination changes brought by weather, static backgrounds methods have difficulties with outdoor scenes. [[1]](https://en.wikipedia.org/wiki/Foreground_detection#cite_note-Piccardi-1)

Conventional approaches

A robust background subtraction algorithm should be able to handle lighting changes, repetitive motions from clutter and long-term scene changes.[[2]](https://en.wikipedia.org/wiki/Foreground_detection#cite_note-cs.utexas-2) The following analyses make use of the function of *V*(*x*,*y*,*t*) as a video sequence where *t* is the time dimension, *x* and *y* are the pixel location variables. e.g. *V*(1,2,3) is the pixel intensity at (1,2) pixel location of the image at *t* = 3 in the video sequence.

**Using frame differencing**

A motion detection algorithm begins with the segmentation part where foreground or moving objects are segmented from the background. The simplest way to implement this is to take an image as background and take the frames obtained at the time t, denoted by I(t) to compare with the background image denoted by B. Here using simple arithmetic calculations, we can segment out the objects simply by using image subtraction technique of computer vision meaning for each pixels in I(t), take the pixel value denoted by P[I(t)] and subtract it with the corresponding pixels at the same position on the background image denoted as P[B].

In mathematical equation, it is written as:

{\displaystyle P[F(t)]=P[I(t)]-P[B]}

The background is assumed to be the frame at time *t*. This difference image would only show some intensity for the pixel locations which have changed in the two frames. Though we have seemingly removed the background, this approach will only work for cases where all foreground pixels are moving and all background pixels are static.[[2]](https://en.wikipedia.org/wiki/Foreground_detection#cite_note-cs.utexas-2) [[3]](https://en.wikipedia.org/wiki/Foreground_detection#cite_note-Motion_Detection-3) A threshold "Threshold" is put on this difference image to improve the subtraction (see Image thresholding).

{\displaystyle |P[F(t)]-P[F(t+1)]|>\mathrm {Threshold} }

This means that the difference image's pixels' intensities are 'thresholded' or filtered on the basis of value of Threshold. [[4]](https://en.wikipedia.org/wiki/Foreground_detection#cite_note-Advanced_Motion_Detection-4) The accuracy of this approach is dependent on speed of movement in the scene. Faster movements may require higher thresholds.

**Mean filter**

For calculating the image containing only the background, a series of preceding images are averaged. For calculating the background image at the instant *t*,

{\displaystyle B(x,y,t)={1 \over N}\sum \_{i=1}^{N}V(x,y,t-i)}

where *N* is the number of preceding images taken for averaging. This averaging refers to averaging corresponding pixels in the given images. *N* would depend on the video speed (number of images per second in the video) and the amount of movement in the video.[[5]](https://en.wikipedia.org/wiki/Foreground_detection#cite_note-5) After calculating the background *B*(*x*,*y*,*t*) we can then subtract it from the image *V*(*x*,*y*,*t*) at time *t* = t and threshold it. Thus the foreground is

{\displaystyle |V(x,y,t)-B(x,y,t)|>\mathrm {Th} }

where Th is threshold. Similarly we can also use median instead of mean in the above calculation of *B*(*x*,*y*,*t*).

Usage of global and time-independent thresholds (same Th value for all pixels in the image) may limit the accuracy of the above two approaches{\displaystyle d=|(I\_{t}-\mu \_{t})|}

Reference : 1) <http://layer0.authentise.com/segment-background-using-computer-vision.html>

2) <https://en.wikipedia.org/wiki/Foreground_detection>

3) <https://docs.opencv.org/3.4.4/de/de1/group__video__motion.html>

cv2.inRange (Thresholding Operations using inRange)

Applies a threshold to each array element on a range of pixel values we need.

The function applies thresholding to a multiple-channel array. The function is typically used to get a bi-level (binary) image out of a grayscale image ( **compare** could be also used for this purpose) or for removing a noise, that is, filtering out pixels with too small or too large values. There are several types of thresholding supported by the function. They are determined by type parameter.

Here we will take a specific color range with is black from the picture to detect the head count of humans.

cv2.threshold (Applies an adaptive threshold to an array.)

Here, the matter is straight forward. If pixel value is greater than a threshold value, it is assigned one value (may be white), else it is assigned another value (may be black). The function used is **cv2.threshold**. First argument is the source image, which **should be a grayscale image**. Second argument is the threshold value which is used to classify the pixel values. Third argument is the maxVal which represents the value to be given if pixel value is more than (sometimes less than) the threshold value. OpenCV provides different styles of thresholding and it is decided by the fourth parameter of the function. Different types are:

* cv2.THRESH\_BINARY
* cv2.THRESH\_BINARY\_INV
* cv2.THRESH\_TRUNC
* cv2.THRESH\_TOZERO
* cv2.THRESH\_TOZERO\_INV
* **THRESH\_BINARY**

dst(x,y) =  \fork{\texttt{maxValue}}{if $src(x,y) > T(x,y)$}{0}{otherwise}

* **THRESH\_BINARY\_INV**

dst(x,y) =  \fork{0}{if $src(x,y) > T(x,y)$}{\texttt{maxValue}}{otherwise}

where T(x,y) is a threshold calculated individually for each pixel.

* For the method ADAPTIVE\_THRESH\_MEAN\_C , the threshold value T(x,y) is a mean of the \texttt{blockSize} \times \texttt{blockSize} neighborhood of (x, y) minus C .
* For the method ADAPTIVE\_THRESH\_GAUSSIAN\_C , the threshold value T(x, y) is a weighted sum (cross-correlation with a Gaussian window) of the \texttt{blockSize} \times \texttt{blockSize} neighborhood of (x, y) minus C . The default sigma (standard deviation) is used for the specified blockSize .

cv2.morphologyEx(Morphological Transformations)

Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images. It needs two inputs, one is our original image, second one is called **structuring element** or **kernel** which decides the nature of operation. Two basic morphological operators are Erosion and Dilation. Then its variant forms like Opening, Closing, Gradient etc also comes into play.

**1. Erosion**

The basic idea of erosion is just like soil erosion only, it erodes away the boundaries of foreground object (Always try to keep foreground in white). So what it does? The kernel slides through the image (as in 2D convolution). A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero).

So what happends is that, all the pixels near boundary will be discarded depending upon the size of kernel. So the thickness or size of the foreground object decreases or simply white region decreases in the image. It is useful for removing small white noises (as we have seen in colorspace chapter), detach two connected objects etc.

### 2. Dilation

It is just opposite of erosion. Here, a pixel element is '1' if atleast one pixel under the kernel is '1'. So it increases the white region in the image or size of foreground object increases. Normally, in cases like noise removal, erosion is followed by dilation. Because, erosion removes white noises, but it also shrinks our object. So we dilate it. Since noise is gone, they won't come back, but our object area increases. It is also useful in joining broken parts of an object.

### 3. Opening

Opening is just another name of **erosion followed by dilation**. It is useful in removing noise.

i.e here we have used cv2.morphologyEX(mask,cv2.MORPH\_OPEN,kernelOp)

where kernelOp=np.ones((3,3),np.uint8)

### 4. Closing

Closing is reverse of Opening, **Dilation followed by Erosion**. It is useful in closing small holes inside the foreground objects, or small black points on the object.

i.e here we have used cv2.morphologyEX(mask,cv2.MORPH\_CLOSE,kernelCl)

where kernelCl=np.ones((11,11),np.uint8)

cv2.findContours(Structural Analysis and Shape Descriptors)

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition.

* For better accuracy, use binary images. So before finding contours, apply threshold or canny edge detection.
* findContours function modifies the source image. So if you want source image even after finding contours, already store it to some other variables.
* In OpenCV, finding contours is like finding white object from black background. So remember, object to be found should be white and background should be black.
* In case of a raster image, the spatial moments \texttt{Moments::m}_{ji} are computed as:
* \texttt{m} _{ji}= \sum _{x,y}  \left ( \texttt{array} (x,y)  \cdot x^j  \cdot y^i \right )
* The central moments \texttt{Moments::mu}_{ji} are computed as:
* \texttt{mu} _{ji}= \sum _{x,y}  \left ( \texttt{array} (x,y)  \cdot (x -  \bar{x} )^j  \cdot (y -  \bar{y} )^i \right )
* where (\bar{x}, \bar{y}) is the mass center:
* \bar{x} = \frac{\texttt{m}_{10}}{\texttt{m}_{00}} , \; \bar{y} = \frac{\texttt{m}_{01}}{\texttt{m}_{00}}
* The normalized central moments \texttt{Moments::nu}_{ij} are computed as:
* \texttt{nu} _{ji}= \frac{\texttt{mu}_{ji}}{\texttt{m}_{00}^{(i+j)/2+1}} .