Background segmentation in video sequences

Objectives:

- To learn how to obtain a scene model to separate (segment) moving objects.
- To learn how to use the interfaces that OpenCV provides for live video input/output (from a webcam) or from files: *cv::VideoCapture*, *cv::VideoWriter*.

For validating your code, use the videos provided in Moodle.

1. Minimum (Up to 7/10)

A very simple technique to detect objects in movements in a scene is to threshold the difference between two consecutive images | I_*t+1* - I_*t* |> U.

Depending on the threshold U, the points of the image with a significant change in its gray value will indicate points of moving objects in the scene (since the scene is assumed to be static).
As this technique is very sensitive to noise, it will cause a high level of “segmentation noise”.
One way to attenuate this noise is to use morphological operations of "opening" + "closing".
Write program called "*vseg.cc*" that opens a capture device (identified with a integer number) or a video file (*cv::VideoCapture*) and captures a sequence of images, being capable of separating moving objects. The program will show in a graphic window the original sequence of images and the segmented sequence.

```
vseg -v <input-pathname|int> -o <output-pathname> [-t <float>] [-s <int>]
```

The program will have a parameter *-t > 0* to set a detection threshold. It will also have a parameter *-s >= 0* for the radius of the structural element for morphological operations. The value 0 indicates not performing these operations and will be the default value.

Add a slider to manually set a threshold *U* [0-255]. Show in another window, frame by frame, the result of the detection of the “Foreground” as a binary image (0/255) and another with the clipping on the input video where only the colors of the detected object are shown. Suggestions: use the *cv::absdiff* function, the operator ">" to generate a segmented mask, and the bit operator '&' to apply the mask.

The program will generate an output video (*cv::VideoWriter*) with the segmented video (0/255).

Pressing the 'spacebar' key (*cv::waitKey*) will save the current image capture (*cv::imwrite*) in RGB with the background removed (set to 0) in a file named *out_xxx.png* where *xxx* will be the frame number starting from 0.

The program will end when either the user presses the "Escape" key (at any moment) or, if the input is from a video file, when it is finished.

2. Optional

A. (+1) If the user provides as an input argument an integer number to capture from a hardware device, add sliders (*cv::createTrackBar*) in the window with live video to set the exposure time / gain / brightness of the device (it depends on your hardware that you can modify all or only some). Investigate the properties that can be read / fixed with the *get/set* methods of *cv::VideoCapture* on your hardware (by trial/error).

B. (+2) Write a more advanced algorithm that consists of learning a model of the static scene to detect objects in motion. Let's assume that the value of a pixel *(y, x)* in time can be modeled as a Gaussian in RGB with mean M(*y, x*) = (*m_r, m_g, m_b*) and standard deviation Dev(*y, x*) = (*d_r, d_g, d_b*). Once the model is learnt, we will say that a pixel is on a moving object if its RGB value is very different with respect to the learnt model. That is, if for some channel |M(*y, x*) - I(*y, x*)| > *t* · Dev(*y, x*). To do this, keep in mind that:

   a. We work in RGB.
   b. During the first *X* frames (parameter *-b*), 100 by default, compute for each pixel its mean and deviation, learning a model of the static scene (the background) for each pixel.
   c. Then, for each new frame, apply the point operation |pixel - mean| >= *U*, with *U*= *t* * Dev, where *t* is a factor (parameter *-t*) that is set with typical values between 0.0 and 6.0, with steps 0.1. Use a slider to set it interactively. If a pixel does not meet the criteria in some channel, it is considered as *foreground*.
   d. The average/deviation model must be updated (so that it adapts dynamically to changes in the static scene), only for background pixels, every *Y* frames (parameter *-c*), 10 frames by default. You can control how fast/slow the system adapts to changes with an equation like: new_model = alpha * old_model + (1-alpha) * new_data with alpha (parameter *-a*) in the interval [0, 1].
   e. Suggestions: apply vector operations always you can. See for example *cv::accumulate*, *cv::accumulateSquare*.
   f. Command line:

```
vseg -v <int|input-pathname> -o <output-pathname> [-t <float>=2.5] [-b
<int>=100] [-c <int>=10 ][-a <float>=0.9] [-s <int>=0]
```