**“Vehicle Detection and Speed Estimation”**

**Explanation of Important OpenCV Functions**

**Perspective transformation**

* lambda = getPerspectiveTransform(inputQuad, outputQuad);

lambda is a pre-defined matrix, inputQuad is an input array which stores coordinates of quadrangle vertices in the source image, while outputQuad is an output array which stores coordinates of the corresponding quadrangle vertices in the destination image. This function is used to calculate a perspective transform from four pairs of the corresponding points.

* warpPerspective(input, output, lambda, output.size());

input is an input image (src), output is the output image (dst) that has the size = output.size(), lambda is the transformation matrix (M). This function applies a perspective transformation to an image using the specified matrix:



**Background subtraction**

* pMOG2 = createBackgroundSubtractorMOG2();

pMOG2 is a pre-defined background subtractor, this function is used to create MOG2 background subtractor.

* pMOG2->apply(frame, fgMaskMOG2);

frame is an input array image, fgMaskMOG2 is an output array image; the apply function is used to compute a foreground mask.

**Contours detection**

* Canny(src\_gray, canny\_output, thresh, thresh \* 2, 3);

src\_gray is a single-channel 8-bit input image, canny\_output is the output edge map, thresh is the first threshold for the hysteresis procedure, thresh \* 2 is the second threshold for the hysteresis procedure, 3 is the default aperture size for the Sobel() operator. This function finds edges in the input image image and marks them in the output map using Canny algorithm. The smallest value between 1st threshold and 2nd threshold is used for edge linking. The largest value is used to find initial segments of strong edges.

* findContours(canny\_output, contours, hierarchy, RETR\_TREE, CHAIN\_APPROX\_SIMPLE, Point(0, 0));

canny\_output is an 8-bit single-channel binary image, contours is an output array of arrays which stores detected contours, each contour is stored as a vector of points. hierarchy is an optional output vector which contains information about the image topology. It has as many elements as the number of contours. For each i-th contour contours[i] , the elements hierarchy[i][0] , hiearchy[i][1] , hiearchy[i][2] , and hiearchy[i][3] are set to 0-based indices in contours of the next and previous contours at the same hierarchical level, the first child contour and the parent contour, respectively. If for the contour i there are no next, previous, parent, or nested contours, the corresponding elements of hierarchy[i] will be negative. RETR\_TREE is the mode that retrieves all of the contours and reconstructs a full hierarchy of nested contours. CHAIN\_APPROX\_SIMPLE is a method that compresses horizontal, vertical and diagonal segments and leaves only their end points. Point(0, 0) is the optional offset by which every contour points is shifted. This is useful if the contours are extracted from the image ROI and then they should be analyzed in the whole image context.

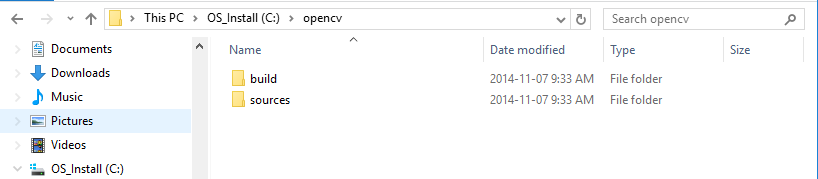
* drawContours(drawing, contours, i, color, 1, 8, hierarchy, 0, Point());

drawing is the destination image, contours is an array of arrays which stores detected contours. i is a parameter indicating which contour to draw. color is the color of the contours. 1 is the thickness of the line, 8 represents the line connectivity type. hierarchy is an optional information hierarchy, it’s only needed if you want to draw some of the contours. 0 is the maximal level for drawn contours, only the specified contour is drawn. Point() is the optional contour shift parameter, shifts all the drawn contours by the specified offset = (dx, dy).

**Approaches**

**Prerequisites:**

* Install and setup OpenCV 3.0.0 on the computer (OS: Windows)
* Usually, download OpenCV to C:\

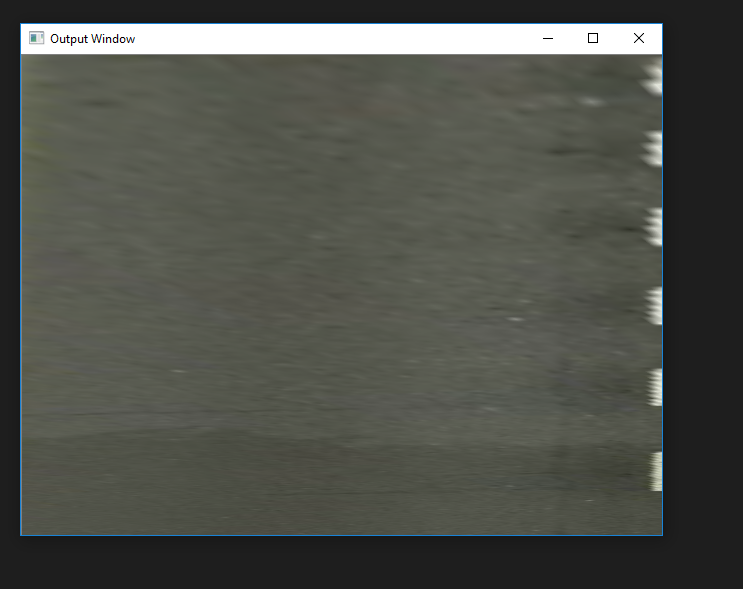
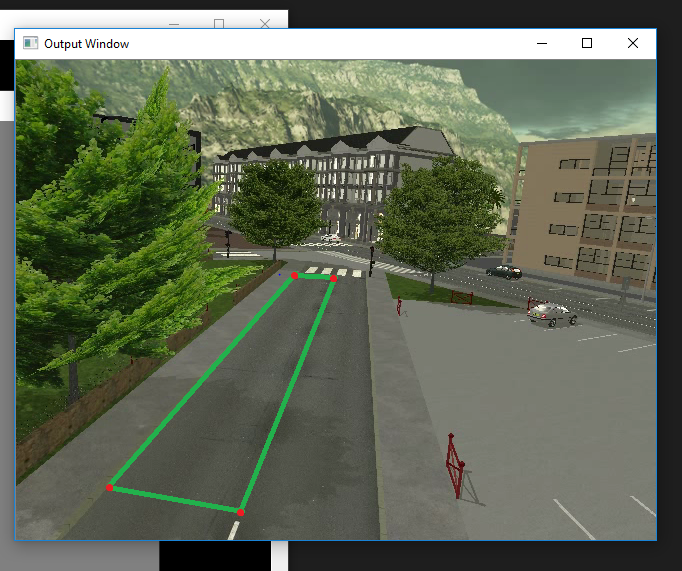


* Add it to Environment Variables by creating a new System Variable called **OPENCV\_DIR** with value **C:\opencv\build\x86\vc12**; and add a value **%OPENCV\_DIR%\bin** to the System Variable **Path**.
* Setup Visual Studio to include OpenCV library
* Create a new C++ project, and set up its properties
* Properties pages -> C/C++ -> General -> Additional Include Directories -> **$(OPENCV\_DIR)\..\..\include**
* Properties pages -> Linker -> General -> Additional Library Directories -> **$(OPENCV\_DIR)\lib**
* Properties pages -> Linker -> Input -> Additional Dependencies -> (add) **opencv\_ts300.lib;opencv\_world300.lib;**
* Download sample input frames to the project root from <http://bmc.iut-auvergne.com/?page_id=24>
* Setup project’s property, in Property Pages -> Configuration Properties -> Debugging -> Command Arguments -> **-img 111\_png/input/1.png**

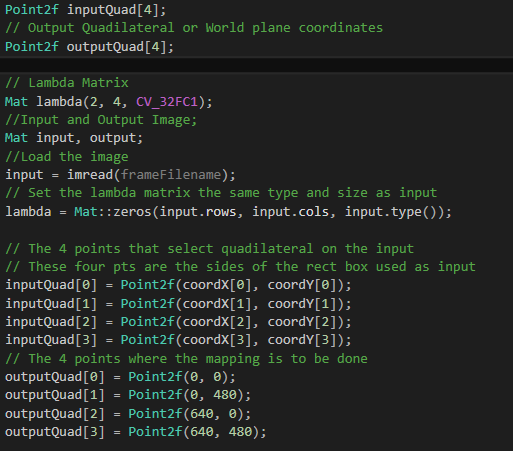
**Main Steps and Corresponding Results:**

1. Read frames in sequence (frames are captured from a video)
2. Perform perspective transformation by selecting 4 corner points from the first input frame. These 4 points consist a rectangle in real life, see the sample below, the left frame is the original input frame, while the right one is the result frame after performing perspective transformation. (Note: it’s important to select these 4 points in order, we should follow UP-LEFT -> BOTTOM LEFT -> UP-RIGHT -> BOTTOM-RIGHT by assumption.)

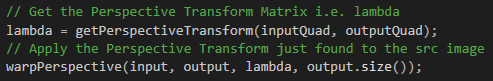
**Sample input & output:**



**Main code of Selecting 4 corner points:**

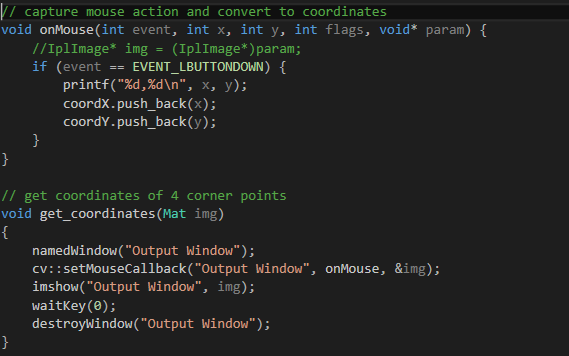


**Main code of performing Perspective Transformation:**



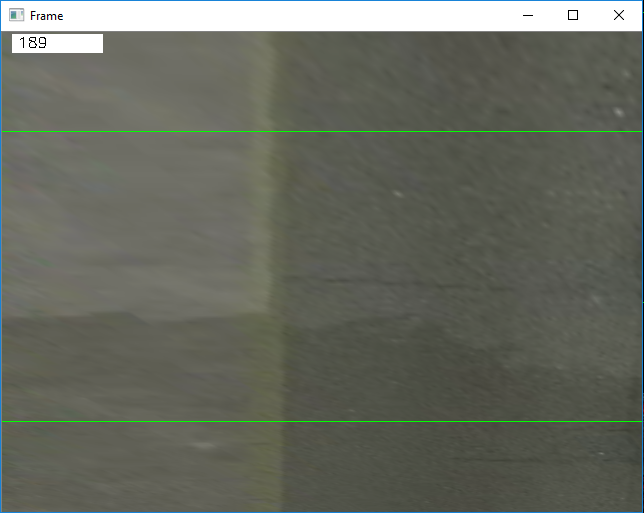
1. Retrieve coordinates of 4 corner points manually by mouse-click

**Main code:**

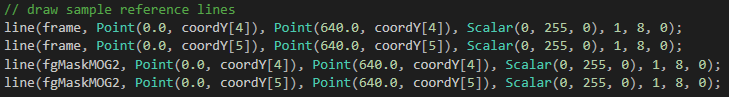


1. Draw reference lines on the transformed frame by mouse-click and also output them on fore ground masks, in order to estimate the speed, the distance between two parallel lines mean the distance the vehicle travelled

**Sample output (transformed frame with two reference lines):**

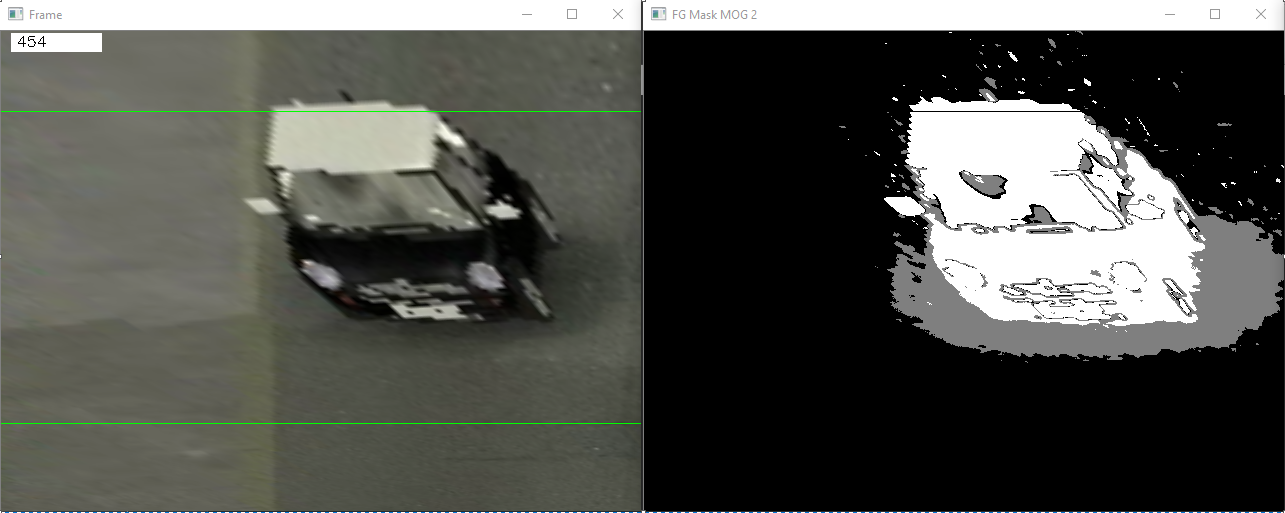


**Main Code of drawing reference lines:**



1. Perform background subtraction on the frames, to generate a foreground mask (a binary image contains the pixels belonging to moving objects). In general, it applies a subtraction between the current frame and a background model.

**Sample input & output:**



**Main code of performing Background Subtraction:**

* Define a global background subtractor variable



* Call the build-in background subtract function in OpenCV (Note: there are different approaches to perform background subtract, we choose to use MOG2 approach, because it has better accuracy and performance.)

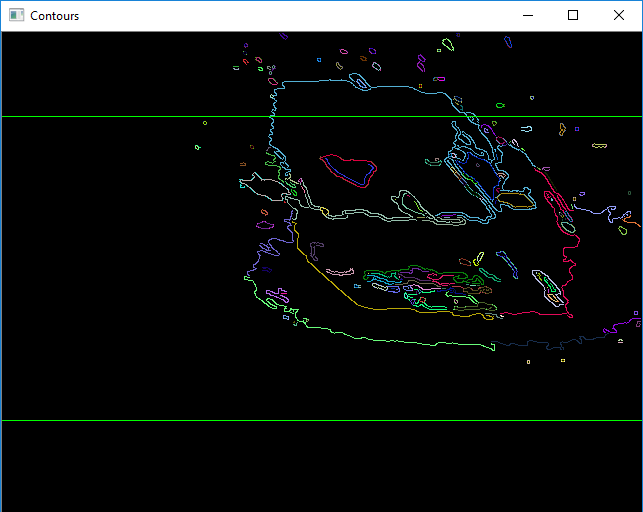


* Apply this approach on the frames

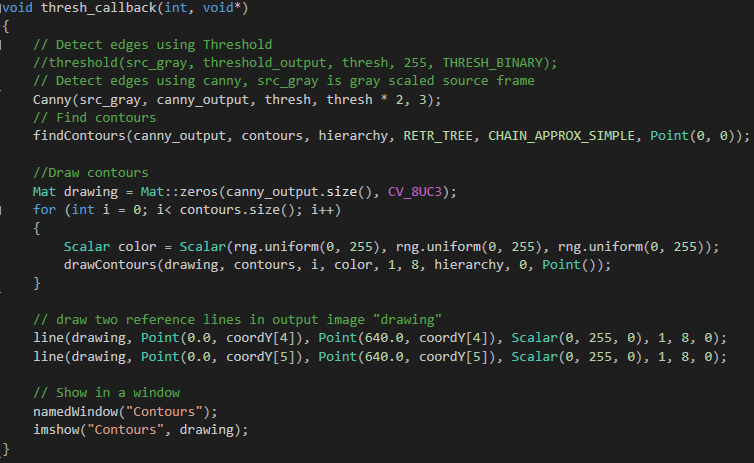


1. Apply canny edge detection and find/draw contours. For better accuracy, use binary images, and before finding contours, apply canny edge detection.

**Sample output (contours of a moving vehicle and two reference lines):**



**Main code of canny edge detection and finding contours:**



1. Estimate the speed of the detected moving vehicle, by assuming the distance between two parallel reference lines and fps, and then output the speed in the console.

**Main code of speed calculation:**

