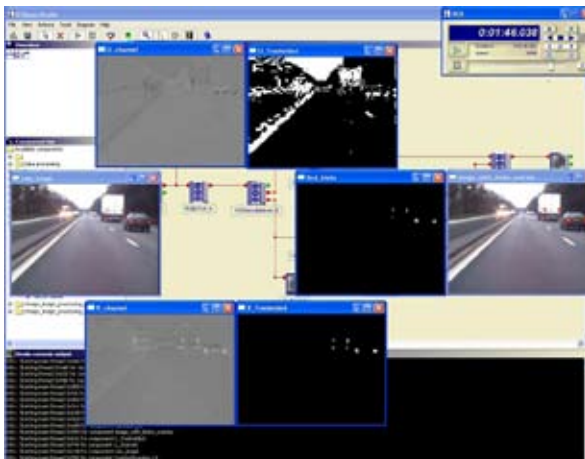


Red blobs detection with ^{RT}Maps...

Summary:

Find and display the red zones in an image.

We start with an RGB images flow, convert it to YUV in order to separate luminance and chrominance informations, apply threshold on the U and V channels of the resulting images, then find the bounding rectangles to the connex pixels that pass the thresholds and display the corresponding rectangle primitives onto the source image.



Duration: 10 minutes

Objectives:

- » Register packages of ^{RT}Maps components in the Studio and include them in your ^{RT}Maps application.
- » Connect components, display images
- » Perform some simple vision algorithms

Keywords:

- » ^{RT}Maps Packages
- » Computer vision
- » Image processing
- » Overlay drawing objects

Prerequisites (Before starting this tutorial, you should know how to):

- » Launch ^{RT}Maps Studio
- » Open an ^{RT}Maps database in a "Player" component and display data.

Used components:

- » Player
- » RGB to YUV converter
- » YUV demultiplexer
- » OpenCV image processing components
- » Labelizer
- » OverlayDrawing
- » ImageViewer

1. DataBase & Packages

Open the C3_highway database and display the video.


Launch RTMaps Studio



Place a "Player" component on your diagram and open the C3_highway database (located in .../Intempora/RTMaps 3.2/samples/databases/C3_highway/)

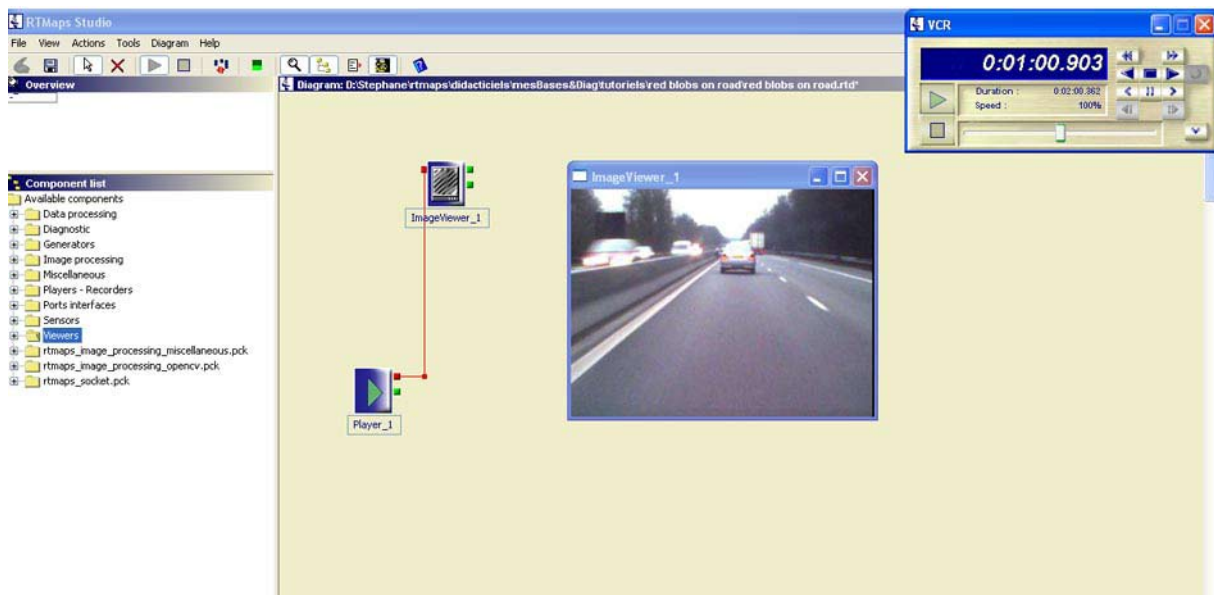
Place an "ImageViewer" component in front of the Player and connect the images output of the Player to the ImageViewer input.

Registering a package

In the RTMaps Studio toolbar, click on the "Register package" button (), or via the menu Actions | Register/Unregister package, then Add...

Browse for the package file called *rtmaps_image_processing_opencv.pck* located in .../Intempora/RTMaps 3.2/packages.

A new section in the *Component list* window appears with the name of the package. This section contains a bunch of image processing components made from the OpenCV (Open Computer Vision) library (<http://www.intel.com/technology/computing/opencv/>)



2. RGB & YUV Colorspace

Converting the image from the RGB colorspace to the YUV colorspace, and splitting the YUV channels

In section *Image processing* of the Component list, choose the *RGB -> YUV* component, and place it on the diagram on the right of the *Player*.

Connect the *Player* image output to the input of this component.

From the *Image processing* section, also choose the *YUV demultiplexer* component and place it on your diagram just after the *RGB to YUV converter*. This component will separate the Y, U and V values of each pixel of the input images and output them into 3 separate grayscale images.

Place 2 new *ImageViewer* components after the *YUV demultiplexer* in order to display the U and V planes (available on the 2nd and 3rd outputs of the *YUV demultiplexer*).

Why converting the RGB colorspace into YUV ?

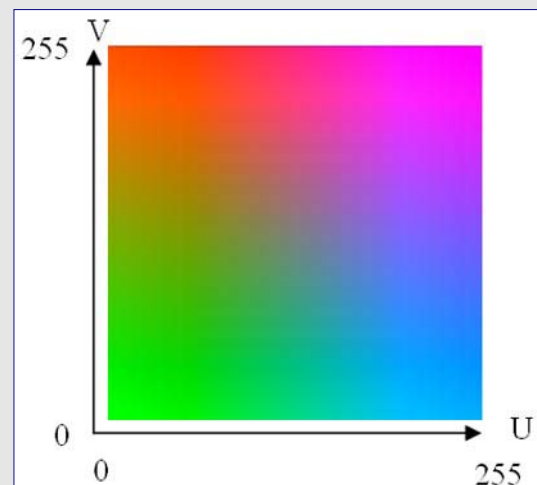
RGB means Red Green Blue. This means that each pixel of an image is represented by a value of Red, a value of Green, and a value of Blue, (the 3 primitive colors for a computer screen), each one between 0 and 255:

- » a pixel with $R=G=B=0$ is a black pixel
- » a pixel with $R=G=B=255$ is a white pixel
- » a pixel with $R=G=B=50$ is a dark gray
- » a pixel with $R=255$ and $G=B=0$ is a bright red and a pixel with $R=50$ and $G=B=0$ is a dark red.
- » etc...

The problem is that this way of representing colors does not separate the luminance information (the brightness) from the chrominance information (the color itself). So it is not possible to determine if a pixel is red or not for example by simply applying threshold on its R, G and B values.

The YUV colorspace transforms the R, G and B values in Y (luminance), plus U and V which contain the chrominance information.

Determining the Y, U and V values for a pixel is made via linear combinations of its R, G and B values. The following image represents the colors in the U, V coordinates plane:



The World Wide Web is full of resources about the many different colorspace that are available to represent images. For more informations about the YUV colorspace, refer to <http://en.wikipedia.org/wiki/YUV> for example.

At this point, it seems quite easy to apply simple thresholds to the U and V values of pixels in order to determine which pixels in the image are "red".

3. Threshold

Applying thresholds on the U and V planes

From the OpenCV components, place 2 *OpenCV_Threshold* components after the *YUV demultiplexer*. Send the U images to the 1st one, and the V images to the 2nd one.

Place 2 new *ImageViewer* components in order to display the results of the *thresholded* U and V pixels.

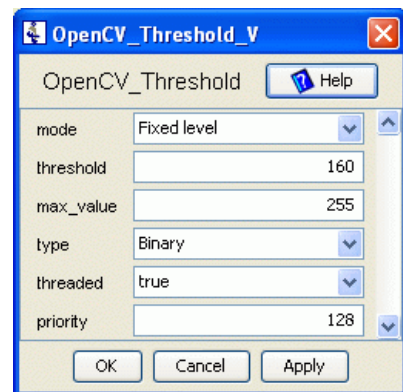
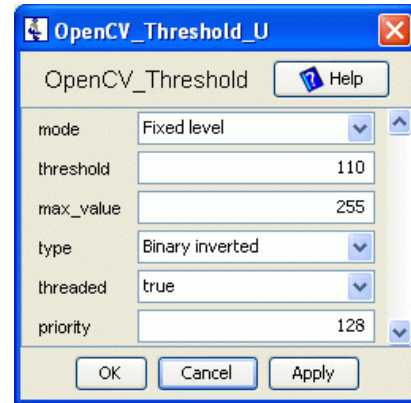
Edit the properties of the 2 *threshold* components (and keep in mind that we want to keep pixels whose U is below 110, and whose V is above 160, which corresponds more or less to the "red" color – see figure X : UV plane).

For the U thresholder, set *threshold* to 110, and *type* to *Binary inverted*. This means that all incoming pixels whose value is below 110 will be set to *max_value* (255) and all the other pixels will be set to 0.

For the V thresholder, set *threshold* to 160. Here we let the *type* property set to *Binary* (not inverted) so that all incoming pixels whose value is below 160 will be set to 0, and all other pixels will be set to "max_value" (255).

Finally, in order to fuse the pixels that have been thresholded, we need an *OpenCV_Logical* component, which will perform a bitwise AND operation on the two incoming images. This means we will only keep the pixels that have passed both the U and the V thresholds (the red ones !!!).

As usual, you can place an *ImageViewer* component to display the result...



4. Overlay Drawing

Determining the bounding rectangles of the connected pixels and overlaying them onto the source image:

The operation of extracting zones of connected pixels is called "Labelization".

Register the package called `rmaps_image_processing_miscellaneous.pck`. Place a *Labelizer* component from this package after the *OpenCV_Logical* component (and connect it of course). Its first output is called *rectDrawingObjects*.

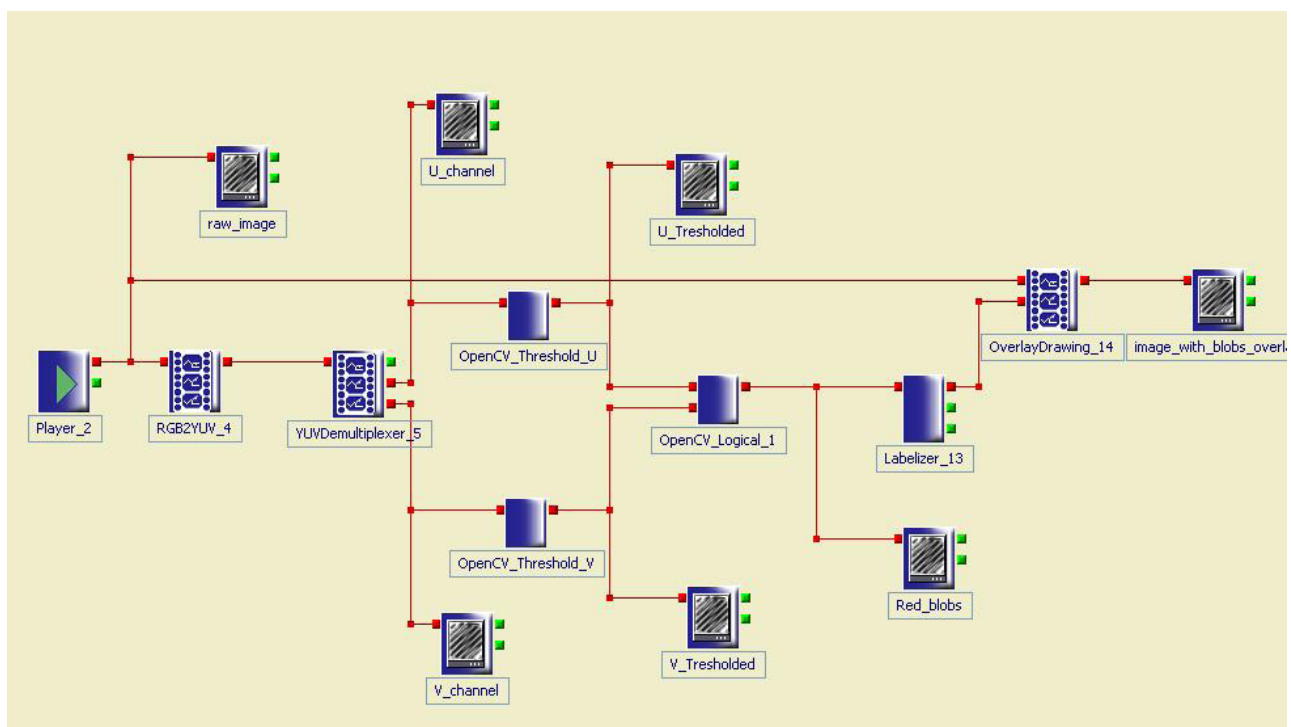
From the *Viewers* section in the component list, place an *Overlay drawing* component on your diagram after the *Labelizer*. Connect its first input to the source images (back to the first *Player* output), and its second input to the first *Labelizer* output (the rectangles).

The Drawing Object type in ^{RT}Maps

DrawingObject is a data type in ^{RT}Maps (the same way as images, integers, floating point numbers, stream of bytes, CAN frames...). They represent graphical primitives (such as rectangles, lines, circles, spots, ellipses, text messages) than can be overlaid onto images (via the *OverlayDrawing* component), or used to display schemas or maps (in the *Objects Viewer* component).

Now,... a new *ImageViewer* (the 7th if you followed me... and the last one), in order to display our image processing chain results !

After some relooking such as components renaming and connections breaks (right-click on a connection), your diagram could look like this:



You can now play with the threshold values in order to try to catch other colors from other databases, or from your webcam...