Laboratory 4: Detection and recognition

In this task you will work in pairs. The details will be announced at the practical session.

The deadline for this task is the second week of the second semester.

As usual follow the guidelines:

1. The script is not a step-by-step tutorial. It introduces the problem but it is your task to solve it.
2. If you get stuck, make sure that you read *Help* section at the end of the document.
3. If you still have problems, ask the lecturer, demonstrator or fellow students for help.
4. Try to do as much work as possible in the class, where it is easier to get help.
5. If you need help when working at home, use the Blackboard discussion board.
6. Finish all assignments a week before the deadline. If you get stuck, you will still have one week to ask for help.

1 Task: Find the sphere

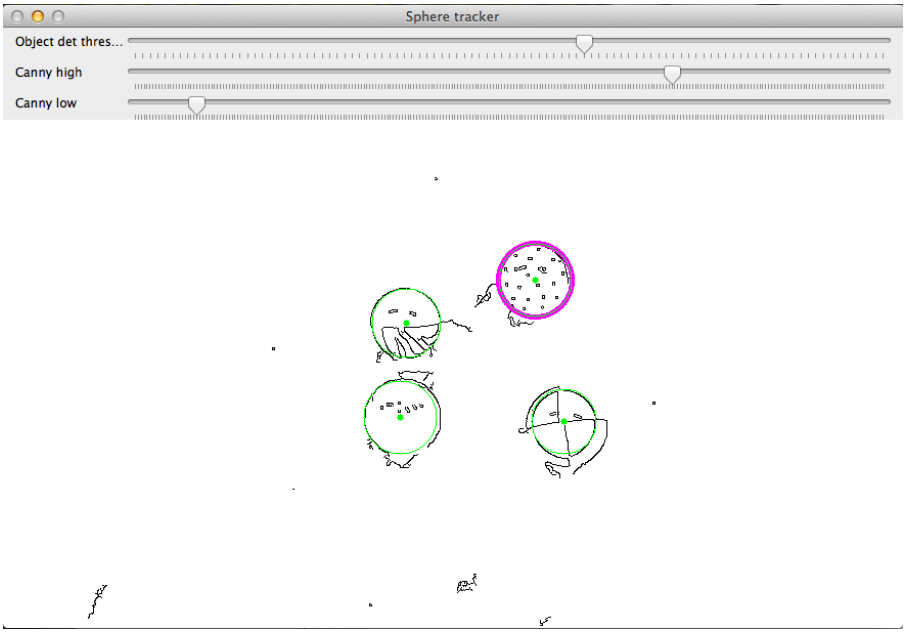

Figure 1: The application window. Detected spheres are marked with green and the recognized object with magenta.

Your task is to create an application that:

1. Reads a movie file whose name is passed as `argv[1]` argument. The movie will be played in a loop continuously. If no argument is passed, the application captures life video from the camera.

2. For each frame:

   (a) Plots the result of the Canny edge detector. This will help you finding the best parameters for the circle detection (next step).

   (b) Detects circles using the Hough transform and plots them in the current frame. Note that the OpenCV function for detecting circles using the Hough transform calls Canny edge detector internally. You will need to call Canny edge detector in addition to that to visualize the edge detection results.

   (c) If the mouse button is clicked on top of one of the circles, it is selected for recognition. The application extracts the descriptor for the selected circle and uses it to recognize the selected sphere in the subsequent frames. Once the sphere is recognized, it is plotted using a different colour than the rest of the detected circles.

3. The application should include the following sliders:

   (a) "Canny low" controlling the low threshold of the Canny edge detector (0–255, used for the Hough transform).

   (b) "Canny high" controlling the high threshold of the Canny edge detector (0–255).

   (c) "Detection threshold" controlling the threshold used for recognizing the sphere (normalise it to the range from 0 to 1).

   Use the sliders to find the best set of parameters and then use them as the default parameter values.

4. The video pauses when the space bar is pressed. The application exits if any other key is pressed.

To simplify testing, record a test video, about 20-30 seconds long with whatever camera you have available, such as your phone camera. The video should contain some spheres of different texture but also other distractors. It is expected that each group will prepare their own test video. To get your started, a sample test video is available in Blackboard.

You should find the following OpenCV functions and classes useful for this task:

```
VideoCapture
setMouseCallback
cvtColor
Canny
HoughCircles
circle
```

An example how the application may look like is shown in Figure 1.

Submit to Blackboard the entire project in a ZIP file. The project must compile on the university computers. Each group may be asked to demonstrate the application.

Good luck!

2 Help

2.1 Simple colour descriptor

It is strongly recommended that for this task you use the following simple colour descriptor.

To describe the selected image region (circle), first convert all pixel values to the chromatic colour space:

$$C_r = \frac{R}{R+G+B+1}, \tag{1}$$

$$C_g = \frac{G}{R+G+B+1}, \tag{2}$$

where R, G and B are pixel values beween 0 and 255, C_r and C_b are the new colour coordinates. Note that the new chromatic values will be in the range 0–1. The goal of this step is to make the colour independent of the brightness (luma) component, which may vary with shading and illumination.

Next, build two-dimensional histogram of C_r and C_b values. Such a histogram is a (2D) matrix in which each element is the count of the number of pixels, for which both C_r and C_b values fall into a specific range. To build such a histogram, first quantise C_r and C_b into the range from 0 to N (where N is usually 15 or less) for each pixel. Then, increase the value of the 2D histogram using the quantized values as the coordinates.

Finally, divide all values in the histogram by the total number of pixels. This will make the descriptor independent of the size of a described region.

2.2 Metric for matching the descriptors

Typical metrics, such as SSD or SAD, are not suitable for comparing histograms. However, histograms can be easily compared using the following metric:

$$D = \sum_{k=1}^{K} \min(A(k), B(k)), \tag{3}$$

where K is the total number of bins in the histogram, A and B are the histograms and D is the metric value. The larger D is, the more similar are both histograms. Can you explain what it is so?

2.3 Extracting region of interest in OpenCV

A region of interest (ROI) can be easily extracted from a matrix in OpenCV using the index operator. For example to create a ROI 10×10 pixels located at the coordinates $x = 200$, $y = 100$ for the matrix `img`, you can write:

```
Mat roi = img( Rect( Point(100,200), Size(10,10) ) );
```

Note that the coordinates in OpenCV are reversed (first row index, then column index). The `roi` matrix is a header that uses the same memory storage as the matrix `img`. So when `roi` is modified, also `img` will be modified.

ROI can be also specified using row and column ranges. The code:

```
Mat roi = img( Range(100, 109), Range(200, 209) );
```

is equivalent to the one above.