ESE650 Project 1: Color Segmentation

Code due date: **1/31/2017 at 1:20pm** on Canvas, <pennkeyID>_project1.zip
Report due date: **02/01/2017 at 11:59pm** on Canvas, <pennkeyID>_project1.pdf

In this project, you will train a probabilistic color model from image data, use it to segment and detect a target of interest (red barrel), and find the relative world coordinates of the target with respect to the camera frame. Given a set of training images, you should hand-label examples of different colors. From these training examples, you should build color classifiers for several colors (e.g., red, yellow, brown, etc.) and finally a red barrel detector. You will then use your algorithm to obtain the bounding box of a detected barrel in the image frame and use the camera parameters to calculate the distance to the barrel on a set of new test images.

Training Data Download: Now available at https://upenn.box.com/v/ese650pj1-train
Additional Data Download: Now available at https://upenn.box.com/v/ese650pj1-train-additional
Test Data Release: 1/31/2017 3:00pm, https://upenn.box.com/v/ese650pj1-test
Upload: on Canvas (**Late submission is not allowed for this project!**)

(1) **Code** (<pennkeyID>_project1.zip): upload all your code (do not include image data but include dependencies) and a README file with clear instructions for running it. You should include your trained model for the test. (TAs will check your results with your trained model.)

(2) **Write-up** (<pennkeyID>_project1.pdf): Write a project report which should include the following sections: Introduction, Problem Formulation, Technical Approach, Results and Discussion. Make sure your results include (a) segmented integer image, (b) the bounding box coordinates of the barrel, (c) the distance to the barrel estimate **in meters** for each test image.

Grading: Rubrics can be found on the Canvas assignment page.

Instructions and Tips:

1. Hand-label appropriate regions in the training images with discrete color labels. For this project, we will be especially interested in regions containing the red barrel (positive examples) and images containing similar colored-areas that are not a barrel (negative examples). If you are more ambitious, you could try to implement more automated ways of labeling images, e.g., by an unsupervised image segmentation, or an adaptive region flooding algorithm. Lighting invariance will be an issue, so you should think carefully about the best color space to use, and perhaps some low-level adaptation on the image.

2. Use a learning algorithm to partition the color space into appropriate class color regions. You **must** implement and present results from the approach discussed in class but you are also free to try other machine learning approaches if you have time, e.g., decision trees, support vector machines, etc. You need to make your algorithm so that it is able to robustly generalize to new images. To prevent overfitting the training images, split them into training and validation sets. Train your algorithms using the training set and evaluate their performance on the validation set. This will allow you to compare different parameters for the probabilistic models and different color space representations.

3. Once the color regions are identified, you can use shape statistics and other higher-level features to decide where the barrel is located in the images. Use your algorithms to identify the coordinates of a

bounding box for each detected barrel. You should also compute an estimate of the distance to the barrel. You'll be expected to quickly be able to classify and display your results on a new set of test images. Write a test script to display your result as follows:

```
import cv2, os
folder = "Test_Set"
for filename in os.listdir(folder):
    # read one test image
    img = cv2.imread(os.path.join(folder,filename))

    # Your computations here!
    blX, blY, trX, trY, d = myAlgorithm(img)

    # Display results:
    # (1) Segmented image
    # (2) Barrel bounding box
    # (3) Distance of barrel
    # You may also want to plot and display other diagnostic information
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

4. Your write-up upload should include your test result in the following manner:

ImageNo = [01], BottomLeftX = 507.0605, BottomLeftY = 506.7571,
TopRightX = 662.0208, TopRightY=826.7004, Distance = 5.4418
ImageNo = [02], BottomLeftX = 507.0605, BottomLeftY = 826.7004,
TopRightX = 662.0208, TopRightY=826.7004, Distance = 2.3840
….

5. You may use some useful python functions for this project:
   - hand-labeling: roipoly: https://github.com/jdoepfert/roipoly.py
   - color conversion: cvtColor: http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_tutorials.html
   - region analysis: regionprops: http://scikit-image.org/docs/dev/api/skimage.measure.html

   However, please do not use any built-in function that would be the core part of this project. If you are not sure, then ask TAs. Examples of allowed code:

```
import cv2
img2 = cv2.cvtColor(img, cv2.COLOR_BGR2YCR_CB)

from skimage import data, util
from skimage.measure import label, regionprops
img = util.img_as_ubyte(data.coins()) > 110
label_img = label(img, connectivity=img.ndim)
props = skimage.measure.regionprops(label_img)
```

6. Barrel and camera parameters:
   - **Barrel height**: 93cm; **Barrel radius**: 53cm
   - **Rectified projection matrix**: this matrix specifies the intrinsic (camera) matrix of the processed (rectified) image. It projects 3D points in the camera coordinate frame to 2D pixel coordinates using the focal lengths (fx, fy) and principal point (cx, cy)

$$P = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 413.8794311523438, & 0.0, & 667.1858329372201 \\ 0.0, & 415.8518917846679, & 500.6681745269121 \\ 0.0, & 0.0, & 1.0 \end{bmatrix}$$