Exercise 1

Deadline: 24.04.2013, 16.00

Regulations:

You may hand in the exercises in groups of *up to three persons*.

Hand in a printed version of your solutions at the beginning of the exercises. Please always include a syntax-highlighted printout of all the code you have written.

In addition, send a zipped directory containing your source code to `thorben.kroeger@iwr.uni-heidelberg.de`. Your subject line should be `[IA13][EX%02d] your names`. Please include the `pdf` file of your writeup, too.

Please cross-reference your code files in your writeup, such that it is clear which file has to be run.

1 Introduction to Scientific Python and Color Space (20 pt)

Download the `lena`, `mandrill`, `f16` and `wildflower` images from the lecture's website (`http://hci.iwr.uni-heidelberg.de//MIP/Teaching/ip/exercises.php`).

Plot the color distributions of these images both in RGB and in HSV space (use cylindrical coordinates for HSV). In the 3D scatter plot, color each pixel's associated circle with the pixel's color at full opacity. For your solution, select an appropriate viewing position and save a screenshot. Don't forget to add axis labels and a plot title.

Can you relate the shape of these distributions to the image content?

Implementation hints.
Read the images using
`vigra.readImage(filename).view(numpy.ndarray)`. Make sure the images are properly normalized!

For sake of speed, subsample the images (for example $S = 16$) using the appropriate slicing notation.

The `colorsys` module contains functions to transform between different color spaces. In order to apply a function to each entry of a numpy array, **do not try** writing a loop in Python. Because Python is an interpreted language, this is going to be too slow. Instead, use `numpy.vectorize`.

Useful functions: `from mpl_toolkits.mplot3d import Axes3D`, `plot.figure()`, `ax = fig.add_subplot(111, projection='3d')`, `ax.scatter` (useful keyword arguments: `facecolor`, `edgecolor`, `alpha`, `s`), `ax.set_title`, `ax.set_xlabel`, `ax.set_ylabel`, `ax.set_zlabel`, `fig.savefig`, `numpy.sin`, `numpy.cos`.

Print the R, G, B, S and V components as grayscale images and the H component as a color image in which saturation and value are constant throughout the image. Which details are visible in which component?

Implementation hints.
Useful functions: `plot.gray()`, `plot.imshow()`.

If the image appears rotated, use `numpy.transpose` or `numpy.swapaxes`.

2 Bonus exercise: k-NN classification of handwritten digits (20 pt)

In this exercise, we are going to make use of the *scikit-learn* machine learning package for Python (`scikit-learn.org`). This package contains ready to use implementations of many state-of-the-art machine learning algorithms, such as Support Vector Machines and Randomized Decision Trees. They also provide the `KNeighborsClassifier` class.

In the lecture, you have seen the MNIST benchmark dataset for handwritten digit recognition. It consists of 70′000 images (size 28×28) of handwritten digits (numbers 0 to 9) together with their ground-truth labels.

Using scikit-learn, this data can be loaded via

```
from sklearn.datasets import fetch_mldata
mnist = fetch_mldata("MNIST original")

print "labels have shape = %r" % (mnist.target.shape,)
print "data has shape    = %r" % (mnist.data.shape,)
```

- We are going to focus on the binary prediction task of distinguishing the numbers `5` and `6`.
  - Create a training set consisting of the first 10 examples of `5` and the first 10 examples of `6`.
  - Plot the training set as a grid of images
  - Create a test set consisting of the next 500 examples of `5` and the next 500 examples of `6`. Make sure not to include the images from the training set. **Why is this necessary?**
- Use the `KNeighborsClassifier` class's method `fit` to learn the classifier on your training set and the `predict` method to evaluate the performance on the test set. Report the accuracy (#correctly classified samples over all samples). The constructor of the class takes a `n_neighbors` keyword argument, report the effect of setting it to 1, 3, 5. Why are we only using odd numbers?
- Next, we will transform our training samples by rotating them slightly to the left and right and adding these modified versions to the training set. This way, we are approximating the *tangent distance*.
  To rotate an image `img` by 10°, use
  `img_rot = vigra.sampling.rotateImageDegree(img.astype(numpy.float32), 10)` For each training images, add additional samples that have been rotated -10°, -5°, 5°, 10°. How does the accuracy of the prediction change?
- If RAM memory permits: How good is the accuracy for 3-NN with 60'000 samples on a 10'000 training set?

Implementation hints.
Useful functions: `plot.subplots`, `numpy.reshape`, `numpy.where`, `numpy.hstack`, `numpy.vstack`, `zip`.