Abstract: In this project, I'll present an application for detecting an image orientation, based on low features only, and fix the orientation accordingly.
1. Motivation

Nowadays, digital cameras have come very common in our life. Every phone, laptop or tablet is equipped with one. Digital images are more common and easily produced than ever.

One can take a photo in any place and time. After uploading the images to the PC, We notice that some of the images are rotated, and need to go through your album manually preform the fixes. The process is a time consuming and therefore nobody is doing so and the album staying as is - unfixed.

2. Color

Color is the visual perceptual property corresponding in humans to the categories called red, blue, yellow, green and other. Color is derived from the spectrum of light interacting with the spectral sensitivities of the light receptors in the eye. The retina in the human eye contains 2 receptor cells called Rods and Cons. Rods are cells that more sensitive to light but have no preference to a specific wave length. On the other hand, the Cons have 3 types of cells, which react to a different range of wave length; these 3 cells are sensitive to the "red", "blue" and "green" wave length.

a. RGB color space

An RGB color space is any additive color space based on the RGB color model. RGB stands for the RED, the GREEN and the BLUE. These colors are the base axis for this color space. It can be view as a cube (see figure 1).

To determine if color A \((r_1, g_1, b_1)\) is similar to color B \((r_2, g_2, b_2)\), one can simply measure the distance between these coordinates in the Euclidean space. The problem in RGB model (shown in Fig 2) is when trying to determine similarity of colors; colors with the same distance don’t look the same.
b. **HSV color space**

HSV is a common cylindrical-coordinate representation of points in an RGB color model. HSV stand for hue, saturation, and value.

![Figure 3: HSV color space](image)

In the cylinder, the angle around the central vertical axis corresponds to "hue", the distance from the axis corresponds to "saturation", and the distance along the axis corresponds to "value".

Trying to define a "distance" between colors, is by comparing each component separately: the angle of the hue component, the defense in the saturation and the value components.

Colors that are perceptually close in the human vision will be close by the distance formula.

3. **Machine learning and feature vector**

Machine learning algorithms are tools that from study on test set, we can conclude on input that not in our data base.

These algorithms use a feature vector - an n-dimensional vector of numerical features that represent some object, in this case an image.

![Figure 4: classify function in MatLab](image)

MatLab's "classify" function, needs training vectors and a group labels for all the vectors. The function builds an n-dimensional space, and then groups the vectors together according to their proximity to the same group label, as shown in Figure 4.
4. Problem Domain and Assumptions
   Orientation of an image can be in the range of $0^\circ - 360^\circ$.
   But when taking a picture in a digital camera, the image orientation can only
   have one of four values:
   
   $\omega = 0^\circ, \omega = 90^\circ, \omega = 180^\circ, \omega = 270^\circ$

5. Peripheral blocks
   By analyzing images, I’ve concluded that the peripheral area of an image has enough
   information to decide its orientation. Therefore, The image was split to NXN blocks,
   and only the peripheral sub-blocks were taken.

   ![Image Sub Blocks](image.png)

   Figure 5: image sub blocks: (a) Original image   (b) Image grid with N = 4
   (c) Peripheral sub blocks

6. Principles of the Algorithm
   The approach for solving the problem was divided to 4 steps:
   6.1 Creating the database
      a. The image must be of orientation $0^\circ$
      b. Resize the image to resolution of 800 X COL, COL is the value which keeps
         the ratio of the original image. Doing so reduce the size of the image for
         reducing calculation in the following steps.
      c. Rotate the image to all 4 possible orientation: $0^\circ, 90^\circ, 180^\circ, 270^\circ$ and save
         the images in the DB folder

   6.2 Calculating the feature vector for an image
      a. Convert the image to HSV color space.
      b. Find the peripheral blocks for the image.
      c. For each sub block calculate the first and second moments (the mean and the
         variance) of the sub block.
      d. Calculate the ratio between edges in the each sub block, using canny edge
         detection algorithm, and the area of the sub block.
      e. Combine all previous calculations into a single vector.
6.3 defining the training data and the group labels
    a. Combine all the feature vectors from the same orientation to a matrix of the same orientation.
    b. Create a column vector of grouping label (0, 90, 180, 270), and place it by the vector of the same orientation.

6.4 detect orientation of an input image
    a. Resize the image to 800XCOL like in 6.1.a
    b. create a feature vector for the image as describe in step 6.2
    c. Classify the test feature vector with the training group to define its orientation
    d. Rotate the image to the opposite direction for fixing.

7. Experiments and Results
In order to test the algorithm and find the parameters suitable for the task, I worked with database images of different sizes.
N, the number of blocks, was determent arbitrary.
At First I tried using the RGB scheme color, and after some experiments, decided to change to HSV.
At first, I’ve worked only with the mean property of a block, and afterwards I decided to add the variance and the edge ratio, and noticed that the accuracy increased.

<table>
<thead>
<tr>
<th>N</th>
<th>DB size</th>
<th>Color scheme</th>
<th>Feature Vector</th>
<th>Vector size</th>
<th>Test size</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>200</td>
<td>RGB</td>
<td>Mean</td>
<td>24</td>
<td>50</td>
<td>62%</td>
</tr>
<tr>
<td>3</td>
<td>200</td>
<td>RGB</td>
<td>Mean+var</td>
<td>48</td>
<td>50</td>
<td>34%</td>
</tr>
<tr>
<td>4</td>
<td>300</td>
<td>HSV</td>
<td>Mean</td>
<td>24</td>
<td>70</td>
<td>76%</td>
</tr>
<tr>
<td>4</td>
<td>300</td>
<td>HSV</td>
<td>Mean+edge</td>
<td>37</td>
<td>400</td>
<td>79%</td>
</tr>
<tr>
<td>4</td>
<td>300</td>
<td>HSV</td>
<td>Mean+Var+edge</td>
<td>88</td>
<td>400</td>
<td>82%</td>
</tr>
<tr>
<td>8</td>
<td>300</td>
<td>HSV</td>
<td>Mean+Var+edge</td>
<td>200</td>
<td>200</td>
<td>81%</td>
</tr>
</tbody>
</table>

a. Subset of images with orientations correctly detected
b. Examples for images whose orientations are not correctly detected

8. Conclusion

There is need to consider the database space on the disk, and the vector size for running time, and in order to produce highly accurate results. Most of the time was needed to create the database (saving images in all four orientations) and calculating the feature vectors. These actions are performed only once, and for a DB of size 300 and vector feature of size 88 it takes about 10 minutes. After the feature vectors are ready and saved on the disk, it takes about a second, per image, to return an answer. The highest accuracy rate was 82%. It was obtained from a test set of 400 images. That’s about 330 images from 400 that were put into the right orientation.
9. Future work

I try to think how I can improve the accuracy without object recognition aid:

- Improving the feature vector, adding more properties that can help detection of orientation.
- Add rejection criteria – a dark picture might be hard to depict, hence resulting in a non-useful feature vector.
- Add indoor/outdoor classifier – most of the images I’ve worked with were outdoor images. Perhaps a database that contains more indoor images, with the corresponding label, can improve the accuracy.

10. References

a. Y. Wang and H. Zhang, "Content-based image orientation detection with support vector machines".

b. Datar M., Qi X., "Automatic Image Orientation Detection Using the Supervised Self-Organizing Map".

c. Cardani, Darrin, "Adventures in HSV Space".

d. Ben Shahar, Ohad, "Color - Perceptual theories, mathematical abstraction, and cortical infrastructure", Lecture notes from Introduction to Computational and Biological Vision course, 2013.
# Application User Manual

## 1. Files and Folders

<table>
<thead>
<tr>
<th>File name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project.m</td>
<td>Main file</td>
</tr>
<tr>
<td>myMenu.m</td>
<td>Opens the menu</td>
</tr>
<tr>
<td>Menu2.m</td>
<td>Function that creates a menu</td>
</tr>
<tr>
<td>CreateDB.m</td>
<td>Function that gets an image folder, and rotate</td>
</tr>
<tr>
<td>rotateFolder.m</td>
<td>Function that rotate an image an save it on the disk</td>
</tr>
<tr>
<td>createFeatureVector.m</td>
<td>Create DB feature vectors</td>
</tr>
<tr>
<td>Write2File.m</td>
<td>Save feature vector on disk</td>
</tr>
<tr>
<td>get_feature_vec.m</td>
<td>Combining all the features to one vector</td>
</tr>
<tr>
<td>Img_edge.m</td>
<td>Calculate vector of edges ratio in blocks</td>
</tr>
<tr>
<td>HSV_Mean</td>
<td>Calculate vector of mean of H,S,V on image I</td>
</tr>
<tr>
<td>HSV_Var.m</td>
<td>Calculate vector of variance of H,S,V on image I</td>
</tr>
<tr>
<td>Split_Image.m</td>
<td>Split image to NXN blocks</td>
</tr>
<tr>
<td>Border_Image.m</td>
<td>get only the peripheral blocks</td>
</tr>
<tr>
<td>Testing_folder.m</td>
<td>testing folder</td>
</tr>
<tr>
<td>Test_folder.m</td>
<td>testing images in folder by training set</td>
</tr>
<tr>
<td>Load_data.m</td>
<td>load feature vector from disk</td>
</tr>
<tr>
<td>get_direction</td>
<td>get the orientation of image, after the underline</td>
</tr>
<tr>
<td>Print_Border.m</td>
<td>show figure with peripheral blocks only</td>
</tr>
<tr>
<td>Print_Block_Image.m</td>
<td>how figure with NXN blocks</td>
</tr>
<tr>
<td>testingPic.m</td>
<td>testing image</td>
</tr>
<tr>
<td>test_pic.m</td>
<td>testing image by training set</td>
</tr>
<tr>
<td>vec_0.mat</td>
<td>vector feature for orientation 0°</td>
</tr>
<tr>
<td>vec_90.mat</td>
<td>vector feature for orientation 90°</td>
</tr>
<tr>
<td>vec_180.mat</td>
<td>vector feature for orientation 180°</td>
</tr>
<tr>
<td>vec_270.mat</td>
<td>vector feature for orientation 270°</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Images folder</th>
<th>0° image folder</th>
<th>90° image folder</th>
<th>180° image folder</th>
<th>270° image folder</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DB folder</td>
<td>Test folder</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2. How to use

Run project.m and it show you a menu:

![Menu Image]

**Create Database:**
Please make sure that you have images in "images/DB" whose orientation is 0°. After pressing the button, the application will create all four orientations and save the images in the correct folder.
- Be patient, this process could take a while, depending on the DB size.

**Create Feature Vector:**
Pressing the button will create the files: "vec_0.mat", "vec_90.mat", "vec_180.mat", and "vec_270.mat".
For each image in each of the folders, it will calculate its vector feature
- Be patient, this process could take a while, depending on the DB size and the features that needs to be calculated.

**Testing folder:**
Pressing the button will open a window for you to select folder with images. It then run the classify algorithm on each of the images. In the end the accuracy of the process will be displayed.
If you want the application to show the images during the process, go to MyMenu.m and change the variable toShow to 1.

**Testing image:**
Pressing the button will open a window for you to select an image. It then than runs the classify algorithm in order to detect its orientation. After the classification is done, it shows the images of both before and after detection.

**Close all:**
Close all the figures open on screen.