EC601 A1 Product Design in Electrical and Computer Engineering

Mini_project 3: Image visualization in Python

Author: Xushan Hu

Part 1: Description of the project

Study and demonstrate visualizations tools to be used with DSP (Javascript or Python). In my project, I chose to analyze the image and show the visualization. It is a project of the image processing in python. The project includes displaying an image, blurring and grayscaling the image, edge detection, count objects, and face detection.

Part 2: Demonstrate the visualization tool -- OpenCV & Matplotlib

OpenCV is an open-source computer vision and machine learning software library. It is a BSD-licence product thus free for both business and academic purposes. The Library provides more than 2500 algorithms that include machine learning tools for classification and clustering, image processing and vision algorithm, basic algorithms and drawing functions, GUI and I/O functions for images and videos. Some applications of these algorithms include face detection, object recognition, extracting 3D models, image processing, camera calibration, motion analysis, etc. Currently, OpenCV supports a wide variety of programming languages like C++, Python, Java, etc and is available on different platforms including Windows, Linux, OS X, Android, iOS, etc. Also, interfaces based on CUDA and OpenCL are also under active development for high-speed GPU operations.
Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatterplots, etc., with just a few lines of code. So Matplotlib is a very convenient tool for image processing and visualization.
Why I chose Python?
Python is a general-purpose programming language started by **Guido van Rossum**, which became very popular in a short time mainly because of its simplicity and code readability. It enables the programmer to express his ideas in fewer lines of code without reducing any readability. And the support of Numpy makes the task easier. **Numpy** is a highly optimized library for numerical operations. It gives a MATLAB-style syntax. All the OpenCV array

structures are converted to-and-from Numpy arrays. So whatever operations you can do in Numpy, you can combine it with OpenCV, which increases the number of weapons in your arsenal. Besides that, several other libraries like SciPy, Matplotlib which supports Numpy can be used with this. So OpenCV-Python is an appropriate tool for fast prototyping of computer vision problems.

Part 3: Image processing tutorials:

3.1 Display an Image

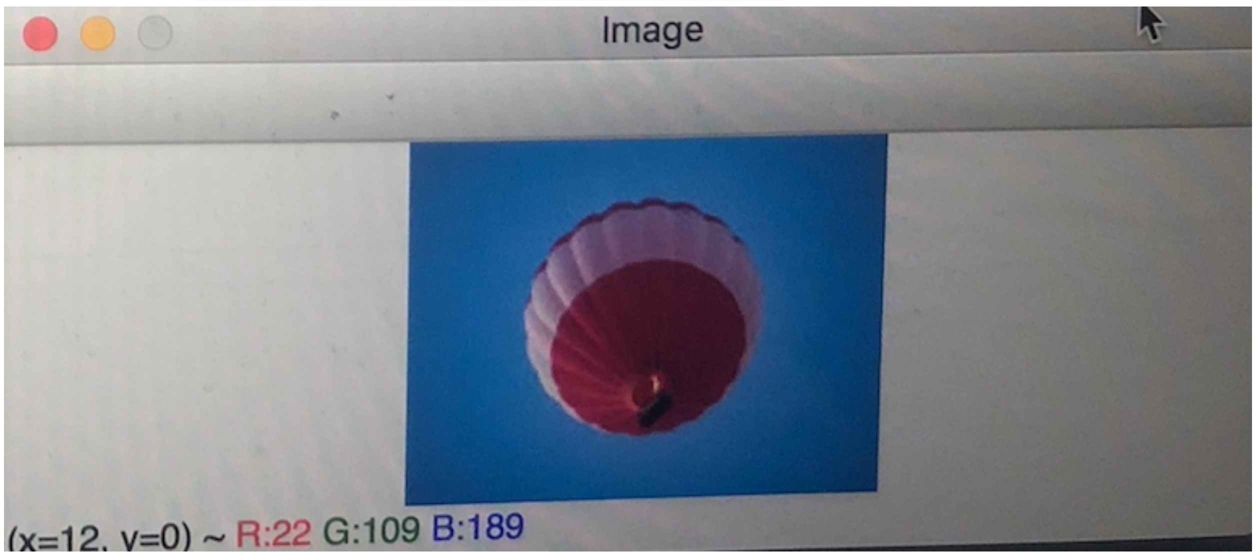
Use the imshow() function to display an image:

Syntax: cv2.imshow(window_name, image)

Parameter:

Window_name - A string representing the name of the window in which image to be displayed.

image - It is the image that is to be displayed.



(x=12, y=0) ~ R:22 G:109 B:189

3.2 Blur and Grayscale

Use the cvtColor() and GaussianBlur() functions to process the image:

3.2.1 cvtColor(): Converts an image from one color space to another.

Syntax: cv2.cvtColor(src, code[, dst[, dstCn]]) → dst

Parameter:

src – input image: 8-bit unsigned, 16-bit unsigned (CV_16UC...), or single-precision floating-point.

dst – output image of the same size and depth as src.

code – color space conversion code (see the description below).

dstCn – number of channels in the destination image; if the parameter is 0, the number of the channels is derived automatically from src and code.

3.2.2 GaussianBlur(): Using the Gaussian filter for image smoothing

Syntax: cv2.GaussianBlur(src, ksize, sigmaX[, dst[, sigmaY[, borderType=BORDER_DEFAULT]]])

Parameter:

src - input image

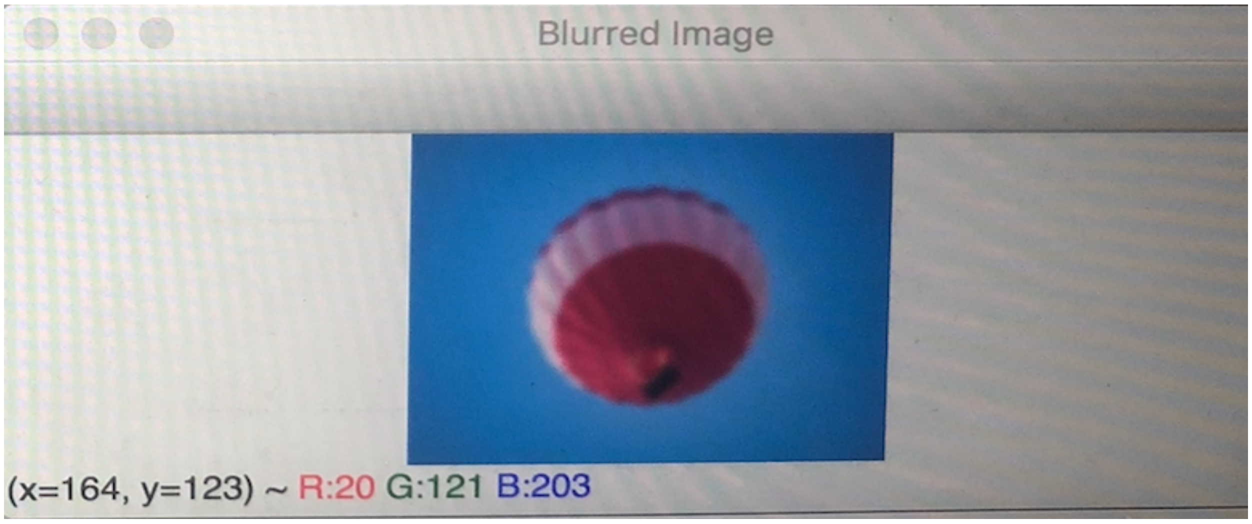
dst - output image

Ksize - Gaussian Kernel Size. [height width]. height and width should be odd and can have different values. If ksize is set to [0 0], then ksize is computed from sigma values.

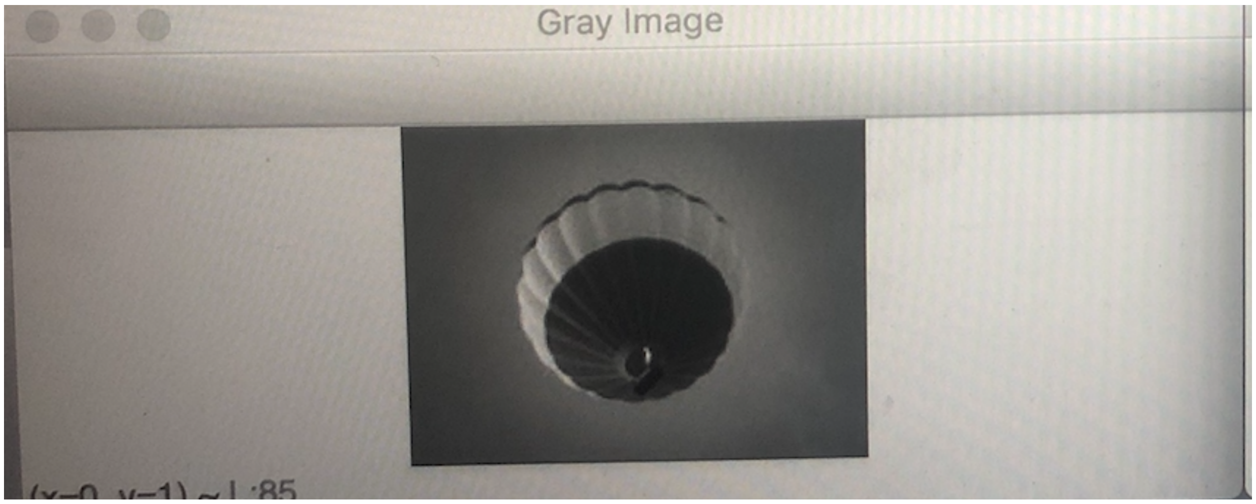
sigmaX - Kernel standard deviation along X-axis (horizontal direction).

sigmaY - Kernel standard deviation along Y-axis (vertical direction). If sigmaY=0, then sigmaX value is taken for sigmaY

borderType - Specifies image boundaries while kernel is applied on image borders.



```
(x=164, y=123) ~ R:20 G:121 B:203
```



3.3 Edge Detection

Edge detection means detecting where the edges of an object in an image are. The algorithm looks for things like change in color, brightness, etc to find the edges. I used the canny() function to process the image.

Syntax: edges = cv2.Canny('/path/to/img', minVal, maxVal, apertureSize, L2gradient)

Parameter:

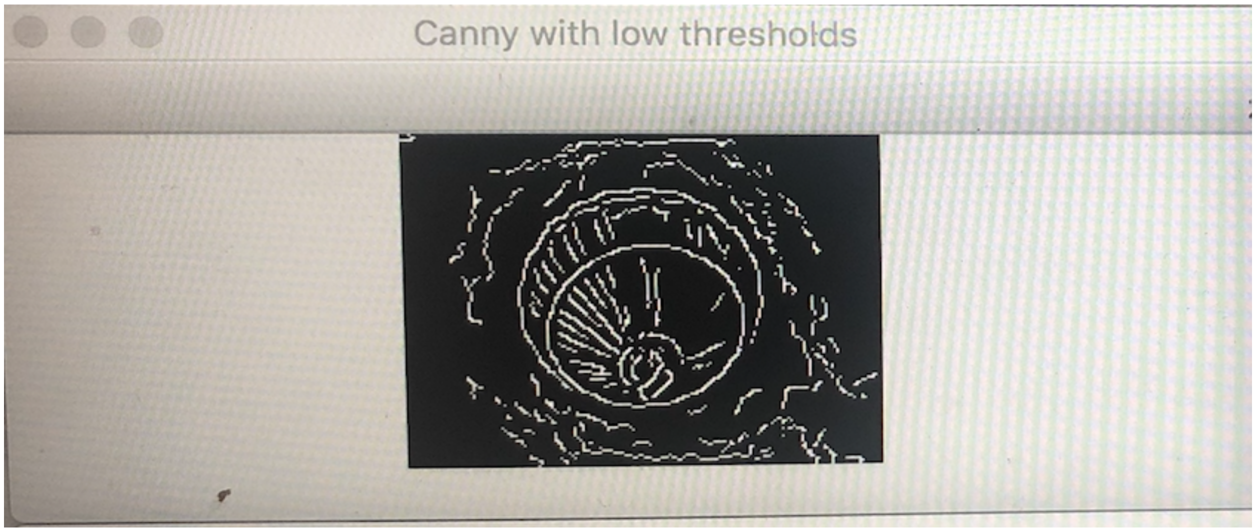
/path/to/img - file path to the image.

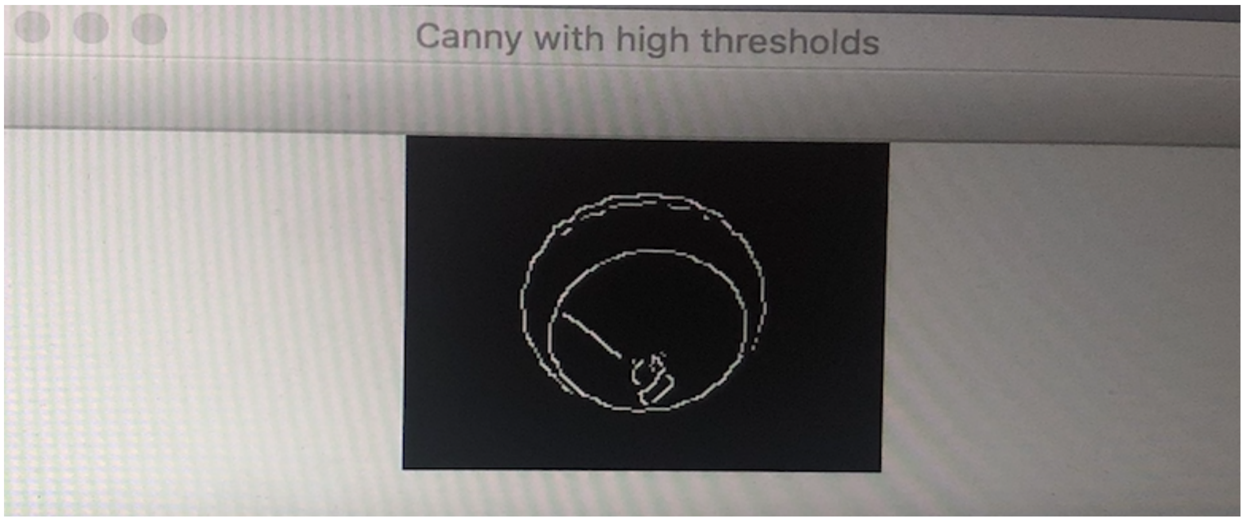
minVal - Minimum intensity gradient

maxVal - Maximum intensity gradient

apertureSize

L2gradient - If true, Canny() uses a much more computationally expensive equation to detect edges, which provides more accuracy at the cost of resources.





3.4 Count Objects:

The *drawContours()* finds the contours in the given image.

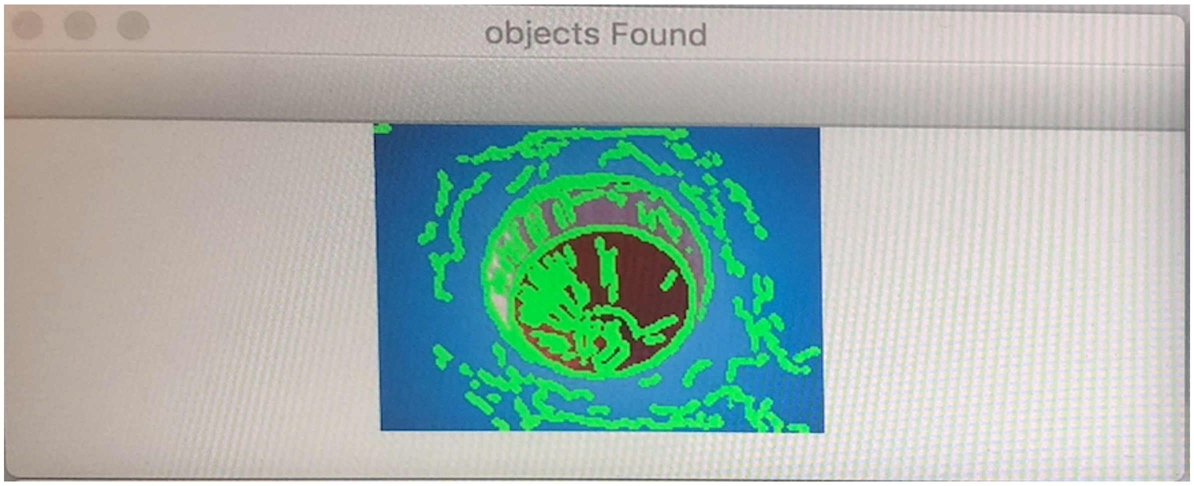
Syntax: cv2.moments(array[, binaryImage]) → retval

Parameter:

array – Raster image (single-channel, 8-bit or floating-point 2D array) or an array ($1 \times N$ or $N \times 1$) of 2D points (Point or Point2f).

binaryImage – If it is true, all non-zero image pixels are treated as 1's. The parameter is used for images only.

moments – Output moments.



3.5 Face Detection:

'OpenCV' contains many pre-trained classifiers for face, eyes, smile, etc. The XML files of pre-trained classifiers are stored in `opencv/data/`. For face detection specifically, I tried the Haar Cascade Classifier:

It is a machine learning-based approach where a cascade function is trained from a lot of positive (images with face) and negative images (images without face). The algorithm is proposed by Paul Viola and Michael Jones.
The algorithm has four stages:
1. Haar Feature Selection: Haar features are calculated in the subsections of the input image. The difference between the sum of pixel intensities of adjacent rectangular regions is calculated to differentiate the subsections of the image. A large number of haar-like features are required for getting facial features.
2. Creating an Integral Image: Too much computation will be done when operations are performed on all pixels, so an integral image is used that reduce the computation to only four pixels. This makes the algorithm quite fast.
3. Adaboost: All the computed features are not relevant for the classification purpose. 'Adaboost' is used to classify the relevant features.
4. Cascading Classifiers:** Now we can use the relevant features to classify a face from a non-face but algorithm provides another improvement using the concept of `cascades of classifiers`. Every region of the image is not a facial region so it is not useful to apply all the features in all the regions of the image. Instead of using all the features at a time, group the features into different stages of the classifier. Apply each stage one-by-one to find a facial region. If on any stage the classifier fails, that region will be discarded from further iterations. Only the facial region will pass all the stages of the classifier.
I used plt.imshow() to show the image.

Syntax: mshow(X, cmap=None, norm=None, aspect=None, interpolation=None, alpha=None, vmin=None, vmax=None, origin=None, extent=None, shape=<deprecated parameter>, filternorm=1, filterrad=4.0, imlim=<deprecated parameter>, resample=None, url=None, *, data=None, **kwargs)

Parameter:

X - array-like or PIL image
cmap - str or Colormap, optional
norm - Normalize, optional
aspect - {'equal', 'auto'} or float, optional
interpolation - str, optional
alpha - scalar, optional
vmin, vmax - scalar, optional
origin - {'upper', 'lower'}, optional
extent - scalars (left, right, bottom, top), optional
filternorm - bool, optional, default: True
filterrad - float > 0, optional, default: 4.0
resample - bool, optional

url - str, optional

The result is shown as follows: