CV Handbook

Shahzeb Asif

July 7, 2014

References

OpenCV Tutorials

All of the work below is based on the tutorials at the OpenCV Python Tutorial .

NumPy

NumPy is a Python library made for really fast arrays. It's used extensively in OpenCV.

Here are some notes on using NumPy.

Indexing

It's inefficient to get an item from the numpy array by indexing: `arr[10]`.

Instead, use the numpy array methods: `item()` and `itemset()`.

Use the old way of indexing for slices, i.e. `arr[:10, :10, :10]`.

We can unpack arrays with `split()` but it's slow. Use slicing, if possible.

Array Variables

- `shape` describes its shape in n-dimensions.
- `size` is the total number of items in an array.
- `dtype` is the data type. Note that arrays must be homogeneous.

IO

We load images using the `imread` function. It takes in a filename and returns a numpy array.

We write images using `imwrite`. It takes in a filename and a numpy array.

Arithmetic

Addition

We can add values to arrays. But remember that numpy arrays are a defined type.

- `cv2.add()` will add two arrays but it puts a ceiling on the values from the data type.
- `a + b` is using numpy. This will modulo the result with the ceiling for the data type.

There is also `addWeighted()`. This takes in two arrays, and their relative weightings. And simply does: $result = (1 - x) * img1 + x * img2$ The result appears to have transparency.

There are also bitwise operations.

Shape

This is annoying enough to get its own section.

Numpy arrays have a shape property. `arr.shape` will return the dimensions of the array but they return rows, cols, z-dim.

Remember that rows are basically y-values and cols are x-values. It actually gives `y, x, z` which is backwards compared to mostly everything.

OpenCV, on the other hand, does use `cols, rows`. Remember to keep track of this annoying issue.

BGR & Formats

OpenCV does things in the BGR format. This means that the pixels are stored with blue first, then green, and finally red.

Luckily, there is a function called `cvtColor` that allows you to easily convert from various formats to others.

It takes in an image and a type as arguments. One very common argument is `COLOR_BGR2GRAY` which is used to convert BGR to grayscale.

Basic Image Properties

Brightness

Brightness is a lot simpler than it sounds. It is actually almost like the average value of all the pixels.

If you increase the value of all the pixels by using `numpy.add` or `cv2.add`, it will increase the brightness.

Contrast

Contrast ilike increasing brightness more for already bright pixels.

We can do this by multiplying by > 1 or < 1. Multiplying by a value like 2 will increase all the bright pixels by a much larger margin than the darker ones.

Use `cv2.multiply` or `numpy.multiply` to adjust contrast.

Template Matching

The idea in template matching is to compare a small template to a larger image.

The comparison is done with some math in the frequency domain.

Template matching will identify areas that appear to be the same as the template.

But there are a few problems with template matching:

- It is somewhat slow.
- It doesn't work easily with varying sizes or rotations.

It may be better to go with other methods.

Windows

Windows are fairly easy to create. Use `imshow(window_name, image)` to display something. It will automatically create a window for you.

You must use `waitKey` to get any output. `waitKey` just waits for key input and then returns that in hex. But there's a small catch.

The returned number is a little stupid and instead of returning something reasonable, it returns three bytes. So use an AND mask on it to just get the lower byte.

Contours

Instead of using template matching, my pynotescan program uses contours.

The idea of contours is just to identify a connected line. That really helps identify pages in the program.

Use the `findContours(img, heirarchy_mode, extensive_mode)` function. `img` is just the image you want to find contours in. `heirarchy_mode` can be used to get all contours at different heirarchies, or just the top most one, or not put them in heirarchies at all. `extensive_mode` has two options `CHAIN_APPROX_NONE` and `CHAIN_APPROX_SIMPLE`. `_SIMPLE` will reduce long lines to just corners but `_NONE` will have everything.

The `findContours` function returns three values: image, contours, heriarchy.

Warp Image

We can warp images from their original dimensions to new dimensions.

The `getPerspectiveTransform` and `warpPerspective` functions are used to warp an image.

First pass in numpy arrays with float32 type into `getPerspectiveTransform(old_dims, new_dims)`. This returns a warper that is actually used to warp an image.

This warper is then passed in to `warpPerspective(img, warper, size)`. The image is just the image to warp. The size is the width x height of the new_dims.

Threshold

It's often very useful to apply thresholds to images. There are a few different types of thresholds to be aware of.

Basic

The most basic type of threshold is using `cv2.threshold` which takes four parameters: image, threshold, maximum, mode.

The way the function applies the threshold and maximum value depends on the mode. Check the documentation for modes.

One basic mode is `THRESH_BINARY` which makes all pixels above thresh to the maximum value and all pixels below thresh to zero.

Another useful mode used in pynotescan is `THRESH_TOZERO`. This makes everything below thesh zero but leaves everything above thresh as-is.

Adaptive

Adaptive threshold is used when you want the threshold to be applied to a local area instead of applying it to the entire image.

The idea is that being very bright in one area but not that bright in another area will not have as much of an effect as simple thresholding.

We use the function `adaptiveThreshold` to do adaptive thresholding. The function takes many arguments:

```
adaptiveThreshold(img, max_val, mode, type, k_sz, const)
```

The img, max_val are self-explanatory. The mode can be either `ADAPTIVE_THRESH_MEAN_C` or `ADAPTIVE_THRESH_GAUSSIAN_C`. Try both to see the differences. The type can be either `THRESH_BINARY` or `THRESH_BINARY_INV`. The k_sz is the kernel size used and the const is just some constant subtracted from the result.

Look to the documentation for details on each parameter.

Smoothing

Smoothing is very useful in removing noise from images. In pynotescan, most of the noise came from bad pictures and texture of backgrounds. Here were a few options to remove that noise.

Median Blur

This blurring ended up being incredibly useful in removing texture, like carpet or cloth, from images.

Pass in an image and a kernel size to the function `medianBlur(img, kernel_sz)`. A larger kernel size removes more detail.

Histogram

We can equalize images in many ways. Here are two of them.

Basic

We can use the `equalizeHist(img)` function to equalize the histogram of an image.

This is not smart equalization. It just distributes the pixels over different intensities.

Adaptive

We can use something called CLAHE to have adaptive equalization. Like adaptive thresholding, CLAHE works on small sections of the image instead of the entire image at once.

Use the function `createCLAHE` to create and return a clahe object. It has no mandatory arguments but you can add `clipLimit` and `tileGridSize` to control clipLimit? and just the size of the area CLAHE will use.

To actually apply it to an image, use the `apply` method from the object created with `createCLAHE`. `apply` just takes an image as an argument and returns a new image.

Installation

Dependencies

Install the following dependencies:

Required

- `build-essential`
- `cmake`
- `git`
- `libgtk2.0-dev`
- `pkg-config`

- `libavcodec-dev`
- `libavformat-dev`
- `libswscale-dev`

Optional

- `python3-dev`
- `python3-numpy`

The list of dependencies can be found here.

Steps

- CMake is some useful tool to create and configure a bunch of makefiles.
- Use the following flags, or similar ones, on the cmake in OpenCV:

```
cmake
-D WITH_IPP=OFF
-D PYTHON_EXECUTABLE=/usr/bin/python3.3
-D PYTHON_LIBRARIES=/usr/lib/python3.3/
-D PYTHON_INCLUDE_DIRS=/usr/include/python3.3m/
-D PYTHON_PACKAGES_PATH=/usr/lib/python3.3/
```

OpenCV Flags

- Also note that `PYTHON_INCLUDE_DIRS` has `python3.3m`. I've no idea why but that works and `python3.3` doesn't.
- But if have Python3, it may not work. In that case, do the following:
  - Install `python3-dev`.
  - Run `python3.3-config --includes`.
  - That will give you a few directories, check them for `pyconfig.h` to the directory in the cmake above.
- Then, `make`.
- Finally, `sudo make install`.
- The last line written will have something to do with cascading trains.
- It could also give an error of `Failed to load OpenCL runtime`.
- In that case, install `ocl-icd-opencl-dev`.
- It's also possible that your camera won't change resolutions. In that case, install the `libv4l-dev` package. Re-make the source, re-install everything. And then restart. It doesn't seem to take effect until restarting.