Automatic Number Plate Recognition in Python on RPi3 / UP²

SJM@MCL 10-2-18 *www.marvellconsultants.com*

UKRegOCR: Cross-platform UK number-plate reading software written in Python3

Limitations

The code is optimised only for the standard UK number-plate font, some older UK plates do not use the proper font and so may fail to read correctly. It can only read single-line plates, 2-line plates are not recognised. The plate must be reasonably horizontal (within 10 degrees) in order to be recognised. It's also fairly slow - budget for about 1s/read on a fast, multi-core Intel/AMD x86 linux PC, 1.6s/read average on an Ubuntu UPsquared Pentium N4200, but much slower on a Raspbian Raspberry Pi 3b @ about 7.5s/read (10% faster on a 3b+). If these limitations aren't too serious in your application then please evaluate UKRegOCR for yourself and let me know how you get on. Also let me know if you find any bugs or have any ideas for improvements. The code is provided free and completely without warranty as per the usual legal blah blah.

Installing and evaluating UKRegOCR

Python v3.5 or later is required along with recent (2017/18) versions of NumPy, OpenCV and PILlow python packages - see resources section below. Python is fully supported on Unix and Windows platforms (and others), I developed and tested with Python v3.6.3 on Windows / v3.5.2 on Ubuntu, OpenCV (cv2) v3.3.1 (v3.4 is now available and should be a simpler pip install on Windows), NumPy v1.14.0 and PILlow (PIL) v4.3.0. Note that OpenCV v3.x is confusingly referred to as cv2. I also used the 32-bit option for compatibility reasons - if you can get it all working in 64-bit please let me know, it could significantly improve speed.

NumPy, OpenCV and PILlow should first be installed following any of the detailed guides available on the internet for your platform, google it. See below for detailed Raspberry Pi and UPsquared installation instructions.

Now download MCL's UKRegOCR components from:

http://www.marvellconsultants.co.uk/ANPR/UKRegOCR.zip

Extracting to an empty directory, you should see 4 files & a samples folder:

`UKRegOCR.py` is the core OCR module
`Example.py` is a sample python script which imports and demonstrates the above module
`UKPlates66x20P.xml` is the number-plate level Haar-cascade classifier
`UKChars33_16x25_11W.xml` is the character level Haar-cascade classifier
`samples/` contains some test images including a couple of failure examples

Download UKNumberPlate.ttf true-type font file from:

https://www.dafont.com/uk-number-plate.font

Save it to the same directory. Finally, run the example script with python v3.x:

```
python3 Example.py
```

All being well Example.py should sequence through the images in the samples folder and print the results of each attempted read. On the last image (only) the script should also display an annotated thumbnail. A calculation of the average read-time is also printed.

To test it on your own images you can add a file or folder name to the Example.py command-line, eg:

```
python3 Example.py my_folder/*.jpg
```

Camera set-up

The reader works best with characters of at least 25 pixels in height. To minimise motion-blur use a faster shutter speed where possible. Minimise any artificial/added "sharpness" in the camera's output image if at all possible. Minimise JPG artefacts by prioritising image quality rather than small file size. Arrange for the plate to appear as horizontal as possible in the image, you may need to apply your own affine transform prior to the read attempt. The code works identically with monochrome images, for example from infra-red cameras, although obviously it will be unable to determine the plate's b/g colour.

Under the Hood

UKRegOCR employs a 6 stage process as follows.

Stage1, plate detection: this vital first step uses OpenCV's Haar-cascade classification function trained on several thousand images of UK number-plates. With the correct training this function can reliably detect the presence and location of the desired target object at any scale in a source image. The result of the massively long winded training process (it can take days of processing time) is embodied in a surprisingly compact xml file which somehow, magically, defines the salient features of the target. The output of the multi-scale detector is a list of bounding-boxes indicating where in the source image the target object appears. The code narrows the output down to just one candidate and crops the input image to a more manageable size around the target area.
A more meaningful contrast-stretch (using histogram equalisation) can now be performed on the smaller image. To find out more about Haar-cascade classification simply Google it, there's plenty of information out there.

Stage 2, affine transformation: this stage detects and corrects any small rotational deviation from the horizontal (up to 10 degrees), this is done using a sequence of three OpenCV functions: adaptive thresholding followed by edge-detection then a Hough Lines scan. Correction uses an affine transform. A similar process is deployed to detect and correct any skew (up to 10 degrees) in the individual characters - especially important after a rotation. Since the number-plate classifier will not recognise plates that deviate too much from the horizontal there's little point improving the code to cope with larger angles. Some very old UK plates feature white characters on a black background, these are detected here and the image is simply inverted if necessary, in readiness for the subsequent stages.

Stage 3, estimate character height: here again we employ a Haar cascade classifier this time trained to look for individual UK number-plate font characters. This classifier works less well than the plate classifier but usually provides enough information to make a better estimate of the character height. The results from this detection are also used to feed back into the rotation detector/corrector above. An alternative method makes use of a Sobel filter followed by thresholding and a morphology closure stage to produce a solid block around character-like features, the block size is measured to produce a second estimate of character height which is statistically combined with the Haar-cascade result.

Stage 4, template matching: given a reasonable estimate of character height, template characters can now be produced to that dimension (using the PILlow package and the UKNumberPlateFont TTF file) to match against the plate image with OpenCV's matchTemplate function. The output from which is an array of cross-correlation coefficients for each pixel of the source image, one array per character. Note that there are really only 34 characters in the font as O & 0 and I & 1 are identical. The locations of the highest peaks in the output arrays indicate where characters are likely to be found. This process is repeated for incrementally taller and shorter character sizes each time searching for the best cross-correlation peaks and thus the best match to character height. That whole process is then repeated again for incrementally varying character widths until finally the optimum character size is found. BTW doing these sequences of 34 template-match passes is what takes most of the processing time. A couple of optimisations are employed in the above process to speed things up: the area of interest is reduced as much as possible, as is the character-set, prior to each template-match pass. Some further statistical analysis produces a list of best-guess candidates.

Stage 5, I characters: stage 4 actually only concerns itself with the 33 non-I characters, these are dealt with separately due to the anomalous width of the I (or 1) character. Candidate I characters are now sought in the image and added to the result from stage 4.

Stage 6, finish up: The candidate list is statistically examined for positional anomalies, over-laps, stragglers and notably weak correlation coefficients. It is pruned accordingly before being matched to a list of valid UK number plate format specifications. During this stage 0/O and I/1 choices are made and swap-outs of commonly misread character pairs may be made in order to optimise the match to a valid format. Common misread pairs are: '8':'B', 'B':'8', 'Z':'7', '7':'Z', '6':'G', 'G':'6', '5':'S', 'S':'5', 'D':'0', 'Q':'0'. A more accurate bounding-box for the plate is now computed and if the source image was not monochrome the background colour of the plate can also be determined. Finally an estimate of read confidence is derived from the average correlation coefficient and the quality of match to a valid plate format with format commonness also taken into account.

UKRegOCR module documentation

The UKRegOCR module exposes three functions:

```
UKRegOCR(img, target_size)
lookForPlate(img, target_size)
WhtOrYel(img, AOI)
```

img:

An OpenCV image array, colour or monochrome, as returned by cv2.imread()

target_size:
A (Width, Height) tuple that approximates to the average, mid-field size in pixels of a UK number plate as imaged by the camera. The plate should be roughly horizontal and reasonably undistorted, the reader can theoretically cope with up to 10 degrees of rotation and skew in the input image. Pre-process your images with one of openCV's affine transforms if necessary. Note that a plate height of 50 pixels corresponds to a character height of about 33 pixels, the reader works well down to a character-height lower limit of around 25 pixels, less well below that.

AOI:
Optional, an (X,Y,W,H) tuple defining an area of interest within the source image over which to evaluate background colour. X & Y are measured down from the top left in pixels.

UKRegOCR() is the main plate-reading function it returns either a four element tuple in the event of a successful read or an error message string. The tuple contains:

```
'reg', confidence, (plate-location-AOI), 'plate_colour'
```

as: string, int 0..99, (X, Y, W, H), ['W'|'Y'|'B'|'-'] respectively
For example:

```
'AB12CDE', 92, (100,200,200,60), 'Y'
```

A plate_colour of '-' implies that it was not possible to asses the colour.
A confidence of 0 implies failure to fit the read data to a valid UK plate format.
A confidence of <70 implies a character-swap was requited for a UK format match.
(X,Y,W,H) are in integer pixels wrt to the top-left corner of the source image.
An example error message might be: '!no plate'

lookForPlate() is an interface to the Haar-cascade classifier for UK number plates, pass it an image and a target_size and it will return a tuple containing either (0,0,0,0) if no plate is detected or an (X,Y,W,H) AOI specification.

WhtOrYel() is an interface to the plate background colour detector, pass it an image and an optional AOI tuple and it will return a single character either 'W','Y','B' or '-' indicating respectively White, Yellow, Blue, unknown.

Installation on a Raspberry Pi3B from scratch, step by step.

UKRegOCR does work on the Raspbery Pi 3 but it's much slower than on a regular PC with average read times of about 7.5s. Furthermore installing OpenCV on the Pi currently requires a full compile from sources and is a lengthy and complex procedure, but if you're still undaunted read on.

Download the latest Rasbian (currently Stretch) image from:

https://www.raspberrypi.org/downloads/raspbian/

I chose the lite, non-desktop version. Use an image writer to burn the image to a 16GB SD card, I tried this initially with 8G but ran out of space. Alternatively purchase a pre-initialised SD card. Boot up your Pi (default user/password is

pi/raspberry) & setup networking as required. A wired ethernet is best at this stage as it requires no set-up. After login, configure your Pi & re-boot

```
sudo raspi-config
 → Update the tool
 → Change User Password
 → Advanced → Expand filesystem
 → Interfacing Options → SSH → Yes
 → plus anything else you might need for networking etc
 → Finish
sudo reboot
```

Check you've now got plenty of free disk space:

```
df
```

Once you have a reliable network connection, if you need to, get your Pi's IP address:

```
hostname -I
```

Now you can optionally log in using SSH with Putty from a desktop PC. Or not. First, update the Pi's operating system:

```
sudo apt-get update
sudo apt-get -y upgrade
```

Rasbian stretch currently comes with Pythons v2.7 and v3.5 already installed. If you'd prefer v3.6 (which may be slightly faster) try following this guide:

https://gist.github.com/dschep/24aa61672a2092246eaca2824400d37f

Get and set-up Python's package installer, pip

```
wget -O get-pip.py https://bootstrap.pypa.io/get-pip.py
sudo python3 get-pip.py
rm get-pip.py
```

Install PILlow & numpy:

```
sudo apt-get -y install libjpeg62 libjpeg62-dev
sudo apt-get -y install zlib1g zlib1g-dev
sudo apt-get -y install libfreetype6 libfreetype6-dev
sudo apt-get -y install liblcms1 liblcms1-dev
sudo pip3 install pillow
sudo apt-get -y install python3-numpy
python3
>>> import numpy,PIL
```

```
>>> numpy.__version__
'1.12.1'
>>> PIL.__version__
'5.0.0'
>>> quit()
```

Get OpenCV & numpy, see Adrian Rosebrock's excellent guide at:

https://www.pyimagesearch.com/2017/09/04/raspbian-stretch-install-opencv-3-python-on-your-raspberry-pi/

The guide is for OpenCV v3.3, but since 3.4 has become available we'll be installing that instead consequently there are some subtle differences so I've detailed here the exact steps I took. As we've already installed python3-numpy above there's need to duplicate that and I didn't bother with the optional virtual environment as recommended in the guide. First we must install all OpenCV's dependencies, I saved the following to a shell script and ran from that, you can do it line by line or in groups - but it'll take a while.

```
sudo apt-get -y install build-essential
sudo apt-get -y install cmake
sudo apt-get -y install pkg-config
sudo apt-get -y install libjpeg-dev
sudo apt-get -y install libtiff5-dev
sudo apt-get -y install libjasper-dev
sudo apt-get -y install libpng12-dev
sudo apt-get -y install libavcodec-dev
sudo apt-get -y install libavformat-dev
sudo apt-get -y install libswscale-dev
sudo apt-get -y install libv4l-dev
sudo apt-get -y install libxvidcore-dev
sudo apt-get -y install libx264-dev
sudo apt-get -y install libgtk2.0-dev
sudo apt-get -y install libgtk-3-dev
sudo apt-get -y install libatlas-base-dev
sudo apt-get -y install gfortran
sudo apt-get -y install python2.7-dev
sudo apt-get -y install python3-dev
```

Get OpenCV sources & start the build...

```
wget -O a.zip https://github.com/Itseez/opencv/archive/3.4.0.zip
wget -O b.zip https://github.com/Itseez/opencv_contrib/archive/3.4.0.zip
unzip a.zip
unzip b.zip
rm a.zip
rm b.zip
cd opencv-3.4.0
mkdir build
cd build
cmake -D CMAKE_BUILD_TYPE=RELEASE ¥
      -D CMAKE_INSTALL_PREFIX=/usr/local ¥
```

```
-D INSTALL_PYTHON_EXAMPLES=ON ¥
-D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib-3.4.0/modules ¥
-D BUILD_EXAMPLES=ON ..
```

Check the cmake output as per the guide. If there's a problem I'm afraid it's beyond my pay grade. Next, again as per the guide, temporarily increase your swap space to 1GB.

```
sudo /etc/init.d/dphys-swapfile stop
sudo nano /etc/dphys-swapfile
// change line to CONF_SWAPSIZE=1024
^O RETURN ^X
sudo /etc/init.d/dphys-swapfile start
```

Now to compile & install, grab a coffee. And lunch, this will take some time...

```
make -j4
sudo make install
sudo ldconfig
```

Look for the cv2.so output file:

```
ls -l /usr/local/lib/python3.5/dist-packages
→ cv2.cpython-35m-arm-linux-gnueabihf.so
```

That's good, but unfortunately it's incorrectly named, fix the problem like so:

```
sudo cp /usr/local/lib/python3.5/dist-packages/cv2.cpython-35m-arm-linux-
gnueabihf.so
/usr/local/lib/python3.5/dist-packages/cv2.so
```

Hopefully that all worked. Again, if it didn't then I'm afraid I really can't help. To check...

```
cd ~
python3
>>> import cv2
>>> cv2.__version__
'3.4.0'
>>> quit()
```

Important to restore swap space back to its original setting:

```
sudo /etc/init.d/dphys-swapfile stop
sudo nano /etc/dphys-swapfile
// change line to CONF_SWAPSIZE=100
^O RETURN ^X
sudo /etc/init.d/dphys-swapfile start
```

Now delete the bulky source codes if you're sure you no longer need them, this should free-up over 5GB of disk space.

```
cd ~
rm -rf opencv-3.4.0
rm -rf opencv_contrib-3.4.0
```

Thanks again to Adrian Rosebrock https://www.pyimagesearch.com
Now get UKRegOCR from us at marvellconsultants.co.uk:

```
mkdir ocr
cd ocr
wget -O c.zip http://www.marvellconsultants.co.uk/ANPR/UKRegOCR.zip
unzip c.zip
rm c.zip
```

Then continue from the number plate font ttf file download above. Good luck.

Installation on an UPsquared board from scratch, step by step.

Follow instructions at:
https://downloads.up-community.org/download/up-squared-iot-grove-development-kit-ubuntu-16-04-server-image/
to download ubuntu server 16.04 server image & Tuxboot. Next make the bootable USB stick as instructed but noting that step 3 should point you to "ubuntu_16.04_server_image_for_up_squared_r1.zip".

Boot the UP2 from the USB stick (screen, kbd & network attached) and allow tuxboot/clonezilla to set up the on-board flash. The UP2 will finish by powering itself down, after which remove the bootable USB media and switch back on (with the little button). Checkout the 1st ubuntu boot sequence on the monitor which'll disconcertingly blank immediately afterwards. Figure out the UP2's IP address - AaeonTec mac addrs are labeled on the CAT5 ports - and SSH (port22, putty) into the UP2 using 'upsquared' as both user-name and password.If that all worked...

```
sudo apt-get update
sudo apt-get upgrade
```
DO NOT: `sudo apt-get install python-opencv` # => python2
check that python3 is pre-loaded, then get its pip...
```
sudo apt-get install python3-pip
```
upgrade pip(3) at your own risk, it didn't end well for me!
get opencv and numpy...
```
sudo pip3 install opencv-python
```
get PILlow dependencies & PILlow...
```
sudo apt-get install libtiff5-dev libjpeg8-dev \
 zlib1g-dev libfreetype6-dev liblcms2-dev \
 libwebp-dev tcl8.5-dev tk8.5-dev
sudo .local/bin/pip3.5 install Pillow
```
or maybe try: `sudo -H pip3 install Pillow`

Please change the default password and continue with the get UKRegOCR step as above. Good luck.

Resources:

OpenALPR:	http://www.openalpr.com/cloud-api.html
	http://doc.openalpr.com/opensource.html
Python3:	https://www.python.org/downloads/
OpenCV:	https://opencv.org/opencv-3-4.html

NumPy: http://www.numpy.org/
PILlow: http://pillow.readthedocs.io/en/4.0.x/installation.html
UKFont: https://www.dafont.com/uk-number-plate.font
Adrian RoseBrock:
https://www.pyimagesearch.com