**Notes on Autonomous Camera Image Processing**

Daniel Buscombe

September 19, 2016

The following sequence of processing steps are required to use an arbitrary image from an autonomous camera system for visualizing, and estimating the planform area of, a sandbar

1. Computing a transformation matrix from image to real-world coordinates. This allows rectification (or georectification) of an oblique image into a planform image with a projected coordinate system. The approach I take it to find the homography between two sets of points: ground control points in image reference system, and their real world coordinates. Note that only 2D points [x,y] are considered. The same transformation matrix will apply to all images from a site/deployment if they are all in the same image reference frame (i.e. they have been ‘registered’, see below) . I use the opencv (<http://opencv.org/>) library in python. The homography and transformation matrix is found using the ‘findHomography’ function (<http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html?highlight=findhomography#findhomography>) and the image rectification is carried out using the warpPerspective (<http://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html#warpperspective>) function
2. Image registration. This is the process of getting all sample (‘raw’) images in the same image reference system.
3. Sandbar segmentation. This is the process of segmenting the sandbar from the background from the registered image. It might be (in order of preference) fully automated, semi-automated, or completely manual. The approach I have worked up is semi-automated (aka ‘supervised’, or ‘interactive’). Perhaps a fully automated approach could be worked up. So far, and after several attempts at trying, this has alluded me.

The main workflow is encoded within two python programs,

* regis\_dft.py. This is a python script that registers (<https://en.wikipedia.org/wiki/Image_registration>) a series of images to a common image. The purpose is to place all images for a given site/deployment in a common image reference frame. For example, all (>15,000) images from the site RC0307Rf are registered to the image, ‘RC0307Rf\_20091012\_1130.jpg’. There are many approaches to this problem. I have tried a few, and the one I settled on (at least for now!) is a standard and simple one: compute the cross-correlation function between the master and given sample image, by means of the fast Fourier transform (FFT), and locating its peak. The peak in two dimensions represents the shift in x and y locations, which are accordingly applied to the sample image to register it. It is therefore a ‘translation only’, frequency-domain image transformation. It is reasonably fast and appears to be robust to lighting variations. It is set up to run in parallel but more could be done to optimize. Initial tests indicate this is just as effective as using a similarity transform, which uses FFT techniques to also estimate any scale and rotation differences (and which is also encoded in the python script) but is slower. Note that the algorithm I have developed for image translation offers no subpixel precision. If necessary, this might be achieved by implementing the algorithm of Guizar-Sicairos et al. (2008) (<http://scikit-image.org/docs/dev/auto_examples/plot_register_translation.html>). Also note that I tried to register images based on common scale invariant features in master and sample images, implementing the SIFT algorithm with RANSAC. It worked ok, but was slow, and when it failed, it failed badly. The 2D-DFT approach is faster and apparently more robust (at least at RC0307R).
* sandbar\_gui.py. This is a python script that interactively segments a sandbar from an image (preferable, a registered image still in an image reference frame, i.e. unrectified) using a GUI. The user selects a set of images (in various ways – see below), and interactively segments them, one by one. The results from each image are saved in a binary python format (a pickled object: <https://docs.python.org/2/library/pickle.html>), one file per image (with the ‘.p’ extension). There are 3 tabs in the program, each with slightly different functionality. The first (‘gold’) tab allows the user to select multiple arbitrary files, make a movie of them, and/or ‘process’ them. The act of ‘processing’ in this case refers to the interactive segmentation which is explained below. The second (‘green’) tab allows the user to find images from the database, based on site/deployment, date range, time of day (including the ‘ideal time’ for a given site, based on average lighting conditions), and discharge level. Right now, it will find images within +/- 100 cfs of the desired discharge level, at the nearest gage. Also, it doesn’t yet implement the stage-discharge relationships for a given site. This isn’t important for the R0307Rf site that will be analysed first, but will matter more as sites far from gages are analyzed. Again, once the images are ‘found’, the user can process and make a movie of these images (the movie this time will include a plot of the discharge curve for the selected date range). The third (purple) tab allows the user to find images from the database, based on site/deployment, date range, and time of day (including the ‘ideal time’ for a given site, based on average lighting conditions). In other words, it does the same as the green tab except the specification of discharge. Ok, so how it works: you draw a box around the bar and press the ‘n’ key. It makes an initial guess at finding the sandbar within the box. Press ‘n’ again, and it updates its guess. After a few ‘n’ presses, it doesn’t change. At that point, you either accept that it has done a good job, and save the result and move on to the next image, or you give it new information that allows it to improve its guess. This is done by specifying what is foreground (sandbar) or background in the image with the mouse. Foreground marking is done by hitting the ‘1’ key, then drawing on the image where there is sandbar it has missed. Background marking is done by hitting the ‘0’ key, then drawing on the image where there is no sandbar. See the movie for an example. What’s going on in the background is the GrabCut algorithm (<https://en.wikipedia.org/wiki/GrabCut>) implemented in python’s OpenCV (<http://opencv.org/>) implementation. See this link for an example, which I used to base the current program: <http://docs.opencv.org/3.1.0/d8/d83/tutorial_py_grabcut.html>. As well as the Grabcut segmentation, some additional filtering is taking place based on texture and image intensity that helps refine the process. I will write up a more detailed description in due course.