Mosaic-Library

This is a python library for manipulating images for underwater mapping applications. There are several examples to get you familiar with the concepts of correcting an images color balance, contrast, and lighting, applying a static rotation to an image to change the perspective, and combining all of these with image registration to mosaic an image.

Getting Started

First, install Python 3.8.10 for your system.

(Optional): install an Integrated Development Environment, I recommend PyCharm Community

Download the mosaic-library from learn.inside.dtu.dk. Extract the folder somewhere convenient for you to find.

Make sure you copy down the path location of your mosaic-library folder (i.e. where this .pdf is being read from!), I will refer to this directory as $WORKSPACE from now on.

If Using PyCharm

1. Open PyCharm
2. Select 'Open Project'
3. Navigate to $WORKSPACE, click open.
4. Choose to initialize Python 3.8.10 as a new virtualenv.
5. Open a terminal tab at the bottom of the window, type `python3 -m pip install --update pip setuptools`
6. Install the package dependencies `python3 setup.py install`

If Using Windows (No PyCharm)

1. Press `windows key + r`
2. Type 'cmd' into the run dialog box and press ok.
3. Type `python3 -m pip install --update pip setuptools` *Note*: if there is an error complaining about missing python3, try python instead. Otherwise we will have to fix your system path, just ask.
4. Change directory to your workspace by `cd $WORKSPACE`
5. Install the package `python3 setup.py install`

If Using OSX (No PyCharm)

1. Open a terminal (search for `Terminal` in LaunchPad)
2. Type `python3 -m pip install --update pip setuptools` *Note*: if there is an error complaining about missing python3, try python instead. Otherwise we will have to fix your system path, just ask.

3. Change directory to your workspace by `cd $WORKSPACE`
4. Install the package dependencies `python3 setup.py install`

Preprocessing Images

Images are essentially 3-Dimensional Tensors (Videos are 4-Dimensional), and so typical linear algebraic operations (matrix multiplication, inversion, *etc.*) will work on them. Images can appear faded or have low-contrast due to underwater effects. Histogramming techniques can balance the color intensity between channels (color fixing), balance the overall light intensity of the image, or balance the contrast of the image. Three of these methods are demonstrated in color-correction.py to balance channels, intensity and contrast to obtain clearer pictures.

1. In a terminal, navigate to $WORKSPACE/examples
2. Run `python3 color-correction.py ../images/corrections/underwater-fishes.png`
3. Four images are shown, from left to right: Original, Color Corrected, Contrast Corrected, Lighting Corrected
4. Press any key to quit.

5. Experiment with the inputs to each the fix_color, fix_light and fix_contrasts function in examples/color-correction.py.

How do these parameters affect the quality of the output image?

Transforming Images

Applying geometric transformations to images is relatively easy. A rotation matrix, R, that specifies the roll, pitch, and yaw angles can rotate an about its principle axes in sequence to appear as if looked at from another position. This is useful for lining images up so that they match closely with each other (this is called image registration in the literature). Additionally it is useful to correct for perspective. If we have some idea of where the camera is relative to the image, then we can warp the image by rotating it about the approximate camera position.

This example investigates and applies a transformation to the image.

1. In a terminal, navigate to $WORKSPACE/examples
2. Run `python3 perspective-test.py ../images/homography/train.jpg`

Why do the rails appear to vanish to a single point?

What are we doing with the camera when we shift the perspective like this?

Are we able to get the tracks to appear parallel (as they are in reality)?

Mosaicking

Mosaicking is the process of finding correspondences between images and arranging them so that they create a larger image. Video can be used to incrementally expand a mosaic by adding in new tiles with each new frame. The images are preprocessed first to add clarity (making it easier to find matching features), then a transformation is computed between the existing mosaic and the new image that maximises correlation between the matched features. Then a perspective transformation (similar to the previous example) and translation are applied to the image (like the previous example). This is done over and over for each new image, and the mosaic grows.

1. Navigate to $WORKSPACE
2. Run the following: `python3 simple-similarity.py images/mosaicking/GOPR0009_Trim.mp4 --save_freq=10 --alpha=0.9 --max_mosaic_size=20000 --output_directory=test --show_mosaic --scale_factor=0.5`
3. Observe how the mosaic changes over time.
4. Press q to quit early.
5. The mosaic will be saved in output_mosaic.png.

Do any parts of the mosaic appear poorer in quality than others? If so, why?

Control of the camera is particularly important as the new image will become smaller and smaller with respect to the mosaic as the camera moves "into" or "through" the mosaic. Eventually the number of common features between the mosaic and the image will diminish to the point where no good transformation can be found. Or the camera may view something that does not have enough features to estimate a transformation. Or, the mosaic size might get too big. The current implementation simply just dumps the mosaic to a file (called tmp_001.png, tmp_002.png etc.) and starts a new mosaic on the latest image.

What could be a way to reduce this diminishing matching features problem?

Activities

1. Create your own video and see how large a mosaic you can create (substitute your video path in for the one given in the instructions).
2. What kind of camera control and survey works best?
3. Apart from color, contrast and lighting transforms, are there any geometric transformations that we could apply to get better results?