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 clone or extract this repository into a location with an environment setup with Python 3.7.

```
git clone https://fftxtreme@bitbucket.org/mosaick-toolbox/mosaic-library.git
```

NOTE: REPO SNAPSHOT AVAILABLE ON CN.INSIDE

Create a python virtual environment, then install the dependencies:

```
pip install -r requirements.txt
```

Look in the examples folder for the example scripts we will be playing with today :)

Preprocessing Images

Images are essentially 3-Dimensional Tensors (Videos are 4-D), and so typical linear algebraic operations (matrix multiplication, inversion, *etc.*) will work well 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 used in color-correction.py to balance channels, intensity and contrast to obtain clearer pictures.

1. Copy the examples/color-correction.py script into the same directory level (up a level) as the mosaicking package (alternatively, set up a configuration file with the working directory specified on the level containing the mosaicking package)
2. Run python color-correction.py ./images/corrections/underwater-fishes.png
3. A directory will be created in your current folder called "outputs", the three outputs from the color correction process will be saved there. Close the displayed graph (which is the same 3 + the original image)

Activity

The mosaicking/preprocessing.py file contains the functions that perform these corrections, they have additional parameters that can be modified.

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). This example first rotates an image by 45 degrees, then applies a 20 degree pitch (the image appears as if on a slope), then finally a 60 degree roll.

1. Copy the examples/euler-rotate.py script into the same directory level (up a level) as the mosaicking package (alternatively, set up a configuration file with the working directory specified on the level containing the mosaicking package)
2. Run python euler-rotate.py -y 45 -p 20 -r 60 -d ./images/corrections/underwater-fishes.png
3. A directory will be created in your current folder called "outputs", the transformed version of underwater-fishes.png will appear there.

What does applying rotations of more than 90 degrees in pitch and roll do?

Mosaicking

Mosaicking is the process of finding relationships 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. Copy the examples/mosaic.py script into the same directory level (up a level) as the mosaicking package (alternatively, set up a configuration file with the working directory specified on the level containing the mosaicking package)
2. Run python mosaic.py ./images/mosaicking/Fishes.MOV
3. A directory will be created in your current folder called "outputs", the output mosaic (and sub mosaics) will appear here.

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. 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. Does applying preprocessing or static transforms help the mosaic (*i.e.* more features, less diminishing matches)?