mosaic-library

This is a python library for manipulating images for underwater mosaicking applications. Currently, this libary is capable of:

1. Reading input videos with the VideoPlayer class
2. Applying color, contrast, and lighting balancing as well as detail enhancement.
3. Other image manipulations such as resizing for faster mosaicking.
4. Image registration, such as feature detection and description, matching, and transformation estimation in pure translation, similarity, affine, and perspective classes.
5. Infinite and finite homography transformations on imagery given extrinsic rotations and translations between the camera and a new "virtual" camera.
6. Basic time synchronisation between camera position and rotation in the world frame with the video.
7. Visualisation of the preprocessing, feature detection / registration, homographic transform, and mosaicking steps.
8. Basic mosaicking map generation, with minimal memory management support (tiling).

Future Roadmap:

1. Estimation of a ground plane given additional extrinsic information from the mobile camera platform.
2. Estimation and removal of vanishing line to reduce OutOfMemory errors due to excessive image warping.
3. Memory management through iterative updating of mosaic tiles according to a common mosaic coordinate system.

Getting Started

First, make sure you have installed either Python 3.8.10 or Python 3.9 for your system.

(Optional): install an Integrated Development Environment, for example: PyCharm Community.

Create and activate virtual environment (guide here for non-pycharm users).

Install the mosaic-library package: `python -m pip install mosaic-library` (internet connection required).

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. Additionally, a sharpness filter cv2.detailEnhance has been exposed in `mosaicking.preprocessing.enhance_detail`

Example Color Correction

```
from mosaicking import preprocessing
from skimage.data import astronaut

img = astronaut()  # Load a good image
img = img[:,:,[2,1,0]]                # Convert to OpenCV BGR color channel ordering (not nec
img[:, :, -1] = 0.4 * img[:, :, -1]  # Reduce the Red channel to make the image seem underwa

color_balance = preprocessing.fix_color(img, 0.9)  # Apply Color balancing
# Display the comparison
preprocessing.cv2.imshow("astronaut (push any key to quit)",
                         preprocessing.np.concatenate(
                         (img, color_balance), axis=1))
preprocessing.cv2.waitKey()
# Cleanup
preprocessing.cv2.destroyWindow("astronaut (push any key to quit)")
```

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 image about its optical center in sequence to appear as if looked at from an equivalent camera from a different orientation. This is useful for projecting an image onto a desired plane (such as a ground plane). Additionally, translation transformations can be applied to the image. This, in combination with rotation, can align images taken at different positions. In both situations it is essential to know the intrinsic properties of the camera in order to correct for changes in 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 extrinsic and intrinsic transformations to the image. See if you can make the train tracks appear parallel. To exit, press ESC with either image window active.

Other questions:

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?

Mosaicking

Mosaicking is the process of finding correspondences between images and arranging them so that they create a larger image. Videos or sequences of images can be used to incrementally expand a mosaic by adding in new tiles with each new frame. The images are preprocessed first to increase clarity and sharpness (making it easier to find features), then a transformation is computed between the existing mosaic and the new image that maximises correlation between the matched features. Then similarity, affine or perspective transformations (similar to the train example) are applied to the image. These steps are iterated over each new image, and the mosaic grows.

The mosaicking script has been exposed as a command (make sure you have it added to your path) as well as through the module.

To see the usage:

```
python3 -m mosaicking.mosaic -h

usage: mosaic.py [-h] [--output_directory OUTPUT_DIRECTORY] [--start_time START_TIME | --sta
                 [--finish_time FINISH_TIME | --finish_frame FINISH_FRAME] [--frame_skip FRA
                 [--sync_points SYNC_POINTS SYNC_POINTS | --time_offset TIME_OFFSET] [--min_
                 [--max_warp_size MAX_WARP_SIZE [MAX_WARP_SIZE ...]] [--max_mosaic_size MAX_
                 [--alpha ALPHA] [--show_rotation] [--show_mosaic] [--show_preprocessing] [-
                 [-k INTRINSIC INTRINSIC INTRINSIC INTRINSIC INTRINSIC INTRINSIC INTRINSIC I
                 [-y YROTATION] [-z ZROTATION] [-g GRADIENTCLIP] [-f] [--homography {rigid,s
                 [--features {ORB,SIFT,SURF,BRISK,KAZE,ALL} [{ORB,SIFT,SURF,BRISK,KAZE,ALL}
                 video

positional arguments:
  video                 Path to video file.

optional arguments:
  -h, --help            show this help message and exit
  --output_directory OUTPUT_DIRECTORY
                        Path to directory where output mosaics are to be saved. Default is i
  --start_time START_TIME
                        Time (secs) to start from.
  --start_frame START_FRAME
                        Frame number to start from.
  --finish_time FINISH_TIME
                        Time (secs) to finish at.
  --finish_frame FINISH_FRAME
                        Frame number to finish at.
  --frame_skip FRAME_SKIP
                        Number of frames to skip between each mosaic update.
  --orientation_file ORIENTATION_FILE
```

```
                        Path to .csv file containing orientation measurements that transfor
  --sync_points SYNC_POINTS SYNC_POINTS
                        Time points (sec) where video and orientation file are in sync, use
                        orientation file.
  --time_offset TIME_OFFSET
                        Time offset (sec) between video and orientation file timestamps, use
  --min_matches MIN_MATCHES
                        Minimum number of matches to proceed with registration.
  --min_features MIN_FEATURES
                        Minimum number of features to detect in an image.
  --max_warp_size MAX_WARP_SIZE [MAX_WARP_SIZE ...]
                        Maximum size of warped image (used to prevent OOM errors), if 1 argu
                        height, width.
  --max_mosaic_size MAX_MOSAIC_SIZE
                        Largest allowable size (width or height) for mosaic. Creates a new
  --save_freq SAVE_FREQ
                        Save frequency for output mosaic (if less than 1 then output saves a
  --scale_factor SCALE_FACTOR
                        Scale the input image with constant aspect ratio.
  --alpha ALPHA         Alpha blending scalar for merging new frames into mosaic.
  --show_rotation       Flag to display the rotation compensation using rotation data.
  --show_mosaic         Flag to display the mosaic output.
  --show_preprocessing  Flag to display the preprocessed image
  --fix_color           Flag to preprocess image for color balance.
  --fix_contrast        Flag to preprocess image for contrast equalization.
  --fix_light           Flag to preprocess image for lighting equalization.
  -c CALIBRATION, --calibration CALIBRATION
                        Path to calibration file, overrides --intrinsic and --distortion.
  -k INTRINSIC INTRINSIC INTRINSIC INTRINSIC INTRINSIC INTRINSIC INTRINSIC INTRINSIC INTRINS
                        Space delimited list of intrinsic matrix terms, Read as K[0,0],K[0,
                        calibration file if intrinsic present.
  -d DISTORTION [DISTORTION ...], --distortion DISTORTION [DISTORTION ...]
                        Space delimited list of distortion coefficients, Read as K1, K2, p1,
  -x XROTATION, --xrotation XROTATION
                        Rotation around image plane's x axis (radians).
  -y YROTATION, --yrotation YROTATION
                        Rotation around image plane's y axis (radians).
  -z ZROTATION, --zrotation ZROTATION
                        Rotation around image plane's z axis (radians).
  -g GRADIENTCLIP, --gradientclip GRADIENTCLIP
                        Clip the gradient of severely distorted image.
  -f, --fisheye         Flag to use fisheye distortion model.
  --homography {rigid,similar,affine,perspective}
                        Type of 2D homography to perform.
  --features {ORB,SIFT,SURF,BRISK,KAZE,ALL} [{ORB,SIFT,SURF,BRISK,KAZE,ALL} ...]
                        Set of features to use in registration.
  --show_matches        Display the matches.
  --inliers_roi         Only allow the convex hull of the inlier points to be displayed.

  --demo                Creates a video of the mosaic creation process. For demo purposes o
  --show_demo           Display the demo while underway.
```

Try running the mosaicking.mosaic script on a video you've taken with your phone or otherwise.

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

It is important the camera's view is mostly occupied by a scene that lies on a plane (like the seabottom). Control of the camera to keep the ground plane in sight 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 tiles (called tile_001.png, tmp_002.png etc.) and starts a new mosaic on the latest image if the memory consumption goes above a threshold.

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?