Submission By: *Zuhaib Ul Zamann*
Entry Number: 2018MT60798

Approach

Part 1: Harris corner detector

File: Harris.py
Class HarriCornerDetector
Fields:

- **Kernels**: Dictionary of sobel filters

```
Kernels['x']: #Filter for derivative across x-coordinate
Kernels['y']: #Filter for derivative across y-coordinate
```

Methods:

- **Convolve2D**: 2D-Convolution operation with padding set as same, so that the input and output image size is same
- **Smoothen**: Applying gaussian blur to smoothen the picture
- **HarrisMatrix**: Calculates harris matrix values given by $det(H) - 0.05tr(H)^2$. Elements of harris matrix are calculated by taken gaussian weighted sum of each of the terms involved
- **nonMaxSupression**: Perform nonMaxSupression in a window of windowSize. 8-Way local optima is found
- **SelectTopK**: After non-max Supression select the first K points with maximum harris corner value
- **getCorners**: Method for getting the final keypoints(corners)

File: Matching.py Class Matching
Fields:

- **detector**: HarrisCornerDetector. Used for finding Key points

Methods:

- **findNearestMatches**: matches keypoints based on the value of $0.1 \times SSE + \text{Manhat}(\text{pixel coordinates of points})$, where $\text{Manhat}(\text{x,y})$ is the manhattan distance

between x and y given by $\sum_{i=1}^{n} |x_i - y_i|$

- **match**:- Wrapper function over `Matching.findNearestMatches`
- **findAffine**: Wrapper function over cv2 library function `cv2.findHomography` . Uses `Matching.match` to get the matches and then calls `cv2.findHomography`
- **trim**: Function to crop the image to get rid of the extra 0s
- **findHomograpgies**: calls `findAffine` over adjacent images to get the homographies between adjacent images
- **generatePanoroma**: Takes folder address, loads images using `os.listdir` and `cv2.imread` and then calls `self.detector` to get the key points and `Matching.findhomographies` to get the homographies. Using `cv2.warpPerspective` , applies homography on right images and overlays left image(Because the homography calculates the view in coordinates of left image). Saves the final panorama at the folder address with name as `pan.jpg` Image is resized to (400, 400) to reduce computational cost

File: AffineModel.py
Class AffineModel
Fields:

- **detector**: HarriscCornerDetector. Used to calculate keypoints
- **match**: Matching, used to calculate matching between keypoints
- **createMatrix**: Given matches as $((x_1, y_1) \to (x'_1, y'_1), (x_2, y_2) \to (x'_2, y'_2), \ldots, (x'_n, y'_n))$ creates the matrix. $\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_n & y_n & 1 \end{bmatrix}$.
  We only pass the source points and denote the matrix by A
- **createLoad**: Given matches as $((x_1, y_1) \to (x'_1, y'_1), (x_2, y_2) \to (x'_2, y'_2), \ldots, (x'_n, y'_n))$ creates the matrix. $\begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ \vdots \\ x'_n \\ y'_n \end{bmatrix}$

We only pass the destination points and denote the matrix by b

- **estimateParams**: Linear regression estimate for the equation $At = b$, where $t = \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix}$ and the affine transform is then given by $\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}$
- **builtinEstimator**: Use `cv2.estimateAffine2D` to estimate the affine matrix params
- **getMatches**: Same as `Matching.match` function
- **getAffine**: similar to `Matching.findAffine` . Instead of homography, it finds the affine matrix
- **getAffines**: Similar to `Matching.findHomographies`
- **generatePanorama**: Similar to `Matching.generatePanorama` . Uses `AffineModel.getAffines` in place of `Matching.findHomographies` and `cv2.warpAffine` instead of `cv2.warpPerspective`

Requirements

opencv-python==4.7.0
matplotlib==3.6.2
numpy==1.23.4
tqdm==4.64.1
scipy==1.10.0

How to run

In the file affine Model change the line at last

```
A.generatePanorama('NewData/5')
```

to

```
A.generatePanorama(<yourfolderAddress>)
```

Make sure that if the frames are sorted in lexicographical order, then the images should run from left to right

Now from command line run the script `AffineModel.py` and the output will be saved as **pan.jpg** in the folder from which data is loaded

Results

One drive Link