3D Computer Vision: HW1

Panoramic Image Generation

1 Submission

- **Assignment due: 11:59 PM Thursday April 17th**
- Skeletal code and data can be downloaded from: `https://drive.google.com/file/d/1mzTxYh4yRd345bTdKFetxHqSLgZ6ujfJ/view?usp=sharing`.
- Individual assignment
- Submission through ETL. The submission file format should be [Student_ID]_HW1.py.
- Use Python
- You will complete HW1.py including the following functions:
  - `MatchSIFT`
  - `EstimateH`
  - `EstimateR`
  - `ConstructCylindricalCoord`
  - `Projection`
  - `WarpImage2Canvas`
  - `UpdateCanvas`
- DO NOT SUBMIT THE PROVIDED IMAGE DATA
- The function that does not comply with its specification will NOT BE GRADED.
- You are not allowed to use computer vision related package functions unless explicitly mentioned here. Please consult with TAs if you are not sure about the list of allowed functions.
- This homework has been borrowed from 3D Computer Vision Class of University of Minnesota. Special thanks to Professor Hyun Soo Park for allowing the use of this homework.

3D Computer Vision: HW1

Panoramic Image Generation

2 Overview

In this assignment, you will implement a method to create a panoramic image from a set of images using image transformation as shown in Figure 1. The pseudo code can be found in Algorithm 1.

(a) Input images

(b) Result panoramic image

Figure 1: (a) Given a set of images of pure rotation, you will implement an algorithm (b) to automatically create a panoramic image.

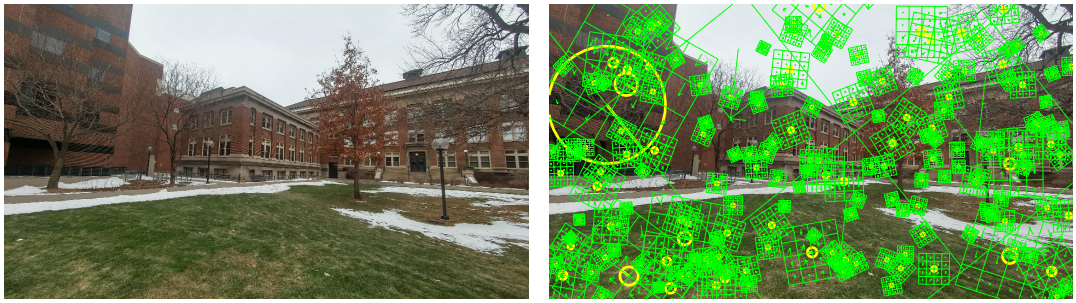
Algorithm 1 Create Panoramic Image

Read all images.
$\mathbf{R}_1 = \mathbf{I}_3$ ▷ Rotation for the first image.
for i^{th} image in all images except the last **do**
    Load consecutive images $\mathbf{I}_i$ and $\mathbf{I}_{i+1}$.
    Extract SIFT feature using `OpenCV` for each image
    Find the matches between two images ($\mathbf{x}_1 \leftrightarrow \mathbf{x}_2$) ▷ `MatchSIFT`
    Estimate the homography between images using RANSAC ▷ `EstimateH`
    Compute the relative rotation matrix, ${}^{i+1}\mathbf{R}_i$ ▷ `EstimateR`
    Compute the $(i+1)^{\text{th}}$ rotation matrix, $\mathbf{R}_{i+1}$.
end for
Generate 3D points on the cylindrical surface ▷ `ConstructCylindricalCoord`
for i^{th} image in all images **do**
    Project the 3D points to the i^{th} camera plane ▷ `Projection`
    Warp the image to the cylindrical canvas. ▷ `WarpImage2Canvas`
    Update the canvas with the new warped image. ▷ `UpdateCanvas`
end for

3D Computer Vision: HW1

Panoramic Image Generation

3 Sift Feature Extraction



(a) Image (b) SIFT descriptor

Figure 2: Given an image (a), you will extract SIFT features using OpenCV.

To stitch images, you need to match keypoints in images using image local descriptor. You will use SIFT descriptor to enable matching (Figure 2). We will use OpenCV library for SIFT extraction given your images.

(Note) You will use this library only for SIFT feature extraction and its visualization. All following visualizations and algorithms must be done by your code.

(Note) The function for SIFT feature extraction lie in the contrib module of OpenCV library, so you need to install opencv-contrib-python package additionally.

- pip3 install opencv-python
- pip3 install opencv-contrib-python

If you have any patent issues, you can reinstall opencv-python and opencv-contrib-python package to an earlier version 3.4.2.16 easily through following two command lines.

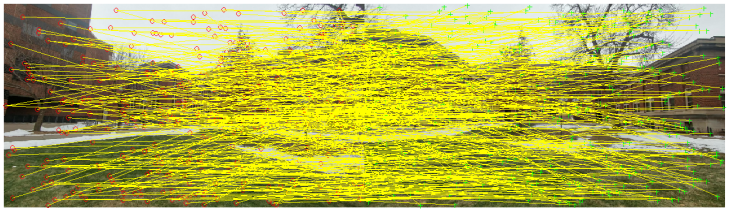
- pip3 install opencv-python==3.4.2.16
- pip3 install opencv-contrib-python==3.4.2.16

(SIFT visualization) Use `OpenCV` to visualize SIFT features with scale and orientation as shown in Figure 2 (OpenCV may different colors to visualize). You may want to follow the following tutorial:
`https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html`

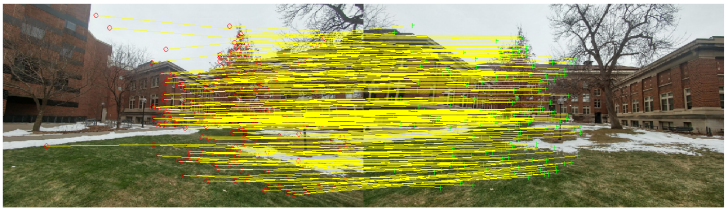
3D Computer Vision: HW1

Panoramic Image Generation

4 SIFT Feature Matching



(a) Raw matches



(b) Filtered matches after ratio test

Figure 3: You will match points between the template and target image using SIFT features.

The SIFT is composed of scale, orientation, and 128 dimensional local feature descriptor (integer), $\mathbf{f} \in \mathbb{Z}^{128}$. You will use the SIFT features to match between two images, I_1 and I_2. Use two sets of descriptors from the template and target, find the matches using nearest neighbor with the ratio test. You may use `NearestNeighbors` function imported from `sklearn.neighbors` (You can install `sklearn` package easily by "pip3 install -U scikit-learn"), or `scipy.spatial`.

```
def MatchSIFT(loc1, desc1, loc2, desc2):
  ...
  return x1, x2
```

Input: `loc1` $\in \mathbb{R}^{n_1 \times 2}$ and `desc1` $\in \mathbb{R}^{n_1 \times 128}$ are keypoint locations and their SIFT descriptors where n_1 is the number of keypoints detected in image 1.
Output: `x1, x2` $\in \mathbb{R}^{n \times 2}$ are matches (correspondences) where n is the number of matches.
Description: Each row of `x1` and `x2` contains the (x, y) coordinate of the point correspondence in I_1 ad I_2, respectively. The matching function is supposed to filter the correspondences based on bi-directional consistency and the ratio test.

(Note) You can only use SIFT module of `OpenCV` for the SIFT descriptor extraction. Matching with the ratio test needs to be implemented by yourself.

5 Robust Homography Estimation

Figure 4: You will compute a homography using SIFT matches filtered by RANSAC.

Given the matches based on SIFT descriptors, you will compute a homography that fits to the image transformation (i.e., pure rotation).

```
def EstimateH(x1, x2, ransac_n_iter, ransac_thr):
  ...
  return H, inlier
```

Input: `x1` $\in \mathbb{R}^{n\times 2}$ and `x2` $\in \mathbb{R}^{n\times 2}$ are the set of correspondences. `ransac_n_iter` and `ransac_thr` are the number of iterations and the error threshold for RANSAC.
Output: `H` $\in \mathbb{R}^{3\times 3}$ homogaphy and the set of inlier indices `inlier` $\in \mathbb{Z}^k$ where k is the number of inliers.
Description: The estimated homography will map $\mathbf{x}_1$ to $\mathbf{x}_2$, i.e., $\mathbf{x}_2 = \mathbf{H}\mathbf{x}_1$. You can visualize the filtered inliers based on RANSAC as shown in Figure 4.

```
def EstimateR(H, K):
  ...
  return R
```

Input: `H` $\in \mathbb{R}^{3\times 3}$ and `K` $\in \mathbb{R}^{3\times 3}$ are the estimated homography and camera intrinsic parameter.
Output: `R` $\in \mathbb{R}^{3\times 3}$ is the relative rotation matrix (i.e., orthogonal matrix) from image 1 to image 2.
Description: The computed rotation matrix is the relative rotation between the image 1 and image 2. This rotation can be used to compute the absolute rotation of the $(i{+}1)^{\text{th}}$ image given that of the i^{th} image.

3D Computer Vision: HW1

Panoramic Image Generation

6 Coordinate Transformation

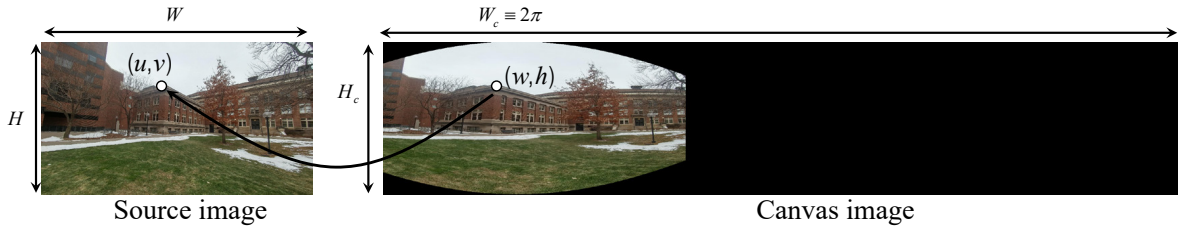


Figure 5: You will find a mapping from the panoramic canvas to the image to transport the pixel, i.e., $(w, h) \rightarrow (u, v)$.

You will warp an image to the panoramic canvas that is the unfolded surface of a cylinder where the width and height of the canvas are W_c and H_c, respectively. A point in the cylindrical surface can be parametrized by angle $\phi \in [0, 2\pi]$ and height h as shown in Figure 6. Note that the angle ranging $[0, 2\pi]$ can be converted to the pixel coordinate ranging $[0, W_c]$ using $\phi = w\frac{2\pi}{W_c}$ (we assume the width of the canvas is equivalent to a full rotation, i.e., $W_c \equiv 2\pi$. For the height of the canvas, it is identical to the height of the source image, i.e., $H_c = H$.

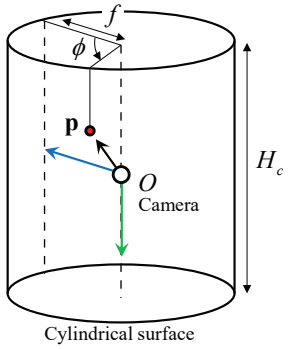


Figure 6: Cylindrical coordinate.

To warp the image, you will apply the backward (inverse) warping scheme, i.e., you transport the pixel at the transformed coordinate in the image to the canvas coordinate, $(w, h) \rightarrow (u, v)$ as shown in Figure 5. This mapping from the canvas to the source image can be derived as a composition of mappings through a 3D point on the cylindrical surface (Figure 6): a point in the canvas (w, h) is mapped to a 3D point $\mathbf{p} \in \mathbb{R}^3$ on the surface of the cylinder, and in turn, the 3D point is mapped to the point in the source image (u, v) through projection, i.e., $(\phi, h) \rightarrow \mathbf{p} \rightarrow (u, v)$.

Figure 6 defines the coordinate system. The camera center is located at the cylinder center, O where the green and blue arrows indicate the y and z axes, respectively. We assume the source image and canvas have the same focal length, i.e., the radius of the cylinder is the focal length, ($f = \mathbf{K}_{11}$ where $\mathbf{K}$ is the intrinsic parameter of the source image). Given the geometry, it is possible to derive the coordinate of the 3D point (red) in terms of ϕ, h, f, W_c and H_c.

3D Computer Vision: HW1

Panoramic Image Generation

```
def ConstructCylindricalCoord(Wc, Hc, K):
  ...
  return p
```

Input: `Wc` and `Hc` are width and height of the canvas. You will set the `Wc` to be (# of images)$\times W/2$ where W is the width of the image, and the `Hc` to be the height of the image H. $\mathbf{K}$ is the intrinsic parameter of the source images.
Output: $\mathtt{p} \in \mathbb{R}^{\mathtt{Hc}\times\mathtt{Wc}\times 3}$ is the 3D points corresponding to all pixels in the canvas.
Description: Given the size of the canvas, you can generate the canvas coordinates using `numpy.meshgrid`, and express the corresponding 3D points using the geometry in Figure 6.

```
def Projection(p, K, R, W, H):
  ...
  return u, mask
```

Input: $\mathtt{p} \in \mathbf{R}^{\mathtt{Hc}\times\mathtt{Wc}\times 3}$ is a set of 3D points that correspond to every pixel in the canvas image. W and H are the source image size.
Output: $\mathtt{u} \in \mathbb{R}^{\mathtt{Hc}\times\mathtt{Wc}\times 2}$ is the 2D projection of the 3D points $\mathbf{p}$, and $\mathtt{mask} \in \{0,1\}^{\mathtt{Hc}\times\mathtt{Wc}}$ is the corresponding binary mask indicating valid pixels.
Description: If a point is valid, the corresponding mask entry is one, and zero otherwise. A projected point is not valid:

- if the 3D point is behind the camera
- if the projected point is beyond the image boundary.

For instance, the black pixels in Figure 5 are the ones that are not valid for the first source image.

```
def WarpImage2Canvas(image_i, u, mask_i):
  ...
  return canvas_i
```

Input: $\mathtt{image\_i} \in [0,255]^{\mathtt{H}\times\mathtt{W}\times 3}$ is the i^{th} RGB image (`uint8` format) with W width and H height, $\mathtt{u} \in \mathbb{R}^{\mathtt{Hc}\times\mathtt{Wc}\times 2}$ is the mapped 2D pixel locations in the source image for pixel transport, and $\mathtt{mask\_i} \in \{0,1\}^{\mathtt{Hc}\times\mathtt{Wc}}$ is the valid pixel indicator.
Output: $\mathtt{canvas\_i} \in [0,255]^{\mathtt{Hc}\times\mathtt{Wc}\times 3}$ is the canvas image generated by the i^{th} source image (`uint8` format).
Description: You will use the mapping $(w,h) \rightarrow (u,v)$ to generate the i^{th} canvas image. A new canvas image can be created, and for each pixel location in the canvas image, its pixel value can be copied from `u` that specifies the mapped location in the source image. You would expect to generate a canvas image similar to Figure 5. You may use `scipy.interpolate` for batch pixel warping with bilinear interpolation.

7 Update Canvas Image

Source image Canvas image

Figure 7: Given the previous canvas image and new canvas image, you composite them using the valid pixels.

You will generate the composite of the canvas image, recursively. In Figure 7, given the canvas image generated by the first image (Figure 5) and newly generated canvas image from the new source image, the canvas image can be updated based on the valid pixels.

```
def UpdateCanvas(canvas, canvas_i, mask_i):
  ...
  return canvas
```

Input: `canvas` $\in [0, 255]^{\texttt{Hc}\times\texttt{Wc}\times 3}$ is the previously generated canvas, `canvas_i` $\in [0, 255]^{\texttt{Hc}\times\texttt{Wc}\times 3}$ is the newly generated canvas image from `WarpImage2Canvas`, and `mask_i` $\in \{0, 1\}^{\texttt{Hc}\times\texttt{Wc}}$ is the valid pixel indicator.
Output: `canvas` $\in [0, 255]^{\texttt{Hc}\times\texttt{Wc}\times 3}$ is the updated canvas image (`uint8` format).
Description: This updating function composites two canvas images based on the valid pixel indicator.