CIS 580, Machine Perception, Spring 2023
Homework 4
Due: Thursday, April 13th 2023, 11:59pm ET

In this homework, you are going to implement a two-view stereo algorithm to convert multiple 2D viewpoints into a 3D reconstruction of the scene.

The main code is implemented in an interactive jupyter notebook `two_view.ipynb` which imports several functions from `two_view_stereo.py` which you are responsible for.

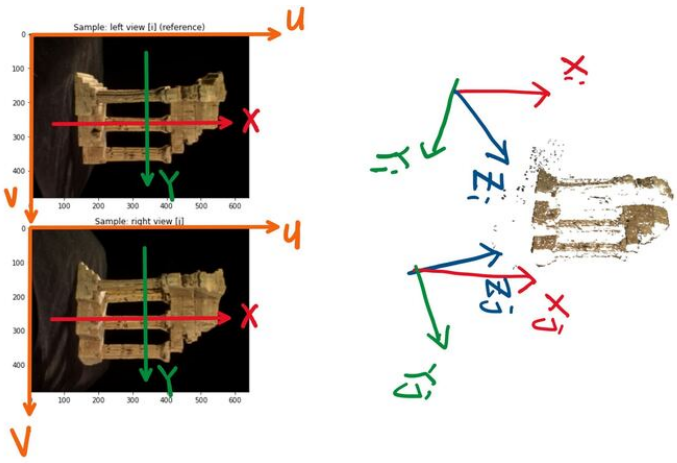
The prerequisite python libraries you will need to install to run the code are included in `requirements.txt` which if you are a pip user, you can install via `$ pip install -r requirements.txt`. Note that for the K3D library, which is used to visualize the 3D point clouds in the jupyter notebook, there are some additional steps after pip installation (If you are using VS-Code, VS-Code will automatically help you to handle this). Namely, you may run the following lines after installation to explicitly enable the extension:

```
$ jupyter nbextension install --py --user k3d
$ jupyter nbextension enable --py --user k3d
```

You need to submit your code to our auto-grader in gradescope.

Note that in the readme file, we provide some common mistakes and hint to help you better debug.

In `two_view.ipynb` notebook, we first would like to get an understanding of the dataset we are working with and visualize the images:



The row index of the image corresponds to pixel coordinate v and the horizontal direction Y of the camera frame. The column index of the image corresponds to pixel coordinate u and the vertical direction X.

To further help your understanding of the scene, you can uncomment the function `viz_camera_poses([view_l, view_r])` to interactively visualize the coordinate frames of the viewpoints. You can press `[i]` to show the world coordinate frame.

Two View Stereo (100pts)

We use `view_l` as the left view and `view_r` as the right view.

1. (25pts) Rectify Two view.

   (a) (5pts) Understand the camera configuration. We use the following convention: $P^i = R^i_w P^w + T^i_w$ to transform coordinates in the world frame to the camera frame. In the code, we use `R_iw` to denote R^i_w. You will need to compute the right-to-left transformation $P^i = R^i_j P^j + T^i_j$ and the baseline B. Complete the function `compute_right2left_transformation`

   (b) (10pts) Compute the rectification rotation matrix R^{rect}_i to transform the coordinates in the left view to the rectified coordinate frame of the left view. Complete `compute_rectification_R`. Important note: You can find the derivation for rectification in the two-view stereo slides (Lec22-slides.pdf), but remember that the images in our dataset are rotated clockwise and thus the epipoles should not be placed at x-infinity but instead at y-infinity. **Hint**: move the epipole to the y-infinity using: $[0, 1, 0]^T = R^{rect}_i e_i$

   (c) (10pts) We implemented half of the two-view calibration for you after getting R^{rect}_i. Complete the function `rectify_2view` by first computing the homography and then using `cv2.warpPerspective` to warp the image. When warping the image, use the target shape we computed for you as `dsize=(w_max, h_max)`. We are using the K_{corr} here to enlarge the pictures and eliminate black areas after warping. **Hint**:

$$K^{-1}_{corr} \begin{bmatrix} u_{rect} \\ v_{rect} \\ 1 \end{bmatrix} = R^{rect}_i K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

2. (45pts) Compute Disparity Map

   (a) (5pts) We are going to compare the patch, and complete the function `image2patch` using zero padding on the border (the padding means when you extract the patch of some pixel on or near the border of the image, you may find missing positions in the patch, then you can fill in the missing pixels as zeros.). This function should take an image with shape $[H, W, 3]$ and output the patch for each pixel with shape $[H, W, K \times K, 3]$. The function should work when `k_size=1`.

   (b) (15pts) Complete the three metrics in function `ssd_kernel`, `sad_kernel` and `zncc_kernel`. In `zncc_kernel`, you should return the negative zncc value, because we are going to use `argmin` to select the matching latter. You can find the definition of these three matching metrics below. The metrics treat each RGB value as one grayscale channel and finally you should sum the three (R,G,B) channels (sum across the channels at each pixel, i.e. [H,W,3] would go to [H,W]). The input of each kernel function is a src `[M,K*K,3]` that contains M left patches and a dst `[N,K*K,3]` that contains N right patches. You should output the metric with shape `[M,N]`. Each left patch should compare with each right patch. Try to use vectorized numpy operation in the kernel functions. You are going to get an example plot for pixel (400,200) of the left view and its matching score on the right view. **Note**: We define a small number EPS, please add the EPS to your denominator for safe division in zncc.

- SSD (Sum of Squared Differences) $\sum_{x,y} |W_1(x,y) - W_2(x,y)|^2$
- SAD (Sum of Absolute Differences) $\sum_{x,y} |W_1(x,y) - W_2(x,y)|$
- ZNCC (Zero-mean Normalized Cross Correlation), sometimes just "NCC"

$$\frac{\sum_{x,y}(W_1(x,y) - \overline{W_1})(W_2(x,y) - \overline{W_2})}{\sigma_{W_1}\sigma_{W_2}}$$

  - where $\overline{W_i} = \frac{1}{n}\sum_{x,y} W_i \qquad \sigma_{W_i} = \sqrt{\frac{1}{n}\sum_{x,y}(W_i - \overline{W_i})^2}$

(c) L-R consistency: When we find the best-matched right patch for each left patch, i.e. argmin along the column of the returned [M,N] shape value matrix, this match must be consistent with the match found from the other direction: for each right patch find the best matched left patch, i.e. argmin along the row of returned [M,N] shape value matrix. We provide an example code of the LR consistency check, please understand this code.

(d) (25pts) Implement the full function `compute_disparity_map` using what you understand from the above examples (you can directly copy the example code and then expand upon it).

Hint: one call of `compute_disparity_map` might takes 1-2min, you can use `tqdm` to get a progress-bar.

3. (15pts) Compute Depth Map and Point Cloud: given the disparity map computed above, complete the function `compute_dep_and_pcl` that returns a depth map with shape [H,W] and also the back-projected point cloud in camera frame with shape [H,W,3] where each pixel store the xyz coordinates of the point cloud.

4. (5pts) We implemented most of the post-processing for you to remove the background, crop the depth map and remove the point cloud outliers. You need to complete the function `postprocess` to transform the extracted point cloud in the camera frame to the world frame.

5. (5pts) We implemented the visualization for you with the K3D library; you can directly visualize the reconstructed point cloud in the jupyter notebook. **Upload on the autograder** the screenshot of your Reconstruction using SSD, SAD, and ZNCC kernel.

6. (5pts) Multi-pair aggregation: We call your functions in the full pipeline function `two_view`. We use several view pairs for two view stereo and directly aggregate the reconstructed point cloud in the world frame. **Upload on the autograder** the screenshot of your fully reconstructed point cloud of the temple. Reconstruction may take around 10 min on a laptop.

Students need to complete in `two_view_stereo.py`:
`compute_right2left_transformation`
`compute_rectification_R`
`image2patch`
`ssd_kernel,sad_kernel,zncc_kernel`
`compute_disparity_map`
`compute_dep_and_pcl`
`postprocess`