ECE661 Fall 2024: Homework 9

TA: Rahul Deshmukh (deshmuk5@purdue.edu)

Due Date: 18 Nov 2024, 11.59 PM

Late submissions will be accepted only for special cases.

Turn in typed solutions via BrightSpace. Additional instructions can be found at BrightSpace. This can be a challenging homework, **start early**!

1 Theory Question

Refer to the Epipolar Geometry illustrated in the figure on Page 23-1 of the Lecture 23 handout: Given a pixel $\vec{x}$ in the left camera, we can be sure that its corresponding pixel $\vec{x'}$ in the right camera will be on the Epipolar Line l' shown in the Fig. 1.

The above claim can be justified by the stereo imaging geometry shown in the figure.

You are asked to prove the above claim **theoretically** using the concepts you learned in Lecture 20.

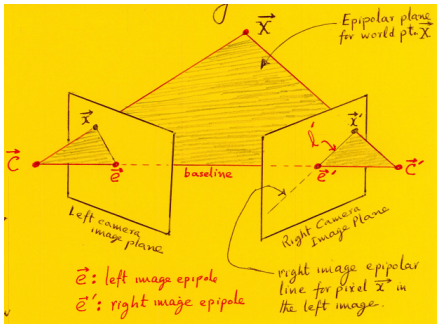


Figure 1: Epipolar Geometry

2 Programming Tasks

This homework consists of three parts:

1. **Projective Stereo Reconstruction:** Create a 3D reconstruction from a pair of images recorded with an uncalibrated camera (such as your own cellphone). Such reconstructions are related to the actual scene structure by a 4×4 homography as you will see.

2. **Dense Stereo Matching:** Given two rectified images, calculate the disparity map for the left image vis-a-vis the right image. Note that rectification means that the two images have been subject to appropriate homographies so that their epipolar lines correspond to the rows in the images and the corresponding rows in two images are also in epipolar correspondence.

3. **Depth Map and Automatic Extraction of Dense Correspondences:** Given a pair of images and their depth maps, automatically extract a dense set of correspondences between the two images. This is a critical step in training deep-learning based keypoint extraction and matching networks.

3 Task 1: Projective Stereo Reconstruction

A 3D reconstruction is called projective if it is related by a 4×4 homography to the real scene. Obviously, this means that what we obtain from projective reconstruction might appear distorted when compared to the actual scene. In practice, depending on how rich the scene structure is and how much prior knowledge one has about the objects in the scene, one may be able to use additional constraints derived from the reconstruction to remove the projective, the affine, and the similarity distortions. In this task, however, our focus is on just creating a projective reconstruction of a scene.

Take a pair of stereo images with your camera, with no particular constraints on how the second image is recorded vis-a-vis the first, as long as the two views are of the same scene. Using the two stereo images perform the following tasks:

3.1 Image Rectification

The goal of image rectification is to find the homographies $\boldsymbol{H}$ and $\boldsymbol{H}'$ for the stereo images such that in the transformed image pair, the same world point appears on the same row. It can be a rather complicated process involving multiple steps. Here is a high-level step-by-step guide:

1. Obtain an initial estimate of the fundamental matrix $\boldsymbol{F}$. Manually extract a set of corresponding points (a minimum of 8) between the two images. Use these correspondences to estimate the fundamental matrix $\boldsymbol{F}$ using the 8-point algorithm, as described in Lecture 23 page 21-6. You will need to enforce the rank constraint on the fundamental matrix $\boldsymbol{F}$ as described on the same page.

2. Estimate the left and right epipoles: $\boldsymbol{e}$ and $\boldsymbol{e}'$. They are the right and left null vector of the fundamental matrix $\boldsymbol{F}$, respectively.

3. Obtain the initial estimate of the projection matrices in canonical form: $\boldsymbol{P}$ and $\boldsymbol{P}'$. Refer to the middle of page 21-6 of Lecture 23 for what it is meant by two cameras being in *canonical configuration*.

4. Refine the right projection matrix $\boldsymbol{P}'$ using nonlinear optimization. The geometric error d^2_{geom} will be your cost function and its calculation will involve triangulating the image points back to their world coordinates, followed by projections back to the two image planes. For details on both the form of

the geometric / reprojection error and the triangulation procedure, refer to page 22-2 of Lecture 24. You only need to refine the right projection matrix since we assume the cameras are in a canonical configuration.

5. Obtain the refined fundamental matrix $\boldsymbol{F}$ using the refined projection matrix $\boldsymbol{P}'$. The relationship between $\boldsymbol{F}$ and $\boldsymbol{P}$ has been given in Lecture 23.

6. Estimate the right homography matrix $\boldsymbol{H}'$ using the refined epipoles. Refer to the last page of Lecture 24 for more details.

7. Estimate the left homography matrix $\boldsymbol{H}$. You can find the full procedure of this estimation in Corollary 11.4 on page 306 of the textbook [1]. What it essentially boils down to is estimating the parameters (a, b, c) in the $\boldsymbol{H}_A$ matrix, which you can solve by forming a inhomogeneous linear system.

8. Apply the homographies to your stereo pair to rectify your images.

3.2 Interest Point Detection

Your next goal is to construct automatically a large set of correspondences between the two images. Toward that end:

- Use the Canny edge detector to extract edge features in your scene. Obtain pixel coordinates from the Canny edge mask. Use these points as your interest points.
- If your image rectification is "perfect", given a pixel in the first image, all you'd need to do for finding the corresponding pixel in the second image is to look at the same row in the latter. Ordinarily, you'd look in the same row and in a small number of adjoining rows for the correspondences.
- In most cases, for any given pixel in the first image, you will end up with multiple candidates in the second image. In such cases, use the SSD or the NCC metric to select the best candidate for each pair of correspondences.
- You also need to ensure that the ordering of the interest points on an epipolar line (which would be a row if your rectification procedure is working well) in the second image is the same as that of the corresponding points in the second image.

3.3 Projective Reconstruction

Once you have established a large quantity of correspondences between your rectified images, repeat steps 1 – 4 of Section 3.1 to back-project your Canny points into their world coordinates. Note that we will still be assuming that the cameras are in the canonical configuration for finding the new projection matrix after rectification. Since our cameras are uncalibrated, the scene will be reconstructed up to a projective distortion.

3.4 3D Visual Inspection

Finally, for visualization:

- Make a 3D plot of the reconstructed points. Draw some "pointer lines" between the corresponding pixels in the two images, on the one hand, and between those pixels and the reconstructed 3D points on the other.
- In the above 3D plot, plot your camera poses as was explained in Homework-8.
- Remember to include a display of the projective distortion in a fashion similar to the one on page 267 of the textbook [1]. You can also refer to the previous years' solutions too for that. You should show at least two different views of your reconstruction.

4 Task 2: The Loop and Zhang Algorithm

In this task, we ask you to run an implementation of the Loop and Zhang algorithm for image rectification and compare the results with those by your own pipeline from Task 1. You will be using the excellent C++ implementation of the Loop and Zhang algorithm that can be found at [2]. For this homework, the original repo has been slightly modified in order to be compatible with modern OpenCV versions. We have also provided the necessary `CMakeLists.txt` to make compilation straightforward. Now here are the steps for this task:

1. Extract the code provided to you in `LoopAndZhang.zip`. Navigate to the root directory of the extracted repo in a Linux terminal.
2. Make sure that you have OpenCV installed in your system. We tested the code with `OpenCV == v4.5.1` but any version `>= v4.0` should also work. For source installation instructions, a helpful guide can be found here [3]. You can also use the OpenCV installation in your conda environment for building the code by just activating your conda environment before the build process.
3. Before compiling the program, change the image paths in `src/main.cpp` to point to your own images or the demo images provided. Then, run the following instruction in a Linux terminal to build the program:
   ```
   $ mkdir build && cd build && cmake ..  && make
   ```
   If the build is successful, you can run the resulting executable by:
   ```
   $ ./main
   ```
4. Examine your rectified output images by the Loop and Zhang algorithm. Compare the correspondences with those by your own image rectification pipeline on the same stereo pair. Discuss in a couple of sentences on what you observe in terms of both how the images have been warped and the quality of the correspondences.

5 Task 3: Dense Stereo Matching

Given a stereo rectified pair, the dense stereo matching problem essentially involves finding as many accurate pixel correspondences as possible and marking occluded regions invalid. For this task, you will implement a rudimentary dense stereo matching algorithm using the Census Transform, you won't be doing the additional test for finding the "invalid" pixels. The goal of this task is not to outperform the state-of-the-art approaches. Instead, the focus is to gain some insights on the challenges that are involved in dense stereo matching.

Before diving into the programming task, you need to understand the notions of the disparity map and Census Transform.

5.1 What is a Disparity Map?

Dense stereo matching essentially boils down to finding per-pixel correspondences in the given stereo rectified pair. Concretely speaking, to compute the disparity map for the left image vis-a-vis the right image, for every pixel in the left image we scan all the potential matching candidates along the same row in the right image. After establishing the correspondences, we simply store the difference in the x-coordinates as the output disparity map. As one can guess, in untextured regions you would find many possible correspondences, such as the black background in the scene in Figure 2. To overcome this ambiguity, there are more sophisticated approaches that aggregate global or semi-global information from a pair. However, for this homework you will only use the local context around each pixel to find the correspondences in the right image, as explained in the next section. The example in Figure 2 is from the Middlebury 2014 high resolution stereo dataset. For this homework you will use a low-resolution stereo pair from the Middlebury 2003 dataset [4].

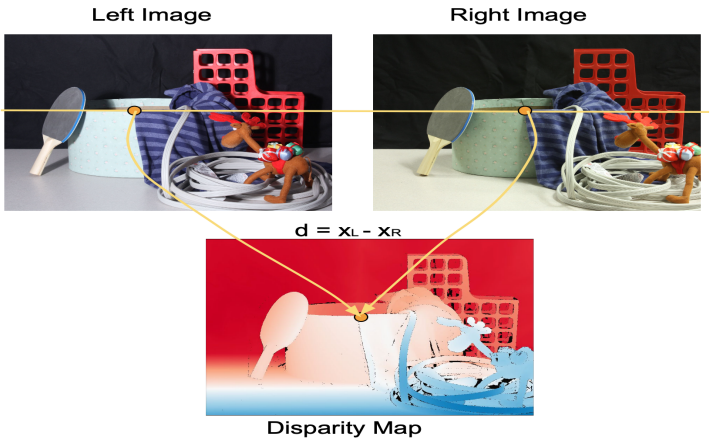


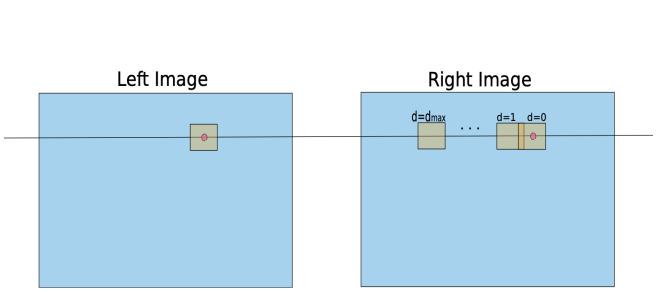
Figure 2: Example groundtruth disparity map. The disparity map stores the per-pixel difference in x-coordinates. The black pixels in the disparity map are occluded and therefore invalid pixels. The stereo pair is a part of the Middlebury 2014 stereo dataset [5].

5.2 Census Transform

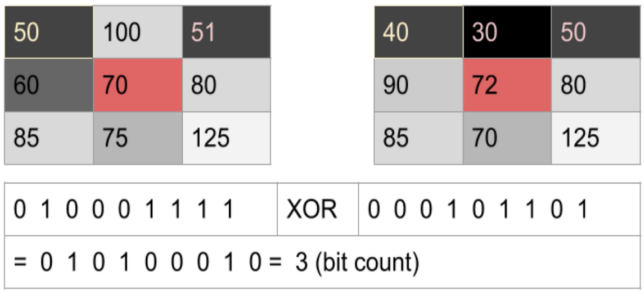
Figure 3 shows the summary of computing the Census transform. On the left hand side of Figure 3(b), you have a local pixel intensity distribution within the $M \times M$ window in the left image centered at pixel $\boldsymbol{p}$ and on the right you have the corresponding local view of a pixel $\boldsymbol{q} = [\boldsymbol{p}_x - d, \boldsymbol{p}_y]$ at disparity d, where the pixel position $\boldsymbol{q}$ is defined in the right image.

For both pixels, we compute a bitvector of size M^2, wherever the pixel intensity value is strictly greater than the center pixel value we make that bit one. This gives us two M^2 bitvectors at pixel $\boldsymbol{p}$ in the left image and at pixel $\boldsymbol{q}$ in the right image. After the bitwise XOR operation between the two bitvectors, we simply compute the number of ones in the output bitvector. This gives us the data cost between the two pixels. Note that $d \in \{0, \cdots, d_{max}\}$. We pick the disparity value d such the data cost is minimized. In the case of multiple minima, pick the first disparity value that results in the minimum data cost while iterating from 0 to d_{max}.

Compute the disparity maps for at least *two* different shapes of the local window. Note that it doesn't have to be a square window, e.g. , 9×7 is also a valid size for the census window.



(a) Scanning a rectified stereo pair for dense stereo matching.



(b) The Census Transform. Note that the red cells are the center pixels and the numbers in each cell represent pixel intensity value. The corresponding M^2 bitvector is computed by setting a bit one wherever the pixel value is *strictly greater* than the center pixel value.

Figure 3: Dense stereo matching using the Census Transform.

5.3 Programming Task

You're provided with a stereo rectified image pair and the corresponding ground truth disparity map for evaluating your stereo matching accuracy. Figure 4 shows the given stereo pair for Task 2. Your task are as follows:

- Estimate the *left* disparity maps using the Census transform for at least two different window sizes and evaluate the accuracy as explained in the next bullet.
- Let $\boldsymbol{D}$ be the given ground truth disparity map and $\widehat{\boldsymbol{D}}$ be the estimated disparity map. Let N be the

total number of valid pixels, i.e. wherever you see the non-black pixels in the groundtruth disparity map in Figure 4(c).

- In order to highlight the challenging regions in the given stereo pair, also show a binary error mask with the value of 255 for pixels where the disparity error is $\leq \delta$ and 0 otherwise. Repeat for the different local window sizes.
- **Special note:** After reading the groundtruth disparity map, convert it into `float32` and divide the disparity values by 4, then convert it back to `uint8` values. You can obtain the d_{max} value from the ground truth disparity map, after the aforementioned adjustment. We need this adjustment because the ground truth disparity map was generated for a higher resolution stereo pair. This is how it's available in the Middlebury 2003 dataset.

(a) Left Image (b) Right Image (c) Left GT Disparity (d) Right GT Disparity

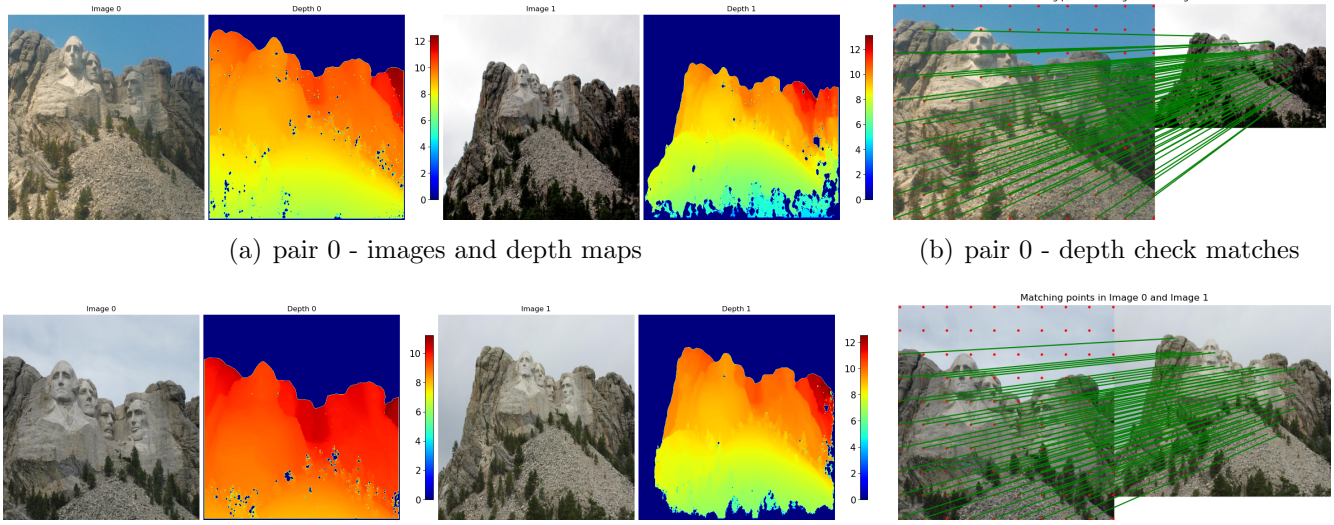
Figure 4: The input stereo pair for Task 2 and the ground truth disparity maps. Note that all the images are of the same size.

6 Task 4: Depth Map and Automatic Extraction of Dense Correspondences

For this task we will first explain what is a depth map and how it can be used for automatically extracting dense correspondences. Then we will give a brief description of the dataset provided for this task and what is expected from the students.

6.1 Depth Map

Depth Map is a popular term in robotics and navigation applications. For a given image (I), its depth map is a 2D matrix that represents the distance of each point in a scene from the camera's principal plane. More formally, for a pixel location ($\boldsymbol{x}$) in the image, its depth value is the Z_{cam} coordinate. As you would guess, the size of the Depth Map should be the same as the image. Fig 5 shows a few example of depth maps. These depth maps were sourced from the MegaDepth dataset [6]. To generate a depth map from images, you would need the camera parameters (intrinsic and extrinsics) and the 3D reconstructed scene. This means that depth map generation can only be done after stereo matching.



(a) pair 0 - images and depth maps

(b) pair 0 - depth check matches

(c) pair 7 - images and depth maps

(d) pair 7 - depth check matches

Figure 5: Example of Depth Maps and matches extracted for pair of images (from [6])

Depth Maps are useful for navigation applications as they give you the information about how far any object is from the camera. In the case of autonomous driving applications, such information can be critical to avoid any collisions with any objects in the scene. However, conventional methods like stereo reconstruction require you to collect several images of a scene in order to estimate depth. To address this limitation, researchers have worked on estimating the depth only from a single image using deep-learning. This task is known as "Monocular Depth Estimation". If you end up working in this domain, you will

probably use the MegaDepth dataset [6]. In addition to depth estimation, the depth maps are useful for automatic extraction of dense correspondences which is discussed in the next section.

6.2 Automatic Extraction of Dense Correspondences using Depth Maps

Given a pair of images (I_A, I_B) and their depth maps (D_A, D_B), you can automatically extract a dense set of correspondences $(\{\boldsymbol{x}_i^A \leftrightarrow \boldsymbol{x}_i^B\})$ between the two images. The idea is to use the depth map to estimate the 3D coordinates of each pixel in the image. Once you have the 3D coordinates, you can project them back to the image plane of the other image and carry out a "Depth Check" through which you can declare them as true correspondences. This process is known as "Automatic Extraction of Dense Correspondences". The extracted correspondences can be used for various applications like stereo matching, image registration, and 3D reconstruction. This technique is critical to training of deep-learning based keypoint extraction and matching networks like SuperPoint + SuperGlue [7, 8], LoFTR [9], CAPS[10], and many more.

We will now explain the the "Depth-Check" process in detail. Given a pixel location $(\boldsymbol{x}_i^A)$ in image I_A and the camera projection matrix $(\boldsymbol{P}_A = \boldsymbol{K}_A \left[\boldsymbol{R}_A | \boldsymbol{t}_A\right])$ we cannot directly estimate the 3D coordinates of the pixel. We can only calculate the 3D ray that passes through the pixel. However, if you have the depth value $(D_A(\boldsymbol{x}_i^A))$, you can estimate the 3D coordinates $(\boldsymbol{X}_i)$ of the pixel as follows:

1. We first calculate the inverse of the camera matrix $(\boldsymbol{K}_A)$ as per Eq. 1. You should note that $\boldsymbol{K}_A^{-1}$ is also a upper triangular matrix with the 3,3 element as 1. This is important for the next step.

2. We now calculate the 3D coordinate $(\boldsymbol{X}_{cam}^A)$ of any point on the ray corresponding to the pixel in the camera coordinate system using Eq. 2. Using Eq. 1 we can see that the last element of the 3D coordinate (i.e. Z_{cam} coordinate) is 1.

3. To get the 3D coordinate of the world point with depth value $D_A(\boldsymbol{x}_i^A)$. We simply multiply Eq. 2 by this depth value as shown in Eq. 3. We now have the 3D coordinate of the point in terms of camera coordinate frame of the image I_A.

4. For the next step, we convert the above 3D coordinate to the world coordinate frame using camera extrinsic parameters $(\left[\boldsymbol{R}_A | \boldsymbol{t}_A\right])$. This conversion is shown in Eq. 4

5. Now using the camera extrinsic of I_B, we can calculate the Z_{cam} coordinate of X_{world} wrt camera coordinate frame of I_B. This is done using $\boldsymbol{X}_{cam}^B = \left[\boldsymbol{R}_B | \boldsymbol{t}_B\right] \boldsymbol{X}_{world}$. The Z_{cam} coordinate is the **estimated depth value**.

6. Next, we project the 3D world coordinate to I_B image and obtain the corresponding pixel location $\boldsymbol{x}_i^B$. Then using the depth map $\boldsymbol{D}_B$, we obtain the **true depth** $\boldsymbol{D}_B(\boldsymbol{x}_i^B)$.

7. Finally, we calculate the absolute difference between the two depth values and if this value is less than a threshold (δ) then the two points $\boldsymbol{x}_i^A \leftrightarrow \boldsymbol{x}_i^B$ are declared as matching correspondences.

8. This whole process is known as the "Depth Check". We show an example of matches obtained using this process in Fig. 5.

$$\boldsymbol{K} = \begin{bmatrix} \alpha_x & s & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow \boldsymbol{K}^{-1} = \begin{bmatrix} \frac{1}{\alpha_x} & -\frac{s}{\alpha_x \alpha_y} & \frac{s y_0 - \alpha_y x_0}{\alpha_x \alpha_y} \\ 0 & \frac{1}{\alpha_y} & -\frac{y_0}{\alpha_y} \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

$$\boldsymbol{x}_i = \boldsymbol{K}\boldsymbol{X}_{cam} \quad \Rightarrow \boldsymbol{X}_{cam} = \boldsymbol{K}^{-1}\boldsymbol{x}_i = \begin{bmatrix} \frac{1}{\alpha_x} & -\frac{s}{\alpha_x \alpha_y} & \frac{s y_0 - \alpha_y x_0}{\alpha_x \alpha_y} \\ 0 & \frac{1}{\alpha_y} & -\frac{y_0}{\alpha_y} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{2}$$

$$\boldsymbol{X}_{cam} = D(\boldsymbol{x}_i) * \boldsymbol{K}^{-1}\boldsymbol{x}_i \tag{3}$$

$$\begin{aligned} \boldsymbol{X}_{cam} &= \left[\boldsymbol{R}\middle|\boldsymbol{t}\right] \boldsymbol{X}_{world} \\ \Rightarrow \begin{bmatrix} \boldsymbol{X}_{cam} \\ 1 \end{bmatrix} &= \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{X}_{world} \\ 1 \end{bmatrix} \\ \Rightarrow \boldsymbol{X}_{cam\_hc} = \boldsymbol{T}\boldsymbol{X}_{world\_hc} \Rightarrow \boldsymbol{X}_{world\_hc} &= \boldsymbol{T}^{-1}\boldsymbol{X}_{cam\_hc} \end{aligned} \tag{4}$$

where hc denotes homogeneous coordinates and $\boldsymbol{T}_{4\times4} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix}$ is a transformation matrix

6.3 Student Task

We have created a small subset dataset for you to practice the automatic extraction of dense correspondences. This subset was created for the scene #1589 of MegaDepth which corresponds to pictures of Mount Rushmore. The subset consists of 10 pairs of images and depth maps. We also provide the camera intrinsic and extrinsic parameters for every pair. Your task is the following:

1. **Implement "Depth Check"**: You need to implement the process as explained above and extract the dense correspondences for each pair. For this task, you are provided a python script `plot_depth_check.py` that has the code for the depth check process. You need to fill in the missing parts of the code indicated by `"""<Student Code> ..."""`. Then generate the plots similar to Fig. 5. You need to attach these plots for each pair in your submission.

2. **Plot the 3D points**: Given a pair of images, you need to estimate the 3D world points of the scene using the depth maps. This can be done independently for the images (I_A, I_B) and the 3D points $(\boldsymbol{X}^A_{world}, \boldsymbol{X}^B_{world})$ can be plotted in a single 3D plot. For every pair, you need to plot the 3D world coordinates as a scatter plot and attach these plots in your report. Make sure that we can see the 3D structure of the scene from the plots.

Note that the depth maps can have nodata values which are represented as 0 (blue regions in Fig. 5). These are the pixels which were not matched by stereo matching. You need to ignore these pixels in the depth check process. Please see the comments in the python script for more details.

Finally, the depth maps are written in HDF5 file format. You need to install the `h5py` library (https://pypi.org/project/h5py/) to read the depth maps.

7 Submission Instructions

Include a typed report explaining how did you solve the given programming tasks.

1. Turn in a zipped file, it should include (a) a typed self-contained pdf report with source code and results and (b) source code files (only .py files are accepted). Rename your .zip file as hw9_<First Name><Last Name>.zip and follow the same file naming convention for your pdf report too.

   There should be two items in your submission - (1) Homework PDF (2) ZIP file containing source code (*.py files) and text files (if any).

2. Your pdf must include a description of

   - Task 1
     - A good description how you implemented each of the sub-tasks with relevant equations.
     - The manually selected points used to estimate the fundamental matrix as well as the matched correspondences between interest points.
     - A plot showing your stereo images before and after image rectification.
     - A plot showing the large set of correspondences based on the Canny points on the rectified pair.
     - The 3D reconstructed results with some marker or guide lines to understand the scene structure.
     - Display two different views of your 3D reconstruction vis-a-vis your two stereo images, demonstrating the projective distortion in your reconstruction.
   - Task 2
     - Explain in a couple sentences the key high-level ideas behind the Loop and Zhang algorithm for image rectification.
     - A plot showing the rectified images and the resulting correspondences by the Loop and Zhang algorithm for same stereo pair that you used for Task 1.
     - A one paragraph comparative discussion on what you observe in terms of both how the images have been warped and the quality of the correspondences by your own pipeline and by the Loop and Zhang algorithm.
   - Task 3
     - Explain in your own words the dense stereo matching algorithm using the Census Transform.

  - Your estimated disparity maps for different window sizes.
  - Accuracy and error masks for $\delta = 2$ and for different window sizes.
  - Your observations on the output quality for different local window sizes.
- Task 4
  - Explain in your own words the automatic extraction of dense correspondences using depth maps and why it is useful.
  - The plots of image, depth maps, and depth check process for each pair.
  - The 3D scatter plots for the 3D world coordinates for each pair.
- Your source code. Make sure that your source code files are adequately commented and cleaned up.
- To help better provide feedbacks to you, make sure to **number your figures**.
- The sample solutions from previous years are for reference only. **Your code and final report must be your own work.**

Homework File Size

For your homework submissions, please ensure that your reports (PDFs) are **under 10MB**.

Some ways to reduce your report sizes are:

- Downsample your input/output image when including in the reports.
- When including plots in your reports, save them as PDFs instead of PNG/JPEG. PDFs are vector formats which are lightweight and do not pixelate when zooming into the plot.

Strike a balance between the file size and image quality displayed in your report. This will help your TA in easy distribution of homeworks for grading without maxing out the disk space. Moreover, this will help you when you upload your homeworks to your individual git repos. You don't want to upload 40 MB PDF to your git repo!

Finally, do not include the images in your ZIP files for your homework submissions. Ideally, the ZIP file should be under 1MB because it only contains ASCII (*.py /*.txt) files.

References

[1] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge university press, 2003. 3, 4

[2] A. G. Montoro, "Image rectification." https://github.com/agarciamontoro/image-rectification. 4

[3] "Install OpenCV on Ubuntu or Debian." https://milq.github.io/install-opencv-ubuntu-debian/. 4

[4] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International journal of computer vision*, vol. 47, no. 1, pp. 7–42, 2002. 5

[5] D. Scharstein, H. Hirschmüller, Y. Kitajima, G. Krathwohl, N. Nešić, X. Wang, and P. Westling, "High-resolution stereo datasets with subpixel-accurate ground truth," in *German conference on pattern recognition*, pp. 31–42, Springer, 2014. 5

[6] Z. Li and N. Snavely, "MegaDepth: Learning Single-View Depth Prediction from Internet Photos," in *Proceedings of IEEE Intl. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018. 8, 9

[7] D. DeTone, T. Malisiewicz, and A. Rabinovich, "SuperPoint: Self-Supervised Interest Point Detection and Description," in *Proceedings of IEEE Intl. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018. 9

[8] P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich, "SuperGlue: Learning Feature Matching with Graph Neural Networks," in *Proceedings of IEEE Intl. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020. 9

[9] J. Sun, Z. Shen, Y. Wang, H. Bao, and X. Zhou, "LoFTR: Detector-free local feature matching with transformers," in *Proceedings of IEEE Intl. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021. 9

[10] Q. Wang, X. Zhou, B. Hariharan, and N. Snavely, "Learning feature descriptors using camera pose supervision," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020. 9