Exercise 5

Deadline: 30. January, 2022

Please send your results as a zip file either to **tewodros_amberbir.habtegebrial@dfki.de** or **torben.fetzer@dfki.de**.
Use the **Latex template** provided in Exercise_0 for the report.
Hand-written results will not be accepted!

Preliminaries

In this exercise we will cover two instances of multi-view stereo:

1. Depth estimation from multiple views.
2. View synthesis from nearby views.

Depth estimation from multiple views - or simply multi-view 3D reconstruction - uses a set of images to recover the scene geometry. The expected output is a depth map representation of the three-dimensional structure of the scene.
View synthesis, on the other hand, uses a set of images to generate a visually consistent image of the scene from a novel viewing angle.

Practical Part

To perform the tasks above, you are given 5 images with known intrinsics and extrinsics. The cameras are named by C_i for $i = 0, 1, 2, 3, 4$. Camera 2 will represent the world (or reference) coordinate system. Rotation matrices and translation vectors for the other cameras follow the convention adopted in the lecture i.e. $\mathbf{R}_i$ converts coordinates of a point from the world/reference to the coordinate frame of camera C_i. Translation vector $\mathbf{t}_i$ corresponds to the location of the origin of the world coordinate system, expressed in the coordinate system of camera C_i. In the `main.py` script there are functions *load_camera_pose* and *load_imgs_and_k_mats* to load rotation matrices, translation vectors, intrisic calibration matrices and images. Moreover, there are two datasets *KITTI* and *CARLA*. You can choose between these two datasets inside the `main.py` file.

Testing Depth Proposals The principle for solving both tasks is nearly the same: Given a set of depth proposals d_i in a range $[d_{\min}, d_{\max}]$, for each pixel select the d_i that is most likely the correct depth for that pixel. The selection is done based on confidence assigned to each depth using photometric consistency. In other words, the method keeps a list of depth proposals and chooses the most likely depth.

1. Depth Estimation

The goal is to predict depth for each pixel in the reference camera. For each pixel p_{ref} in the reference camera image (C_2) and for each depth proposal d_i, perform the following steps:

(a) Compute the corresponding 3D point.

(b) Project the 3D point onto all other camera images ($C_0, C_1, C3, C_4$). Let's call these projected image locations $\{p_0, p_1, p_3, p_4\}$.

(c) Use SSD to determine a similarity score (also referred as photometric consistency) between the source pixel p_{ref} and the pixel values at $\{p_0, p_1, p_3, p_4\}$. Remember, pixel-wise comparison is not accurate and instead apply SSD matching on a patch centered around p_{ref} against the patches centered around $\{p_0, p_1, p_3, p_4\}$.

(d) Store the similarity computed above in a list (this list will have the same size as the number of depth proposals).

(e) Iterate over the list of depth proposals and select the depth with highest similarity score.

2. View Synthesis

Now, the goal is to generate a novel image from neighboring views. In this case, assume the image from camera 2 is no longer available. The task is then to synthesize image 2 using the known pose of this camera. Here, camera 2 is referred to as target view. The procedure is analogous to the one presented above for depth estimation. However, since the color values of the target view are unknown, step (c) must be adapted accordingly: the similarity score is now calculated using patches centered around p_0, p_1, p_3 and p_4. Note that you still use calibration and pose of camera 2 to generate the 3D points projected onto the other views (steps 1 and 2 above). For each pixel location in camera 2 do the following steps:

(a) Compute the corresponding 3D point.

(b) Project the 3D point onto all other camera images ($C_0, C_1, C3, C_4$). Let's call these projected image locations $\{p_0, p_1, p_3, p_4\}$.

(c) Use SSD to determine a similarity score (also referred as photometric consistency) between the patches centered around $\{p_0, p_1, p_3, p_4\}$.

(d) Store the similarity computed above in a list (this list will have the same size as the number of depth proposals).

(e) Iterate over the list of depth proposals and select the depth with highest similarity score.

After the best depth is chosen for the pixel in camera 2, we will have 4 candidate RGB color values (c_0, c_1, c_3 and c_4) to assign to it; one RGB value coming from each of the input views $\{0, 1, 3, 4\}$. Then, we can either take the median or the mean of candidate colors. You should try both mean and median aggregation methods.

Results to be reported Please include the following results in your report:

- Depth map of the reference view from **Depth Estimation**.
- Synthesized novel-view from **View Synthesis**. Report results using both median and mean aggregation of candidate color values. Does the median aggregation give better results ? Why ?

Recommendations for computational efficiency Suppose you are interested in computing the confidence associated with a depth proposal. As mentioned in steps (a)-(c) in the **Depth Estimation**, a given depth proposal d_i allows you to project pixels from the source view to the target view. Then comparing colors at the source and destination pixels we can compute "confidence" values for d_i. You can do these steps, for each pixel in the source camera one-by-one using a for loop. This method is going to be slow. Therefore, step (a) to (c) could be greatly simplified using homographies. For each depth proposal d_i we take the four corners of the reference/source camera C_s (let's call them $c_{s1}, c_{s2}, c_{s3}, c_{s4}$) and project them to one of the input views ($C_i, i \in \{0, 1, 3, 4\}$). The projection will give us four locations in C_i (let's call them $c_{i1}, c_{i2}, c_{i3}, c_{i4}$). Using the four pairs of correpondences, we can compute the homography matrix $\mathbf{H}_s^i$ that maps points in the source view C_s to the input view C_i. Then, by applying the homography matrix on the input view, we can warp it to the source view. Then, we can simply compare the warped image and reference image C_s using SSD. You can use opencv's *cv.findHomography* to calculate the homography matrix and *cv.warpPerspective* to perform warping using the homography matrix.

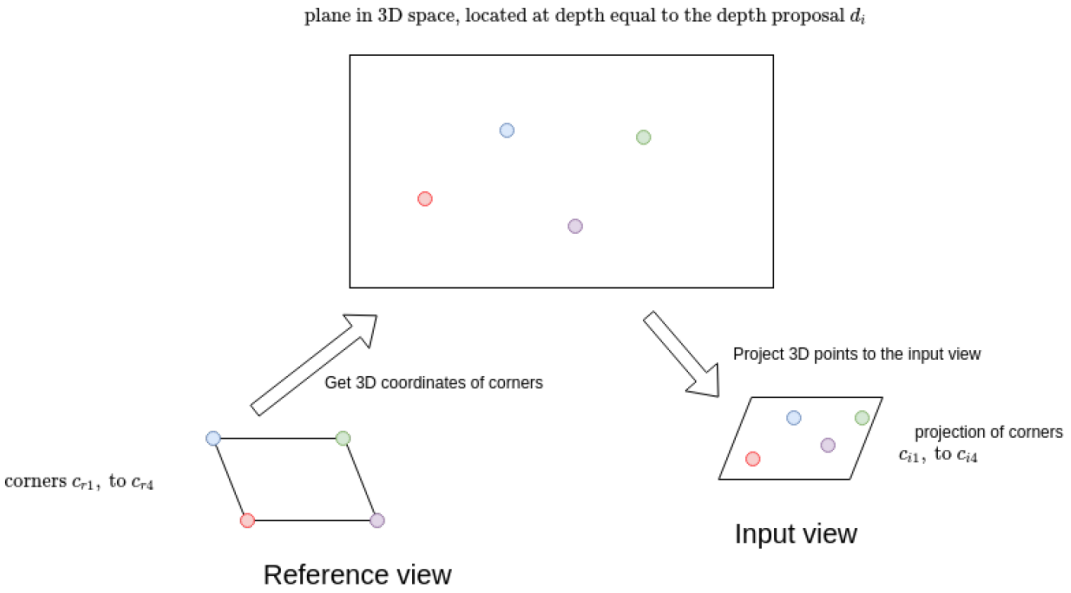

Figure 1: Projecting corners of the source camera to the one of the input views.

Remark:
Make sure your code executes the tasks above sequentially by simply calling `python3 main.py` (include the data directory alongside your scripts in your `.zip` file). Please make sure to run the command above with python3.

Good Luck!