2.1 Stereo Estimation

In triangulate function, I used 2 for loops to traverse disparity image dispI. I got each disparity value as dispI[i, j]. If this disparity equals to 0, I just said continue without doing anything with this disparity, because it will result in a divide by 0 error if we count that disparity too. After getting the disparity, I used the depth formula depth= B*f/disparity. B and f values are already given, with the disparity I got, I calculated the depth values corresponding to the disparities. Then I calculated the point coordinates as follows:

- x=(i-cx)*depth/f
- y=(j-cy)*depth/f
- z= depth

after then, I used append to add the 3D points to points, and colors as follows:

- points.append([x,y,z])
- colors.append(all_colors[i,j])

I make points and colors numpy arrays and write the outputs to the .ply files.

To the bonus part of this question, I defined a function called outlier_ratio(dispI_est, dispI_gt, threshold). I chose threshold as 3 because in lecture slides it is stated that threshold is 3 for KITTI. In the function, I calculated the error with getting the absolute difference between estimated disparity image and ground truth disparity image. Then if the error is larger than the threshold, I used them to calculate the outlier ratio. And we got 65% outlier ratio.

2.2 pykitti

In part 1, basically I used the given functions to find the things that we need to find. In the stereo processing part, I used stereo.compute.

2.3 Monocular VO

For this part, I inspired from the link provided: bitbucket.org/castacks/visual_odometry_tutorial/src/master/visual-odometry

I used the KITTI parameters. I looped through the each frame, read the current image.

Extract keypoints and descriptors using ORB. Match keypoints between the current and previous frames using a brute-force matcher bf. Estimate essential matrix and recover the relative pose between frames using RANSAC and triangulation. Convert rotation and translation to quaternion representation. Update current camera pose and store it in the output file. Update visualization of the camera trajectory on trajMap. Display keypoint matches and camera trajectory. Save the camera trajectory visualization as an image.

2.4 Bullet Time

I used mapping function from cv2 after running inference.py, to conduct the task on np_flow_12. Code creates a mesh grid representing the pixel coordinates of the first image. In the loop, for each intermediate frame, it computes the warped image by incrementally displacing the pixel coordinates based on the divided flow. It then uses OpenCV's remap() function to warp the first image towards the second image using the computed flow. The resulting synthesized frames are stored in a list and displayed using plt.imshow(). Finally, the synthesized frames are visualized in a grid layout using Matplotlib