**Problem 1 - Ben**

The Part1 task works on overlaying the positive and negative grasping rectangles for each image on the raw image using the coordinates of the rectangles provided from the “cpos.txt” file and the “cneg.txt” file. The positive rectangles are colored in green while the negative rectangles are colored in red to differentiate between positive and negative grasping rectangles for each image. The image with “overlay.txt” files are the raw image with the grasping rectangles. The reason we did not directly write it into the raw image is to make sure that it is convenient for others to perform their specific tasks without the interference of the grasping rectangles. The table shown in Image 1 illustrates the grasping rectangles for the positive and negative grasping rectangles. The filename for image 1 is “pcd0100overlay.png”. Most folders contain 100 images to overlay rectangles while the 09 folder and 10 folder contain less than 100 images to overlay. Therefore, the loops were slightly modified when going through 09 folder and 10 folder. The grasping rectangles on each image are also located in a folder named “grasping rectangles”.

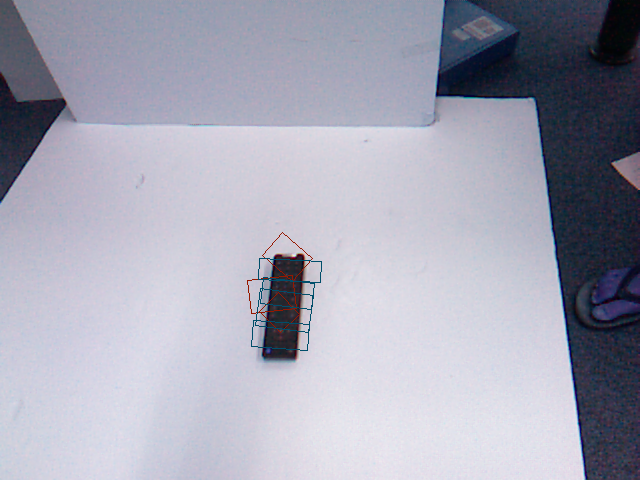


Figure 1: Positive and negative grasping rectangles overlaid on image

**Problem 2 - Julia**

The Part2 program converts the PCD formatted data to the XYZ cartesian point cloud. This program first creates an Image object with the RGB image and then a Depth map subclass object by inheriting the image class. The PCD file was incorporated into the Depth map object, assigning depth data to each pixel location to create the XYZ point cloud. Finally, an inpaint algorithm enhanced the depth map, filling the gaps between objects with neighboring values. One example of the generated depthmaps is shown as Fig. 2. The Part2 program is based on the program provided by this link.

https://github.com/skumra/robotic-grasping/blob/master/utils/dataset\_processing/image.py



Figure 2. One example of depth map generated by programe Part2.

Another contribution I made for this project is implementing the file iteration code for the group.

**Problem 3 - Ryan**

The program of problem 3 first converts the RGB png files into YUV color space and then crops both the obtained YUV image and the depth images from problem 2 into rectangular grasping sub patches. The cropping code reads in the coordinates of the grasping rectangles from the txt files and crops out the area of interest through masking and slicing.

The main challenge of this problem lies within the fact that many of the grasping rectangles are tilted, which causes the cropped out patches to have large black bezels around them (see upper Fig. 3). In fear that these bezels would introduce unnecessary bias to the data, we used perspective transform from the previous assignment to remove them. To be specific, we use the boundingRect() function to find the anchor points as well the dimensions of the bounding rectangle to map the anchor points to the following set of coordinates [(0, height), (width, height), (width, 0), (0, 0)] to ensure the transformation preserves the original aspect ratio. An example of the patches is shown in the lower part of Fig. 3.

Figure 3: The original cropped patch with bezels (above) compared with the patch went through perspective transform (below)

**Problem 4 - Felipe**

The images created in Problem 3 were grayscale PNGs representing the depth of features. As per the documentation on PCA whitening, each dimension in the dataset represents a pixel on the image. Therefore, it was necessary to take the depth images and resize them so that comparisons could be made based on a similar pixel in the images. Each image was read in and resized to 35 by 35 using cv2.resize().

After the images are resized, the mean value is subtracted, so as to center the depth values around zero. The images are then cast to a vector with 1225 dimensions (one for each pixel in the provided 35x35 frame). A single vector containing all depth images in the data set is then constructed. Given m images, the resulting matrix (x) has shape (1225, m). The covariance of this matrix is then calculated using NumPy, and the singular value decomposition is found. Following the equation for x\_pca on the provided resource, the final whitened image can be obtained as a row in this matrix, and then resized for visualization:



Figure 4: Original image (below) compared with whitened image (above)

From there, it is possible to loop over all the created images and save a PNG next to the original depth image.

**Problem 5 - Felipe**

After doing research on Rviz, it was determined that converting to a .ply file was not how visualization of a point usually works. Instead, first step was to take the .pcd file and publish it as a ROS topic:

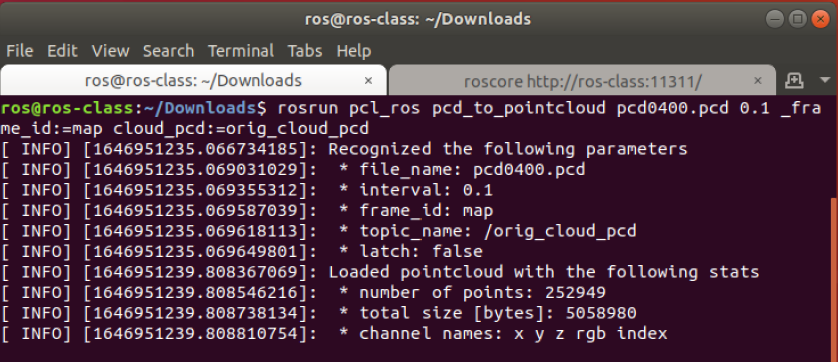


Figure 5: Starting to publish to a rostopic

Running a rostopic list indicates that the topic /orig\_cloud\_pcd is being published, and it can be read by Rviz:

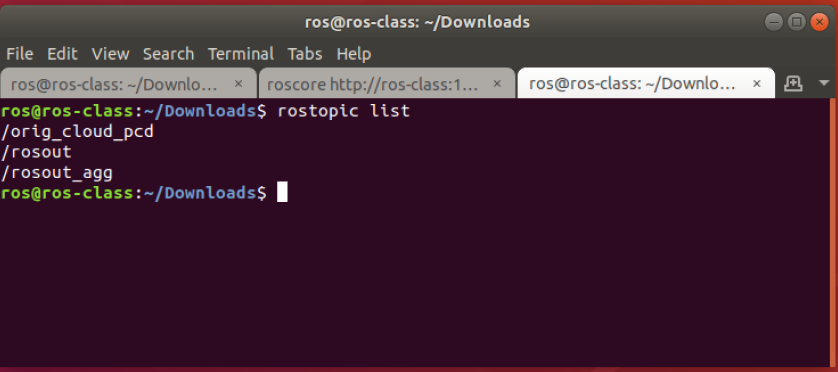


Figure 6: Verification that rostopic is created

Once the topic was published, it was easy to select the relevant topic (/orig\_cloud\_pcd, highlighted below) in the Rviz GUI, and visualize the point cloud:

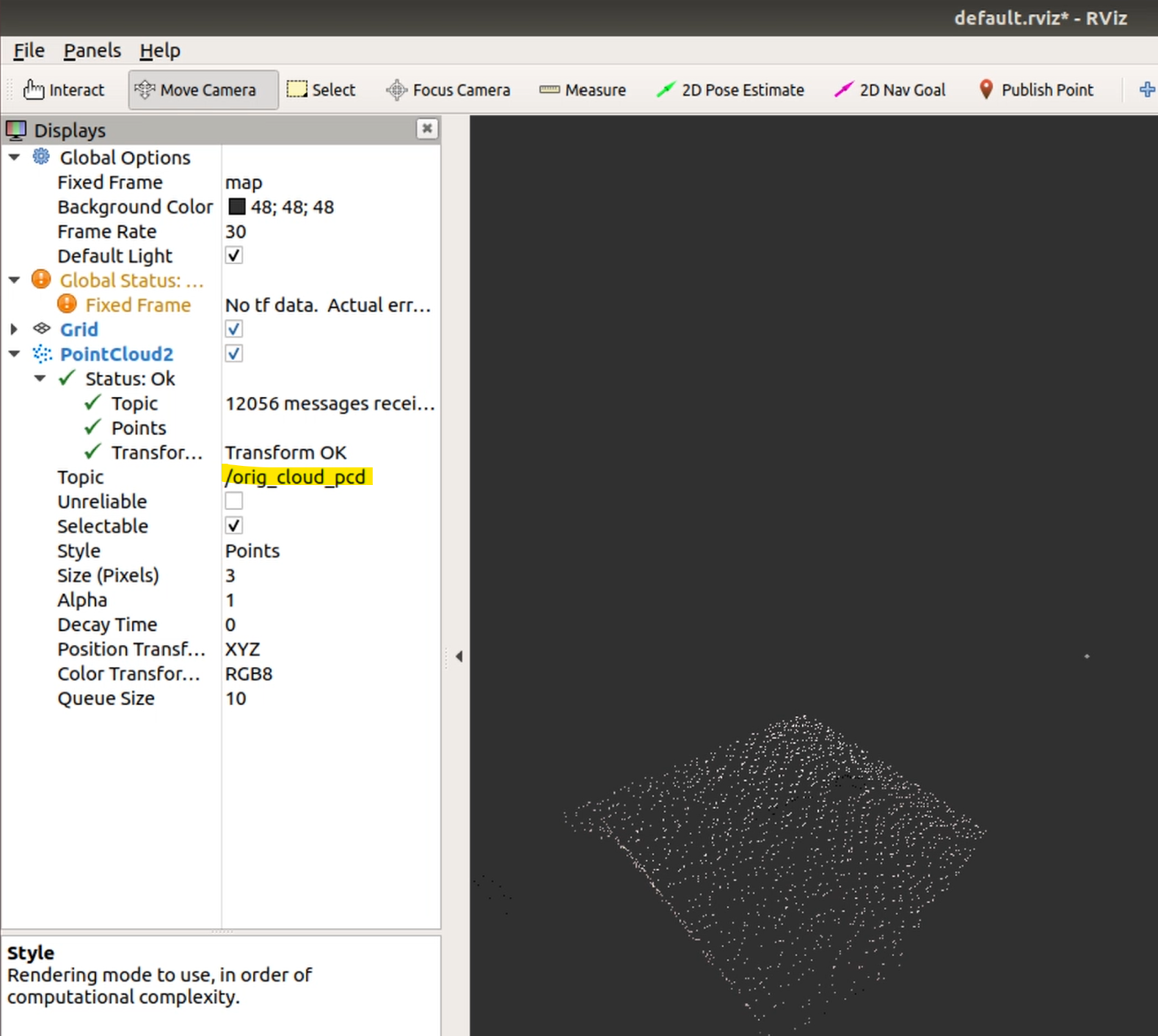


Figure 7: Visualizing point cloud in Rviz