Github Repo link (please use the “main” branch): <https://github.com/rho-selynn/592-HW2>

1. With the provided Grasping Dataset, unzipped and loaded the “01” folder into the working directory of the Google Colab notebook. The goal was to edit the original pictures to overlay grasp rectangles. There was some confusion as to how these rectangle coordinates were defined in the text files, so the team decided to use ordered pairs of rectangles, with each ordered pair describing the top left and bottom right coordinates of the rectangles. So for example, in a text file for the positive rectangles, a list of (300, 350) and (200, 250) was given. With those coordinates, a positive rectangle would be created, whose top left corner is at (300,250) and bottom right corner is at (200,250). The team could not find any documentation to describe whether this is the correct interpretation or not, but if it is not the correct interpretation, it would be very easy to slightly edit our code to fix the problem.

A picture containing text, indoor, envelope

Description automatically generated

Figure 1: An output image of the remote overlaid with our generated grasp rectangles

1. This step was done by loading the [linked GitHub repo](https://github.com/skumra/robotic-grasping/blob/master/utils/dataset_) from the assignment into our Colab notebook. The team was not sure what an RGB-D image was, so we looked it up. We found the following image:

A picture containing text, indoor

Description automatically generated

Figure 2: an example of an RGB-D image

Using the [linked GitHub repo](https://github.com/skumra/robotic-grasping/blob/master/utils/dataset_) given, we created code that created two similar pictures, one picture representing the RGB channels of the image, the other picture representing the depth (D) channel. Some odd holes can be seen in the D pictures generated, and we believe this is caused by improper data capture in the point cloud files (Fig. 3). Further support can be found for this in problem 5, discussed later.

A picture containing text, indoor, television, monitor

Description automatically generated

Figure 3: (Left) the original image. (Right) the point cloud generated image

1. We started by converting the RGB-D images to the YUV color format. Then, we saving the YUV and depth images of each picture taken (Fig. 4).

A picture containing text

Description automatically generated

Figure 4: (Left) the original image. (Middle) image in YUV color format. (Right) depth image

After this, we cropped these images to the sizes of the grasp rectangles given in the data to get sub-patches of the image, again following the same convention stated in Problem 1. After cropping the YUV images, we displayed them. However, for the depth images, we had to normalize them to properly display the depth information after cropping them.

1. We imported the PCA function from the Scikit-Learn (sklearn) Python package.
2. For this problem, we used open3d, an open-source python library, to convert the text files to .ply files. After doing this, we saved the .ply files locally to our computers, and again displayed them using open3d. Dr. Sarkar said it was okay to not use ROS for this problem, as we did not have time to set up a virtual Ubuntu machine, which is required for ROS. A few generated point clouds can be seen below. In the second point cloud shown, it can be seen that the data captured is not of great quality, having many holes around the stapler scanned. These holes are the reason for the odd holes in the problem 2 Depth images generated.

A picture containing stationary, businesscard

Description automatically generated

A picture containing stationary

Description automatically generated