Readme for Biomass_estimation.py

Intro

The overall algorithm for this python file can be broken down into 4 basic steps:

1) The initial ground segmentation and rotation of plane.
2) Identification and isolation of data inside the region of interest (white square).
3) Finding the best distance threshold to identify the plants and determining the accuracy of the ground plane.
4) Identify plant vs ground inside region of interest. Then count the total plant points and estimate the overall volume as an index.

Biomass_estimation.py is meant to be a living file. There are many improvements that can be made given more time and testing which I will describe in their relevant sections.

It also can be split into several files containing important functions. The reason for keeping it in the same file was the variation in approach and testing, and the need to visualize the results each step of the way. In the long term however, different files should be made for each of the calculations.

Step 1 Initial Plane

First the point cloud is loaded from a local directory. The Ransac plane segmentation is run on the 3D image, using the open3D library. The code stores the xyz coefficients and prints the plane equation to the console.

```
pcd = o3d.io.read_point_cloud('C:\\Users\\Robert
Mayo\\PycharmProjects\\OpenCV2\\bag_files\\19700101_122305.bag_ply2.ply
')

plane_model, inliers = pcd.segment_plane(distance_threshold=0.02,
                                         ransac_n=3,
                                         num_iterations=250)
[a, b, c, d] = plane_model
print(f"Plane model: {a:.2f}x + {b:.2f}y + {c:.2f}z + {d:.2f} = 0")
```

Everything within the segmented plane is saved into the ‘inlier_cloud’ pcd while everything outside is saved in the ‘outlier_cloud’.

```
inlier_cloud = pcd.select_down_sample(inliers)
inlier_cloud.paint_uniform_color([1.0, 0, 0])
outlier_cloud = pcd.select_down_sample(inliers, invert=True)
outlier_cloud.paint_uniform_color([0, 0.4, 0])
```

The plane is then rotated to be parallel to the ground based on the plane equation found from the segmentation with the coefficients stored in variables a, b, c, d. The easiest way to do this is to rotate in the y direction by the negative of the x-coefficient (counterclockwise while facing +y direction by the

right hand rule) and in the x direction by the positive of the y-coefficient (clockwise while facing the +x direction).

```
R = pcd.get_rotation_matrix_from_xyz((b, -a, 0))
pcd.rotate(R, center=True)
```

What this does is change the x and y coefficients to be close 0 (there is some error in the ransac calculation due to random sampling, which will account for small differences).

Step 2 Isolate Region of Interest

First the threshold for finding the white values are defined, and the used to identify the points in the pcd color matrix. Next the points from the color matrix are used to find the corresponding points in the xyz position matrix. This is done through scalar multiplication between the two arrays, with the 0's eliminating all points not meeting the criteria.

Next all points with unreasonable z values are eliminated. Then the array is copied and each copy is sorted, one by x values and the other by y values. Points with unreasonable x and y values are eliminated in the next step.

```
elim_above_z = np.where(white_pos[:,2] > (np.median(xyz_in[:,2],axis=0) +
np.std(xyz_in[:,2],        axis=0)),0,1)
elim_z = elim_above_z * elim_below_z
elimz_2d = np.atleast_2d(elim_z)
elimz_t = elimz_2d.transpose()
white_square = elimz_t * white_pos

xsort = np.sort(white_square[:,0], axis=0)
ysort = np.sort(white_square[:,1], axis=0)
xsort_no_zero = xsort[xsort != 0]
ysort_no_zero = ysort[ysort != 0]
xlen = len(xsort_no_zero)
ylen = len(ysort_no_zero)

# Find the Upper and lower 5th of X/Y points as the sides of the square
# Assumption: top 1/4 of x points will correspond to the right side, etc.
Top 1/5 eliminates outlier pts.
xupper = xsort_no_zero[int(xlen/5)*4:(xlen-1)]
xlower = xsort_no_zero[0:int(xlen/5)]
yupper = ysort_no_zero[int(ylen/5)*4:(ylen-1)]
ylower = ysort_no_zero[0:int(ylen/5)]
```

The idea behind this approach is in a square outline with a relatively equal distribution of points, the top and bottom 1/4 of x and y values will contain the 4 sides. I used the top and bottom 1/5 of the points in case the distribution was slightly uneven.

```
x_max = np.where(xyz[:, 0] > (np.median(xupper) + np.std(xupper)), 0, 1)
x_min = np.where(xyz[:, 0] < (np.median(xlower) - np.std(xlower)), 0, 1)
y_max = np.where(xyz[:, 1] > (np.median(yupper) + np.std(yupper)), 0, 1)
y_min = np.where(xyz[:, 1] < (np.median(ylower) - np.std(ylower)), 0, 1)
```

Finally, the points outside of the established x/y boundaries are eliminated, again through scalar matrix multiplication and the resulting zeros are eliminated. The final result is a point cloud only containing the region of interest.

```
elim_max = xmax_t * ymax_t
elim_min = xmin_t * ymin_t
elim_outside = elim_max * elim_min
inside_square = elim_outside * xyz
just_inside_square = inside_square[inside_square[:, 2] != 0]
```

This section could be improved by adding a way to estimate the square if it is covered by multiple leaves. However this could easily turn into a large and complex algorithm. Another solution is to identify the corners of the square when the video is taken and use the script to draw the corners.

Step 3 Choosing Distance Threshold

Plane segmentation is run again for each distance threshold within the range 0.01 and 0.07 (range of minimal plane thickness to >99% of the pcd included in the plane). This is done through a while loop.

```
dist = .01
while(dist < 0.07):
 plane_model, inliers =
 pcd_white_pos.segment_plane(distance_threshold=dist, ransac_n=3,
 num_iterations = 250)

 dist + 0.01
```

Relevant statistical measures are saved in corresponding arrays for each iteration of the loop and are later index by the selected distance thresholds. After the loop is finished, the collected data is used to select the distance threshold, shown below. The best way to do this currently is select the threshold that gives the lowest possible median value for the ground plane. The reasoning is that

```
dist_thresh = in_med.index(min(in_med))
```

Also determined after the loop is whether or not the segmented plane accurately represents the ground. Exceptions to this would be when the vegetation is too thick for the camera to sense the ground. The current approach to this is determining the ground plane is inaccurate if the range of outliers below the median of the ground (very small range, if the ground is visible) is larger than the range of the ground, or the range of outliers below the median is larger than the range of outliers above the median (plants).

```
if (range_below_med[dist_thresh] < inlier_range[dist_thresh]) &
(range_below_med[dist_thresh] < range_above_med[dist_thresh]):
      ground_ref = 1
  else:
      ground_ref = 0
```

Finally, if the ground plane is correct, only points above the ground are counted as plants. If the ground plane is incorrect it is assumed that since the vegetation is so dense, all of the points visible are plants.

One improvement that can be made to this section is to add more statistical measures to the selection of the distance threshold. There could be certain scenarios where more measurements are needed to make an accurate choice.

Step 4 Estimating Biomass

The ground and plant matrices are assigned to new variables and used to determine the length (in Open3d units) of each side of the square region of interest.

```
x_side = np.max(ground_array[:,0]) - np.min(ground_array[:,0])
y_side = np.max(ground_array[:,1]) - np.min(ground_array[:,1])
```

The measures of each side are then compared to the known length in order to convert the default Open3D units to cm.

```
conversion = 50/((x_side+y_side)/2)
```

The conversion is used with the average plant height and the ground coverage to estimate the total plant volume.

```
area = (x_side*conversion) * (y_side*conversion)
avg_z = np.mean(plant_array[:,2])
avg_plant_height = avg_z - median_ground
height_cm = avg_plant_height * conversion

total_coverage = num_plant_pts/(num_ground_pts+num_plant_pts)
volume_estimation = avg_plant_height * area * total_coverage
```

Another estimation that could be added to this is the convex hull volume estimation. This is tricky because the pcd.normals dataset must be recovered after isolating the region of interest, something I could not figure out how to do in the time given.