InnoBioDev_RandomWatering\src\TrayImageProcessor.py

```python
# This script is used to generate random watering points for the farmbot tray.

# Import the required libraries
import os # for file operations
import cv2 # for image processing
# from matplotlib import pyplot as plt # for image display if needed
import numpy as np # for numerical operations
from plantcv import plantcv as pcv # for plantcv operations
import time # for calculating the execution time
import random # for generating random numbers
import string # for generating random strings


class TrayImageProcessor:
    # Class variables
    IMAGE_DIR = './InnoBioDev_Randomwatering/data/farmbot_tray/'
    IMAGE_EXT = ('.jpg', '.jpeg', '.png')
    IMAGE_WIDTH = 1280 # in pixels
    IMAGE_HEIGHT = 960 # in pixels
    TRAY_CENTER_X = 0.0 # in mm, this will be updated later
    TRAY_CENTER_Y = 0.0 # in mm, this will be updated later
    TRAY_CENTER_Z = 0.0 # in mm, this will be updated later
    DIA_OF_POT = 60 # in mm
    DIA_OF_ROI = 50 # in mm DIAMETER OF REGION OF INTEREST (ROI), should be less than DIA_OF_POT
    LENGTH_TRAY_A = 150 # in mm
    LENGTH_TRAY_B = 135 # in mm
    ROTATION_ANGLE = -2 # in degrees
    RATIO_MM2PIX = 3.2 # in pixels per mm
    OFF_SET_CAM_X = 0 # in mm
    OFF_SET_CAM_Y = 5 # in mm

    def _rotate_img(self, img, angle):
        angle = - angle
        height, width = img.shape[:2] # image shape has 3 dimensions
        # Calculate the rotation matrix
        rotation_matrix = cv2.getRotationMatrix2D((width / 2, height / 2), angle, 1)
        # Apply the rotation to the image
        rotated_image = cv2.warpAffine(img, rotation_matrix, (width, height))
        return rotated_image

    def ref2img(self, x_ref, y_ref):
        x_img = int(x_ref + self.IMAGE_WIDTH/2 + self.OFF_SET_CAM_X*self.RATIO_MM2PIX)
        y_img = int(self.IMAGE_HEIGHT/2 - y_ref + self.OFF_SET_CAM_Y*self.RATIO_MM2PIX)
        return x_img, y_img

    def img2ref(self, x_img, y_img):
        x_ref = int(x_img - self.IMAGE_WIDTH/2 - self.OFF_SET_CAM_X*self.RATIO_MM2PIX)
        y_ref = int(self.IMAGE_HEIGHT/2 - y_img + self.OFF_SET_CAM_Y*self.RATIO_MM2PIX)
        return x_ref, y_ref

    def get_image(self):
        # Get the list of files in the directory
        file_list = os.listdir(self.IMAGE_DIR)

        # Filter out non-image files
        image_files = [file for file in file_list if file.endswith(self.IMAGE_EXT)]

        # Sort the image files alphabetically
        image_files.sort()

        # Check if there are image files
        if not image_files:
            raise Exception("No image files found in the directory.")

        # Read the first image file
        first_image_path = os.path.join(self.IMAGE_DIR, image_files[0])
        first_image = cv2.imread(first_image_path)
        # Do further processing with the first image

        # Update the tray center coordinates
        filenameparts = image_files[0].split('_')
        self.TRAY_CENTER_X = float(filenameparts[0])
        self.TRAY_CENTER_Y = float(filenameparts[1])
        self.TRAY_CENTER_Z = float(filenameparts[2])
        # check the tray center coordinates, x,y should be positive, z should be 0.0
        if self.TRAY_CENTER_X > 0 and self.TRAY_CENTER_Y > 0 and self.TRAY_CENTER_Z == 0.0:
            print("Tray center coordinates are valid:", "x:", self.TRAY_CENTER_X, "y:", self.TRAY_CENTER_Y)
        else:
            raise Exception("Tray center coordinates are not valid.")
        return first_image, image_files[0]

    def drop_image(self, imagefile):
        # Delete the image file
        imagepath = os.path.join(self.IMAGE_DIR, imagefile)
        os.remove(imagepath)
        print(f"{imagefile} has been deleted.")

    def _center_CAM(self):
        # Calculate the middle point of the image
        middle_x = int(self.IMAGE_WIDTH / 2 + self.OFF_SET_CAM_X*self.RATIO_MM2PIX)
        middle_y = int(self.IMAGE_HEIGHT / 2 + self.OFF_SET_CAM_Y*self.RATIO_MM2PIX)
        return middle_x, middle_y

    def locate_pots(self):
        # Calculate the center of the 6 pots
        center_of_pots = []
            # caleculate the center of the 1. pot
        center_x_1_ref = 0 - self.LENGTH_TRAY_B * self.RATIO_MM2PIX
        center_y_1_ref = (self.LENGTH_TRAY_A * self.RATIO_MM2PIX)/2
        center_x_1_img, center_y_1_img = self.ref2img(center_x_1_ref, center_y_1_ref)
        center_of_pots.append((center_x_1_img, center_y_1_img))
            # caleculate the center of the 2. pot
        center_x_2_ref = 0 - self.LENGTH_TRAY_B * self.RATIO_MM2PIX
        center_y_2_ref = -(self.LENGTH_TRAY_A * self.RATIO_MM2PIX)/2
        center_x_2_img, center_y_2_img = self.ref2img(center_x_2_ref, center_y_2_ref)
        center_of_pots.append((center_x_2_img, center_y_2_img))
            # caleculate the center of the 3. pot
        center_x_3_ref = 0
        center_y_3_ref = (self.LENGTH_TRAY_A * self.RATIO_MM2PIX)/2
        center_x_3_img, center_y_3_img = self.ref2img(center_x_3_ref, center_y_3_ref)
        center_of_pots.append((center_x_3_img, center_y_3_img))
            # caleculate the center of the 4. pot
        center_x_4_ref = 0
        center_y_4_ref = -(self.LENGTH_TRAY_A * self.RATIO_MM2PIX)/2
        center_x_4_img, center_y_4_img = self.ref2img(center_x_4_ref, center_y_4_ref)
        center_of_pots.append((center_x_4_img, center_y_4_img))
            # caleculate the center of the 5. pot
        center_x_5_ref = 0 + self.LENGTH_TRAY_B * self.RATIO_MM2PIX
        center_y_5_ref = (self.LENGTH_TRAY_A * self.RATIO_MM2PIX)/2
        center_x_5_img, center_y_5_img = self.ref2img(center_x_5_ref, center_y_5_ref)
        center_of_pots.append((center_x_5_img, center_y_5_img))
            # caleculate the center of the 6. pot
        center_x_6_ref = 0 + self.LENGTH_TRAY_B * self.RATIO_MM2PIX
        center_y_6_ref = - (self.LENGTH_TRAY_A * self.RATIO_MM2PIX)/2
        center_x_6_img, center_y_6_img = self.ref2img(center_x_6_ref, center_y_6_ref)
        center_of_pots.append((center_x_6_img, center_y_6_img))
        return center_of_pots

    def show_control_image(self, roh_image, center_of_pots, save_image=False, show_image=True):

        # Copy the first image to a control image
        control_image = self._rotate_img(roh_image, self.ROTATION_ANGLE)
# Draw the horizontal and vertical refference lines on the image
    # Calculate the start and end point of the horizental line based on the angle
        start_x_ref = 0
        start_y_ref = int(self.IMAGE_HEIGHT / 2)
        end_x_ref = self.IMAGE_WIDTH
        end_y_ref = int(self.IMAGE_HEIGHT / 2)
        # Draw the line on the image
        cv2.line(control_image, (start_x_ref, start_y_ref),(end_x_ref, end_y_ref), (0, 255, 0), 2) # green line
        cv2.line(control_image, (start_x_ref, start_y_ref + int(self.OFF_SET_CAM_Y*self.RATIO_MM2PIX)),
                 (end_x_ref, end_y_ref + int(self.OFF_SET_CAM_Y*self.RATIO_MM2PIX)), (0, 255, 255), 2) # yellow line

        # Calculate the start and end point of the vertical line based on the angle
        start_x_ref = int(self.IMAGE_WIDTH / 2)
        start_y_ref = 0
        end_x_ref = int(self.IMAGE_WIDTH / 2)
        end_y_ref = self.IMAGE_HEIGHT
        # Draw the line on the image
        cv2.line(control_image, (start_x_ref, start_y_ref),(end_x_ref, end_y_ref), (0, 255, 0), 2) # green line
        cv2.line(control_image, (start_x_ref + int(self.OFF_SET_CAM_X*self.RATIO_MM2PIX) , start_y_ref),
                 (end_x_ref + int(self.OFF_SET_CAM_X*self.RATIO_MM2PIX), end_y_ref), (0, 255, 255), 2) # yellow line
        # Draw a circle around the center of the pots
        for i,(x,y) in enumerate(center_of_pots):
            cv2.circle(img=control_image, center=(x, y), radius=int(self.DIA_OF_POT*self.RATIO_MM2PIX), color=(0, 0, 255), thickness=2)
            cv2.circle(img=control_image, center=(x, y), radius=int(self.DIA_OF_ROI*self.RATIO_MM2PIX), color=(255, 0, 0), thickness=2)
            cv2.putText(control_image, str(i+1), (x, y), cv2.FONT_HERSHEY_SIMPLEX, 3, (0, 0, 255), 3, cv2.LINE_AA)
        # Display the control image
        if show_image:
            display_image = cv2.resize(control_image, (int(self.IMAGE_WIDTH/2), int(self.IMAGE_HEIGHT/2))) # Resize the image for better display
            s = 'Press "q" to save and close'
            cv2.putText(img=display_image, text=s, org=[2,22], fontFace=cv2.FONT_HERSHEY_SIMPLEX, fontScale=0.8, color=(0, 0, 0), thickness=2, lineType=cv2.LINE_AA) # create a shadow
            cv2.putText(img=display_image, text=s, org=[0,20], fontFace=cv2.FONT_HERSHEY_SIMPLEX, fontScale=0.8, color=(0, 255, 255), thickness=2, lineType=
cv2.LINE_AA)
            cv2.imshow('Control Image', display_image)
            key = cv2.waitKey(0)
            if key == ord('q'):
                cv2.destroyAllWindows()
        if save_image:
            # Create a directory for saving images if it doesn't exist
            save_dir = 'saved_img'
            if not os.path.exists(save_dir):
                os.makedirs(save_dir)

            # Save the control image as a .jpg file
            save_path = os.path.join(save_dir, 'control_image.jpg')
            cv2.imwrite(save_path, control_image)

    def split_roi(self, center_x, center_y, image):
        # Calculate the coordinates of the top-left and bottom-right corners of the ROI
        roi_x1 = int(center_x - self.DIA_OF_POT * self.RATIO_MM2PIX)
        roi_y1 = int(center_y - self.DIA_OF_POT * self.RATIO_MM2PIX)
        roi_x2 = int(center_x + self.DIA_OF_POT * self.RATIO_MM2PIX)
        roi_y2 = int(center_y + self.DIA_OF_POT * self.RATIO_MM2PIX)

        # Crop the ROI from the image
        roi = image[roi_y1:roi_y2, roi_x1:roi_x2]

        # Set pixels outside of the ROI circle to black
        mask = np.zeros_like(roi)
        radius = int(self.DIA_OF_ROI * self.RATIO_MM2PIX)
        center = (roi.shape[1] // 2, roi.shape[0] // 2)  # Set center as the middle of the ROI
        cv2.circle(mask, center, radius, (255, 255, 255), -1) # fill the circle with white color, -1 means fill the circle
        roi = cv2.bitwise_and(roi, mask)

        # Return the ROI
        return roi

    def split_multi_roi(self, center_of_pots, image):
        # Calculate the area of the ROI for each pot
        roi_areas = []
        for center_x, center_y in center_of_pots:
            print(center_x, center_y)
            roi_area = self.split_roi(center_x, center_y, image)
            roi_areas.append(roi_area)
        return roi_areas

    def random_watering_points(self, img, num_watering_points=20, save_image=False, show_image=False, filename="no_name"):
        # Add .jpg extension to the filename if it doesn't have one
        if not filename.endswith(".jpg"):
            filename = filename + ".jpg"
        else:
            filename
        # Define the pot center and radius
        pot_x = int(img.shape[1] / 2)
        pot_y = int(img.shape[0] / 2)
        roi_radius = int(self.DIA_OF_ROI * self.RATIO_MM2PIX)
        # Set a timer for the execution time
        start_time = time.time()
        print('############################# START #############################')
        print('processing image:')
```

```python
        # mask in H channel
        img_HSV = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
        img_H = img_HSV[:, :, 0] #all rows, all columns, first channel (Hue)
        img_H_thresh = cv2.inRange(img_H, 20, 40)
        # mask in A channel
        img_LAB = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)
        img_A = img_LAB[:, :, 1] #all rows, all columns, second channel (A)
        img_A_hist_EQU = cv2.equalizeHist(img_A)
        _, img_A_thresh = cv2.threshold(img_A_hist_EQU, 31, 255, cv2.THRESH_BINARY)
        img_A_thresh = cv2.bitwise_not(img_A_thresh)
        # mask in V channel
        img_V = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)[:, :, 2]
        _, img_V_thresh_up = cv2.threshold(img_V, 250, 255, cv2.THRESH_BINARY_INV)
        _, img_V_thresh_down = cv2.threshold(img_V, 50, 255, cv2.THRESH_BINARY)
        img_V_thresh = cv2.bitwise_and(img_V_thresh_up, img_V_thresh_down)
        # combine H A V masks
        img_H_thresh_erode = cv2.erode(img_H_thresh, kernel=np.ones((5, 5), np.uint8), iterations=1)
        img_A_thresh_erode = cv2.erode(img_A_thresh, kernel=np.ones((5, 5), np.uint8), iterations=1)
        img_thresh = cv2.bitwise_or(img_A_thresh_erode, img_H_thresh_erode)
        img_thresh = cv2.bitwise_and(img_thresh, img_V_thresh)
        # closing method to the mask
        mask_dilated = cv2.dilate(img_thresh, kernel=np.ones((5, 5), np.uint8), iterations=2)
        mask_erode = cv2.erode(mask_dilated, kernel=np.ones((5, 5), np.uint8), iterations=3)
        mask_dilated = cv2.dilate(mask_erode, kernel=np.ones((5, 5), np.uint8), iterations=3)
        mask = mask_dilated

        # labeled the regions on the mask image
        _, labeled_mask = cv2.connectedComponents(mask)
        num_mask = np.max(labeled_mask)
        print('{}'.format('\t'),'total', num_mask, 'region(s) found!')

        # just keep the first 10 biggst region on the mask
        count = 0
        region_info={}
        for region_id in range(1,num_mask+1,1):
            mask_region_cnt = cv2.inRange(labeled_mask,region_id,region_id)
            count = cv2.countNonZero(mask_region_cnt)
            region_info[region_id]= (region_id, count)
        list_of_region = list(region_info.values())
        sorted_data = sorted(list_of_region, key=lambda x: x[1], reverse=True)
        sorted_data_cop = sorted_data[:10]

        mask_cop = np.zeros(np.shape(mask),dtype=np.uint8)
        for region_id in sorted_data_cop:
            id = (int)(region_id[0])
            mask_cop+=cv2.inRange(labeled_mask,id,id)

        # calculation the center of mass of the region
        # this will locate the plant
        contours, _ = cv2.findContours(mask_cop, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        mask_RGB=cv2.cvtColor(mask_cop,cv2.COLOR_GRAY2BGR)

        # let us define the watering point.
        # create watering point
        # the previous mask will be enlarged, so that there will be a safty zone, that we will not water the leaves
        mask_with_saftyzone = cv2.dilate(mask_cop, np.ones((15,15), np.uint8), iterations=3)
        watering_points_list = []
        count = 0
        while (count<=num_watering_points):
            angel = random.randint(0,360)
            rel_radius = random.random()
            x_watering_point = (int)(np.cos(np.radians(angel))*(roi_radius)*rel_radius+pot_x)
            y_watering_point = (int)(np.sin(np.radians(angel))*(roi_radius)*rel_radius+pot_y)
            if mask_with_saftyzone[y_watering_point, x_watering_point] != 255:
                watering_points_list.append((x_watering_point, y_watering_point))
                count+=1
        end_time = time.time()
        print('{}'.format('\t'),'Execution time:', round(end_time - start_time, 2), ' seconds')
        print('############################## END ##############################')
        if show_image*save_image:
            img_out = img.copy()
            # lets draw everything on image
            for i in watering_points_list:
                cv2.circle(img_out,i,4,(0,255,0), -1)
            cv2.drawContours(img_out, contours, contourIdx=-1, color=(255,0,0), thickness=3)
        if show_image:
            img_display = img_out.copy()
            s = 'Press "q" to save and close'
            cv2.putText(img=img_display, text=s, org=[2,22], fontFace=cv2.FONT_HERSHEY_SIMPLEX, fontScale=0.8, color=(0, 0, 0), thickness=2, lineType=cv2.LINE_AA) # create a shadow
            cv2.putText(img=img_display, text=s, org=[0,20], fontFace=cv2.FONT_HERSHEY_SIMPLEX, fontScale=0.8, color=(0, 255, 255), thickness=2, lineType=cv2.LINE_AA)
            cv2.imshow('Control Image', img_display)
            key = cv2.waitKey(0)
            if key == ord('q'):
                cv2.destroyAllWindows()

        if save_image:
            # Create a directory for saving images if it doesn't exist
            save_dir = 'saved_img'
            if not os.path.exists(save_dir):
                os.makedirs(save_dir)
            # Save the mask image as a .jpg file
            save_path = os.path.join(save_dir, filename)
            cv2.imwrite(save_path, img_out)
            print(f"image watering points saved to {filename}")
        return watering_points_list

# to do: add a function to save the watering points to a file
    def save_points_to_csv(self, watering_points_list, filename):
        # Add .txt extension to the filename
        filename = filename + ".csv"
        # Open the file in write mode
        with open(filename, 'w') as file:
            # Write the header
            file.write("X,Y\n")
            # Write each watering point as a new line in the file
            for point in watering_points_list:
                file.write(f"{point[0]},{point[1]}\n")
        print(f"Watering points saved to {filename}")
```

```python
# to do: calculate the watering points back to the real world coordinates
    def roi2real(self, watering_points_list, center_of_pot):
        # Calculate the real world coordinates of the watering points
        # Calculate the middle point of the image with the whole tray
        middle_x, middle_y = self._center_CAM()
        # Calculate the offset of the middle point of the image with the whole tray
        center_x_img = center_of_pot[0]
        center_y_img = center_of_pot[1]
        # Calculate the real world coordinates of the watering points
        real_world_coordinates = []
        for x_roi, y_roi in watering_points_list:
            x_img = x_roi - self.DIA_OF_ROI + center_x_img
            y_img = y_roi - self.DIA_OF_ROI + center_y_img
            x_ref, y_ref = self.img2ref(x_img, y_img)
            real_x = self.TRAY_CENTER_X + x_ref/self.RATIO_MM2PIX
            real_y = self.TRAY_CENTER_Y + y_ref/self.RATIO_MM2PIX
            real_world_coordinates.append((int(real_x), int(real_y)))
        return real_world_coordinates
```