```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Nov 16 16:04:29 2022
CPSC 472 Robot navigation computer vision analysis
To identify and work out ZOMBIE, BERRY type
@author: yujun
"""

from PIL import Image
import matplotlib.pyplot as plt
import os
os.chdir('/Users/yujun/Desktop/CPSC472/Vision')  # set to local folder
import numpy as np

aqua_zombie = Image.open('aqua_legs.png')
blue_zombie = Image.open('blue_legs.png')
green_zombie = Image.open('green_legs.png')
purple_zombie = Image.open('purple_legs.png')
background = Image.open('background_normal.png')


# PNG pixels are left-right, top-bottom order. Need to check
# if Webots uses this ordering too! Horizontal scanline order
# using getpixel method: aqua_zombie.getpixel(coordinate);
# This gives four numbers of the type RGBA
# e.g pixel(40,30) is (33, 210, 183, 255)
# A (alpha) parameter is 0.0 for fully transparent, 1.0 for fully opaque


"""------------ FUNCTION DEFINITIONS ------------"""

# Dictionary of items of interest, obtained from testing
visual_dict = {"Aqua_bright":(37, 221, 194),
               "Aqua_shadow":(10, 69, 67),
               "Blue_bright":(33, 143, 230),
               "Blue_shadow":(10, 40, 99),
               "Green_bright":(35, 192, 39),
               "Green_shadow":(9, 53, 12),
               "Purple_bright":(112, 48, 182),
               "Purple_shadow":(87, 37, 154),
               "Sky":(90, 109, 152),
               "Mountain":(77, 73, 78),
               "Earth":(217, 182, 169)
               }


def make_image_array(Image, x_size, y_size):
    """
    Function to get [px11, px12, px13...] array from PIL data structure
    Likely not necessary in actual Webots environment; this is a
    helper function for Spyder IDE
    """
    output = []
    for y in range(y_size):
```

```
        scanLine = [] # (re)initialize empty list
        for x in range(x_size):
            scanLine.append(Image.getpixel((x,y)))
        output.append(scanLine)
    return output



def analyze_image(image_array, x_size, y_size):
    """
    Input is image array of [[px11, px12, px13...],
                             [px21, px22, px23...],
                             ...]
    array where
    each pxij is a tuple of (R,G,B,A) numbers; dimensions of image
    Outputs R, G, B, A average score
    Global distribution -- Strawman identification method, doesn't really work
    """
    sumR = 0
    sumG = 0
    sumB = 0
    sumA = 0
    for y in range(y_size):
        for x in range(x_size):
            sumR += image_array[y][x][0]  # scanline method
            sumG += image_array[y][x][1]
            sumB += image_array[y][x][2]
            sumA += image_array[y][x][3]
    norm = 1.0/(x_size*y_size*255)
    output = (sumR*norm, sumG*norm, sumB*norm, sumA*norm)

    print("Global image color %:", np.round(output,3))
    # note numpy won't work in my Webots environment
    return output


# Get image dimensions from .size method (128, 64)
image_x_size = aqua_zombie.size[0]
image_y_size = aqua_zombie.size[1]

print("\nAqua zombie:")
aqua_zombie_array = make_image_array(aqua_zombie, image_x_size, image_y_size)
analyze_image(aqua_zombie_array, image_x_size, image_y_size)

print("\nGreen zombie:")
green_zombie_array = make_image_array(green_zombie, image_x_size, image_y_size)
analyze_image(green_zombie_array, image_x_size, image_y_size)

print("\nBlue zombie:")
blue_zombie_array = make_image_array(blue_zombie, image_x_size, image_y_size)
analyze_image(blue_zombie_array, image_x_size, image_y_size)

print("\nPurple zombie:")
purple_zombie_array = make_image_array(purple_zombie, image_x_size, image_y_siz
analyze_image(purple_zombie_array, image_x_size, image_y_size)
```

```
print("\nBackground:")
background_array = make_image_array(background, image_x_size, image_y_size)
analyze_image(background_array, image_x_size, image_y_size)


"""------------ Dominant color method ------------"""


# Green zombie has very green part at (x,y)=(50,35) and shadow at (50,42)
print(background_array[5][20])   # [y][x] indexing
print(background_array[30][80])   # shadow
print(background_array[25][15])


def compute_background_features(background_array):
    """
    Utilizing our knowledge of the background (blue sky, grey mountains
    and peach earth), this function computes the mean and standard
    deviations of each block by statistics of lines
    Need to know x_size, y_size of camera image
    numpy.mean, numpy.std
    """
    #print("~~~ Development testing of background features ~~~")
    all_R = np.array(()) # list
    all_G = np.array(())
    all_B = np.array(())
    for row_idx in range(40, 60):
        # change index to cover sky, mountain or earth
        for x in range(128):
            # appends to a copy of array, need =
            all_R = np.append(all_R, background_array[row_idx][x][0])
            all_G = np.append(all_G, background_array[row_idx][x][1])
            all_B = np.append(all_B, background_array[row_idx][x][2])
    #print("R mean, standard deviation:",np.mean(all_R), np.std(all_R))
    #print("G mean, standard deviation:",np.mean(all_G), np.std(all_G))
    #print("B mean, standard deviation:",np.mean(all_B), np.std(all_B))

    return


def is_pixel_match(pixel_RGB, target_RGB):
    """
    Helper function. Input: pixel RGB is tuple of three values
    Assume standard deviation (error bar) of +/-10 for all values
    Returns True if match, False if not
    10 is empirically tested; if too big we can't pin down zombie type
    """
    flag = True       # True if pixel is target
    for idx in (0,1,2):
        if pixel_RGB[idx] not in range(target_RGB[idx]-12, target_RGB[idx]+12):
            flag = False
            break
    return flag
```

```
def zombie_lookout(image_array, x_size, y_size, threshold):
    """
    Main vision-based zombie alert function
    Need to test vigorously with different inputs!
    Inputs:
        - Array
        - Image dimensions
        - Pixel threshold for zombie warning
    Psuedocode:
        - Filter out pixels with sky, mountain or earth
        - Of remaining pixels cluster pixels to determine if a zombie is near
        - From average of cluster, determine relative horizontal position
    Return None if no zombie, else return type and angle
    """
    filtered_array = []   # new list of RGB pixels for non-background objects
    filtered_pos = []     # positions to save x,y information
    for row_idx in range(y_size):
        for col_idx in range(x_size):
            pix_RGB = image_array[row_idx][col_idx]    # a tuple
            # Now compare againsst known Sky, Mountain and Earth data
            if(is_pixel_match(pix_RGB, visual_dict["Sky"]) == False):
                if(is_pixel_match(pix_RGB, visual_dict["Mountain"]) == False):
                    if(is_pixel_match(pix_RGB, visual_dict["Earth"]) == False):
                        filtered_array.append(pix_RGB)
                        filtered_pos.append((col_idx, row_idx))  # note order!


    # Find the color via counting
    print("Filtered length:",len(filtered_array))
    aqua_score = [0,0,0]  # pixel count, Sum of x (col_idx), Sum of y (row_idx)
    blue_score = [0,0,0]
    green_score = [0,0,0]
    purple_score = [0,0,0]
    #debug_array_x = []    # for debugging to see which pixels are picked up
    #debug_array_y = []

    for idx in range(len(filtered_array)):
        if (is_pixel_match(filtered_array[idx],visual_dict["Aqua_bright"]) or i
            aqua_score[0] += 1.0  # increment count
            aqua_score[1] += filtered_pos[idx][0] # col_idx
            aqua_score[2] += filtered_pos[idx][1] # row_idx
        if (is_pixel_match(filtered_array[idx],visual_dict["Blue_bright"]) or i
            blue_score[0] += 1.0  # increment count
            blue_score[1] += filtered_pos[idx][0] # col_idx
            blue_score[2] += filtered_pos[idx][1] # row_idx
            #debug_array_x.append(filtered_pos[idx][0])
            #debug_array_y.append(filtered_pos[idx][1])

        if (is_pixel_match(filtered_array[idx],visual_dict["Green_bright"]) or
            green_score[0] += 1.0  # increment count
            green_score[1] += filtered_pos[idx][0] # col_idx
            green_score[2] += filtered_pos[idx][1] # row_idx
```

```
            if (is_pixel_match(filtered_array[idx],visual_dict["Purple_bright"]) or
                purple_score[0] += 1.0  # increment count
                purple_score[1] += filtered_pos[idx][0] # col_idx
                purple_score[2] += filtered_pos[idx][1] # row_idx

        # Primary zombie ID complete, analyze results
        # We estimate scaling factors for range and angle to zombie
        print("\nDebugging scores:", aqua_score, blue_score, green_score, purple_sc

        if aqua_score[0] < threshold and blue_score[0] < threshold and \
            green_score[0] < threshold and purple_score[0] < threshold:
            # No zombie nearby; use threshold instead of 0
            return None


        if aqua_score[0] >= blue_score[0] and aqua_score[0] >= green_score[0] and a
            # aqua biggest
            zombie_type = "aqua"
            #zombie_distance = (1-aqua_score[0]/(x_size*y_size))*5.0 # in meters
            zombie_x = aqua_score[1] / aqua_score[0]    # float, x-center of mass
            angle = (zombie_x - x_size)/x_size * 28.5   # angles - estimation from

        elif blue_score[0] >= green_score[0] and blue_score[0] >= purple_score[0]:
            # check if blue biggest
            zombie_type = "blue"
            #zombie_distance = (1-blue_score[0]/(x_size*y_size))*5.0 # in meters
            zombie_x = blue_score[1] / blue_score[0]    # float, x-center of mass
            angle = (zombie_x - x_size)/x_size * 28.5   # angles - estimation from

        elif green_score[0] >= purple_score[0]:
            # check if green biggest
            zombie_type = "green"
            #zombie_distance = (1-green_score[0]/(x_size*y_size))*5.0 # in meters
            zombie_x = green_score[1] / green_score[0]  # float, x-center of mass
            angle = (zombie_x - x_size)/x_size * 28.5   # angles - estimation from

        else:
            # so purple is biggest
            zombie_type = "purple"
            #zombie_distance = (1-purple_score[0]/(x_size*y_size))*5.0 # in meters
            zombie_x = purple_score[1] / purple_score[0] # float, x-center of mass
            angle = (zombie_x - x_size)/x_size * 28.5    # angles - estimation from

        # debug
        #plt.plot(debug_array_x,debug_array_y, ".")
        #plt.show()

        return zombie_type, angle



# Get background statistics
compute_background_features(background_array)

# ID zombie presence, size, location
```

```
plt.imshow(aqua_zombie)  # show in Spyder console
plt.show()
print(zombie_lookout(aqua_zombie_array, image_x_size, image_y_size, 50))


plt.imshow(blue_zombie)
plt.show()
print(zombie_lookout(blue_zombie_array, image_x_size, image_y_size, 50))

plt.imshow(green_zombie)
plt.show()
print(zombie_lookout(green_zombie_array, image_x_size, image_y_size, 50))

plt.imshow(purple_zombie)
plt.show()
print(zombie_lookout(purple_zombie_array, image_x_size, image_y_size, 50))


plt.imshow(background)
plt.show()
print(zombie_lookout(background_array, image_x_size, image_y_size, 50))


"""
Aqua (37, 221, 194) and (10, 69, 67)
Blue (33, 143, 230) and (10, 40, 99)
Green (35, 192, 39) and (9, 53, 12)
Purple (112, 48, 182) and (87, 37, 154)

Background feature
Sky (90, 109, 152) std (9, 9, 7)
Mountain (77, 73, 78) std (11, 9, 7)
Earth (217, 182, 169) std (4, 4, 6)
"""

print("\nDone")
```