This repository Search Pull requests Issues Marketplace Gist

articuno144 / DMT2017

Unwatch 2 Star 1

Code Issues 0 Pull requests 0 Projects 0 Wiki Settings Insights

Branch: master DMT2017 / main / camera_functions.py Find file

articuno144 Finished project 69c5d8f 3

1 contributor

172 lines (152 sloc) 6.12 KB

Raw Blame History

```python
import cv2
import time
import math
import numpy as np
import imutils
from threading import Thread

# assign new item lower['blue'] = (93, 10, 0)
lower = {'blue': (75, 120, 50), 'orange': (150, 120, 0)}
upper = {'blue': (150, 255, 255), 'orange': (200, 255, 255)}

# define standard colors for circle around the object
colors = {'blue': (0, 0, 255), 'orange': (0, 140, 255)}


def get_angle(x, y, w=640, h=480):
    """
    From the x,y location read from the camera, get tangent of the
    horizontal angle alpha and vertical angle beta.
    """
    x = x-w/2
    y = h/2-y
    ta = x*math.tan(math.radians(30))/(w/2)
    tb = y*math.tan(math.radians(25))/(h/2)
    return [ta, tb]


def get_coordinates(cam1_tan, cam2_tan, cam3_tan, a=0.815, b=0.815, c=0.815):
    # tans ==> array of 6 tangents
    [ta1, tb1], [ta2, tb2], [ta3, tb3] = cam1_tan, cam2_tan, cam3_tan
    mx1y1 = np.array([[1, -ta2], [ta1, 1]])  # cam1,2
    [x1, y1] = np.linalg.inv(mx1y1).dot(np.array([-b*ta2, a*ta1]))
    mx2z1 = np.array([[1, ta3], [tb1, 1]])  # cam1,3
    [x2, z1] = np.linalg.inv(mx2z1).dot(np.array([c*ta3, a*tb1]))
    my2z2 = np.array([[1, tb3], [tb2, 1]])  # cam2,3
    [y2, z2] = np.linalg.inv(my2z2).dot(np.array([c*tb3, b*tb2]))
    x = 0.5*(x1+x2)
    y = 0.5*(y1+y2)
    z = 0.5*(z1+z2)
    return np.array([x, y, z])


def simplified_loop(coordinates, read_failed, printing=False, imshow0=None, imshow1=None, imshow2=None):
    vc0, vc1, vc2 = Init()
    while True:
        colored_Cam(coordinates, read_failed, vc0, vc1, vc2, imshow0, imshow1,imshow2)
        if printing:
            print(coordinates, read_failed)
        key = cv2.waitKey(10)
        if key == 27:  # exit on ESC
            break


def Init():
    # Cam 0
    vc0 = cv2.VideoCapture(0)
    vc0.set(3, 640)
    vc0.set(4, 240)
    vc0.set(15, -6)  # exposure
    # Cam 1
    vc1 = cv2.VideoCapture(1)
    vc1.set(3, 640)
    vc1.set(4, 240)
    vc1.set(15, -6)  # exposure
    # Cam 2
    vc2 = cv2.VideoCapture(3)
    vc2.set(3, 640)
    vc2.set(4, 240)
    vc2.set(15, -6)  # exposure
    rval0, frame0 = vc0.read()
    rval1, frame1 = vc1.read()
    rval2, frame2 = vc2.read()
    assert vc0.isOpened(), "can't find camera 0"
    assert vc1.isOpened(), "can't find camera 1"
    assert vc2.isOpened(), "can't find camera 2"
    return vc0, vc1, vc2


def frame_loc(vc, imshow=None):
    """
    Takes the videoCapture object and cam_num as the input,
    returns the drone location.
    """
    ox, oy, bx, by = 0, 0, 0, 0
    rval, frame = vc.read()
    blurred = cv2.GaussianBlur(frame, (11, 11), 0)
    hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
    # for each color in dictionary check object in frame
    for key, value in upper.items():
        # construct a mask for the color from dictionary`1, then perform
        # a series of dilations and erosions to remove any small
        # blobs left in the mask
        kernel = np.ones((9, 9), np.uint8)
        mask = cv2.inRange(hsv, lower[key], upper[key])
        mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)
        mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
        if imshow != None:
            cv2.imshow(key+imshow, mask)
        # find contours in the mask and initialize the current
        # (x, y) center of the ball
        cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
                                cv2.CHAIN_APPROX_SIMPLE)[-2]
        center = None

        # only proceed if at least one contour was found
        if len(cnts) > 0:
            # find the largest contour in the mask, then use
            # it to compute the minimum enclosing circle and
            # centroid
            c = max(cnts, key=cv2.contourArea)
            ((x, y), radius) = cv2.minEnclosingCircle(c)
            M = cv2.moments(c)
            center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))

            # only proceed if the radius meets a minimum size. Correct this
            # value for your obect's size
            if radius > 0.5 and radius < 60:
                if key == 'orange':
                    ox, oy = x, y
                elif key == 'blue':
                    bx, by = x, y
                # draw the circle and centroid on the frame,
                # then update the list of tracked points
                cv2.circle(frame, (int(x), int(y)),
                           int(radius), colors[key], 2)
    return ox, oy, bx, by


def colored_Cam(coordinates, read_failed, vc0, vc1, vc2, imshow0=None, imshow1=None, imshow2=None):
    """
    coordinates and read_failed have length two, for orange and
    blue balls.
    """
    ox0, oy0, bx0, by0 = frame_loc(vc0, imshow0)
    ox1, oy1, bx1, by1 = frame_loc(vc1, imshow1)
    ox2, oy2, bx2, by2 = frame_loc(vc2, imshow2)
    loc_orange = get_coordinates(
        get_angle(ox0, oy0), get_angle(ox1, oy1), get_angle(ox2, oy2))
    loc_blue = get_coordinates(
        get_angle(bx0, by0), get_angle(bx1, by1), get_angle(bx2, by2))
    coordinates[0][:] = list(loc_orange)[:]
    coordinates[1][:] = list(loc_blue)[:]
    if ox0*ox1*ox2 != 0 or oy0*oy1*oy2 != 0:  # captured by all 3 cams
        read_failed[0][0] = 0
        print("orange found drone")
    else:
        read_failed[0][0] = 1
    if bx0*bx1*bx2 != 0 or by0*by1*by2 != 0:  # captured by all 3 cams
        read_failed[1][0] = 0
        print("blue found drone")
    else:
        read_failed[1][0] = 1

if __name__ == '__main__':
    coordinates = [[0, 0, 0], [0, 0, 0]]
    read_failed = [[1], [1]]
    # vc0, vc1, vc2, first_frame0, first_frame1, first_frame2 = Init()

    # Thread(target=threaded_loop_test, args=(vc0, first_frame0, "0",)).start()
    # Thread(target=threaded_loop_test, args=(vc1, first_frame1, "1",)).start()

    # camera_Thread = Thread(
    #     target=threaded_loop, args=(coordinates,
    #                                  vc0, vc1, vc2, first_frame0, first_frame1,
    #                                  first_frame2, "0", "1", "2",))
    camera_Thread = Thread(target=simplified_loop,
                           args=(coordinates, read_failed, True,"0","1","2"))
    camera_Thread.start()
    while 1:
        time.sleep(1)
    # threaded_loop(vc0, vc1, vc2, first_frame0, first_frame1, first_frame2, "0", "1", "2",)
```