# ransacTrack

March 22, 2024

[ ]:
```
import cv2 as cv
from matplotlib import pyplot as plt
import numpy as np
import os
from ransac import ransac
from imageProcessor import imageProcessor
# Import necessary modules for 3D plotting
from mpl_toolkits.mplot3d import Axes3D

# Create lists to store coordinates


filename = "videos/coin.mov"
capture = cv.VideoCapture(filename)
framecnt = 0

if not os.path.exists('Frame_Dump'):
    os.mkdir('Frame_Dump')
else:
    print("Directory already exists, proceeding with overwriting directory\n")

frame_count_list = []
center_xlist = []
center_ylist = []
plt.ion()  # Turn on interactive mode

try:
    while capture.isOpened():
        state, frame = capture.read()
        if state:
            framecnt += 1
            centerX, centerY = imageProcessor(frame, framecnt)

            # If center x-coordinate is found, append it to the list
            if centerX and centerY is not None:
                frame_count_list.append(framecnt)
                center_xlist.append(centerX)
```

```
                center_ylist.append(centerY)
                # Plot the center coordinates with respect to frame count
                plt.clf()  # Clear the previous plot
                plt.plot(frame_count_list, center_xlist, label='Center X␣
 ↪Coordinate', marker='o', linestyle='-')
                plt.plot(frame_count_list, center_ylist, label='Center Y␣
 ↪Coordinate', marker='o', linestyle='-')
                plt.xlabel('Frame Count')
                plt.ylabel('Center Position')
                plt.legend()
                plt.title('Center Position over Frame Count')
                plt.draw()
                plt.pause(0.1)  # Adjust the pause duration as needed

            print('FrameCount:' + str(framecnt) + '\n')
        else:
            print("No remaining frames to process\n")
            break

except KeyboardInterrupt:
    print("Interrupted by user")

finally:
    plt.ioff()  # Turn off interactive mode
    capture.release()
    plt.show()
```

The video is broken up into frames, and each individual frame is passed into the python function imageProcessor, defined below. The function below performs morphological operations such as binary masks, dilation and erosion, and finally edge detection. The edge points detected are then passed into the RANSAC algorithm.

[ ]:
```
import cv2 as cv
from matplotlib import pyplot as plt
import numpy as np
import os
from ransac import ransac

def imageProcessor(img, framecnt):
#Currently converts image to HSV to test functionality. We can build algorithm␣
 ↪here.


    #show the original image
    #cv.imshow('video reader', img)

    height, width = img.shape[:2]
```

```
# Define the ROI parameters
    top_margin = height // 8  # 1/8th of the height from the top
    bottom_margin = height // 8  # 1/8th of the height from the bottom


# Applying ROI mask
    roi = img[top_margin:height - bottom_margin]
    avgkernel = np.ones((8,8),np.float32)/(64)

# Convert the image to grayscale
    gray_img = cv.cvtColor(roi, cv.COLOR_BGR2GRAY)
    #cv.imshow('region of interest', gray_img)
# Ensure that the image is of type CV_8UC1
    if gray_img.dtype != np.uint8:
        gray_img = cv.convertScaleAbs(gray_img)

#Apply Otsu's thresholding
    ret, mask = cv.threshold(gray_img, 0, 255, cv.THRESH_BINARY + cv.
 ↪THRESH_OTSU)

#create structuring element and use it to perform opening mask
    disc = cv.getStructuringElement(cv.MORPH_RECT, (5,5))
    #cv.imshow('mask', mask)

#dilate, erode, and dilate again.
    eroded_mask = cv.erode(mask, disc,1)
    dilated_mask = cv.dilate(eroded_mask, disc, 5)
    #cv.imshow('mask', dilated_mask)


    edges = cv.Canny(dilated_mask, 0, 220)
    #cv.imshow('edges', edges)

    Circle = ransac(edges,350, 50,10, framecnt, roi)

    cv.waitKey(200)
    if Circle is not None:
        return Circle[0]  # Return only the center coordinates (x, y)
    else:
        return 0,0  # Return None when no circle is found
```

The RANSAC algorithm below takes the arguments in terms of a radius threshold in pixels, the edge points, the max number of iterations per frame, the minimum number of inliers, and the original image. The algorithm uses the Hough Transform of a circle by randomly sampling three edge points, drawing circles around each point in parameter space, and from there, the intersections between the circles in parameter space are stored in an accumulator array. Non-maximal suppression is

applied to find the most dominant circle intersection points in parameter space. The intersection point with the most amount of votes in the accumulator array, is then considered a circle center in image space.

[ ]:
```
import cv2 as cv
import numpy as np

def ransac(img, threshold, max_iterations, min_inline, framecnt, orig):
    #Collect number of edge points
    num_points = np.column_stack(np.where(img > 0))

    #check to ensure there are edge points present in the image.
    if len(num_points) == 0:
        print("No edge points found in the binary image.")
        return None

    #preallocate memory for variables such as best circle, max_inliers, and the accumulator array.
    accumulator = np.zeros_like(img, dtype=int)
    best_circle = None
    max_inliers = 0

    #Iterate through the desired number of RANSAC iterations
    for i in range(max_iterations):
        #randomly sample x-number of points. Best number of points is related to the accuracy of the RANSAC.
        sample_index = np.random.choice(len(num_points), size=3)
        sample_points = num_points[sample_index]

        #append the randomly sampled points
        xi, yi = sample_points[:, 1], sample_points[:, 0]

        #create a circle in parameter space around the sampled points.
        c = np.array([np.mean(xi), np.mean(yi)])
        r = np.mean(np.sqrt((xi - c[0]) ** 2 + (yi - c[1]) ** 2))

        #ensure that the sampled circles are within image bounds.
        if c[0] < img.shape[1] and  c[1] < img.shape[0] and 0 < r < min(img.shape[0], img.shape[1]) / 2:
            #in image space, a and b are the center of the circle. In parameter space, it will be the circular edge coordinates with distinct sample points.
            th = np.arange(0, 2 * np.pi, 0.01)
            a = c[0] + (r * np.cos(th))
            b = c[1] + (r * np.sin(th))
```

```
            #ensures that the generated circle coordinates are converted to␣
↪integers, and any potential coordinates close to the image boundaries are␣
↪clipped within a safe margin
            #to prevent the circle from going outside the valid image region.
            #This is crucial for subsequent operations, such as updating the␣
↪accumulator with votes for the circle in the Hough space.
            margin = 5  # Adjust this margin as required (found experimentally)
            a_idx = np.clip(a.astype(int), 0 + margin, img.shape[1] - 1 -␣
↪margin)
            b_idx = np.clip(b.astype(int), 0 + margin, img.shape[0] - 1 -␣
↪margin)
            #Increment the accumulator at the circle coordinates
            accumulator[b_idx, a_idx] += 1

            # Non-maximal suppression: enhance local maxima in accumulator␣
↪array (dilation), and  apply logical mask to accumulator array to only␣
↪retain local maxima.
            accumulator = cv.convertScaleAbs(accumulator)
            local_max = cv.dilate(accumulator, np.ones((3, 3)), iterations=1)
            lmax_mask = (local_max == accumulator)
            accumulator *= lmax_mask

    #Iterate through the coordinates of the local maxima in the parameter space␣
↪(max_coords).
    for max_coords in np.argwhere(accumulator > 0):

        #The radius and center of the circle are calculated based on the␣
↪relationship between the sampled points (xi and yi) and the coordinates of␣
↪the local maxima.
        radius = np.mean(np.sqrt((xi - max_coords[1]) ** 2 + (yi -␣
↪max_coords[0]) ** 2))
        center = (max_coords[1], max_coords[0])

        #Generate the coordinates of the circle and clip them to ensure they␣
↪are within the image bounds
        th = np.arange(0, 2 * np.pi, 0.01)
        x_circle = (center[0] + radius * np.cos(th)).astype(int)
        y_circle = (center[1] + radius * np.sin(th)).astype(int)

        x_circle = np.clip(x_circle, 0, img.shape[1] - 1)
        y_circle = np.clip(y_circle, 0, img.shape[0] - 1)

    #Count the number of inliers (edge points inside the circle)
        inliers = np.sum(img[y_circle, x_circle] > 0)
    #if the circle meets the criteria for a valid circle (minimum inliers,␣
↪radius within threshold, and more inliers than the current best),
```

```
    #update the best circle if needed
        if inliers >= min_inline and radius <= threshold and inliers >␣
↪max_inliers:
            #print(f"Found Circle with radius {radius} px and center {center}")
            best_circle = (center, radius)
            max_inliers = inliers

    #if there is a best circle detected that satisfies all aformentioned␣
↪criteria, visualize it and save it as an image.
    if best_circle is not None:
        # Visualize the best circle
        circ = orig.copy()
        cv.circle(circ, best_circle[0], int(best_circle[1]), [0, 255, 0], 2)
        cv.circle(circ, best_circle[0], 5, [0, 0, 255], -1)
        cv.putText(circ, f'Center: ({best_circle[0][0]}, {best_circle[0][1]})',␣
↪(10, 30), cv.FONT_HERSHEY_SIMPLEX, 1,
                    (0, 255, 255), 2)

        cv.imshow('Best Circle', circ)
        cv.imwrite(f"Frame_Dump/Best_Circle_Detected{framecnt}.png", circ)
        cv.waitKey(10)

    else:
        print("No circles found")

    return best_circle
```