DIP Project Report

Target Tracking of Table Tennis Using Machine Vision

11912417 Wang Anzhe

Introduction

Target tracking is a popular research field in recent years. Research on this topic has achieved efficient detection and tracking of different objects. The good performance has a price of high requirement of the computers. Our project aims to solve a certain problem in ping-pong ball detection, with high processing speed and little requirement of computational power. By using the traditional image processing methods such as HSV threshold detection, Canny edge detection, and Hough Circle detection, we successfully construct a ping-pong ball detection system. A faster remap method for 3D reconstruction is proposed in this project. Based on the reconstruction this system rebuilds the 3D track of the ball with a high accuracy rate.

Related work

The deep-Learning based methods are dominant in the field of object tracking because of their high accuracy rate and transferable applications. Object detection has achieved fast development in recent years, successful approaches empower researchers to easily detect the object, such as the famous YOLO. [1]
For ping-pong ball detection, the deep-learning-based methods also achieved a good result. Gomez Gonzalez has created highly accurate ping-pong balls based on deep learning. [2]
However, the detection of ping-pong balls is a highly specified work, with much previous knowledge like the shape, size, and color. By using this knowledge, we can build a system with traditional image processing methods. Jonas Tebbe et al. adopt a method of detecting table tennis based on color and contour size. [3]

Overview

In order to develop a high processing speed and high accuracy system, the system will rely on color threshold detection to locate the ball.
The potential problem for the color threshold detection is the interference of the object in the image with similar colors, such as the hands and table tennis racket. A solution for this problem is to do Hough Circle detection to remove the interference of objects with similar colors.
This project will design an algorithm based on the physical properties of the ping-pong ball. The combination of color detection, edge detection, and circle detection is used to determine the position of the ball.

Methods

The overall diagram of the system processing flow is shown in figure 1.

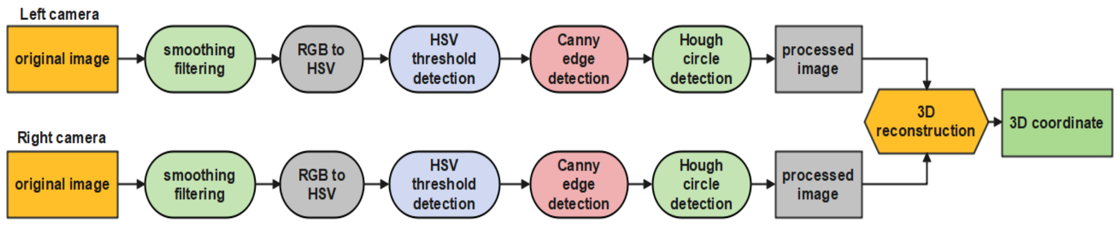


Figure 1 overall diagram

The system will use the images from the binocular camera system as the input. For each side, the system will apply the same image processing system. The main steps are the following:

1. *smoothing filtering.*

The smoothing filtering will help to remove noise and contribute to the following detection.

2. *HSV color threshold detection*

By transferring the original RGB image to the HSV color space, the different colors in the image can be more easily separated from each other. HSV means hue, saturation, and value, and it provides a more intuitive representation of the way that humans sense the colors. The equation of the color space transformation is shown in figure 2.

$$\begin{aligned} &the\ conversion\ from\ RGB\ to\ HSV: \\ &C_{max} = max(R,G,B) \\ &C_{min} = min(R,G,B) \\ &\delta = C_{max} - C_{min} \end{aligned}$$

$$V = max(R,G,B)$$

$$S = \begin{cases} 0 & C_{max} = 0 \\ \delta / C_{max} & C_{max} != 0 \end{cases}$$

$$H = \begin{cases} 0 & \delta = 0 \\ 60 * \dfrac{(G-B)}{\delta} & C_{max}\ is\ R \\ 60 * \dfrac{(B-R)}{\delta + 2} & C_{max}\ is\ G \\ 60 * \dfrac{(R-G)}{\delta + 4} & C_{max}\ is\ B \end{cases}$$

$$H = H/360 * 180$$

Figure 2 RGB to HSV

```
HSV-RGB
for each pxiels
    C_max = max(R,G,B)
    C_min = min(R,G,B)
    delta = C_max - C_min
    if C_max == 0
        S = 0
    elseif C_max != 0
        S = delta / C_max

    V = C_max

    if delta == 0:
        H = 0
    elif C_max == R:
        H = 60 * ((G - B) / delta)
    elif C_max == G:
        H = 60 * ((B - R) / delta + 2)
    elif C_max == B:
        H = 60 * ((R - G) / delta + 4)

    if H[i, j] < 0:
        H[i, j] = H[i, j] + 360

    H[i, j] = H[i, j] * 0.5
```

Figure 3 pseudocode for RGB-HSV

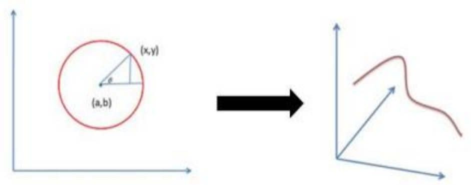
3. *Canny edge detection*

The Canny edge detector will be used in order to provide useful edge information for the following procedures. The Canny edge detection is composed of smoothing, calculation of gradient and its direction, non-maximum suppression, and double-threshold detection. [4] The thin edge information will be helpful to reduce the processing time of the following Hough detection.

4. *Hough circle detection*

Hough circle detection is converting a circle in the 2D image space into a point in the 3D parameter space. A circle is determined by the radius, horizontal and vertical coordinates of the center. Therefore, a circle after the Hough transformation corresponds to a point in the three-dimensional parameter space (abr coordinate system). [5]

For each point in the abr coordinate system, if the cumulative junctions at one point are greater than a certain threshold, then a circle(at XY coordinate system) corresponding to this point(at abr coordinate system) in the image is detected (as shown in figure 3).



$$(x-a)^2+(y-b)^2=r^2 \qquad a = x - r\cos\theta \qquad b = y - r\sin\theta$$

Figure 4 illustration of Hough circle transform

The Hough circle detection will be a time-consuming process. Even though the HSV processed image is simple. A location of the position ball position will be helpful to the circle detection.

```
hough detection pseudocode:
input:edge_image/r_min/r_max/step length/num of thetas/threshold

step of theta: 360 / num of thetas
range of radius: r_min to r_max with step length
step of cos value: cos(step of theta)
step of sin value: sin(step of theta)

//then we create a set of candidates
for every step of radius/step of cos value/step of sin value
    candidates: (r, r*cos, r*sin)

//create a dict called voter
voter = defaultdict(int)

for every pixels
    if edge_image[i,j] != 0
        //by checking all the candidates list
        //calculate all the possible (x,y,r) info using:
        x_center = i - r*cos
        y_center = j - r*sin
        voter[(x_center, y_center, r)] += 1

for all possible circle in voter
    if value of possible circle > threshold
        a circle detected and append circle info to output

return the info of (x,y,r) of the circles
```

Figure 5 pseudocode for Hough circle detection

5. *Binocular 3D reconstruction*

Binocular vision is a method that simulates the human vision principle and uses the computer to perceive distance. Observe an object from two or more points, obtain images from different perspectives, and calculate the offset between pixels according to the matching relationship between images through the principle of triangulation to get three-dimensional information about the object.

The first step is to do stereo rectification using the functions provided by the OpenCV. The meaning of this step is to remap the left and right images to be parallel images.

In the following equation, x/y is the ball's circle center coordinate in the left image, c_x and c_y are the centers of the image, f is the focal length, and d is the distance between two cameras. Using this relation, we can get the 3D coordinate of the ball.

$$Q\begin{bmatrix} x \\ y \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} x - c_x \\ y - c_y \\ f \\ \frac{-d+c_x-c_x'}{T_x} \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix}$$

Figure 6 3D reconstruction

The Canny edge detection and stereo rectification of the cameras will be based on the OpenCV function.

Result Images

1. *HSV threshold detection*

In figure 5, there's a yellow ball in the image, which has different color compared to the background. By using the equation in figure 2, it can get figure 6 which is in HSV color space. Finally, the threshold will block out the background and leave the ball in the image as shown in figure 7.

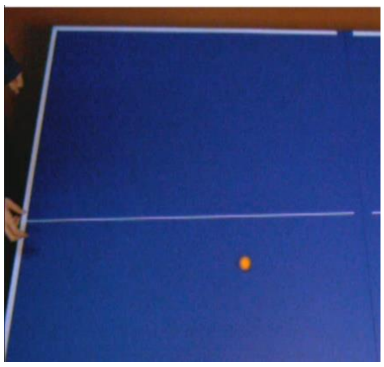


Figure 7 image in RGB color space

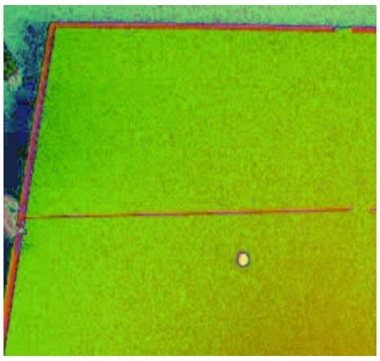


Figure 8 image in HSV color space

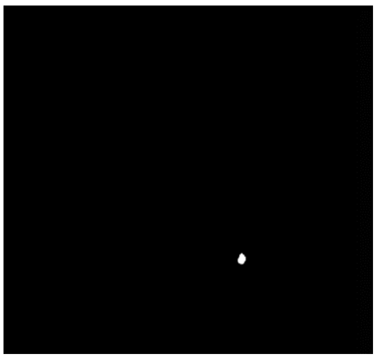


Figure 9 image after HSV threshold detection

2. *Hough circle detection*

Hough circle can be applied not only in our project for ping-pong balls. As long as the object is a circle, the Hough circle can effectively give the position and radius of the circle that meets the requirements.

Figure 8 is an image with a basketball, and figure 9 is the result of Canny edge detection which gives the gradient and gradient direction information required in the Hough circle detection. Figure 10 is the final result with a green circle showing the contour of the ball.

Figure 10 original image

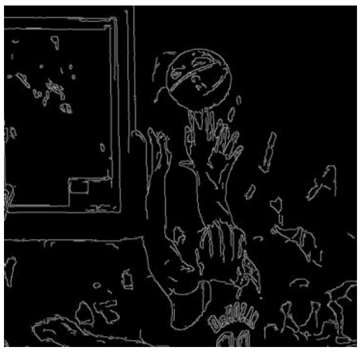


Figure 11 Canny edge detection

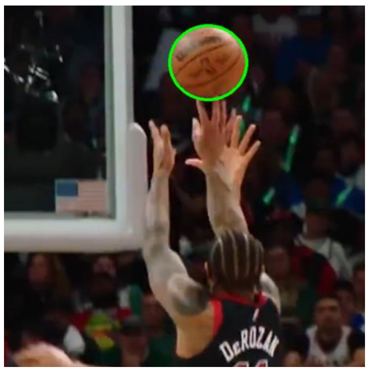


Figure 12 contour of the ball

3. Binocular 3D reconstruction

After 3D reconstruction, the coordinates represented the ball information can be plot in the model. The input is the image of two cameras in figure 11. The reconstructed trajectory can be plotted using matplotlib in figure 12 or using a 3D model in figure 13 to plot the 3D trajectory of a ping-pong ball.

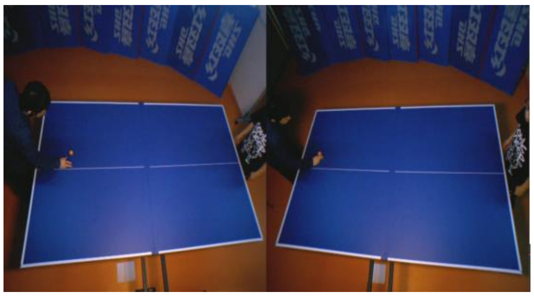


Figure 13 left/right images

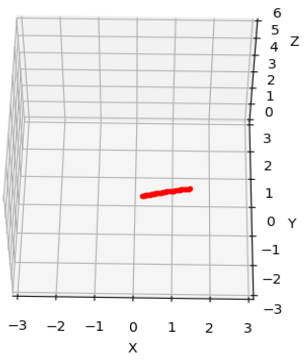


Figure 14 trajectory of a rolling ball

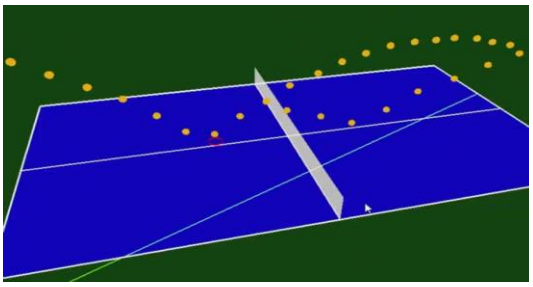


Figure 15 trajectory of a ball in a 3D model

Implementation and Analysis

The main criterion of the system is processing speed. The HSV threshold detection and the Hough circle detection are implemented and improved in this project.

A. HSV Threshold detection

The HSV threshold detection will calculate the hue, saturation, and value of every pixel in the image. It's a time-consuming process that contains a large number of simple but repetitive calculations for every pixel. In this case, a process containing mainly "for loops" can be accelerated using GPU to calculate. A python package called *numba.jit* can be used to easily implement the GPU acceleration.

type / **size**	OpenCV version	GPU version	Original version
100*100	*0.0001s*	*0.02s*	*0.49s*
200*200	*0.001s*	*0.10s*	*0.60s*
500*500	*0.005s*	*0.47s*	*0.7s*
1000*1000	*0.01s*	*0.49s*	*2.90s*
2000*2000	*0.01s*	*0.52s*	*11.1s*
4000*4000	*0.03s*	*0.67s*	*46.49s*

Table 1 comparison between HSV functions

From Table 1, it's clear to see the advantage of the parallel calculation. Although our method is still slower than the OpenCV version (OpenCV is based on C, which is faster than python), the trend of time costing of the optimization is similar to the OpenCV version. The time cost of the GPU version is even less than linear increasing. This is a huge improvement compared to the original version.

B. Hough circle detection

The Hough circle detection is a time-costing process, which costs around 10 seconds for a 1080p frame. By using the information got from the HSV threshold process, the circle detection time will largely decrease. The Hough detection will only apply to the possible region in which objects have a similar color to the ping-pong ball. The Hough circle detection will be applied to a small window of 25*25 size containing the nonzero region.

Table 2 shows the processing speed of the frame speed before and after locating the possible regions.

Whole image	**Local image**
18.9s	*0.02s*

Table 2 comparison between whole/local processing of Hough circle detection

If we apply Hough detection to the entire image, the time cost will be too high to apply a real-time detection system.

C. Multi-Threading

For the whole system as shown in figure 1, the left and right camera frames can be processed independently before the 3D reconstruction. As a result, using multi-threading can parallelly process the image. Table 3 shows the processing speed of the entire system including image processing, 3D reconstruction, and plotting.

One thread	**Two threads**
5 frame/s	*8 frame/s*

Table 3 one/multi-thread comparison

Conclusion

In this project, a 3D reconstruction based on the HSV threshold detection, edge detection, Hough circle detection, and Binocular 3D reconstruction is successfully implemented in this project. The system can run on 8G memory, i5-9300H CPU @ 2.40GHz, and NVIDIA GTX 1660Ti with 30Hz. The reconstruction of a table tennis ball in a 3D figure is shown on the right figure. Besides, the exact trajectory and drop-point detection using the trajectory will be shown in a visualization animation interface.

The deep-Learning based methods are dominant in the field of object tracking because of their high accuracy rate and transferable applications. But for a specific job, such as the detection of ping-pong balls, one can use much previous knowledge like the shape, size, and color. With this knowledge, we can complete the work with traditional image processing methods in a more intuitive way and receive better results compared to deep-Learning based methods which have wider applications.

Reference

[1] J. Redmon 和 S. Divvala, "You Only Look Once: Unified, Real-Time Object Detection," *Computer Vision & Pattern Recognition,* 2016.

[2] S. Gomez-Gonzalez, " Reliable Real Time Ball Tracking for Robot Table Tennis," *cs.RO,* 2019.

[3] J. Tebbe, "A Table Tennis Robot System Using an Industrial KUKA Robot Arm," *GCPR 2018,* 2019.

[4] J. Canny, "A computational approach to edge detection," *IEEE Trans Pattern Anal Mach Intell,* 1986.

[5] D.H.BALLARD, " GENERALIZING THE HOUGH TRANSFORM TO DETECT ARBITRARY SHAPES, " *Pattern Recoqnition,* 1981.

Hough detection code:

```python
def find_hough_circles_fast(edge_image, r_min, r_max, delta_r, num_thetas, bin_threshold):
    img_height, img_width = edge_image.shape[:2]

    dtheta = int(360 / num_thetas)

    thetas = np.arange(0, 360, step=dtheta)

    rs = np.arange(r_min, r_max, step=delta_r)

    cos_thetas = np.cos(np.deg2rad(thetas))
    sin_thetas = np.sin(np.deg2rad(thetas))

    circle_candidates = []
    for r in rs:
        for t in range(num_thetas):
            circle_candidates.append((r, int(r * cos_thetas[t]), int(r * sin_thetas[t])))

    accumulator = defaultdict(int)

    for y in range(img_height):
        for x in range(img_width):
            if edge_image[y][x] != 0:
                for r, rcos_t, rsin_t in circle_candidates:
                    x_center = x - rcos_t
                    y_center = y - rsin_t
                    accumulator[(x_center, y_center, r)] += 1
    out_circles = []

    for candidate_circle, votes in sorted(accumulator.items(), key=lambda i: -i[1]):
        x, y, r = candidate_circle
        current_vote_percentage = votes / num_thetas
        if current_vote_percentage >= bin_threshold:
            out_circles.append((x, y, r, current_vote_percentage))
    return out_circles
```

RGB-HSV

```python
@jit(nopython=True)
def RGB_HSV_jit(img, H, S, V):
    '''''
    accerelate using the numba.jit
    根据 cv 文档描述编写的 HSV 算法
    如果没有安装 numba.jit 包可以注释掉 @jit(nopython=True)

    H: 0-180
    S: 0-255
    V: 0-255
    '''
    row, col = img.shape[0], img.shape[1]

    img_b = img[:, :, 0]
    img_g = img[:, :, 1]
    img_r = img[:, :, 2]

    for i in range(row):
        for j in range(col):
            R = int(img_r[i, j]) / 255
            G = int(img_g[i, j]) / 255
            B = int(img_b[i, j]) / 255

            C_max = max(R, G, B)
            C_min = min(R, G, B)
            delta = C_max - C_min

            if C_max == 0:
                S[i, j] = 0
            elif C_max != 0:
                S[i, j] = delta / C_max * 255

            V[i, j] = C_max * 255

            if delta == 0:
                H[i, j] = 0
            elif C_max == R:
                H[i, j] = 60 * ((G - B) / delta)
            elif C_max == G:
                H[i, j] = 60 * ((B - R) / delta + 2)
            elif C_max == B:
                H[i, j] = 60 * ((R - G) / delta + 4)

            if H[i, j] < 0:
                H[i, j] = H[i, j] + 360

            H[i, j] = H[i, j] * 0.5

    return [H, S, V]
```