Chapter18 gym模拟环境下巡线行驶

本章参考资料

gym模拟环境

gym介绍

gym安装及注意事项

- 记得提前在环境中安装好git，再去执行`$pip install git+https://github.com/tawnkramer/gym-donkeycar`
- 如果出现`s_, r, done, info = env.step(a) ValueError: too many values to unpack (expected 4)`这样的报错，一般就是gym版本的问题，可将代码改为`s_, r, done, info, _ = env.step(a)`。但如果严格按照上述文章中的内容安装，是不会出现错误的。
- 如果想要去学习其他环境，需要安装pyglet模块，但本次实现用不到。

gym-donkeycar

提供了多个赛道作为选择

- "donkey-warehouse-v0"
- "donkey-generated-roads-v0"
- "donkey-avc-sparkfun-v0"
- "donkey-generated-track-v0"
- "donkey-roboracingleague-track-v0"
- "donkey-waveshare-v0"
- "donkey-minimonaco-track-v0"
- "donkey-warren-track-v0"
- "donkey-thunderhill-track-v0"
- "donkey-circuit-launch-track-v0"

基于openCV实现巡线行驶

```
# 导入系统库
import cv2 as cv
import numpy as np
import math
import gym
import gym_donkeycar

# 导入自定义库
from tools import region_of_interest, detect_line, average_lines, display_line
```

```
cv.namedWindow("yellow",cv.WINDOW_NORMAL)
cv.namedWindow("white",cv.WINDOW_NORMAL)
cv.resizeWindow("yellow",(300,200))
cv.resizeWindow("white",(300,200))

# 黄色区域检测
yellow_lower = np.array([15, 40, 40])
yellow_upper = np.array([45, 255, 255])

# 白色区域检测
white_lower = np.array([0, 0, 200])
white_upper = np.array([180, 30, 255])

def main():
    # 设置模拟器环境
    env = gym.make("donkey-generated-roads-v0")
    # 重置当前场景
    env.reset()
    # 开始启动
    action = np.array([0, 0.3])  # 动作控制，第1个转向值，第2个油门值
    # 执行动作
    obv, reward, done, info = env.step(action)
    # 获取图像
    frame = cv.cvtColor(obv, cv.COLOR_RGB2BGR)

    for i in range(2000):
        height, width, _ = frame.shape
        hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)

        yellow_mask = cv.inRange(hsv, yellow_lower, yellow_upper)
        white_mask = cv.inRange(hsv, white_lower, white_upper)

        #边缘检测
        yellow_edge = cv.Canny(yellow_mask, 100, 200)
        white_edge = cv.Canny(white_mask, 100, 200)
        #cv.imshow("yellow_edge",yellow_edge)

        #region of interest 获取感兴趣区域的轮廓
        yellow_roi = region_of_interest(yellow_edge, color='yellow')
        white_roi = region_of_interest(white_edge, color='white')

        yellow_frame=frame.copy()
        white_frame=frame.copy()
        both_frame =frame.copy()

        #线段检测
        yellow_lines = detect_line(yellow_roi)
        yellow_lane = average_lines(frame, yellow_lines, direction='left')
        yellow_show = display_line(yellow_frame, yellow_lane)

        white_lines = detect_line(white_roi)
        white_lane = average_lines(frame, white_lines, direction='right')
        white_show = display_line(white_frame, white_lane, line_color=(255, 0, 0))
```

```
        both_frame=display_line(both_frame,yellow_lane)
        both_frame=display_line(both_frame,white_lines,line_color=(255,0,0))

        cv.imshow("yellow",yellow_show)
        cv.imshow("white",white_show)

        key=cv.waitKey(1)
        if key==ord("q"):
            break
        elif key==ord("c"):
            cv.imwrite("cut"+str(i)+".jpg",both_frame)
        # 计算转向角
        x_offset = 0
        y_offset = 0
        if len(yellow_lane) > 0 and len(white_lane) > 0:  # 检测到2条线
            _, _, left_x2, _ = yellow_lane[0][0]
            _, _, right_x2, _ = white_lane[0][0]
            mid = int(width / 2)
            x_offset = (left_x2 + right_x2) / 2 - mid
            y_offset = int(height / 2)
        elif len(yellow_lane) > 0 and len(yellow_lane[0]) == 1:  # 只检测到黄色行道
线
            x1, _, x2, _ = yellow_lane[0][0]
            x_offset = x2 - x1
            y_offset = int(height / 2)
        elif len(white_lane) > 0 and len(white_lane[0]) == 1:  # 只检测到白色行道线
            x1, _, x2, _ = white_lane[0][0]
            x_offset = x2 - x1
            y_offset = int(height / 2)
        else:  # 一条线都没检测到
            print('检测不到行道线，退出程序')
            break

        angle_to_mid_radian = math.atan(x_offset / y_offset)
        angle_to_mid_deg = int(angle_to_mid_radian * 180.0 / math.pi)
        steering_angle = angle_to_mid_deg / 45.0            #归一化
        print("angle:",steering_angle)
        action = np.array([steering_angle, 0.3-abs(steering_angle)/5])

        obv, reward, done, info = env.step(action)
        frame = cv.cvtColor(obv, cv.COLOR_RGB2BGR)

    # 运行完以后重置当前场景
    env.reset()
    cv.destroyAllWindows()

if __name__ == '__main__':
    main()
```

```
#tools.py
```

```python
import cv2 as cv
import numpy as np

def region_of_interest(edges, color:str):
    '''
    感兴趣区域提取
    '''
    height, width = edges.shape
    mask = np.zeros_like(edges)
    # 定义感兴趣区域掩码轮廓
    if color == 'yellow':
        polygon = np.array([[(0, height * 1 / 2),
                             (width * 1 / 2, height * 1 / 2),
                             (width * 1 / 2, height),
                             (0, height)]], np.int32)
    elif color == "white":
        polygon = np.array([[(width * 1 / 2, height * 1 / 2),
                             (width, height * 1 / 2),
                             (width, height),
                             (width * 1 / 2, height)]], np.int32)
    # 填充感兴趣区域掩码
    cv.fillPoly(mask, polygon, 255)
    # 提取感兴趣区域
    roi_edge = cv.bitwise_and(edges, mask)
    return roi_edge


def detect_line(edges):
    '''
    基于霍夫变换的直线检测
    '''
    rho = 1  # 距离精度：1像素
    angle = np.pi / 180  #角度精度：1度
    min_thr = 10  #最少投票数
    lines = cv.HoughLinesP(edges,
                           rho,
                           angle,
                           min_thr,
                           np.array([]),
                           minLineLength=8,
                           maxLineGap=8)
    return lines


def average_lines(frame, lines, direction:str):
    '''
    小线段聚类
    '''
    lane_lines = []
    if lines is None:
        print(direction + '没有检测到线段')
        return lane_lines

    height, width, _ = frame.shape
```

```
    fits = []
    for line in lines:
        for x1, y1, x2, y2 in line:
            if x1 == x2:
                continue
            # 计算拟合直线
            fit = np.polyfit((x1, x2), (y1, y2), 1)
            slope = fit[0]
            intercept = fit[1]
            if direction == 'left' and slope < 0:
                fits.append((slope, intercept))
            elif direction == 'right' and slope > 0:
                fits.append((slope, intercept))
    if len(fits) > 0:
        fit_average = np.average(fits, axis=0)
        lane_lines.append(make_points(frame, fit_average))
    return lane_lines


def make_points(frame, line):
    '''
    根据直线斜率和截距计算线段起始坐标
    '''
    height, width, _ = frame.shape
    slope, intercept = line
    y1 = height
    y2 = int(y1 * 1 / 2)
    x1 = max(-width, min(2 * width, int((y1 - intercept) / slope)))
    x2 = max(-width, min(2 * width, int((y2 - intercept) / slope)))
    return [[x1, y1, x2, y2]]


def display_line(frame, lines, line_color=(0, 0, 255), line_width=2):
    '''
    在原图上展示线段
    '''
    line_img = np.zeros_like(frame)
    if lines is not None:
        for line in lines:
            for x1, y1, x2, y2 in line:
                cv.line(frame, (x1, y1), (x2, y2), line_color, line_width)
    #line_img = cv.addWeighted(frame, 0.8, line_img, 0.2, 1)
    return frame
```

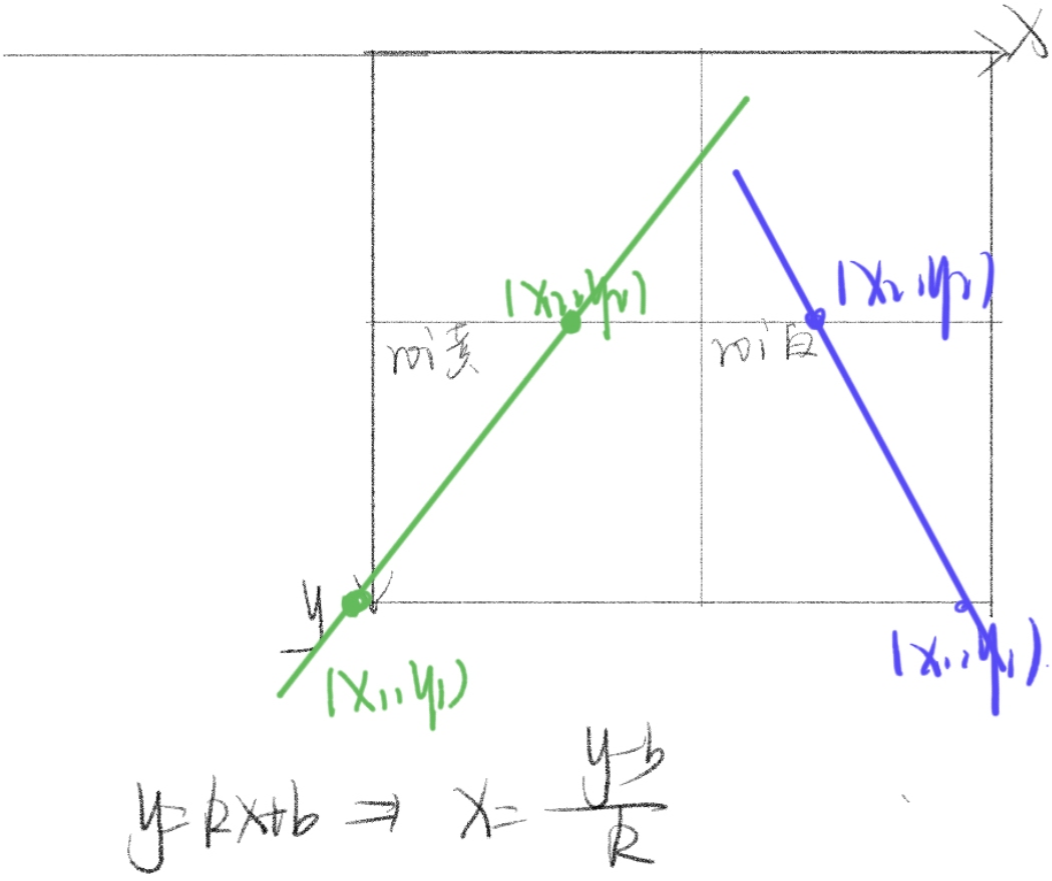
选择重点进行解析：

tools.py:

- 该函数涉及到线段起始坐标的计算

```
76  def make_points(frame, line):
77      '''
78      根据直线斜率和截距计算线段起始坐标
79      '''
80      height, width, _ = frame.shape
81      slope, intercept = line
82      y1 = height
83      y2 = int(y1 * 1 / 2)
84      x1 = max(-width, min(2 * width, int((y1 - intercept) / slope)))
85      x2 = max(-width, min(2 * width, int((y2 - intercept) / slope)))
86      return [[x1, y1, x2, y2]]
```

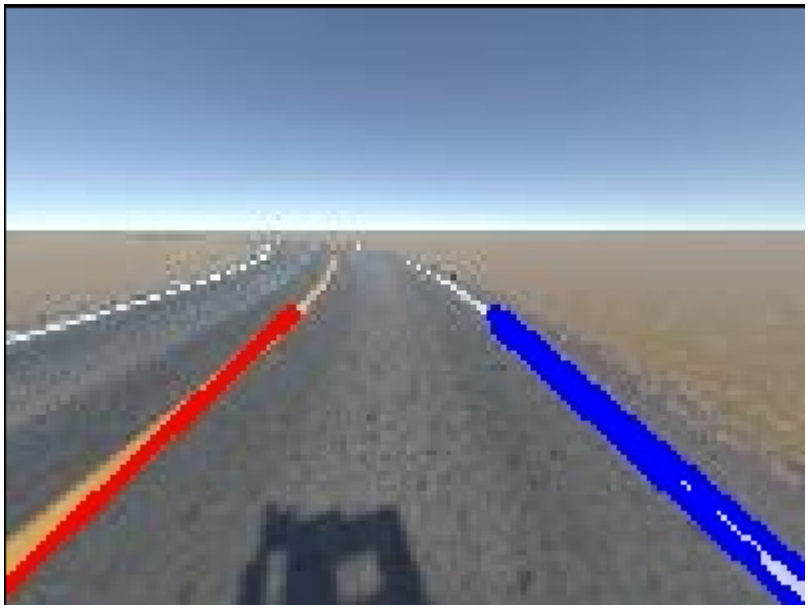
我们选取的roi为左下角和右上角，如图所示，根据霍夫变换的线段检测求出的斜率和截距求出图像中的点，用于将线显示在图像上。第84和85行中的-width和2*width是我们人为界定的边界，因为当x<-width或x>2*width时，线的斜率变化肉眼已经不明显了，简单期间直接界定两个阈值。



sim_car.py:

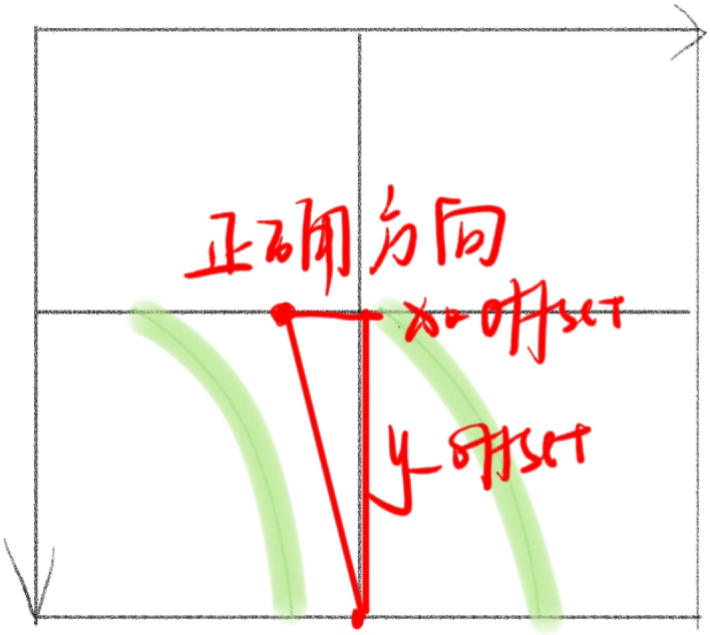
- 59、63行用于将蓝线与白线实时显示于窗口中，便于观察
- 在每一帧中进行捕捉，按“q"关闭窗口，按“c”捕捉当前摄像头捕捉到的图像（附带白黄线）。

```
71          key=cv.waitKey(1)
72          if key==ord("q"):
73              break
74          elif key==ord("c"):
75              cv.imwrite("cut"+str(i)+".jpg",both_frame)
```



- 计算转向偏移

```
79          if len(yellow_lane) > 0 and len(white_lane) > 0:  # 检测到2条线
80              _, _, left_x2, _ = yellow_lane[0][0]
81              _, _, right_x2, _ = white_lane[0][0]
82              mid = int(width / 2)
83              x_offset = (left_x2 + right_x2) / 2 - mid
84              y_offset = int(height / 2)
```



- 执行动作

确定角度与速度，拐弯角度越大速度越慢

```
97          angle_to_mid_radian = math.atan(x_offset / y_offset)
98          angle_to_mid_deg = int(angle_to_mid_radian * 180.0 / math.pi)
99          steering_angle = angle_to_mid_deg / 45.0            #归一化
100         print("angle:",steering_angle)
101         action = np.array([steering_angle, 0.3-abs(steering_angle)/5])
```