7. AI Autopilot

Note: This course is only applicable to the R2L car and Yahboom's autopilot map.

If you use other models or other maps, you need to develop and debug the code by yourself, and the code provided in this chapter cannot be used directly.

This section will be based on the content of the third section to explain how the entire automatic driving is realized. Therefore, before starting this course, it is necessary to calibrate the data required for the data.

7.1. Start up

Terminal input,

```
roslaunch yahboomcar_bringup bringup.launch
python yahboomcar_ws/src/yahboomcar_autodrive/scripts/line_detect.py
rosrun yahboomcar_autodrive test.py
```

The first command is to turn on the chassis control; the second is to turn on the line patrol function; the third command is to turn on the autopilot. After the program starts,

- When no traffic signs are recognized, the car will drive according to the recognized yellow lane lines;
- If a right turn is recognized, it is judged whether the value of the sign box reaches the set value,

  (1) If it is reached, it will stop for 5 seconds to calculate the turning data, then move forward for a short distance, and then turn right. After the right turn, enter the patrol lane again

  (2) If you have not reached it yet, continue to move forward. This step is to prevent turning right as soon as the right turn sign appears on the screen.
- If it is a red light, it will stop;
- If the parking space Parking_lotB is identified, start the movement of side parking, and exit the program after the movement ends.

7.2 Tracking functionline_ detect.py

Code path,

```
~/yahboomcar_ws/src/yahboomcar_autodrive/scripts/line_detect.py
```

This program has the following parts,

- Turn on the camera, get the image data, and pass the image into two main functions

  (1), frame, binary = line_detect.process(frame): This function mainly calculates the center coordinates of the yellow lane line and the angular velocity of the turn based on the image, and then publishes the magnitude of the angular velocity;

```
    def process(self, rgb_img):
        binary = []
        rgb_img = cv.resize(rgb_img, (640, 480))
        #self.pub_rgb.publish(self.bridge.cv2_to_imgmsg(rgb_img, "bgr8"))
        if os.path.exists(self.hsv_text): self.hsv_range =
read_HSV(self.hsv_text)
        if len(self.hsv_range) != 0:
            rgb_img, binary, self.circle = self.color.line_follow(rgb_img,
self.hsv_range)
        if len(self.circle) != 0:
            threading.Thread(target=self.execute, args=(self.circle[0],
self.circle[2])).start()
        return rgb_img, binary

    def execute(self, point_x, color_radius):
        if color_radius == 0: print("stop")
        else:
            twist = Twist()
            b = Bool()
            if self.flag == 1:
                center_x = 320-point_x
                #print(point_x)
                if point_x<320  :
                    point_x = point_x
                    [z_Pid, _] = self.PID_controller.update([(point_x -
30)*1.8/32,0])
                    if self.img_flip == True: self.go_angular_z = -z_Pid
                    else: self.go_angular_z = +z_Pid

                elif point_x>320 or point_x==320:
                    [z_Pid, _] = -self.PID_controller.update([(610-
point_x)*1.8/32, 0])#16
                    if self.img_flip == True: self.go_angular_z = -z_Pid
                    else: self.go_angular_z = +z_Pid

                #print("z_pid" ,z_Pid)
                #print("go_angular_z: ",self.go_angular_z)
                if abs(self.go_angular_z)<0.1:
                    if self.img_flip == True:self.go_angular_z = -
self.go_angular_z
                    else:
                        self.go_angular_z = +self.go_angular_z
                    #self.go_angular_z = 0


                self.go_z.data = self.go_angular_z
                self.pub_go_z.publish(self.go_z)


            elif self.flag == 2:
                [z_Pid, _] = self.PID_controller.update([(point_x -
320)*1.0/32, 0])
                print("zpid: ")
                if self.img_flip == True: self.trun_angular_z = -z_Pid #-
z_Pid
```

```
                else: self.trun_angular_z = +z_Pid
                if abs(twist.angular.z)>1:
                    self.trun_angular_z = 1.0
                else:
                    self.trun_angular_z = abs(self.trun_angular_z)
                self.turn_z = self.trun_angular_z
                self.pub_turn_z.publish(self.turn_z)
```

(2), line_detect.detect(detect_img): This function is mainly for sign recognition, importing the model, and then publishing the recognized results;

```
    def detect(self,detect_img):
        detect_img, result_boxes, result_scores, result_classid =
self.yolov5_wrapper.infer(detect_img)
    # Draw rectangles and labels on the original image
        for j in range(len(result_boxes)):
            box = result_boxes[j]
            self.yolov5_wrapper.plot_one_box(
                box,
                detect_img,
                label="{}:{:.2f}".format(
                    self.yolov5_wrapper.categories[int(result_classid[j])],
                    result_scores[j]
                ),
            )
            self.target.frame_id =
self.yolov5_wrapper.categories[int(result_classid[j])]
            self.target.stamp = rospy.Time.now()
            self.target.scores = result_scores[j]
            # x1, y1, x2, y2
            self.target.ptx = box[0]
            self.target.pty = box[1]
            self.target.distw = box[2] - box[0]
            self.target.disth = box[3] - box[1]
            self.target.centerx = (box[2] - box[0]) / 2
            self.target.centery = (box[3] - box[1]) / 2
            self.target_array.data.append(self.target)
        self.cTime = time.time()
        fps = 1 / (self.cTime - self.pTime)
        self.pTime = self.cTime
        text = "FPS : " + str(int(fps))
        cv.putText(detect_img, text, (20, 30), cv.FONT_HERSHEY_SIMPLEX, 0.9,
(0, 0, 255), 1)
        self.pub_msg.publish(self.target_array)
        #self.pub_imgMsg(frame)
        #print("target.centerx: ",self.target.centerx)
        if self.target.frame_id =="Turn_right":
            print("target.centery: ",self.target.centery)
        action = cv.waitKey(1) & 0xFF
        #self.img = self.detect(self.img)
        cv.imshow("yolov5_frame", detect_img)
        cv.waitKey(1)
```

Let's see what key topic messages are defined for subscription and publication,

```
#Line patrol part
self.pub_go_z = rospy.Publisher("/go_z",Float32,queue_size=1)
self.pub_turn_z = rospy.Publisher("/turn_z",Float32,queue_size=1)
self.subMove_Status =
rospy.Subscriber('move_flag',Int32,self.Move_Statuscallback)
#yolov5部分
self.pub_msg = rospy.Publisher('DetectMsg', TargetArray, queue_size=1
```

self.pub_go_z： Publish the angular velocity of the turn for the line-following turn

self.pub_turn_z： Publish the angular velocity of the turn when the right turn sign is recognized

self.subMove_Status： Subscribe to whether the car is in the line patrol state or the recognition state at this time. When publishing the turning speed, choose to publish the speed of self.pub_go_z or self.pub_turn_z according to the value here.

self.pub_msg：Publish the information of yolov5 recognition results, including recognition results, center coordinates, scores, etc.

Two important variables:

- self.flag：When the topic data of move_flag is received, it will be changed in the Move_Statuscallback, and then used in the execute function. When it is 1, it means that it is in the line patrol state at this time, and the turning speed of the line patrol needs to be released;When it is 2, it means that the right turn is recognized, and the turning speed of the right turn needs to be released.
- point_x：The center coordinate of the yellow lane line. According to the value of this coordinate, we can calculate the size of the angular velocity we need to adjust. For example, we need to keep the car driving a little to the left. After we standardize the picture as 640*480,that is, when the maximum coordinates of the xy pixel are reached, we can estimate that if the vehicle is to be turned to the left, the offset between point_x and 30 needs to be calculated.However, since the yellow lane line on the right may also be recognized, it is also necessary to consider the situation on the right, that is, the difference between point_x and 610. After calculating the offset, pass in the pid adjustment parameter to calculate the specific angular velocity, and then publish it. See the following code for details,

  ```
  if self.flag == 1:
      center_x = 320-point_x
      #print(point_x)
      if point_x<320  :
      point_x = point_x
      [z_Pid, _] = self.PID_controller.update([(point_x - 30)*1.8/32,0])
      if self.img_flip == True: self.go_angular_z = -z_Pid
      else: self.go_angular_z = +z_Pid

      elif point_x>320 or point_x==320:
      [z_Pid, _] = -self.PID_controller.update([(610-point_x)*1.8/32, 0])#16
      if self.img_flip == True: self.go_angular_z = -z_Pid
      else: self.go_angular_z = +z_Pid
  ```

Therefore, if you need to keep the car in the middle of the lane, you need to modify the two offsets of 30 and 610 here.

7.3、Test.py

After this program is started, the automatic driving will start. Let's take a look at the content of the ros part first.

```
self.sub_traffic_sign = rospy.Subscriber('DetectMsg', TargetArray, self.excute,
queue_size=1)
self.pubFlag = rospy.Publisher('move_flag',Int32,queue_size=1)
```

（1）、self.sub_traffic_sign： This is to subscribe to the identification information published by line_detect.py, and the callback function is excute.

```
    def excute(self,msg):
        #print("----")
        '''if msg.data != 1:
            self.turn_right.process()
        elif msg.data ==1 or msg.data ==0 :
            self.turn_right.Go()'''
        #print(msg.data[1].frame_id)
        #print(msg.data[1].frame_id)
        #print(msg.data)
        if msg.data == [] or msg.data[0].frame_id == "Green_light":
            print("Go Now")
            self.move_status.data = 1
            self.pubFlag.publish(self.move_status)
            self.turn_right.Go(self.turn_right.go_angular_z)
            self.turn_right.finish_flag = 0
        elif msg.data[0].frame_id == "Turn_right":
            self.move_status.data = 2
            self.pubFlag.publish(self.move_status)
            if msg.data[0].centery>42 and self.turn_right.finish_flag == 0 : #If
the position height of the camera is changed, the y value here needs to be
calibrated
                self.dist = msg.data[0].centerx * 0.012
                print("self.dist: ",self.dist)
                #print("Turn Now")
                print("self.trun_angular_z: ",self.turn_right.trun_angular_z)
                self.turn_right.Stop()
                sleep(5)

 self.turn_right.process(self.dist,self.turn_right.trun_angular_z)
            else:
                print("Adjust")
                self.move_status.data = 1
                self.pubFlag.publish(self.move_status)
                #self.turn_right.finish_flag = 0
                self.pubFlag.publish(self.move_status)
                self.turn_right.Go(self.turn_right.go_angular_z)
        elif msg.data[0].frame_id == "Parking_lotB":
            print("reversing")
            os.system("rosbag play 2022-11-01-21-29-01.bag")
```

```
            os._exit(0)
        elif msg.data[0].frame_id == "Red_light":
            self.turn_right.Stop()
```

By judging the value of msg.data[0].frame_id to judge the movement to be performed, for example, if Turn_right is recognized, the value of self.move_status.data will be published. Let line_detect.py judge that what is to be released at this time is the angular velocity of the right turn, and then judge whether msg.data[0].centery, that is, the y value of the center coordinate of the recognition frame, reaches the critical value we need to execute . What I set here is that when it is greater than 42, it will start to turn right. If it does not reach the critical value, it will go forward until it reaches the critical value. msg.data == [] means that The flag is not recognized, that is, it enters the line patrol mode.

（2）、self.pubFlag： Publish the status of the car, and send the topic data to line_detect.py according to the recognition result.

Expansion: The source code only writes to judge 4 kinds of recognition results, which can be based on your own needs. For example, it is possible to recognize the deceleration of the sidewalk and the recognition of the Stop sign to stop. Just add or replace it.

A custom library: Turn_Right, the source code path is,

```
~/yahboomcar_ws/src/yahboomcar_autodrive/scripts/Turn_Right.py
```

This library defines an object YahboomCarPatrol. We imported this library in test.py, so we instantiated this type of object.

```
from Turn_Right import *
self.turn_right = YahboomCarPatrol()
```

From this, we can refer to all data and methods of this object. When we need to track the line, we call the Go method inside. Let's look at the specific content of this Go function.

```
def Go(self,go_z):
    #print("Go")
    if self.Joy_active == True:
        return
        self.move_cmd.linear.x = 0.20
        self.move_cmd.linear.y = 0.0
        self.move_cmd.angular.z = go_z
        self.pub_cmdVel.publish(self.move_cmd)
```

It's very simple, just assign a value, and then publish the cmd_vel topic data. Other functions, such as self.turn_right.Stop() for parking at a red light and self.turn_right.process for turning right, can be found here in detail. All in all, the function in Turn_Right is to send out the speed data at the end. After the chassis subscribes to the topic data, it will communicate with the underlying ros expansion board to drive the car.

7.4. Process summary

The entire autonomous driving framework is subdivided into three parts or three functions, which are line_detect.py, test.py and Turn_Right.py according to the program flow.

First, line_detect.py publishes the recognition result to test.py, and then test.py sends the running status of the car back to line_detect.py; line_detect.py calculates and publishes it to Turn_Right.py according to the image and status value, and then Turn_Right.py. py publishes the speed topic cmd_vel to the driver.

The following is the approximate main content data flow diagram.