中山大學
SUN YAT-SEN UNIVERSITY

《视觉巡线小车比赛》

技术报告

（机器人导论 课程设计）

学 院 名 称 ：	计算机学院
专业（班级）：	计算机科学与技术
队 伍 名 称 ：	阳光尖尾雨燕队
学 生 姓 名 ：	吴晓淳　　王若琪　　杨净涵
学　　号 ：	18340174 18340166 18340193
时　　间 ：	2020　年　12　月　26　日

目录

1 实验内容

学会 V-REP 的基本使用方法，包括基础功能，基本参数设置和路径设计与编辑；完成小车视觉巡线任务，利用小车左偏、右偏和直行时，图像某些特征的区别进行反馈控制。

2 实验原理

2.1 图像处理原理

1. 高斯模糊

高斯模糊是指一个图像与二维高斯分布的概率密度函数做卷积。高斯分布即正态分布。由于图像是离散的，所以高斯分布的概率密度函数需要先做采样才能与图像进行卷积。这一过程可以通过 python 中的 OpenCV 中自带的 API，cv2.GaussianBlur() 来完成。

2. 二值化

图像二值化（ Image Binarization）就是将图像上的像素点的灰度值设置为 0 或 255，也就是将整个图像呈现出明显的黑白效果的过程。在数字图像处理中，二值图像占有非常重要的地位，图像的二值化使图像中数据量大为减少，从而能凸显出目标的轮廓。这一过程也可以通过 python 中的 OpenCV 中自带的图像处理 API，cv2.inRange() 来完成。

3. 开、闭运算

开运算和闭运算是比较常用的形态学运算函数。开运算是先腐蚀后膨胀，闭运算是先膨胀后腐蚀，都可以去除噪声

（1）腐蚀：简单的理解，腐蚀的作用就是让暗的区域变大，

（2）膨胀：简单的理解，膨胀的作用就是让亮的区域变大

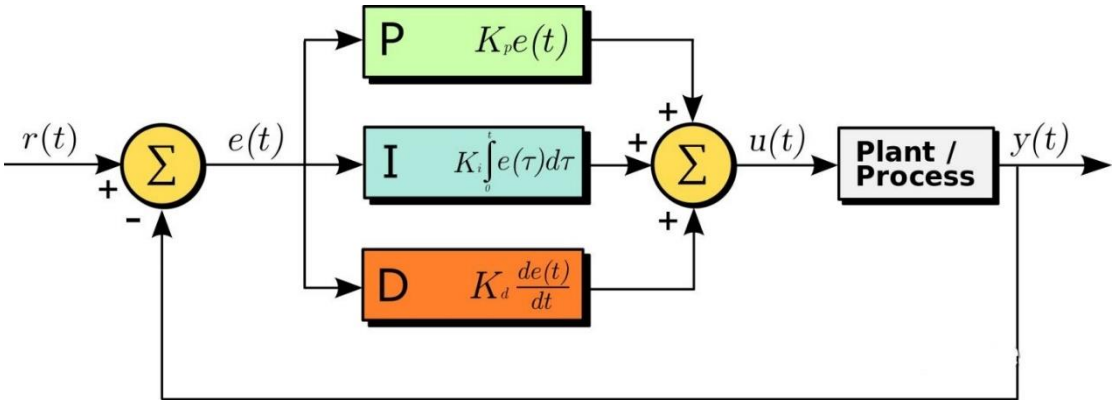
通过开、闭运算，能够去除二值化图像中的干扰点。

2.2 控制算法 PID

1. 基本原理

控制系统根据是否有反馈可分为开环系统和闭环系统，在闭环系统的控制中，由于 PID 算法可以自动对控制系统进行准确且迅速的校正，因此被广泛应用于工业控制系统，PID 的三个部分分别为 P：比例环节；I：积分环节；D：微分环节。

PID 控制器的系统框图如下所示：



输出$u(t)$与输入$e(t)$之间的关系为：

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

其中，K_p是比例增益，K_i是积分增益，K_d是微分增益。积分是误差在一定时间内的和，因此，积分环节满足以下公式：

$$K_i \int_0^t e(\tau)d\tau$$

微分则是误差变化曲线某处的导数，即某一点的斜率，因此，微分

环节满足以下公式：

$$K_d \frac{de(t)}{dt}$$

当误差变化过快时，微分环节会输出较大的负数，抑制输出继续上升，从而抑制过冲。

2. 离散化

在数字系统中进行PID算法控制时，需要对上述算法进行离散化。假设系统采样时间为Δt，则将误差$e(t)$序列化得到$(e_0, e_1, e_2, \cdots, e_{n-1}, e_n)$，在实验中的具体实现为：

```
1.    error = self.SetPoint - feedback      #设定目标值与反馈值之间的偏差
```

将输出$u(t)$序列化得到$(u_0, u_1, u_2, \cdots, u_{n-1}, u_n)$，将比例项$K_p e(t)$离散化得到$K_p e_k$，在实验中的具体实现为：

```
2.    self.P_term = self.Kp * error
```

将积分项$K_i \int_0^t e(\tau)d\tau$离散化得到$K_i \sum_{i=1}^{k} e_i \Delta t$，在实验中的具体实现为：

```
3.    self.I_term += error * delta_time              # 更新积分项
```

将微分项$K_d \frac{de(t)}{dt}$离散化得到$K_d \frac{e_k - e_{k-1}}{\Delta t}$，在实验中的具体实现为：

```
4.    self.D_term = delta_error / delta_time          # 更新微分项
```

最终得到输出u_k与输入e_k之间的关系为：

$$u_k = K_p e_k + K_i \sum_{i=1}^{k} e_i \Delta t + K_d \frac{e_k - e_{k-1}}{\Delta t}$$

在实验中的具体实现为：

```
5. self.output=self.P_term+(self.Ki * self.I_term) + (self.Kd * self.D_
   term)
```

3. 伪代码

```
Function PID(setpoint, measured_value, kp, ki, kd, dt)
/*输入:
  setpoint:  目标设定值
  measured_value: 反馈值
  kp: 比例增益
  ki: 积分增益
  kd: 微分增益
  dt: 采样时间
*/
1. previous_error ← 0;              /* 上一个采样时刻的偏差*/
2. integral ← 0;                    /* 积分项 */
3. while (true)
4.       error ← setpoint − measured_value;          /* 偏差 */
5.       integral ← integral + error ∗ dt;           /* 积分项累加和 */
6.       derivative ← (error − previous_error)/ dt;  /* 微分 */
7.       output ← kp ∗ error + ki ∗ integral + kd ∗ derivative; /* PID 的输出 */
8.    previous_error ← error;/* 保存当前偏差为下一次采样时刻参考的历史偏差 */
```

3 实验过程

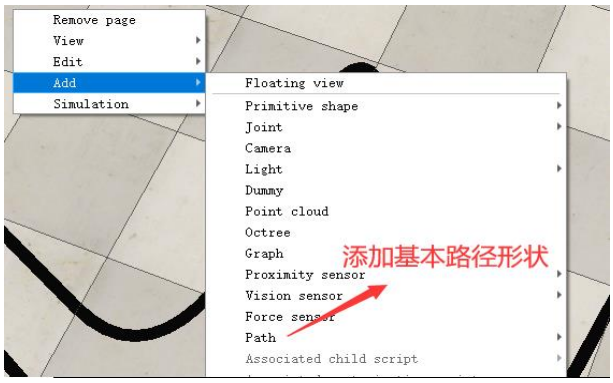
3.1 V-REP 入门

V-REP 是一款动力学仿真软件，主要定位于机器人仿真建模领域，可以利用内嵌脚本、ROS 节点、远程 API 客户端等实现分布式的控制结构，控制器可以采用 C/C++,Python 等语言实现，可以用脚本来控制小车运动，在本次实验中，用到 V-REP 的基本功能有：视角移动及视角切换、模型添加、路径设计、机器人移动与旋转等等。

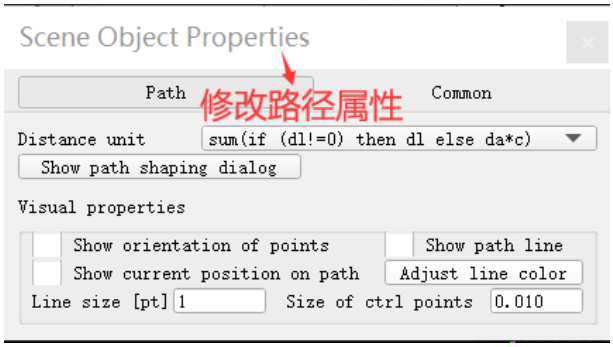
3.2 路径设计

路径设计主要用到【Add->Path->Segment type/Circle type】来

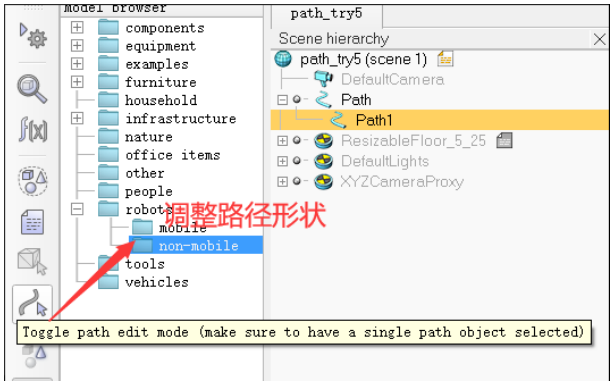
选择基本路径形状是线段还是圆圈。



利用【Scene Object Properties】来修改路径属性：



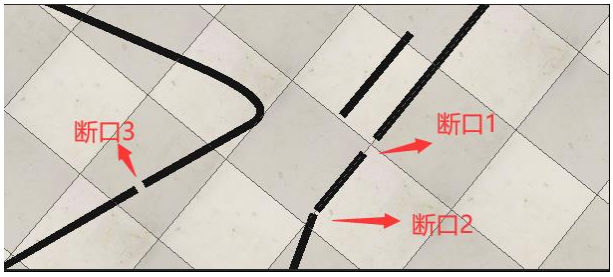
利用【Path Edit Mode】来调整路径形状，



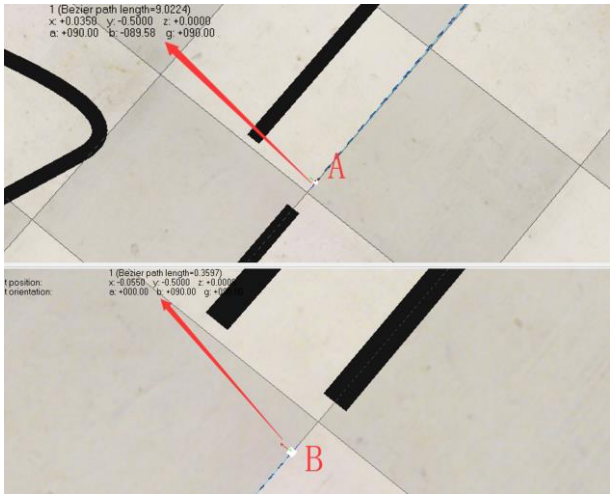
具体路径设计如下：

1. 断口设计

路径中断口需要满足 0.1m 以内，设计如下：



断口 1 的间距最大，相隔的两点坐标为 A(+0.0350,-0.5000,0.0000) 和 B(-0.0550,-0.5000,0.0000)，距离为 0.0350+0.0550=0.09<0.1，满足要求。

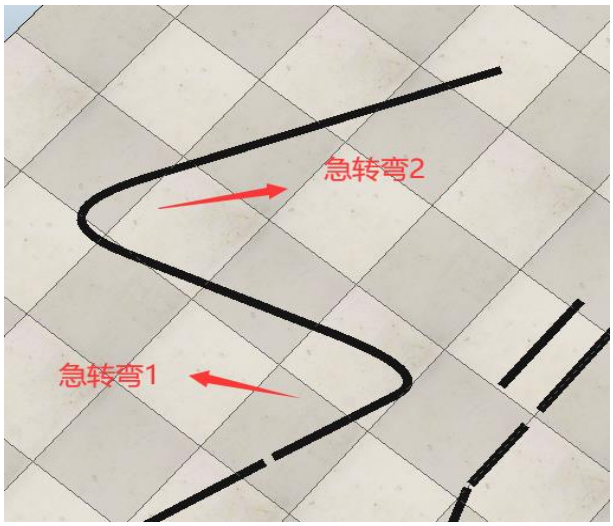


2. 道路交叉

选择基本路径形状为 Segment type，然后调整路径如下：

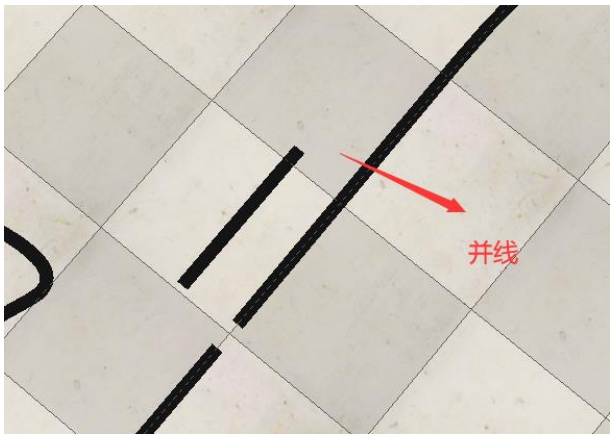
3. 急转弯

转弯角度要求小于 7pi/8，设计如下

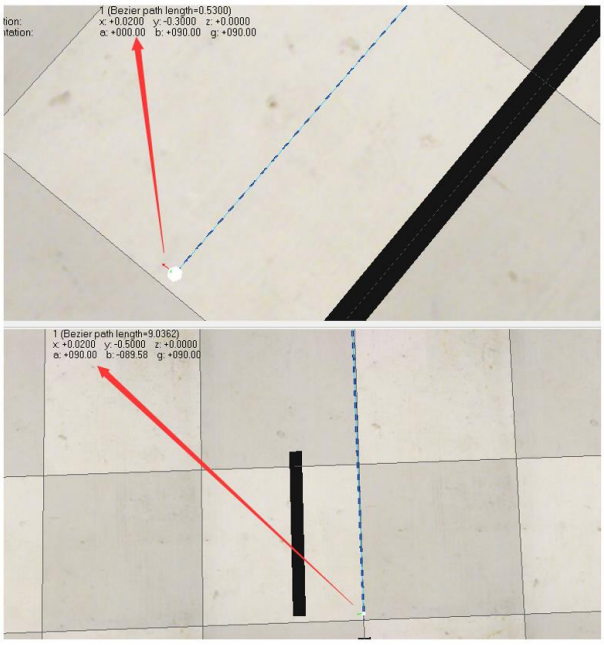


4. 并线

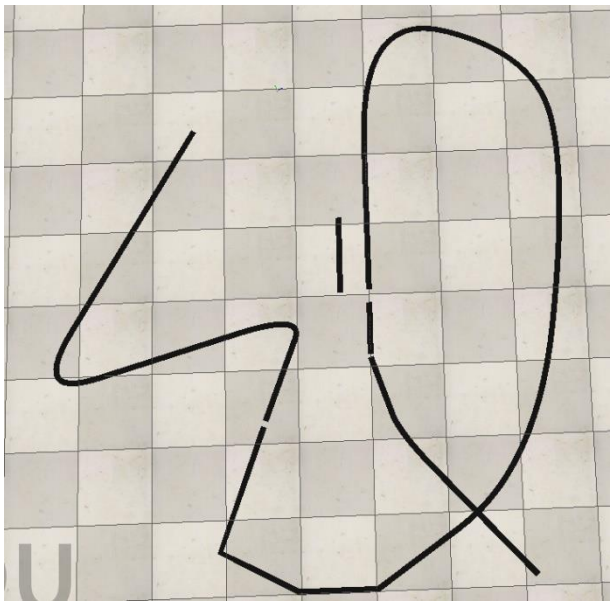
并线距离要求大于 0.15m，设计如下：



相隔的两点坐标为 A(+0.0200,-0.3000,0.0000)和 B(+0.0200,-0.5000,0.0000)，距离为 0.5000-0.3000=0.2000>0.15，满足要求。



完整路径设计图为：



3.3 V-REP 远程 API 控制（Python）

在 V-REP 外部控制中，通常称远程 API 端为 Client 端，称 V-REP 为 Server 端。

1. API 端配置

在 Python 项目文件夹中添加接口文件和链接库，这些文件可以在 V-REP 的安装文件夹中找到，并重命名为 vrep.py 和 vrepConst.py，同时为了方便算法的实现，将文件中的 Sim 接口改名为 vrep，其中，接口文件的路径为：

```
C:\ProgramFiles\CoppeliaRobotics\CoppeliaSimEdu\programming\remoteApiBindings\python\python
```

链接库的路径为：

```
C:\ProgramFiles\CoppeliaRobotics\CoppeliaSimEdu\programming\remoteApiBindings\lib\lib\Windows
```

参照 readMe.txt 文件，将下列文件复制到 Python 项目文件夹下：

```
vrep.py
vrepConst.py
remoteApi.dll
```

2. V-REP 端通信配置

点击模型 Pioneer_p3dx 后面的脚本图标，打开脚本，脚本中有四个子函数，分别为：

```
function sysCall_init()
function sysCall_actuation()
function sysCall_sensing()
function sysCall_cleanup()
```

以上函数分别负责初始化、执行器、传感器和清除等操作的配置。通过在 function sysCall_init()中添加一行代码：

```
repeat until (simRemoteApi.start(19999,1300,false,true)~=-1)
```

从而实现 API 与 V-REP 的临时通信，但只有在 V-REP 先运行的情况下才能实现临时通信。

3.4 图像处理

小车需要根据由传感器传输的图片来确定移动方向，我们小组的思路大致为，先将原始图片进行一系列去噪转化等工作，再在处理后的图片中找出所有的轮廓，接下来从所有轮廓中找到最大的轮廓，并认为这个轮廓就是路径的轮廓。之后，找出路径轮廓的最小外接矩形，并认为这个外接矩形的中心点就是 PID 控制器的输入，整个图像的中心是 PID 控制器的目标点。通过 PID 得出转弯的幅度后，再根据当前所处的是直线还是曲线来判断速度，如果是直线，那么就用较快的速度，如果是曲线，那么就用较慢的速度，下面结合代码进行具体分析：

(1) 使用 V-REP 的 API simxGetVisionSensorImage() 获取传感器的图像：

```
res, resolution, image = vrep.simxGetVisionSensorImage(clientID, camera, 0, vrep.simx_opmode_streaming)
```

(2) 将获得的图像先转换成 cv2 可处理的标准格式，并将图片用 cv2 的 resize 功能缩小，从而减小之后对图像处理的计算量，从而提高之后的计算速度：

```
# 将图像变成标准格式
img = np.array(image, dtype=np.uint8)
img.resize([resolution[1], resolution[0], 3])
# 将图像缩小，提高之后对图像的处理速度
img=cv2.resize(img,(128,128))
```

(3) 调整图像方向，使得图像处于一种便于观察的直观方向，从而便于之后的处理和分析：

```
    # 调整图像方向
img=cv2.flip(img,1)
img=cv2.flip(img,0)
```

```
4. img=cv2.flip(img,1)
```

(4) 从图像中截取需要的部分，避免平台边缘的干扰。由于小车的视觉传感器总是接收到模型环境中平台边缘的图像，非常影响对路径的判断，所以需要将图像截取有效的一小部分，避免视野过大造成干扰：

```
1. # 截取有效视野，排除干扰
2. img = img[ (img.shape[0]//4)*3 : (img.shape[0]//10)*9 , img.shape[1]
   //4 : (img.shape[1]//4)*3]
```

(5) 高斯模糊：先将图片进行高斯模糊处理，高斯模糊是 2D 卷积的一种，常用于给图像提取边缘的预处理阶段。本次实验中用到了 5*5 的高斯内核，标准差取 0。

```
1. orgframe_gas = cv2.GaussianBlur(orgimage, (3, 3), 0)  # 高斯模糊，去
   噪
```

处理前后对比效果如下图：

（前）

（后）

(6) 将图片转化到 LAB 空间中：

```
1. img2 = cv2.cvtColor(img1, cv2.COLOR_BGR2LAB)  # 将图像转换到 LAB 空间
```

(7) 根据 hsv 值对图片进行二值化，即为让整个图像呈现出明显的黑白两色对比效果，代码如下：

```
1. img3 = cv2.inRange(img2, color_range['black'][0],
2.                       color_range['black'][1])  # 根据 hsv 值对图片进行二值化
```

二值化前后对比效果如下：

（前）

（后）

（8） 进行开、闭运算，开运算和闭运算都是比较常用的形态学运算函数。其中开运算是先腐蚀后膨胀，闭运算是先膨胀后腐蚀，都可以达到去除噪声的效果。代码如下：

```
1. img4 = cv2.morphologyEx(img3, cv2.MORPH_OPEN, np.ones((3, 3), np.uint8)) # 开运算
2. img5 = cv2.morphologyEx(img4, cv2.MORPH_CLOSE, np.ones((3, 3), np.uint8)) # 闭运算
```

开闭运算前后的图片效果对比如下图：

（前）

（后）

3.5 小车运动方向的确定

对经过一系列处理之后的图像进行信息提取，确定两个关键点，并通过 PID 控制方法，计算出接下来的控制方向。

（1） 先找出图片中的所有轮廓：

```
cnts = cv2.findContours(final, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_TC89_L1)[-2]  # 找出所有轮廓
```

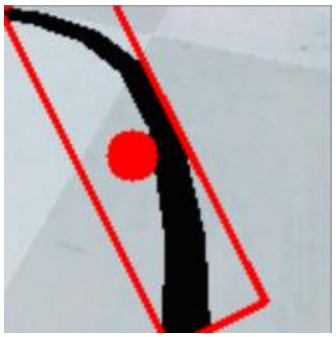
(2) 再从这些找到的轮廓中选出一个面积是最大的，并将其判断为路径的轮廓：

```
cnt_largest = getAreaMaxContour(cnts)  # 找到最大面积的轮廓
```

(3) 找到最小外接矩形，并绘制出最小外接矩形以及它的中心点：

```
if cnt_largest is not None:
    rect = cv2.minAreaRect(cnt_largest)  # 最小外接矩形
    box = np.int0(cv2.boxPoints(rect))  # 最小外接矩形的四个顶点
    p1_x, p1_y = box[0, 0], box[0, 1]
    p3_x, p3_y = box[2, 0], box[2, 1]
    # cv2.drawContours(img, [box], -1, (0, 0, 255, 255), 2)  # 画出四个点组成的矩形
    center_x, center_y = (p1_x + p3_x) / 2, (p1_y + p3_y) / 2  # 中心点
    # cv2.circle(img, (int(center_x), int(center_y)), 10, (0, 0, 255), -1)  # 画出中心点
```

绘制效果如下图：



3.6 小车运动速度的确定

在直线时，小车速度可以适当加快，但是在曲线转弯时，小车速度必须放慢。在本实验中，我们采用了一种简洁有效的速度控制策略，就是用外接矩形的中心点和整个图像的中心点之间的距离大小来确定当前所处的位置是直线还是曲线，如果距离小于 5，就判断为直线，

可以用较快的速度前进，如果距离大于等于 5，那么判断为曲线，用较小的速度转弯：

```
# 确定速度，直道快，弯道慢
abs(x_pid.SetPoint - center_x) < 5:
 speed = 7.5 # 直道
else:
```

3.7 计时方法

先将小车移动到起点处：选中小车，然后【ctrl】选中 start_point，打开位移控制，对其 x 轴，y 轴以及偏转量，将 start_point 移动到路径上自定义的起始位置，x 轴为小车初始朝向，同理将 end_point 移动到路径上自定义的结束位置，最终传感器会侦测到小车经过，将记录时间输出到状态栏。

3.8 仿真步骤

1. Python 端

首先引入头文件 vrep：

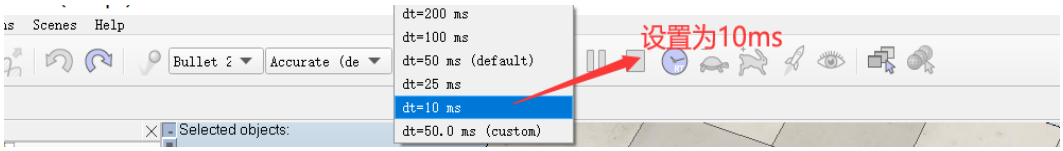
```
import vrep
```

然后通过 vrep.simxFinish(-1)关闭之前的链接，并使用 clientID 连接 V-REP。

```
vrep.simxFinish(-1)
clientID = -1
while(clientID == -1):
    clientID = vrep.simxStart('127.0.0.1', 19999, True, True, 5000, 5)
```

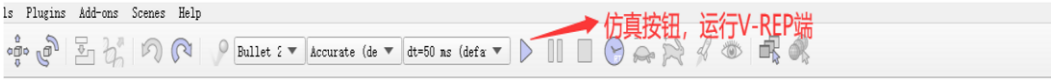
2. V-REP 端

设置仿真时间 dt=10ms：



在终端输入命令，运行 python 文件：

运行 V-REP 端：



4 实验结果

以视频文件中的两次实验数据为例，在助教提供的标准路径上小车完成巡线任务的仿真时间为：18.81s；在小组设计路径上小车完成巡线任务的仿真时间为：34.83s。

5 小组分工

王若琪：加减速算法的设计与实现、图像处理模块的设计与实现、调参、部分实验报告的编写。

杨净涵：仿真软件的调试，比赛路径的设计和测试，部分实验报告的编写，终版实验报告整合。

吴晓淳：PID 控制算法的设计与实现、参数调试以及部分实验报告的撰写。

6 实验总结

本次实验是我们第一次从无到有完成一个完整的仿真实验项目，过程中遇到了太多的困难和问题，都被我们一一解决，最终达到了不

错的实验效果。在一次次解决问题的过程中，我们收获了很多，现总结遇到的问题与解决方案如下：

(1) 在一开始时，我们最先遇到的问题就是用 python 来远程控制小车。经过搜索资料和文档，才慢慢掌握了 vrep 脚本内置语言和 python 中的 vrep 接口的用法。

(2) 接下来我们需要考虑如何根据传感器传来的图片来判断小车的走向，这也是一个困扰我们很久的问题，我们曾经考虑过检测路径边缘，然后取中线，但是中线是一条曲线，很难确定以曲线的哪一个点为 PID 控制器的输入点。后来经过认真搜索资料，发现可以用一个外接矩形框住路径，然后取外接矩形的中点为 PID 控制器的输入点，整个图片的中点为目标点，就能够满足寻迹的要求了。

(3) 经过这些，还有一个困难就是传感器的视野太大了，总是看到平台以外的黑暗部分，这些黑暗部分会让外接矩形框错位置，导致计算出错误的调整方向，最后导致偏离轨迹。这个问题比较容易解决，只需要在 python 代码部分截取需要的一部分视野即可，保证视觉传感器看不到外界部分就可以了。

(4) 还有一个困扰我们最久的问题，就是在其他所有问题都解决之后，我们的小车运行非常不稳定。一开始我们以为是 PID 控制器的参数没有调整好，于是就大量查阅学习有关 PID 控制器调参方法的资料，在没日没夜地调参了 2 个晚上之后，我们发现此时参数大小对小车的影响并不明显，那么问题应该不是出在调参上。我们又输出了仿真过程中很多中间变量来寻找问题，发现有的中间变量的值非常不合

理，甚至称得上离奇，这时我们灵机一动，想到问题可能是因为计算机的计算速度跟不上仿真的速度，导致部分结果没算完就输出了，所以调小了仿真软件中的 dt，变为 10ms，发现小车终于能够顺利前进和转弯了。

(5) 最后，我们考虑继续优化算法，因为要考虑在曲线上运行的稳定性，所以小车速度不能过大，之前完成的小车速度是恒定的，导致在直线上行驶时也很慢，不是很智能，所以需要想一个解决方法。我们想到可以通过对比路径外接矩形的中心点和整幅图像的中心点的距离大小来判断当前所处的位置是直线还是曲线，然后在直线时赋予较大的速度，在曲线时赋予较小的速度，将小车自动寻径的平均速度提升了 20%左右。

总而言之，本次实验非常考验我们发现问题、分析问题、解决问题的能力，也很考验整个团队的合作。经过本次实验，我们不仅学到了有关图像处理、PID 控制等相关知识，更锻炼并提升了探索问题的思维能力，还锻炼了团队组织协作的能力，受益匪浅。

7 附录

7.1 代码文件：code.py

```
1. import vrep
2. import numpy as np
3. import cv2
4. import time
5. import math
6. import threading
7. from PIL import Image
```

```
8.
9.  vrep.simxFinish(-1)
10. clientID = -1
11. while(clientID == -1):
12.     clientID = vrep.simxStart('127.0.0.1', 19999, True, True, 5000,
    5)  # Connect to V-REP
13.
14. color_range = {
15.                 'black': [(0, 0, 0), (62, 255, 250)],
16.                 'white': [(217, 0, 0), (255, 255, 250)],
17.               }
18.
19. class PID:
20.     def __init__(self, P=0.2, I=0.0, D=0.0, sample_time=0.00):
21.         self.Kp = P                                 # 比例增益 Kp
22.         self.Ki = I                                 # 积分增益 Ki
23.         self.Kd = D                                 # 微分增益 Kd
24.         self.sample_time = sample_time              # 采样间隔
25.         self.current_time = time.time()             # 当前时间
26.         self.previous_time = self.current_time  # 历史采样时间点
27.         self.clear()
28.
29.     def clear(self):
30.         self.P_term = 0.0
31.         self.I_term = 0.0             # 积分项
32.         self.D_term = 0.0             # 微分项
33.         self.previous_error = 0.0   # 历史偏差量
34.         self.SetPoint = 0.0         # 设定目标值
35.         self.windup_guard = 20.0
36.         self.output = 0.0           # 输出
37.
38.     def update(self, feedback):
39.         '''''
40.         利用 PID 算法根据设定的目标值和输入的反馈值更新输出调整小车的速度
41.         :param feedback: 闭环系统的反馈值
42.         '''
43.         error = self.SetPoint - feedback                        # 设
    定目标值与反馈值之间的偏差
44.
45.         self.current_time = time.time()                         # 当
    前采样时间
46.
47.         delta_time = self.current_time - self.previous_time     # 时
    间变化量
```

```
48.         delta_error = error - self.previous_error                # 偏差变化量
49.
50.         if (delta_time >= self.sample_time):
51.             self.P_term = self.Kp * error
52.             self.I_term += error * delta_time                    # 更新积分项
53.             self.D_term = 0.0
54.             if delta_time > 0:
55.                 self.D_term = delta_error / delta_time           # 更新微分项
56.
57.             if (self.I_term < -self.windup_guard):
58.                 self.I_term = -self.windup_guard
59.             elif (self.I_term > self.windup_guard):
60.                 self.I_term = self.windup_guard
61.
62.             self.previous_time = self.current_time               # 更新历史采样时间点
63.             self.previous_error = error                          # 更新历史偏差
64.             self.output = self.P_term + (self.Ki * self.I_term) + (self.Kd * self.D_term)
65.
66.
67. #检测并返回最大面积
68. def getAreaMaxContour(contours, area=1):
69.         max_area = 0
70.         ans = None
71.
72.         for c in contours :
73.             temp = math.fabs(cv2.contourArea(c))
74.             if temp > max_area :
75.                 max_area = temp
76.                 if temp > area:
77.                     ans = c
78.         return ans
79.
80. get_line = False
81. center_x = 0
82. speed = 0
83. left_speed, right_speed = 0, 0
84.
85. x_pid = PID(P=0.5, I=0.3, D=0.5)
```

```
86.
87. def Tracing(img0):
88.     global get_line, center_x, speed
89.     global right_speed, left_speed
90.     # cv2.imshow('img0', img0)
91.     img1 = cv2.GaussianBlur(img0, (3, 3), 0)  # 高斯模糊，去噪
92.     # cv2.imshow('img1', img1)
93.     img2 = cv2.cvtColor(img1, cv2.COLOR_BGR2LAB)  # 将图像转换到 LAB
    空间
94.     img3 = cv2.inRange(img2, color_range['black'][0],
95.                                  color_range['black'][1])  # 根据 hsv
    值对图片进行二值化
96.     # cv2.imshow('img3', img3)
97.     img4 = cv2.morphologyEx(img3, cv2.MORPH_OPEN, np.ones((3, 3), np
    .uint8))  # 开运算
98.     img5 = cv2.morphologyEx(img4, cv2.MORPH_CLOSE, np.ones((3, 3), n
    p.uint8))  # 闭运算
99.     # cv2.imshow('img5', img5)
100.
101.         center_x = 0
102.         final = img5
103.         cnts = cv2.findContours(final, cv2.RETR_EXTERNAL, cv2.CHAIN_
    APPROX_TC89_L1)[-2]  # 找出所有轮廓
104.         cnt_largest = getAreaMaxContour(cnts)  # 找到最大面积的轮廓
105.         if cnt_largest is not None:
106.             rect = cv2.minAreaRect(cnt_largest)  # 最小外接矩形
107.             box = np.int0(cv2.boxPoints(rect))  # 最小外接矩形的四个
    顶点
108.             p1_x, p1_y = box[0, 0], box[0, 1]
109.             p3_x, p3_y = box[2, 0], box[2, 1]
110.             # cv2.drawContours(img, [box], -
    1, (0, 0, 255, 255), 2)  # 画出四个点组成的矩形
111.             center_x, center_y = (p1_x + p3_x) / 2, (p1_y + p3_y) /
    2  # 中心点
112.             # cv2.circle(img, (int(center_x), int(center_y)), 10, (0
    , 0, 255), -1)  # 画出中心点
113.             x_pid.SetPoint = img0.shape[-2]/2  # 图像中心
114.
115.             x_pid.update(center_x)  # 将当前获取的跑道中心值作为输入
    值
116.
117.             # pid 输出
118.             out = int(x_pid.output)
119.
```

```
120.             # 确定速度，直道快，弯道慢
121.             if abs(x_pid.SetPoint - center_x) < 5:
122.                 speed = 7.5 # 直道
123.             else:
124.                 speed = 4 # 弯道
125.
126.             # 限制输出大小
127.             if speed - out < -10:
128.                 out = 10 + speed
129.             elif speed - out > 10:
130.                 out = -10 + speed
131.             if speed + out < -10:
132.                 out = -10 - speed
133.             elif speed + out > 10:
134.                 out = 10 - speed
135.
136.             # 根据速度调整拐弯幅度
137.             if speed >= 7:
138.                 out = out / speed
139.             else:
140.                 out = out / 3
141.
142.             get_line = True
143.             # 计算左右速度
144.             left_speed = speed - out
145.             right_speed = speed + out
146.
147.
148.     def move():
149.         global get_line, left_speed, right_speed
150.         while True:
151.             if get_line:
152.                 get_line = False
153.                 vrep.simxSetJointTargetVelocity(clientID, left_motor
    , left_speed, vrep.simx_opmode_streaming)
154.                 vrep.simxSetJointTargetVelocity(clientID, right_moto
    r, right_speed, vrep.simx_opmode_streaming)
155.             else:
156.                 time.sleep(0.001)
157.
158.     th = threading.Thread(target=move)
159.     th.setDaemon(True)
160.     th.start()
161.     if clientID != -1:
```

```
        print('Connected to remote API server')
        # Get the handle of Pioneer_p3dx, Pioneer_p3dx_leftMotor, Pioneer_p3dx_rightMotor, Vision_sensor
        _, body = vrep.simxGetObjectHandle(clientID, 'Pioneer_p3dx', vrep.simx_opmode_oneshot_wait)
        _, left_motor = vrep.simxGetObjectHandle(clientID, 'Pioneer_p3dx_leftMotor', vrep.simx_opmode_oneshot_wait)
        _, right_motor = vrep.simxGetObjectHandle(clientID, 'Pioneer_p3dx_rightMotor', vrep.simx_opmode_oneshot_wait)
        _, camera = vrep.simxGetObjectHandle(clientID, 'Vision_sensor', vrep.simx_opmode_oneshot_wait)

        _, position = vrep.simxGetObjectPosition(clientID, body, -1, vrep.simx_opmode_streaming)
        _, orientation = vrep.simxGetObjectOrientation(clientID, body, -1, vrep.simx_opmode_streaming)


        # 获取传感器的图像
        res, resolution, image = vrep.simxGetVisionSensorImage(clientID, camera, 0, vrep.simx_opmode_streaming)
        while (vrep.simxGetConnectionId(clientID) != -1):
            res, resolution, image = vrep.simxGetVisionSensorImage(clientID, camera, 0, vrep.simx_opmode_buffer)
            if res == 0:
                # 将图像变成标准格式
                img = np.array(image, dtype=np.uint8)
                img.resize([resolution[1], resolution[0], 3])
                # 将图像缩小，提高之后对图像的处理速度
                img=cv2.resize(img,(128,128))

                # 调整图像方向
                img=cv2.flip(img,1)
                img=cv2.flip(img,0)
                img=cv2.flip(img,1)

                # 截取有效视野，排除干扰
                img = img[ (img.shape[0]//4)*3 : (img.shape[0]//10)*9 , img.shape[1]//4 : (img.shape[1]//4)*3]
                Tracing(img)
                # 显示图像
                # cv2.imshow('img', img)
                cv2.waitKey(1)
        cv2.destroyAllWindows()
```