西安交通大學

XI'AN JIAOTONG UNIVERSITY

本科实验报告

机器人导航控制实验

课程名称：	机器人导航实验
姓名：	王海翔
学院：	电子与信息学部
专业：	人工智能
学号：	2203312479
指导老师：	张唐一可

2023 年 4 月 24 日

西安交通大學实验报告

专业：人工智能
姓名：王海翔
学号：2203312479
日期：2023 年 4 月 24 日
地点：科学馆 203

课程名称：机器人导航实验　指导老师：张唐一可　成绩：
实验名称：机器人导航控制实验　实验类型：仿真实验　同组学生姓名：

一、 实验目的和要求

1. 实验目的

轨迹跟踪是指根据某种控制理论，为机器人系统设计一个循迹控制器，使机器人能够到达并最终以期望的速度跟踪期望轨迹。控制系统的任务即是严格按照这个参考路径（以及速度等控制输入量）去控制机器人运动。本实验旨在通过机器人控制实验，使同学们掌握无人驾驶横纵向控制的基本方法，培养和提高同学们应用控制理论解决实际问题的能力。

2. 实验要求

实现基于 PID 的机器人循迹控制。

(1) 掌握实验原理并绘制主要模块框图。

(2) 针对不同的算法进行性能分析。

(3) 提出改善控制性能的可行方案。

二、 实验环境

1. 硬件环境

dell Optiplex 7070 /i7-9700/32G/2T+256GSSD/DVD 刻录/NVIDIA GeForce GTX 1650, 4GB

2. 软件环境

(1) 操作系统:Ubuntu 18.0.4

(2) Python:3.7.3

(3) ROS1:Melodic

三、 实验内容

(1) 实现基于 PID 的循迹控制器。

(2) 实现模糊自适应 PID 控制

(3) 实现模型预测控制 (MPC) 可选

四、 实验原理

1. 算法概述

PID 算法是一种常用的控制算法，它可以通过对被控对象输出和目标值之间的误差进行计算，来实现对被控对象的精确控制。PID 算法由比例控制（P）、积分控制（I）和微分控制（D）三个部分组成。

具体地说，PID 控制器的输出 $u(t)$ 是由三个部分组成的加权和：

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

其中，$e(t)$ 表示目标值与被控对象输出之间的误差，K_p、K_i 和 K_d 分别表示比例系数、积分系数和微分系数。

具体如下流程图：

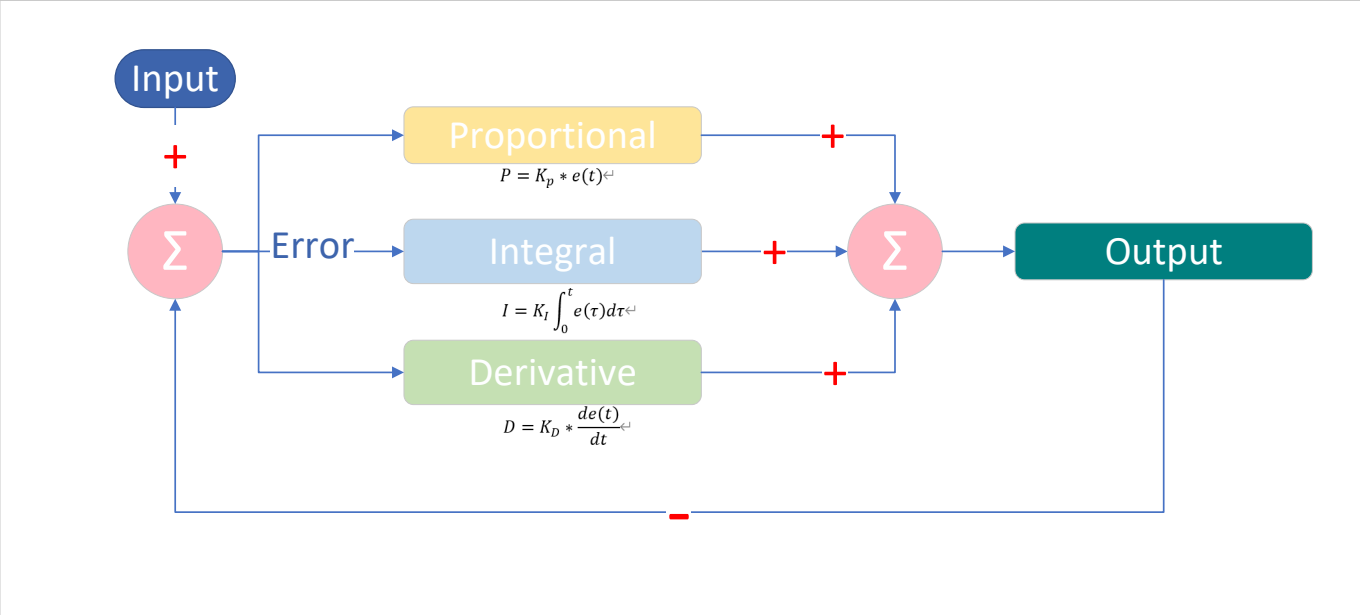


图 1: PID 流程图

其中，输入为 Input，输出为 Output 即为 u(t)

比例控制器 P 的作用是根据误差的大小来控制输出，它可以让被控对象快速地接近目标值。但是，由于比例控制器无法消除误差，因此在误差较大时，它的输出会产生较大的波动。

积分控制器 I 的作用是消除误差，它可以根据误差的积分值来控制输出，从而消除误差。但是，积分控制器会导致系统产生较慢的响应，并可能引起系统的不稳定性。

微分控制器 D 的作用是根据误差的变化率来控制输出，它可以提高系统的稳定性并消除瞬态误差。但是，微分控制器对噪声和干扰非常敏感，因此需要谨慎使用。

通过对比例、积分和微分控制器的加权和进行控制，PID 控制器可以同时兼顾快速响应、消除误差和稳定性，从而实现对被控对象的精确控制。

五、 实验步骤

1. 准备工作

在 Ubuntu 系统中，安装好 ROS 环境，创建好工作空间，配置好环境变量后。打开终端，输入以下代码：

```
$cd ~ # 或者选择合适的路径作为工作环境，后续代码需要修改~为自己的环境
# 创建文件夹
$mkdir -p ~/Robot_Navigation_Control_Sim/src
#### Robot_Navigation_Control_Sim为工作空间名，可根据自己的需要更换;

$cd ~/Robot_Navigation_Control_Sim/src #进入工作空间的src目录下，后续代码均在src目录下运行
$catkin_init_workspace #初始化工作空间
$catkin_create_pkg pid_controller roscpp rospy std_msgs
#pid_controller为功能包名字，roscpp、 rospy、
    std_msgs为功能包依赖的库，可根据自己的需要更换;
```

后续将老师提供的代码将三个功能包进行替换。

接下来继续运行以下代码:

```
$cd ~/Robot_Navigation_Control_Sim #回到工作空间根目录
# 或者在src目录下可输入 cd .. 回到上级目录
$catkin_make #编译
##此时会发现在工作空间下生成了build、devel两个文件夹，用于存放编译过程中产生的一些文件和可执行文件

$source devel/setup.bash #设置工作环境 每次打开新的终端，都需要执行此命令

$roslaunch pid_controller pid_controller.launch #启动功能包 运行仿真小车
```

pid_controller.launch 文件中有三条重要语句，其内容如下:

```
<?xml version="1.0"?>

<launch>

 <include file="$(find vehicle_sim)/launch/vehicle_sim.launch" />

 <include file="$(find pid_controller)/launch/waypoint_loader.launch" />

 <node pkg="pid_controller" type="ebotController" name="ebotController" output = "screen">
 </node>
```

```
11
12
13 </launch>
```

(1) 第一条指令用于加载仿真小车，启动 rviz 界面

```
1          <include file="$(find vehicle_sim)/launch/vehicle_sim.launch" />
```

(2) 第二条指令用于加载路径（运行时需要打开新的终端并设置好工作环境,即 source devel/setup.bash）后续可修改 waypoint_loader.launch 文件中的路径，改变设定的轨迹（绿线）

```
1          <include file="$(find pid_controller)/launch/waypoint_loader.launch" />
```

(3) 第三条指令用于加载控制器

```
1          <node pkg="pid_controller" type="ebotController" name="ebotController"
               output = "screen">
2          </node>
```

2. PID 循迹

若想让小车循迹行驶，需接受两个主要话题，即小车的位姿信息及全局路径（本次控制仅使用全局路径）.

然后根据这些信息计算小车的转角，这里小车的纵向速度我们提前设定好，仅控制横向输出。以下是根目录下/src/pid_controller/src/ebotController.cpp 文件中的示例代码:

```
1  sub_final_waypoints = n.subscribe("/base_waypoints",1,&ebotControllerNode::lane_cb,this);
2
3  // sub_twist_cmd = n.subscribe("/vehicle/cmd",1,&ebotControllerNode::vel_cb,this);
4
5  //test
6  sub_pose = n.subscribe("/odom",1,&ebotControllerNode::pose_cb_test,this);
7  //pub
8  pub_twist_cmd = n.advertise<geometry_msgs::Twist>("/cmd_vel", 1);
9
10 control_timer_ = n.createTimer(ros::Duration(0.02), &ebotControllerNode::ControlCallback,this);
```

以此为例,第一个订阅 sub_final_waypoints 为接受的全局路径,这里我们接受的话题名字为/base_waypoints，调用 lane_cb 函数，得到路径信息。

第二个订阅 sub_pose 接受车辆位姿，话题名为/odom, 调用 pose_cb_test 函数来得到车辆位姿信息。

发布的话题为/cmd_vel，与此相关的内容写在 ControlCallback 函数里面，以便周期调用发布，图例设置频率为 50Hz(cpp 中值设为 0.02)。

修改好控制节点后,打开 pid_controller 功能包下的 CMakeList.txt,在文件尾部可以看到 add_executable(**** src/****.cpp),target_link_libraries(**** $catkin_LIBRARIES) 命令，生成可执行文件。

可参照 ros 小乌龟例子,实现话题的发布与订阅。https://blog.csdn.net/xtark_robot/article/details/107860571

实验过程中，观察车辆偏离车道中心线的程度，以及车辆在循迹过程中是否快速、平稳。如果性能不尽人意需要及时调整相关参数。

3. 一些另外的调整

(1) 将根目录下/src/vehicle_sim/rviz/vehicle_sim.rviz 文件中 keep 参数设置为 10000,keep=10000,保证小车全过程的轨迹线不消失。

(2) 编写 python 文件生成三种不同形状的轨迹路线，分别为方形，圆形，心形，以 csv 文件形式保存，方便后续调试。

(3) 生成路径时尽量使初始点位于原点即小车出发点；使设置路径的初始方向与小车开始方向一致，这样可以减少小车的转弯，更方便实验。

(4) 遵循一般的 PID 调参步骤，先调节 P，再调节 D，最后调节 I。

(5) 每次调整完 ebotController.cpp 文件中的 PID 参数后，需要重新编译，否则不会生效。即需要运行 $catkin_make

六、 实验结果

初始 PID 参数为 (0.4，0.3，0)

1. 直线轨迹

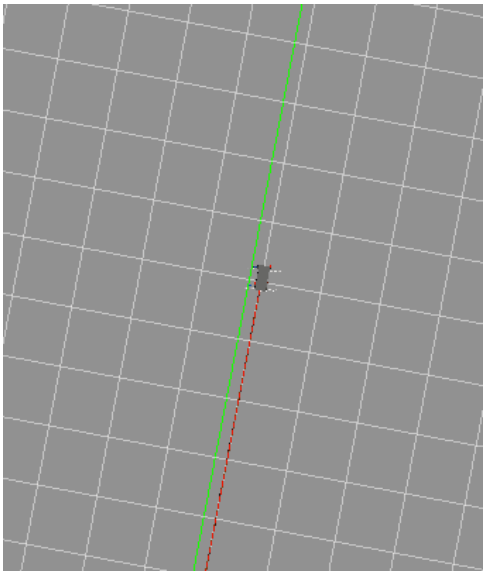


图 2: 直线轨迹 0-0-0

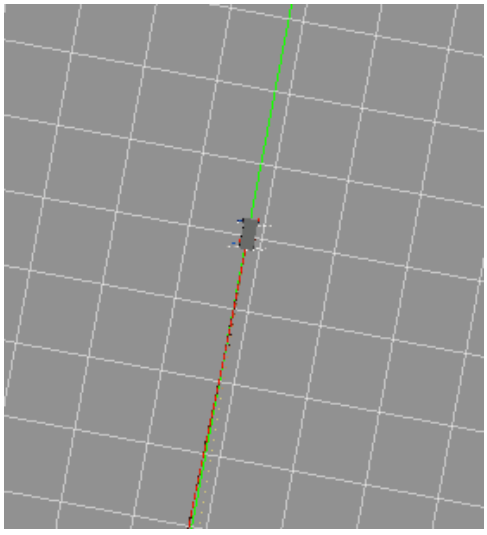
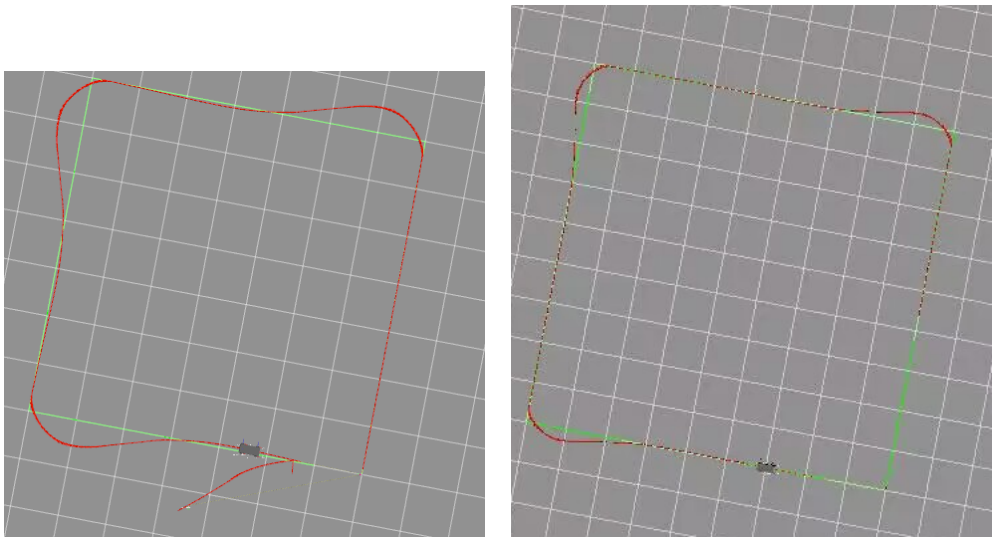


图 3: 直线轨迹 0.3-0-0

由上面两张图片可知，当 P 参数为 0 时，小车无法循迹；当 P 参数为 0.3 时，小车能够正常循迹。

显示了 P 参数的循迹作用，根据误差的大小来控制输出，让被控对象快速接近目标值。

2. 方形轨迹



(a) 方形轨迹 0.4-0.3-0　　(b) 方形轨迹 0.6-0.3-0

图 4: 方形轨迹

由上图可知，调大 P 参数能够使小车更加快速地循迹，

3. 圆形轨迹

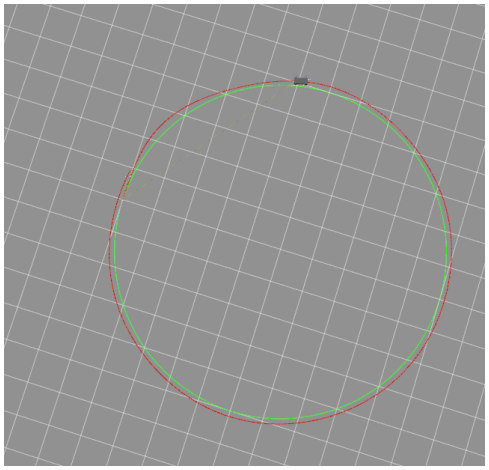
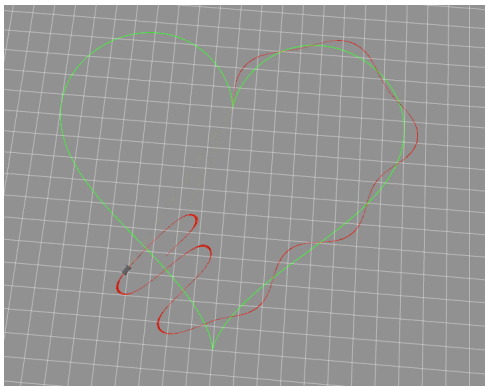


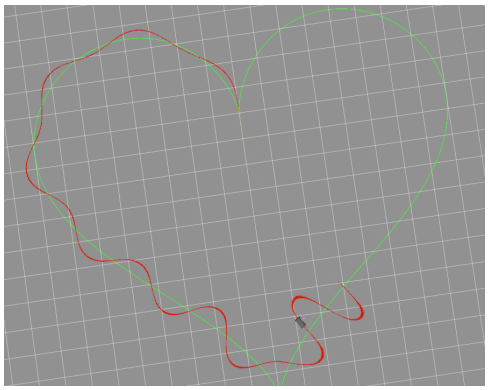
图 5: 圆形轨迹 0.4-0.3-0

默认参数已经可以较好地循迹，但是小车绕圆运动时，运动半径总是大于圆半径，并难以消除误差。后面发现在最后设定的 PID 参数下，小车几乎能够完美地绕圆运动。

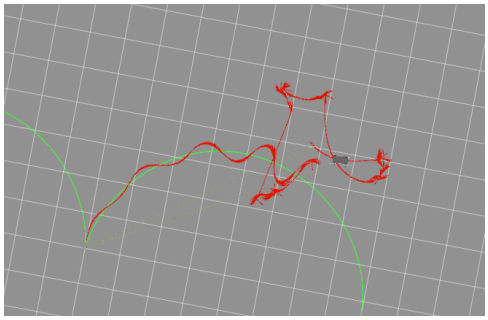
4. 心形轨迹



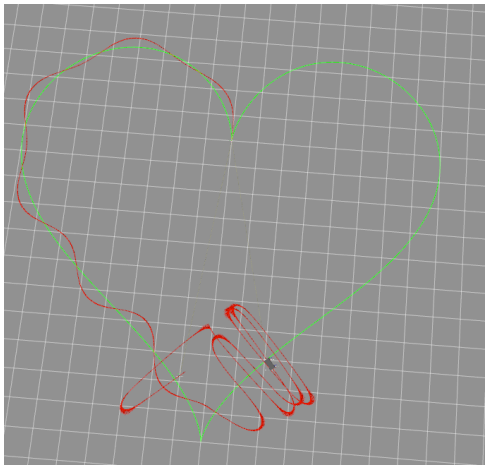
(a) 心形轨迹 0.4-0.3-0



(b) 心形轨迹 0.8-0.3-0



(c) 心形轨迹 0.6-0.3-0.05



(d) 心形轨迹 0.6-0.1-0

图 6: 心形轨迹

(1) 将心形线初始点坐标设为原点即小车出发点，并使初始曲线切向方向为小车运动方向，方便后续调节，也与现实条件相符。

(2) 小车初始向左转或者向右转均有可能发生，猜测是因为心形线取点个数不一样，导致初始曲线切向方向会有人眼无法识别的微小不同。

(3) 从图 a 我们可知，小车一开始能够在设定路径上摆动，进行效果较差的循迹，但是经过心形线的心尖处后，就会不断做椭圆的运动。

(4) 将图 a 至图 b，发现调大 P 参数能够使小车更加快速地循迹，但是会增大初始误差，并难以消除；后续也无法突破转圈圈的问题。

(5) 从图 b 至图 c，将参数 P 取成中间值 0.6，参数 I 不变，参数 D 设置为 0.05，但是小车在进行绕圆运动时就会出现偏离轨迹的情况。

(6) 从图 d，我们将 PID 参数设置为 0.6,0.1,0，发现仍然会出现经过心尖时后陷入椭圆运动的情况，而且比初始参数会更快地陷入困境。

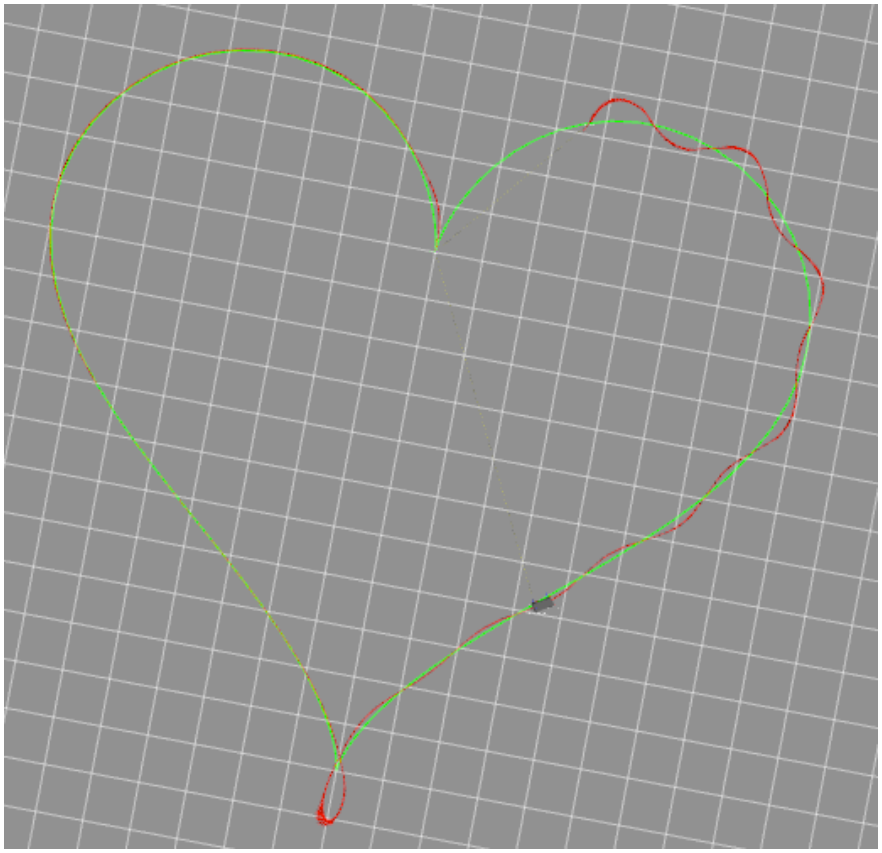


图 7: 心形轨迹 0.8-0.01-0.01

(1) 最后经过多次调参，发现将 PID 参数设置为 0.8,0.01,0.01，小车能够在心形轨迹上进行完美的循迹。（即使增大 I 参数为 0.2，似乎也能够很好地循迹）

(2) 小车经过心尖时，会画一个弧线后回到设定轨迹上。

(3) 美中不足的是，经过心尖后，小车似乎失去了原本能够完美拟合大弯曲线的能力，小车在结束点前的大弧线上运动时，会绕着设定轨道波动前进。

(4) 经过后续检验，此参数组在方形、圆形轨迹上都有几乎完美的循迹曲线。

七、 总结分析

1. 分析

(1) 考虑心形线光滑度对小车循迹实验的结果

首先，生成心形线轨迹的代码包含两个参数，a ,num_points，后者的大小决定了心形线先的光滑度，越大，点越多，心形线轨迹越光滑。

当路径上的点数量较少时，轨迹边界会显得比较粗糙，小车在进行路径跟踪时会出现较大的偏差，小车会进行多次钝角转弯，循迹效果可能会比较差。（参考矩形）

相反，当路径上的点数量较多时，小车的循迹控制会更加精确和平稳，因为控制算法能够更精确

地计算小车当前的位置和偏差。此外，路径上的点数量越多，小车与路径的误差就越小，因此可以实现更高的精度和稳定性。（参考圆形）

但是，路径上的点数量过多也会导致问题。一方面，太多的点会增加算法的计算负担，导致实时性下降；另一方面，路径上的点数量过多会导致路径规划的复杂度增加，可能会导致路径规划算法无法在规定的时间内完成路径规划。

因此，在确定路径上的点的数量时，需要权衡实时性、精度和计算负担等多个因素，并根据具体情况进行调整。故经过调试，我们选择了 600 个点作为最后的参数设置。(需要根据 a 和小车速度自行调节)

(2) 考虑 PID 参数调节方法

在 PID 控制中，PID 参数的合适取值需要根据具体的控制对象和控制要求来确定，通常需要通过试验和调整来得到。在一般情况下，可以按照以下步骤来设置 PID 参数：

a. 将三个参数（比例系数 Kp、积分系数 Ki、微分系数 Kd）都设为 0；

b. 逐个增加参数，首先将 Kp 设为一个较小的值，然后观察系统响应，如果响应过度，可以逐渐减小 Kp 的值，直到响应合适；

c. 在 Kp 合适的情况下，增加 Ki 的值，如果系统存在静态误差，可以通过增大 Ki 的值来减小误差，但是需要注意 Ki 过大会导致系统不稳定；

d. 在 Kp 和 Ki 合适的情况下，增加 Kd 的值，可以通过增大 Kd 的值来减小系统的超调量，但是需要注意 Kd 过大会导致系统出现震荡；

e. 可以通过反复试验和调整来得到合适的 PID 参数。

需要注意的是，PID 参数的取值不仅与控制对象和控制要求有关，还受到控制器的采样周期、噪声等因素的影响，因此在实际应用中需要进行更加细致和精确的调整。

2. 总结

在本次实验中，我们实现了基于 PID 的机器人循迹控制，并对不同算法进行了性能分析，并提出了改善控制性能的可行方案。

首先，我们通过使用网上文档、老师代码及指导书等资源创建工作空间并初始化，成功编译后，粗略地浏览控制代码并着重学习了重点部分代码。

其次，我们实现了基于 PID 算法的机器人循迹控制，该算法将误差的比例、积分和微分进行加权和计算，从而实现对被控对象的精确控制。在实验中，我们可以通过调整 PID 控制器中的三个系数，来获得较好的控制性能。

再次，经过多次调参，我们实现了对直线、方形、圆形、心形等轨迹的机器人循迹控制的 PID 参数调节，使机器人能够很好地以期望误差跟踪轨迹。

最后，我们通过对实验过程和实验图像的分析，总结了实验过程中的 PID 参数调节经验与方法。