

自动控制 实验报告

电子系统导论实验报告

实验10 自动控制

指导教师： 万景

学生姓名： 彭堃	学生姓名： 吴磊	学生姓名： 徐洋
学 号： 22307110109	学 号： 22307130218	学 号： 20300290037
专 业： 保密技术	专 业： 技科	专 业： 计算机

日 期： 2024.05.10

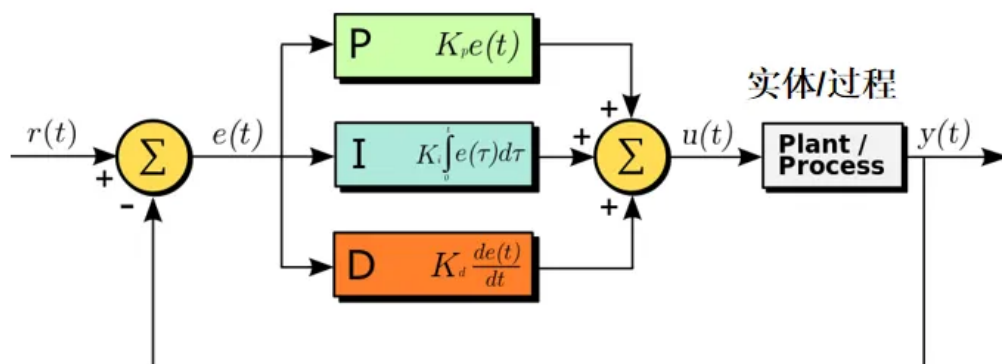
一.实验目的

- 1.了解反馈控制的基本原理
- 2.掌握PID控制的基本方法

二.实验原理

1.PID控制原理

在工程实际中，应用最为广泛的调节器控制规律为比例(proportion)、积分(integral)、微分(derivative)控制，简称PID控制，又称PID调节。



图中展示了PID控制的基本原理。给定输入 $r(t)$ 到系统中运行，其中 $y(t)$ 是过程中得到的实际输出值， $e(t) = r(t) - y(t)$ 是需要修正的值，最后通过PID系统输出控制变量 $u(t)$ 来修正输出。

PID分别是英文proportion,integral和derivative的缩写分别表示对误差 $e(t)$ 进行乘系数，作积分，做微分的操作。

以控制小车速度为例

例如将小车速度设定值 $r(t)$ 为3m/s，由码盘测得速度为 3.2m/s，即过程输出值 $y(t)$ ；偏差 $e(t)$ 为-0.2m/s。

- P：将-0.2m/s乘以一个系数（正）输入到控制器中，以减小输出的占空比，则车轮转速将降低，向设定值靠近。KP越大则调节的灵敏度越大，但过大可能会使实际速度低于3m/s（超调）。
- I：只经过比例调节的小车，可能稳定后的速度为3.1m/s，存在稳态误差-0.1m/s；虽然误差很小，但是因为积分项也会随着时间的增加而加大，它推动控制器的输出增大，从而使稳态误差进一步减小，直到等于0。
- D：小车中有些组件存在较大惯性或者滞后性，其变化总是落后于误差的变化。假设经比例调节后实际速度为3.1m/s，则设定速度与实际速度的差值由-0.2m/s变为-0.1m/s， $e(t)$ 的差分为0.1m/s²，将此差分乘以系数（正）加到控制变量中，相比只有比例环节减缓了速度降低的趋势（减小超调量）。

2.PID的手动调参

根据理论，PID控制可用数学公式表达：

$$u(t) = K_p e(t) + K_i \int_0^t e(t') dt' + K_d \frac{de(t)}{dt} \quad \text{其中 } K_p, K_i, K_d \geq 0.$$

K_p, K_i, K_d 分别为比例系数，微分系数和积分系数。

手动调参参数对输出的影响：

▣ 独立增加参数的影响

调整方式	上升时间	超调量	安定时间	稳态误差	稳定性
↑ K_p	减少 ↓	增加 ↑	小幅增加 ↗	减少 ↓	变差 ↓
↑ K_i	小幅减少 ↘	增加 ↑	增加 ↑	大幅减少 ↓↓	变差 ↓
↑ K_d	小幅减少 ↘	减少 ↓	减少 ↓	变动不大 →	变好 ↑

调参秘诀：先比例后微分有需要调积分。

参数的影响：

- 1.增大比例系数使系统反应灵敏，调节速度加快，并且可以减小稳态误差。但是比例系数过大会使超调量增大，振荡次数增加，调节时间加长，动态性能变坏，比例系数太大甚至会使闭环系统不稳定
- 2.增大微分系数可以减小超调量和稳定时间。
- 3.增大积分系数会减小稳态误差，但会增大超调量和稳定时间；

3.离散PID

由于在计算机上进行PID调节时只能用离散型PID，我们介绍简要介绍离散PID的原理。

实际上就是对积分、微分进行离散化处理。计算机有基本采样时间间隔，设为 T ，那么对于 kT 时刻的系统有：

偏差 $e(k) = u(k) - y(k)$

积分使用加和: $e(k) + e(k - 1) + \dots$

微分考虑斜率: $k = \frac{e(k) - e(k - 1)}{T}$ (实际上每一个 T 均是恒定时长，所以形式上不除以 T 也可以表达微分)

PID改写为：

$$u(t) = K_p e(k) + K_i \sum_{j=1}^k e(j) + K_d (e(k) - e(k - 1)) \quad \text{其中 } K_p, K_i, K_d \geq 0.$$

三.实验内容

PID控制小车直行

调整代码

在预习时我们便发现了代码中有一出明显的错误，因为在上次定时计数实验中，我在单线程代码的尝试中为了保证内存安全，将睡眠时间修改成了0.1，同时也将除数从585.0修改成了58.5

而在本次实验的代码中，睡眠时间被改成了0.1，除数却还是585.0，这显然是有问题的，假如使用原始代码的1.9的速度输入，等价目标速度会变成19，根本达不到，最后就是占空比全拉满

我们修改了代码，并将目标速度设置为最大速度的一半左右，也就是3，这样速度比较合适，不快不慢，对代码来说有较大的调整空间

手动调参方案

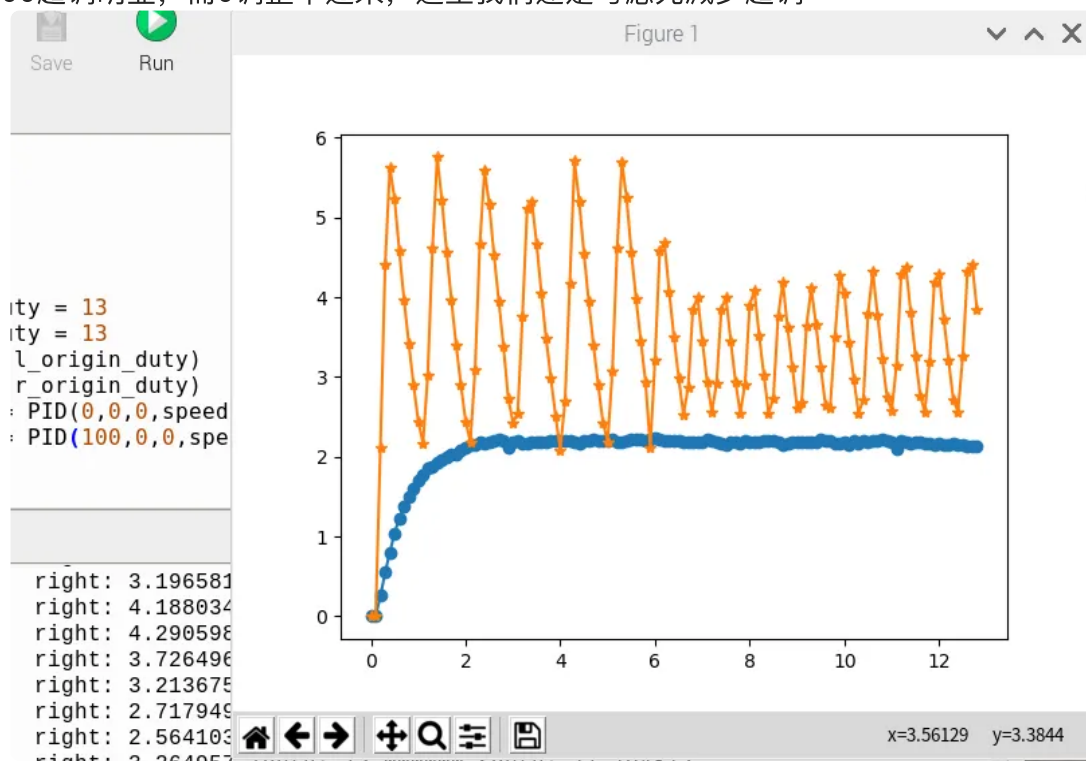
由于这些参数目前对我们来说都是黑盒，所以我们打算按照上面的建议依次调整，并采用类似于二分查找的方案来快速找到较为合适的参数，当然这里的二分查找和算法的二分查找有区别，相较于直接比较大小，这里我们使用我们的大脑来评判调参效果并调整范围，另外二分查找是严格单调，而我们这里是找最好，所以我们可以使用大脑评估、回调

另外我们采用控制变量法加比较法，左右轮使用不同的参数，但仅一个参数不同

P调参

我们参考之前的一些内容和代码提供的原始参数，决定将后面两个参数置0，P选择0-100的范围

1. 第一次，100~0,可以看到100超调明显，而0调整不过来，这里我们还是考虑先减少超调

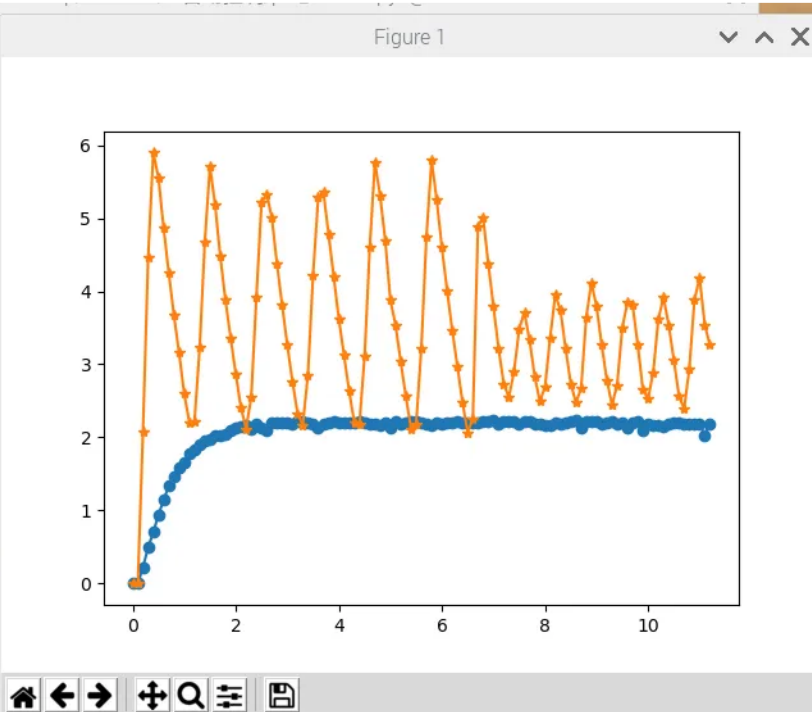


2. 50~0 依旧超调

Save Run

```
ty = 13  
ty = 13  
l_origin_duty)  
r_origin_duty)  
PID(0,0,0,speed,  
PID(50,0,0,speed,
```

```
right: 3.62393  
right: 3.91453  
right: 3.53846  
right: 3.05982  
right: 2.56410  
right: 2.39316  
right: 2.94017
```

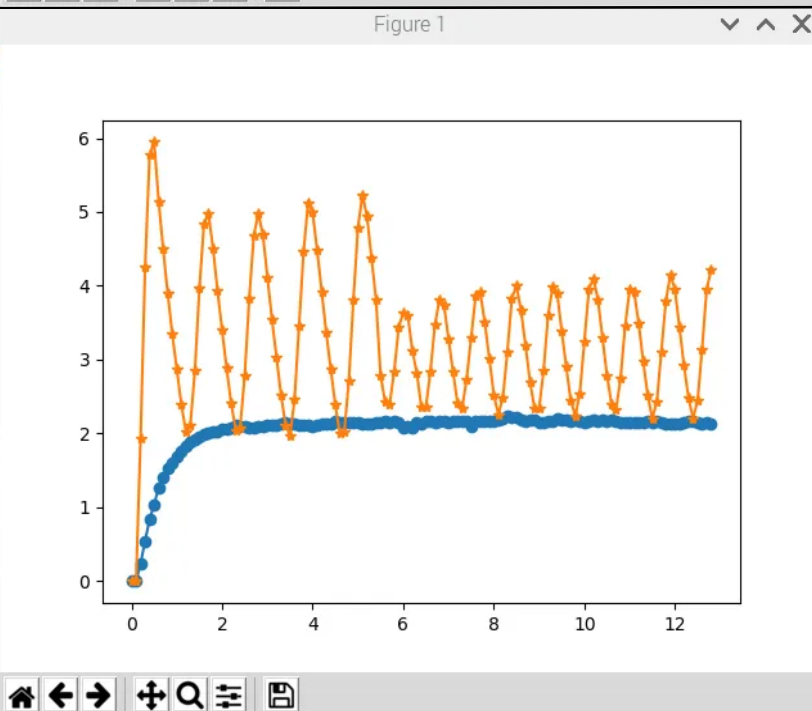


3. 25~0 还是超调

Save Run De

```
ty = 13  
ty = 13  
l_origin_duty)  
r_origin_duty)  
PID(0,0,0,speed,  
PID(25,0,0,speed,
```

```
right: 3.794872  
right: 4.153846  
right: 3.948718  
right: 3.435897  
right: 2.923077  
right: 2.420000
```

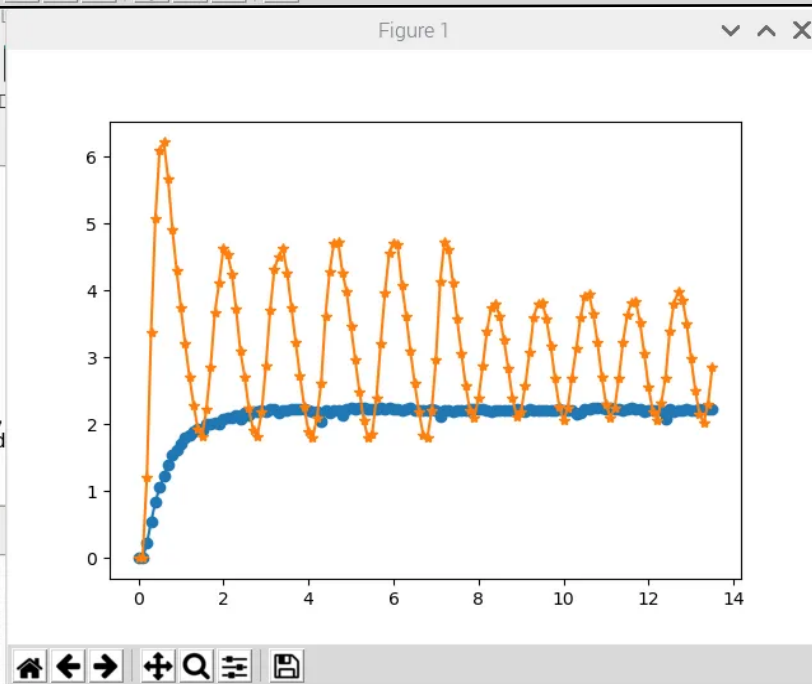


4. 12~0 超调

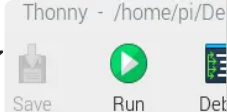
Save Run D

```
/ = 13  
/ = 13  
_origin_duty)  
_origin_duty)  
PID(0,0,0,speed,  
PID(12,0,0,speed,
```

```
right: 3.401709  
right: 3.811966  
right: 3.982906  
right: 3.863248  
right: 3.504274  
right: 2.901453
```



5. 6~0 可以看到超调已经不是很明显，这时我们就需要考虑调整0了



```

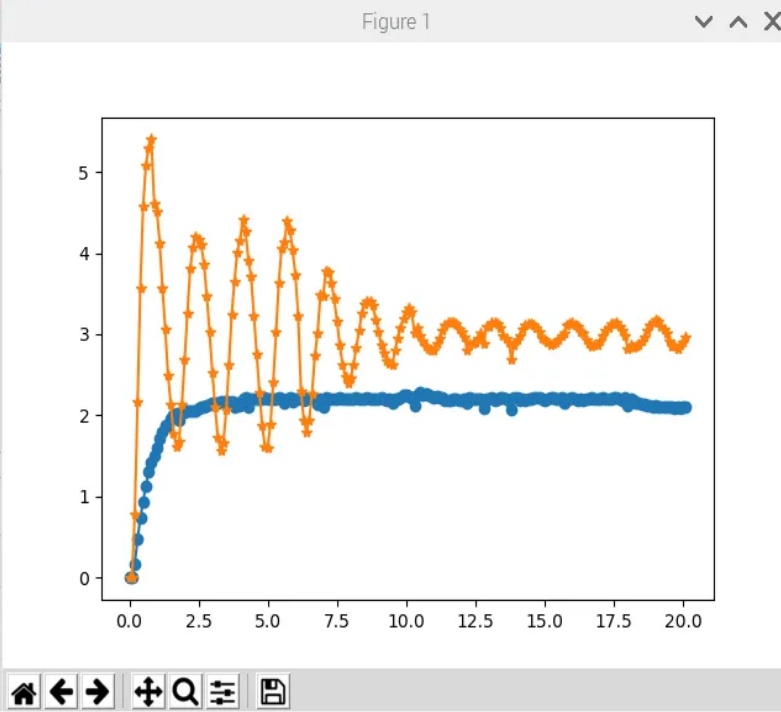
uty = 13
uty = 13
(l_origin_duty)
(r_origin_duty)
PID(0,0,0,speed,l
PID(6,0,0,speed,r

```

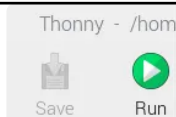
```

right: 3.162393 l
right: 3.094017 l
right: 3.059829 l
right: 3.008547 l
right: 2.923077 l

```



6. 3~6 明显3更优秀



```

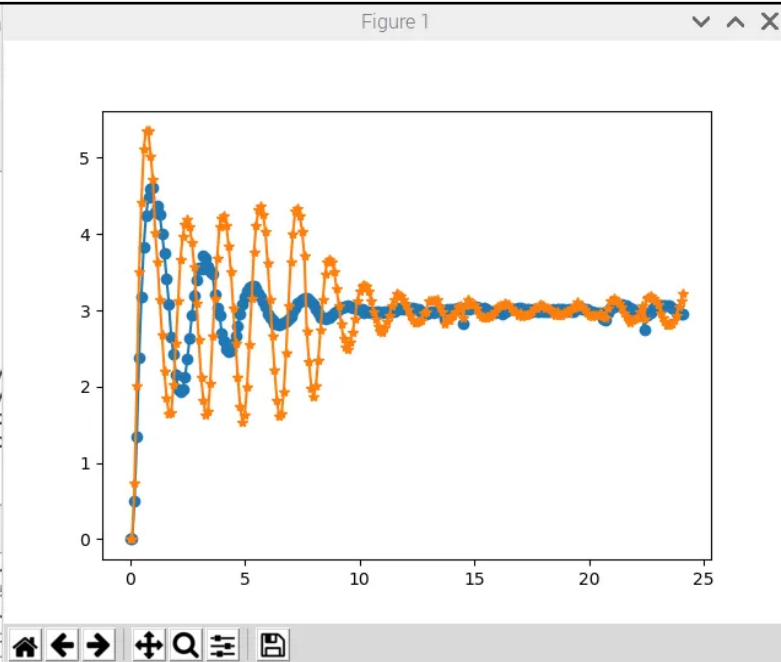
uty = 13
uty = 13
(l_origin_duty
(r_origin_duty
= PID(3,0,0,sp
= PID(6,0,0,sp

```

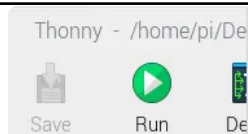
```

right: 2.97
right: 2.90
right: 2.85
right: 2.80
right: 2.80

```



7. 3~5 还是3更好



```

uty = 13
uty = 13
(l_origin_duty)
(r_origin_duty)
= PID(3,0,0,speed,l
= PID(5,0,0,speed,r

```

```

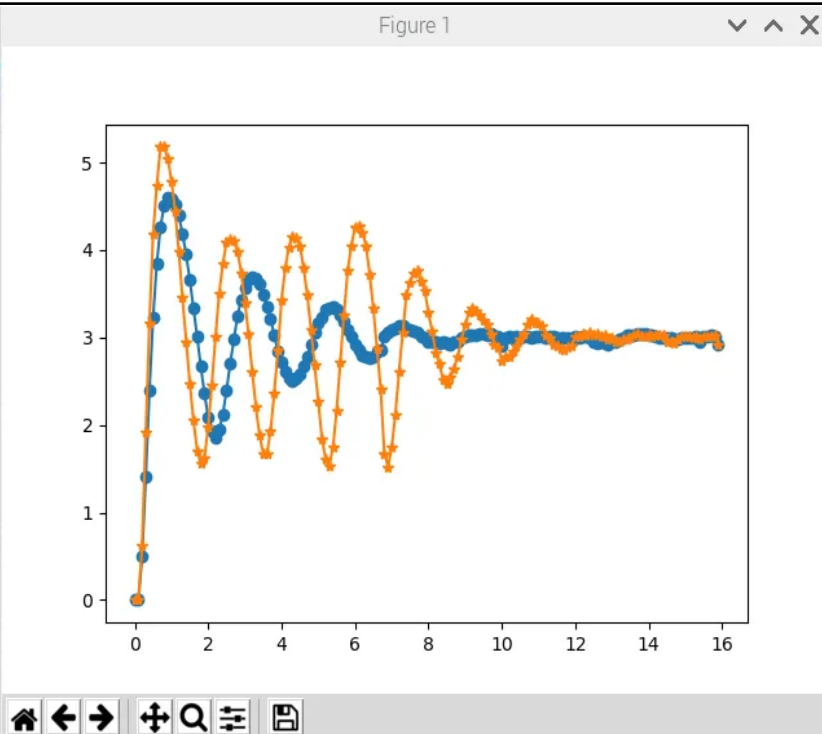
True:
na.ChangeDutyCycle(
ch.ChangeDutyCycle(

```

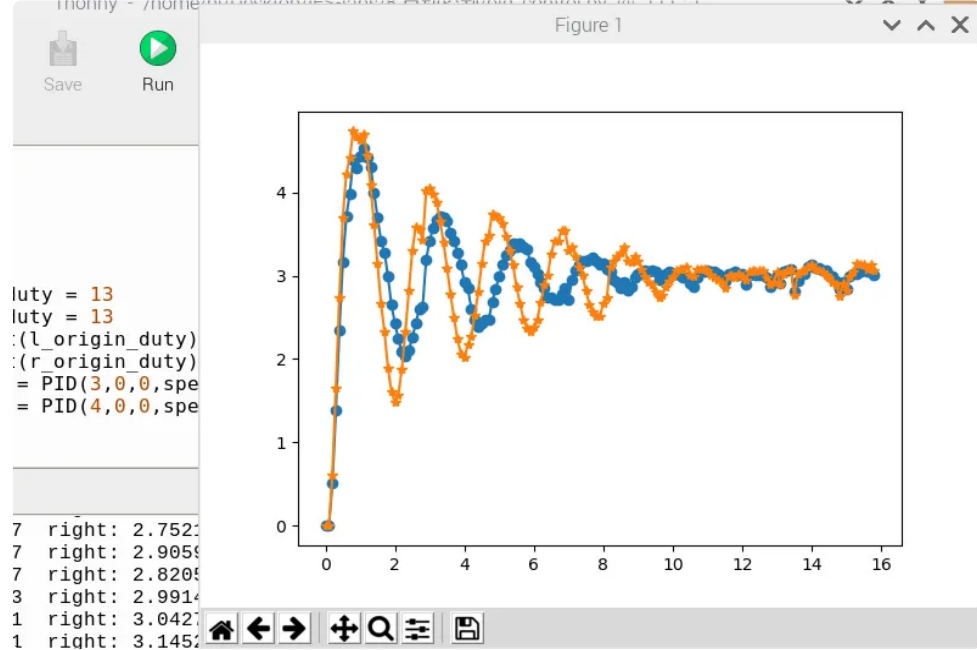
```

right: 3.008547
right: 3.008547
right: 3.008547
right: 2.991453
right: 2.991453

```



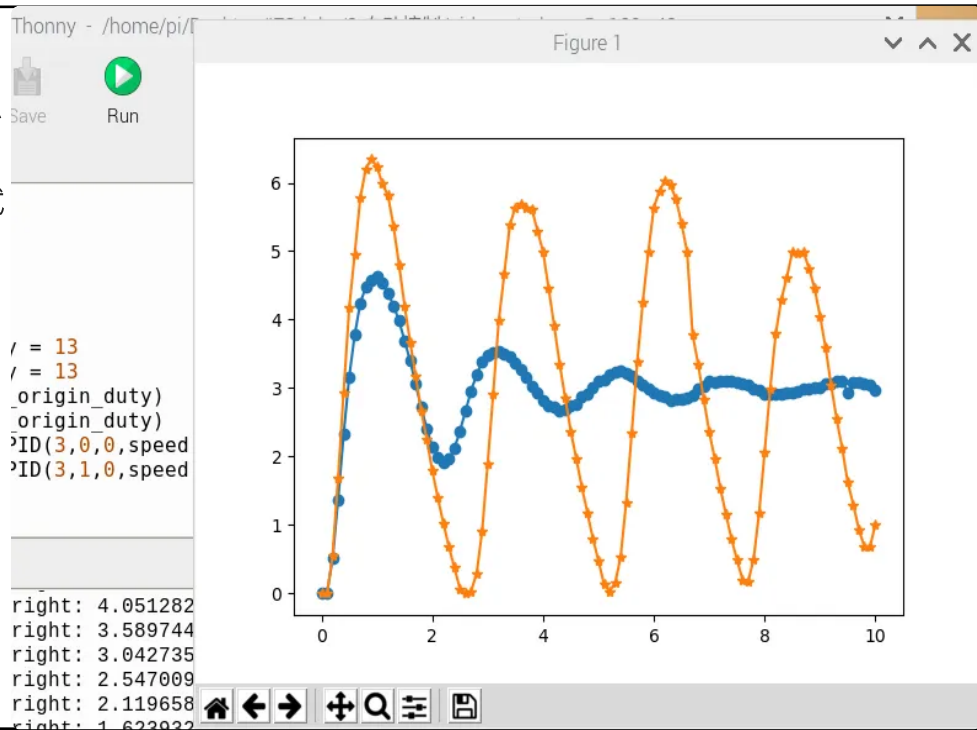
8. 3~4 差距不大, 最终选择3



I调参

参考原始代码给出的值很小, 我们选择从1开始

1. 1~0 明显1太大了, 直接调到0.1试试



2. 0.1并不比0好, 还要调小

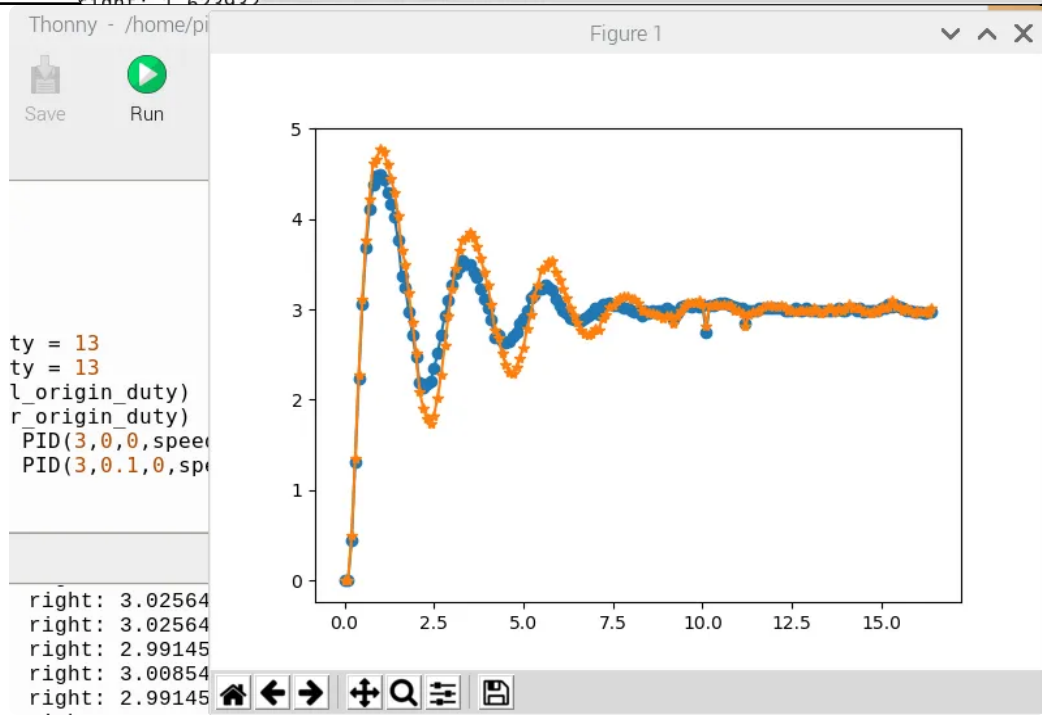


Figure 1

The plot displays the step response of a closed-loop system. The x-axis is labeled 'Time (s)' and ranges from 0 to 15. The y-axis is labeled 'Output' and ranges from 0 to 5. Two data series are shown: one with blue circles and one with orange stars. Both series start at 0, rise to a peak of approximately 4.5 at 1 second, oscillate, and settle around 3.0 after 10 seconds. The blue circles represent the system response with a 0.05s sampling time, and the orange stars represent the system response with a 0.01s sampling time.

Figure 1

The plot displays two data series, one in blue and one in orange, showing a damped oscillation over time. The x-axis represents time from 0 to 12, and the y-axis represents a value from 0 to 4. Both series start at 0, rise sharply to a peak of approximately 4.5 at time 1, and then oscillate with decreasing amplitude, settling around a value of 3.0 after time 6.

Time	Blue Series (approx.)	Orange Series (approx.)
0	0.0	0.0
0.5	2.3	2.3
1.0	4.5	4.5
1.5	3.5	3.5
2.0	2.2	2.2
2.5	2.8	2.8
3.0	3.6	3.6
3.5	3.2	3.2
4.0	2.5	2.5
4.5	3.0	3.0
5.0	3.4	3.4
5.5	2.8	2.8
6.0	3.1	3.1
6.5	2.8	2.8
7.0	3.0	3.0
7.5	3.1	3.1
8.0	2.8	2.8
8.5	3.0	3.0
9.0	3.0	3.0
9.5	2.9	2.9
10.0	3.0	3.0
10.5	2.9	2.9
11.0	3.0	3.0
11.5	2.9	2.9
12.0	3.0	3.0

Thonny - /home/pi/Desktop

Save Run Debug

Figure 1

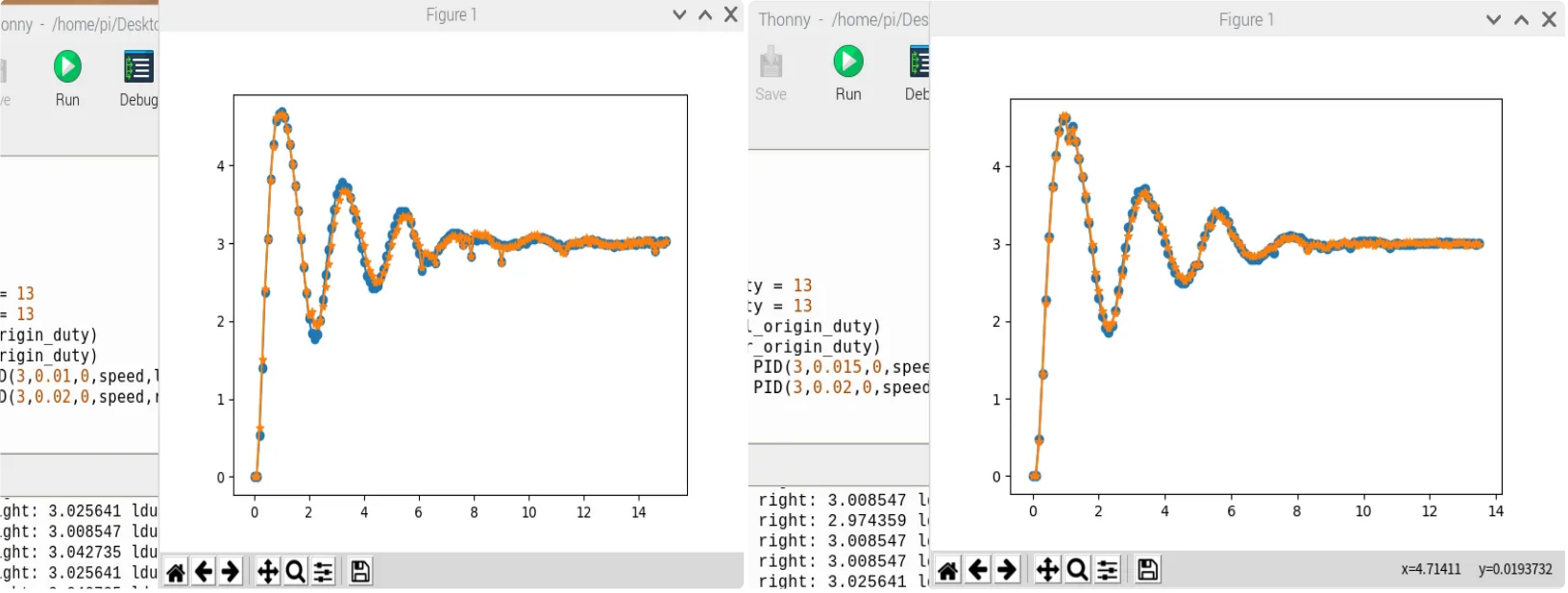
The graph displays the response of a system to a step change in duty cycle. The 'original_duty' (blue circles) and 'duty' (orange stars) series are nearly identical, indicating a very fast and accurate control system. The system exhibits a sharp initial rise, followed by a series of damped oscillations that settle to a steady-state value of approximately 3.0.

Time (s)	original_duty	duty
0.0	0.0	0.0
0.5	2.3	2.3
1.0	4.5	4.5
1.5	3.5	3.5
2.0	2.5	2.5
2.5	2.0	2.0
3.0	2.5	2.5
3.5	3.5	3.5
4.0	3.0	3.0
4.5	2.5	2.5
5.0	3.0	3.0
5.5	3.2	3.2
6.0	3.1	3.1
6.5	2.9	2.9
7.0	2.8	2.8
7.5	2.9	2.9
8.0	3.0	3.0
8.5	2.9	2.9
9.0	2.9	2.9
9.5	2.9	2.9
10.0	3.0	3.0
10.5	2.9	2.9
11.0	3.0	3.0
11.5	2.9	2.9
12.0	3.0	3.0
12.5	3.0	3.0
13.0	3.0	3.0
13.5	3.0	3.0
14.0	3.0	3.0
14.5	2.8	2.8
15.0	3.0	3.0

```
= 13
= 13
original_duty)
original_duty)
ID(3,0,0,speed,l_or
ID(3,0.02,0,speed,
```

right: 3.008547 ldu
right: 3.008547 ldu
right: 2.786325 ldu
right: 3.008547 ldu
right: 3.008547 ldu

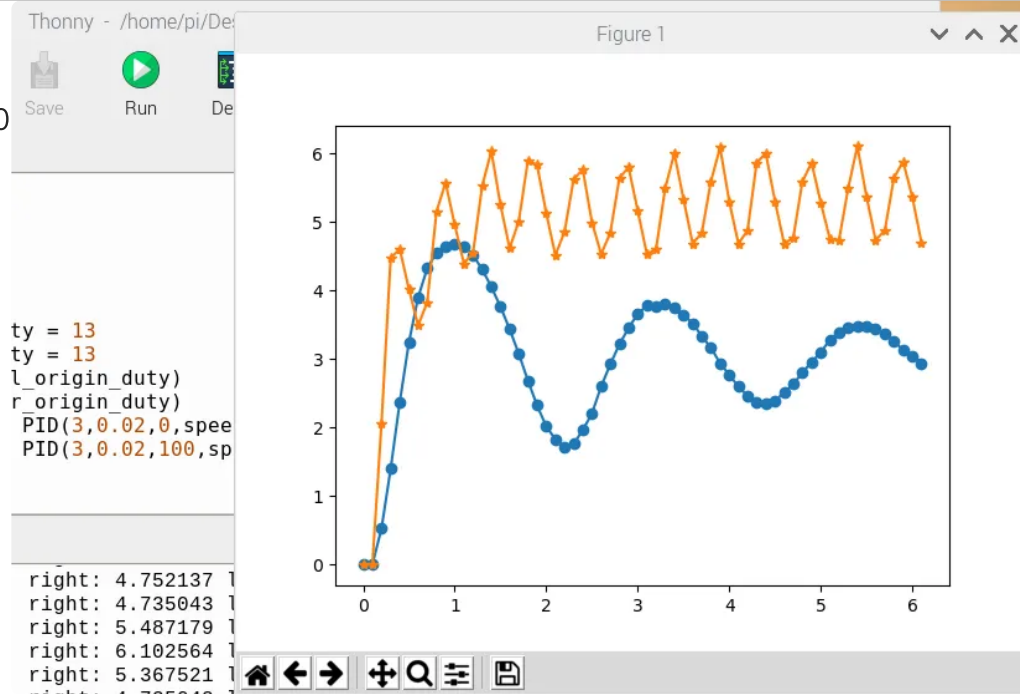
Navigation icons: Home, Back, Forward, Full Screen, Zoom In, Zoom Out, Print



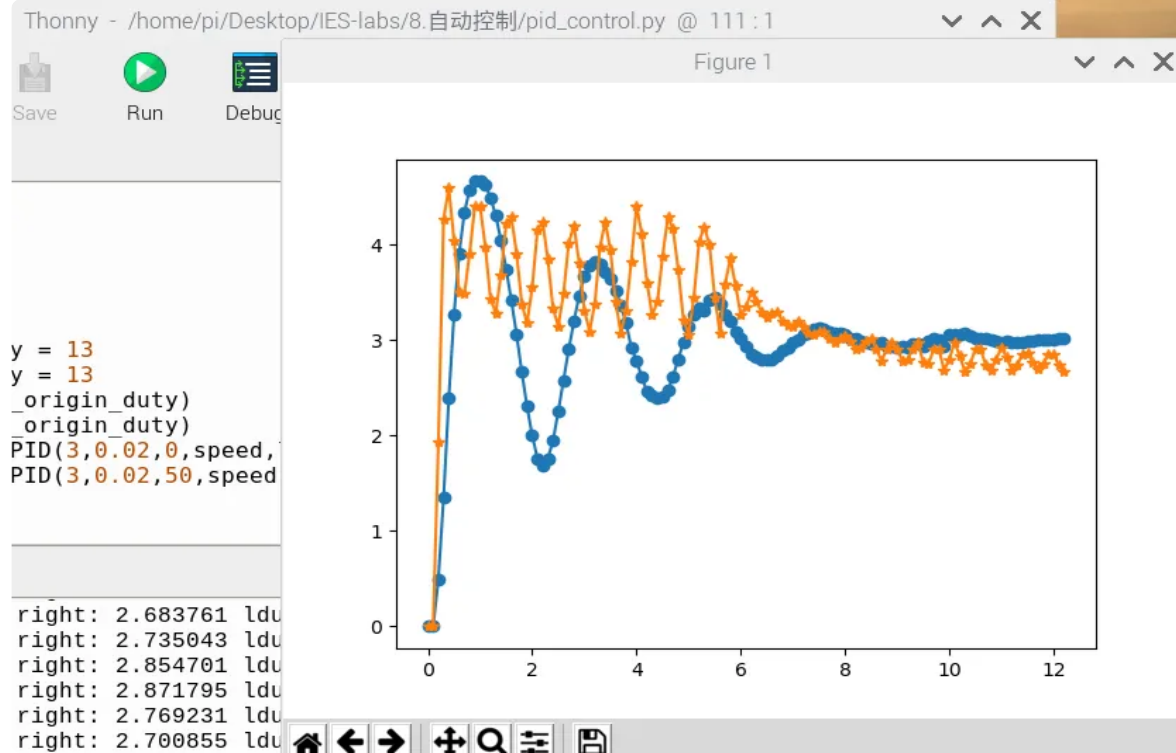
D调参

考虑到之前的数值，我们打算取范围0~100

1. 100~0 明显过度调整了



2. 50~0 还是过度了



3. 25~0 展现出相当不错的优化效果，可以调整0了

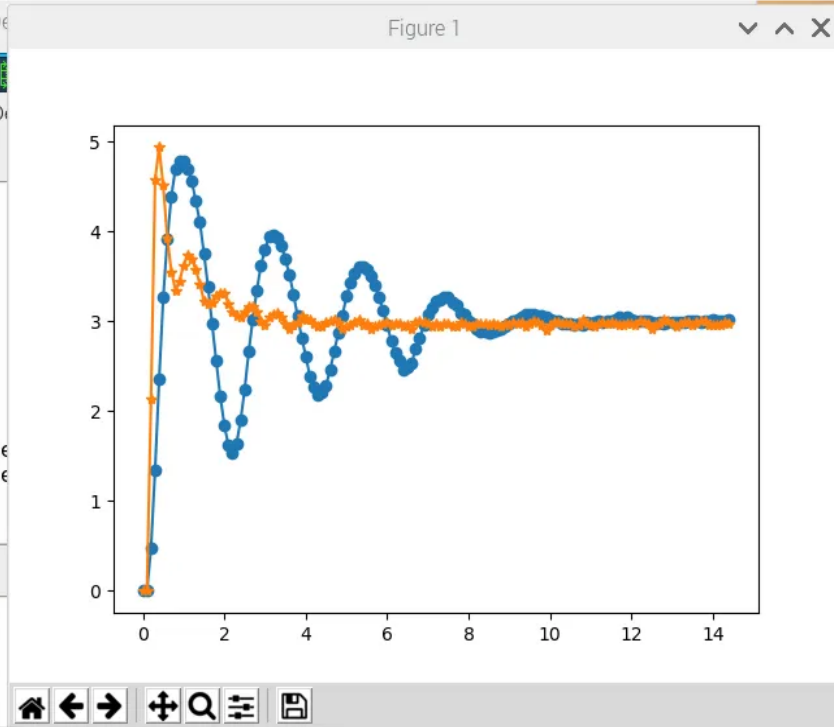
```

Thonny - /home/pi/Desktop/
Save Run De

ty = 13
ty = 13
l_origin_duty)
r_origin_duty)
PID(3,0.02,0,speed)
PID(3,0.02,25,speed)

right: 2.991453
right: 2.957265
right: 2.991453
right: 3.008547
right: 3.008547

```



4. 12.5~25 明显小一点更好

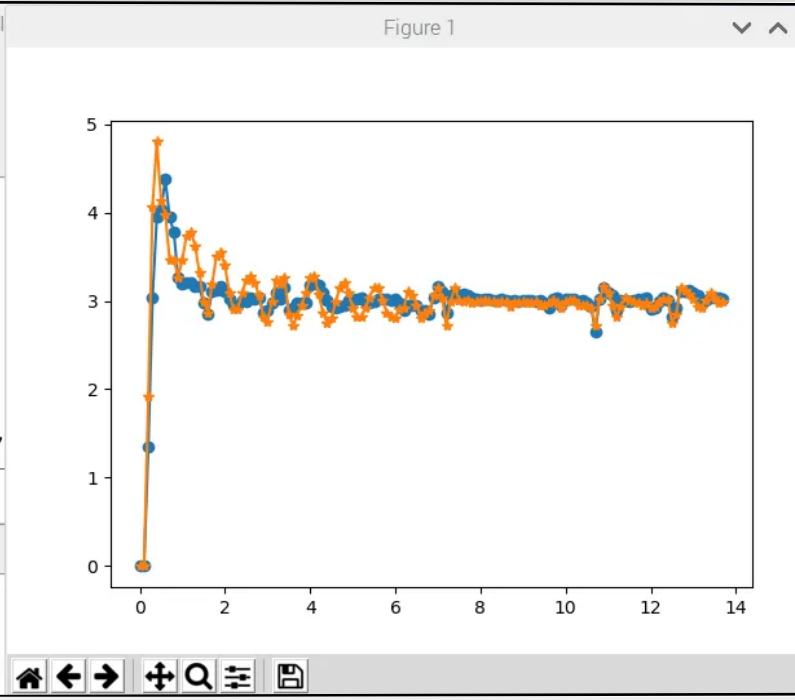
```

Thonny - /home/pi/Desktop/
re Run Debug

= 13
= 13
rigin_duty)
rigin_duty)
D(3,0.02,12.5,speed,
D(3,0.02,25,speed,r_

ght: 3.145299 lduty
ght: 3.128205 lduty
ght: 3.076923 lduty
ght: 3.025641 lduty
ght: 2.940171 lduty

```



5. 12.5~20 20的效果相当不错，大概就在二者之间

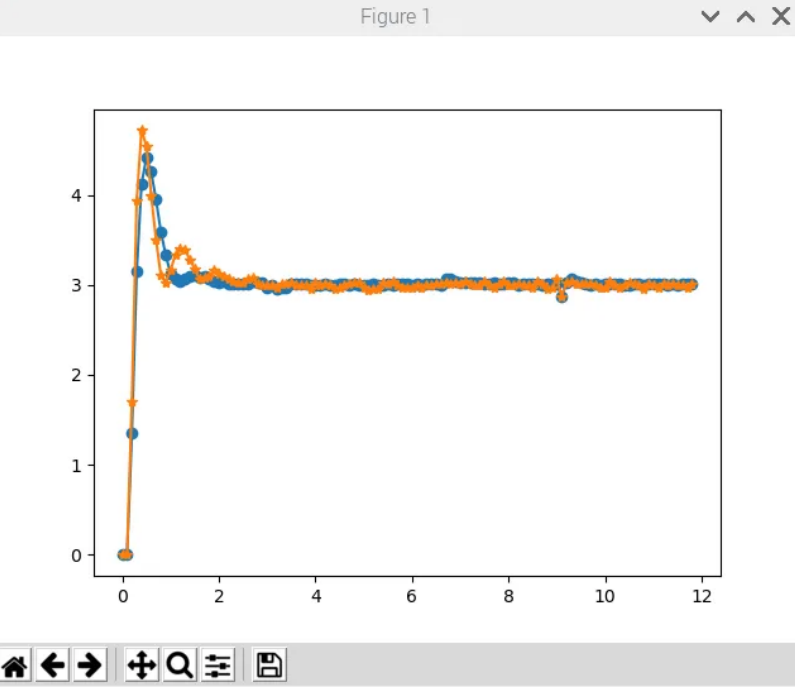
```

Thonny - /home/pi/Desktop/
Save Run Deb

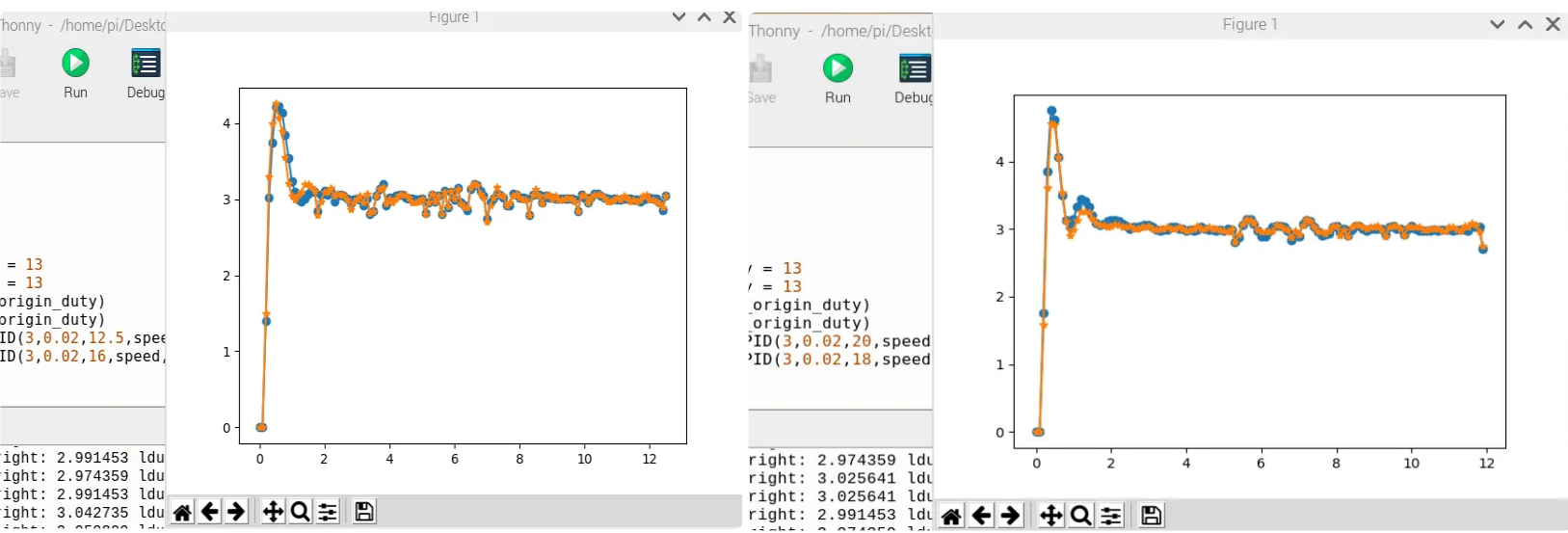
ty = 13
ty = 13
l_origin_duty)
r_origin_duty)
PID(3,0.02,12.5,speed)
PID(3,0.02,20,speed)

right: 2.957265 lduty
right: 3.008547 lduty
right: 2.991453 lduty
right: 2.974359 lduty
right: 3.008547 lduty
right: 2.991453 lduty

```



7. 进行一些比较，实际上18大概是最好的，但是20也不错



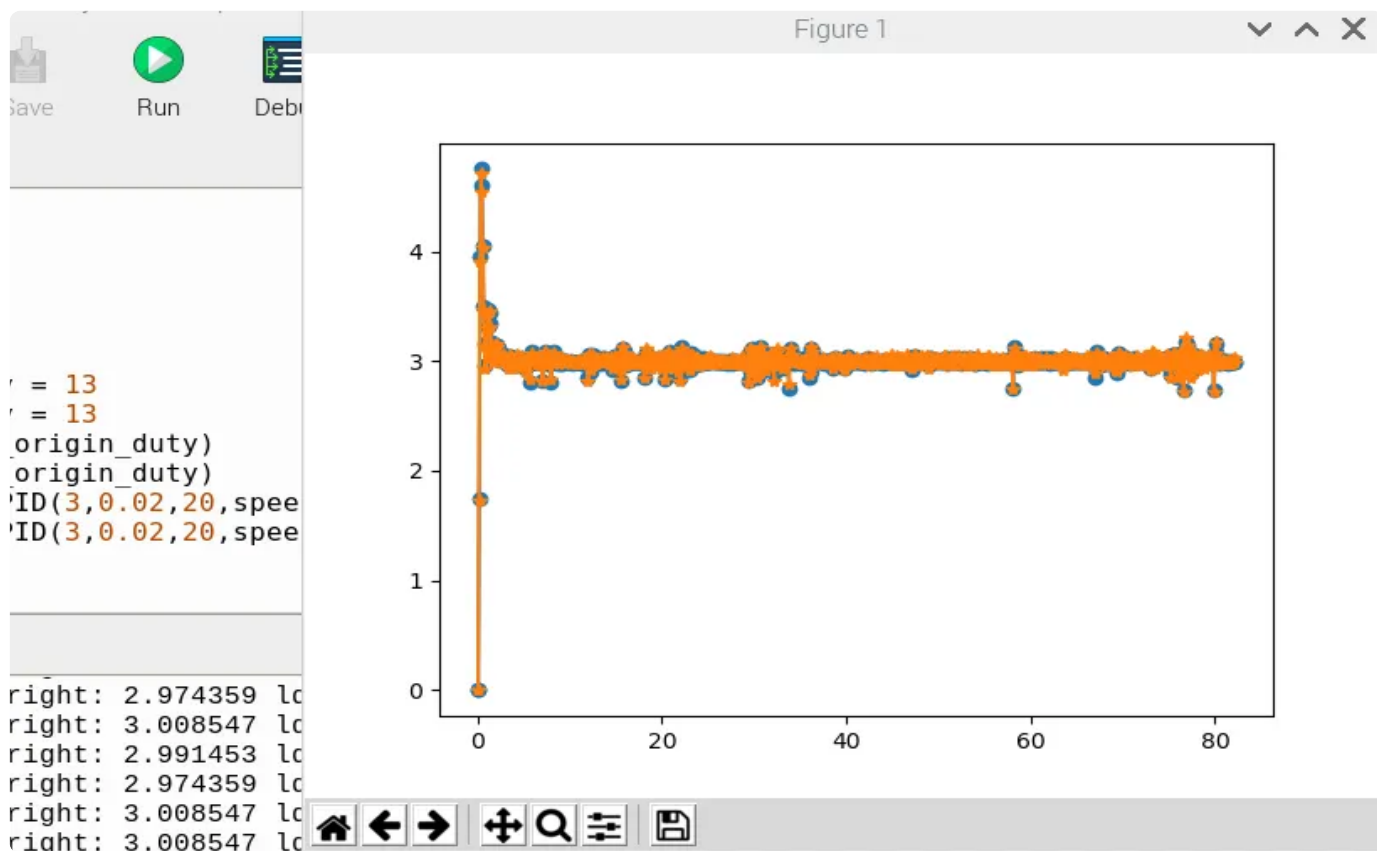
最佳参数范围及稳定时间测试

在进行一些尝试，根据大量数据，我们可以认为最佳参数范围大概是

P:3~4 ; l: 0~0.3（影响较小，且可能不止一个峰值） ; D:17~20

由于三个参数是相互影响的，所以我们暂时还不能简单的得到“最佳”的参数

- 最终我们选择了这样一组参数进行测试

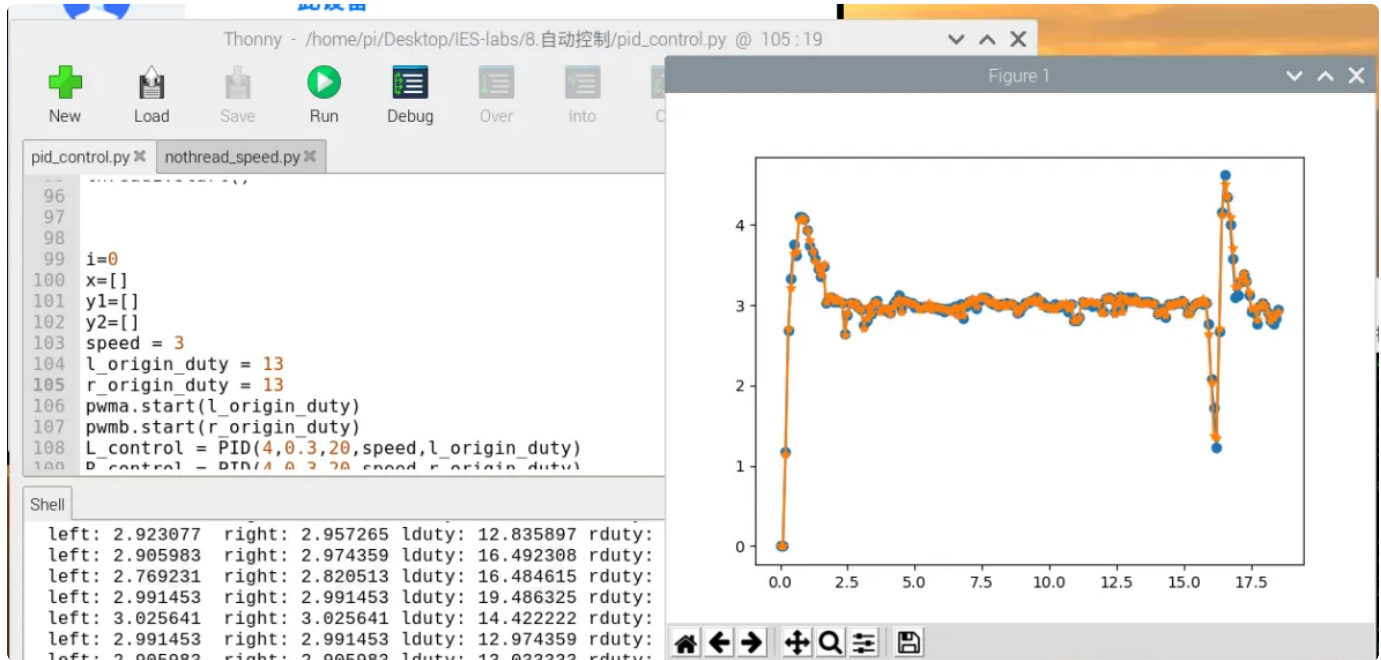


可以看到稳定时间非常长，已经完全足够

地面行走测试

这里我们调整了一点系数，可以看到地面有阻力的情况下第一次达到的最大速度降低了

左右轮的速度基本保持一致，中间我们施加了一个干扰，可以看到调整过程在两个轮子也基本一致



四.实验分析

1.实验代码

代码逻辑并不复杂，就是离散PID算法的代码化实现，这里就不再注释了

唯一要注意的就是上面提到的代码错误，将585.0改成58.5

2.P、I、D的各自作用是什么？

- P即Proportion(比例系数)，这个系数是最直接对误差进行调控的量。例如当误差 $e(t) = 0.8$ 时，假定比例系数设置为 $K_p = 0.8$ 时，P的调控会立马增加 $e(t) * K_p = 0.64$ 的误差修正，从而使得输出值接近设定值。完成一次PID操作后，会再返回 $e(t)$ 值，进行下一轮调控。
- I即Integration(积分系数)，这个系数的出现是为了消除"稳态误差"。"稳态误差"是对设备稳定误差的一种调节方法。假如P调控最后的调控值卡在某一个点。(例如:最后P调控 $K_p * e(t) = 0.04$ ，但是内部组件一直有 -0.08 的稳定损耗，会导致P调控在某一个阈值而无法增加，导致很均匀的误差

时)，I调控就需要被使用。I调控的基本依据是过去所有反馈回来的误差，即 $\sum_{i=1}^n e(i)$ ，积分系数

$K_i \neq 0$ 时，I调控会给予 $K_i * \sum_{i=1}^n e(i)$ 的调控，即收集稳态误差，在通过乘系数去打破稳态误差。

- D即Differential(微分系数), 这个系数的作用可以从手动调参的表格参考得来。表格说对于 K_d 的调整会使得安定时间减少, 但是波动会大幅减少。在离散PID控制中, D调控是离散的, 是基于 $e(t) - e(t - 1)$ 的调控当之前的调控超调时, $e(t) - e(t - 1) < 0$, 会立马进行一个约束的调控, 并且在D调控中, 由于有相对差量 $e(t) - e(t - 1)$ 的输入限制, 它的调整量会比P调控更加稳定。(在参数合理的情况下)故D调控最明显的作用就是可以很优秀的压制超调量, 运行中增加反馈值的稳定性。
- 由于不同系数, 不同的调控方式也会有连结的地方, 谈论它们的作用是基于定性研究的。

3.如果发现小车走不直, 如何确定问题和优化?

- 首先先排查代码层面的问题, 若基础代码没有大问题则可到下一步。
- 再排查器件本身的误差, 比如在同等PWM波占空比下, 两边的轮子是否转速相同, 若不同, 则要针对每一个轮子设计PWM波占空比。
- 其次就要开始考虑PID调控带来的波动。如果PID控制效果的安定时间和稳定性很高就不会在合理路径中出现"走不直"这种情况。所以还需要对PID参数进行调整, 调整方向具体是对D上升, 对P、I下降, 来减小波动。(波动过大会导致两轮的误差被放大, 导致路线蜿蜒曲折)

五.总结与思考

1.本次实验通过采用PID控制原理来让小车能够以给定的速度自动直行。在实验过程中, 调整相关参数相当重要(包括对代码的调整和对PID各系数的调整), 调整合适的参数和使用良好的测试方法(二分法)会让实验过程事半功倍。自动控制的实现也让小车自主移动成为了可能, 为之后接入图像、定轨行走提供了良好的方法基础。