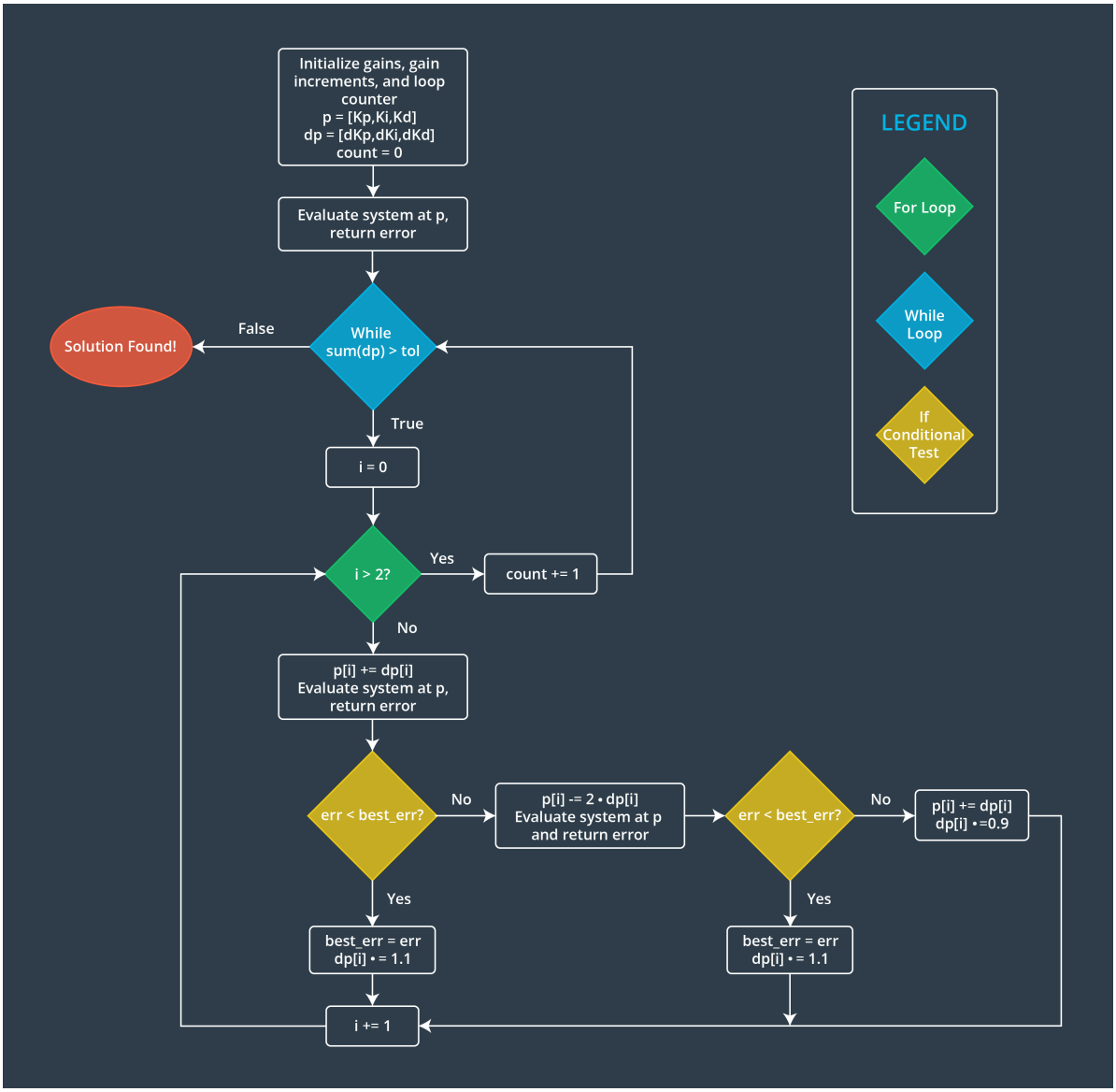
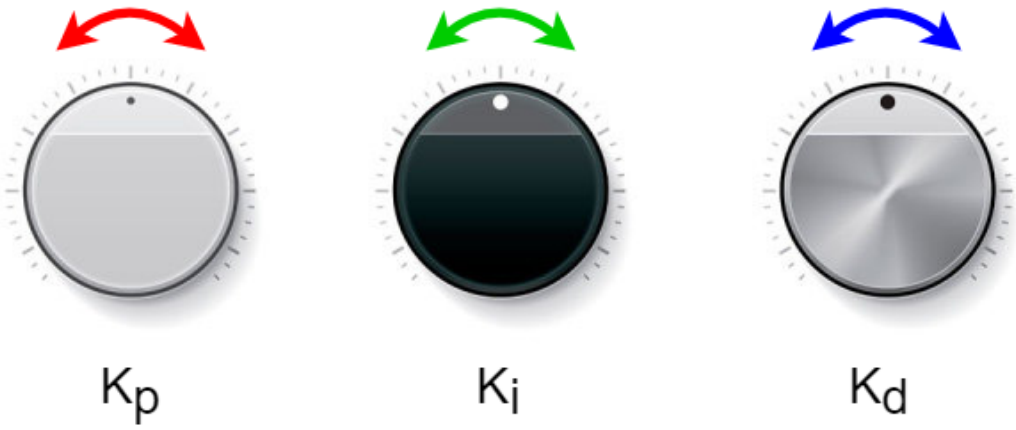
Twiddle 调校最优解

为了避免人为的猜测、试错来获取比较好的PID值，我们可以设计一种自动化的调参逻辑循环，根据使指定目标函数的值达到最小时，得到更加优秀的参数列表。

一种更智能、自动的方法是使用梯度下降算法。 前提是您从三个增益的初始猜测向量开始。 通常对P使用小的非零值，对I和D使用0。 然后，分别对每个收益进行小幅更改，然后测试目标函数是否降低。 如果降低了，将沿相同方向不断更改参数，否则尝试沿相反方向调整参数。 如果增益值的增加或减少均不会降低成本函数，则减小增益增量的大小并重复。 整个循环应继续进行，直到增量大小降至某个阈值以下。



我们可以把以下过程称之为 Twiddle（旋弄、捻弄），可以帮助我们确定合适的P、I、D三个值，其原理是构建一个初始化为 $[0, 0, 0]$ 的参数列表，然后循环多次，根据每次循环返回的误差均值，调大或调小参数（调整的步伐参数由另一个3个值的列表控制）。直到步伐参数之和小于一定阈值时，停止循环，把此时参数列表作为最终的PID参数。



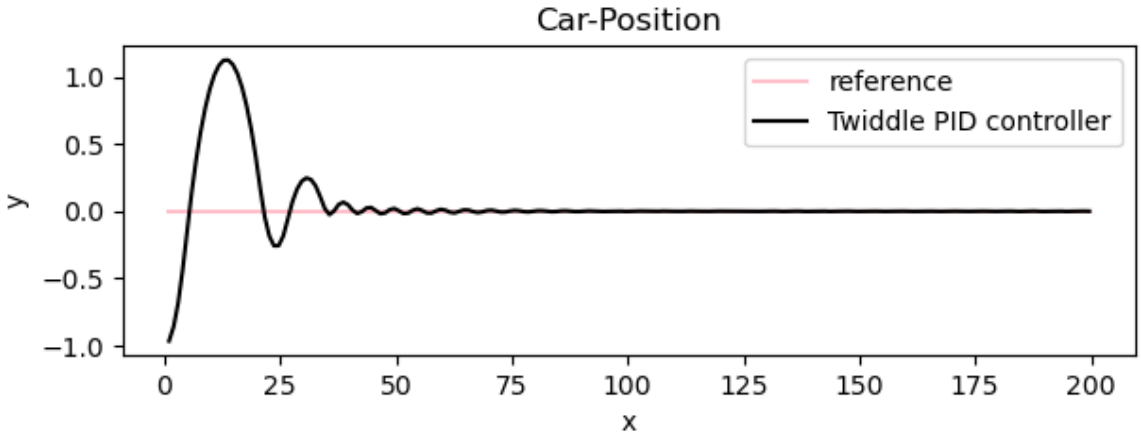
代码实现：

1. 拷贝依赖文件 robot.py 到项目目录
2. 新建文件 `pid_twiddle.py` 并编写如下内容：

```
from robot import Robot, show
import numpy as np

def make_robot():
    """
    创建并初始化机器人小车，设置初始位置为(0, -1)，初始旋转角度为0
    """
    robot = Robot()
    robot.set(0, -1, 0)
    robot.set_steering_drift(10 / 180 * np.pi)
    return robot


# NOTE: We use params instead of k_p, k_d, k_i
def run(robot, params, n=100, speed=1.0):
    x_trajectory = []
    y_trajectory = []
    err = 0
    prev_cte = 0 - robot.y
    int_cte = 0
    for i in range(2 * n):
        cte = 0 - robot.y
        diff_cte = cte - prev_cte
        int_cte += cte
        prev_cte = cte

        steer = params[0] * cte + params[1] * diff_cte + params[2] * int_cte

        robot.move(steer, speed)
        x_trajectory.append(robot.x)
        y_trajectory.append(robot.y)
        if i >= n:
            err += cte ** 2
    return x_trajectory, y_trajectory, err / n
```

```
# Make this tolerance bigger if you are timing out!
def twiddle(tol=0.2):
    p = [0, 0, 0]
    dp = [1, 1, 1]
    robot = make_robot()
    x_trajectory, y_trajectory, best_err = run(robot, p)

    it = 0
    while sum(dp) > tol:
        # 循环，直到系数之和小于等于阈值（默认阈值为0.2，起始值为3.0）
        print("Iteration {}, best error = {}".format(it, best_err))
        for i in range(len(p)):
            p[i] += dp[i]
            robot.reset()
            x_trajectory, y_trajectory, err = run(robot, p)

            if err < best_err:
                best_err = err
                dp[i] *= 1.1     # 此值有助于减少总误差，下次可以多加点
            else:
                p[i] -= 2 * dp[i]# 此值不利于于减少总误差，直接把刚加的去掉，并且向反方向
减一倍
                robot.reset()
                x_trajectory, y_trajectory, err = run(robot, p)

                if err < best_err:
                    best_err = err
                    dp[i] *= 1.1 # 如果反方向有利于减少总误差，扩大此值
                else:
                    p[i] += dp[i] # 恢复为原值
                    dp[i] *= 0.9  # 把缩小变化系数
        it += 1
    return p, best_err


params, err = twiddle()
print("Final twiddle error = {} params = {}".format(err, params))
robot = make_robot()
x_trajectory, y_trajectory, err = run(robot, params)

show(x_trajectory, y_trajectory, label="Twiddle PID")
```

运行结果：



可以与之前使用的PID进行对比，明显可以看出当前PID在x=50附近就开始在目标轨迹附近稳定下来。而之前的要到x=100的位置才可以稳定下来。

输出日志：

根据最后的输出日志可知：

最小的误差均值为 `7.940560962605189e-07`

此时对应的P、D、I分别为： `10.716018504541431, 18.68325573584582, 0.020275559590445292`

```
Iteration 0, best error = 7972.071547906822
Iteration 1, best error = 0.048853806107299856
Iteration 2, best error = 0.03026214567061226
Iteration 3, best error = 0.0077046028132098255
Iteration 4, best error = 0.003222969736312333
Iteration 5, best error = 0.0016693580238629137
Iteration 6, best error = 0.0009763548793623677
Iteration 7, best error = 0.0006143945322198215
Iteration 8, best error = 0.0006143945322198215
Iteration 9, best error = 0.0006143945322198215
Iteration 10, best error = 0.0006143945322198215
Iteration 11, best error = 0.0006143945322198215
Iteration 12, best error = 0.0006143945322198215
Iteration 13, best error = 0.0006143945322198215
Iteration 14, best error = 0.0006143945322198215
Iteration 15, best error = 0.0006143945322198215
Iteration 16, best error = 0.0006143945322198215
Iteration 17, best error = 0.0006143945322198215
Iteration 18, best error = 0.000612580641120018
Iteration 19, best error = 0.0005393890590252054
Iteration 20, best error = 0.0004707073557690764
Iteration 21, best error = 0.00040854195467355256
Iteration 22, best error = 0.00036173191894990056
Iteration 23, best error = 0.0003496794660242756
Iteration 24, best error = 0.00030372592629584626
Iteration 25, best error = 0.00026649497706284647
Iteration 26, best error = 0.00026502850791677363
Iteration 27, best error = 0.00023267758106029443
Iteration 28, best error = 0.00023267758106029443
```

```
Iteration 29, best error = 0.00023267758106029443
Iteration 30, best error = 0.00023267758106029443
Iteration 31, best error = 0.00023267758106029443
Iteration 32, best error = 0.00023222333181571606
Iteration 33, best error = 0.00023222333181571606
Iteration 34, best error = 0.00023222333181571606
Iteration 35, best error = 0.00023222333181571606
Iteration 36, best error = 0.00023222333181571606
Iteration 37, best error = 0.00022978905387007294
Iteration 38, best error = 0.00019428921416175855
Iteration 39, best error = 0.00014919774411248834
Iteration 40, best error = 2.2810673236127803e-05
Iteration 41, best error = 6.65198948619874e-06
Iteration 42, best error = 8.957452817602662e-07
Iteration 43, best error = 8.957452817602662e-07
Iteration 44, best error = 8.957452817602662e-07
Iteration 45, best error = 8.957452817602662e-07
Iteration 46, best error = 8.957452817602662e-07
Iteration 47, best error = 8.957452817602662e-07
Iteration 48, best error = 8.957452817602662e-07
Iteration 49, best error = 8.957452817602662e-07
Iteration 50, best error = 8.957452817602662e-07
Iteration 51, best error = 7.940560962605189e-07
Iteration 52, best error = 7.940560962605189e-07
Iteration 53, best error = 7.940560962605189e-07
Iteration 54, best error = 7.940560962605189e-07
Iteration 55, best error = 7.940560962605189e-07
Iteration 56, best error = 7.940560962605189e-07
Iteration 57, best error = 7.940560962605189e-07
Final twiddle error = 7.940560962605189e-07 params = [10.716018504541431,
18.68325573584582, 0.020275559590445292]
```