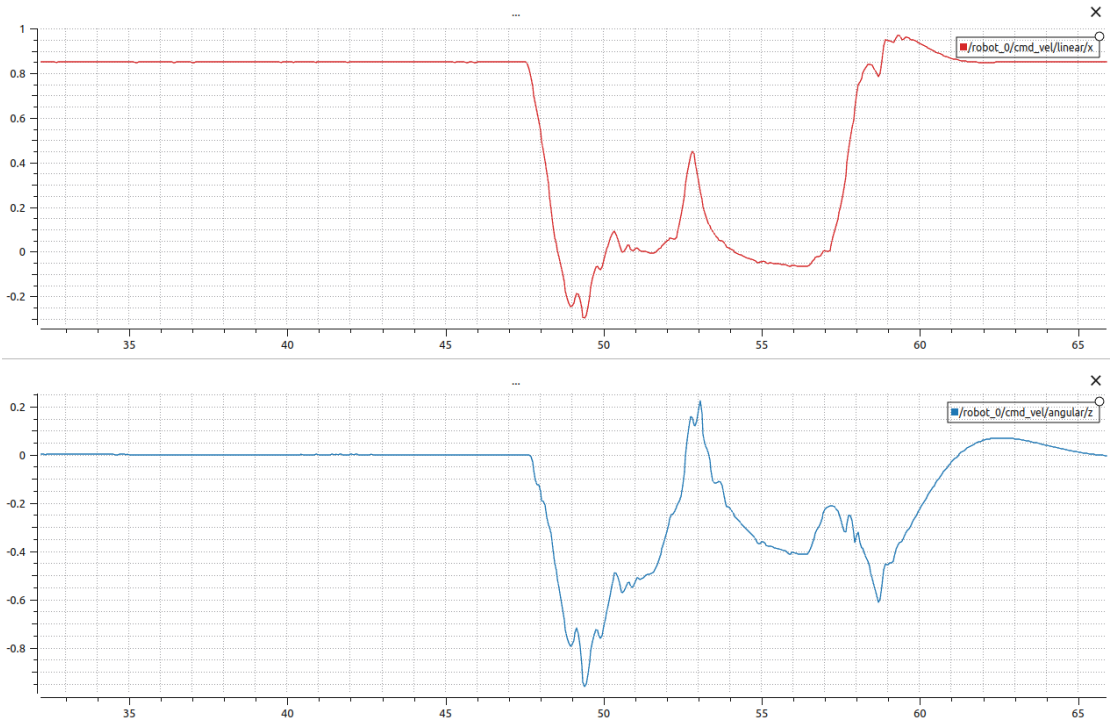
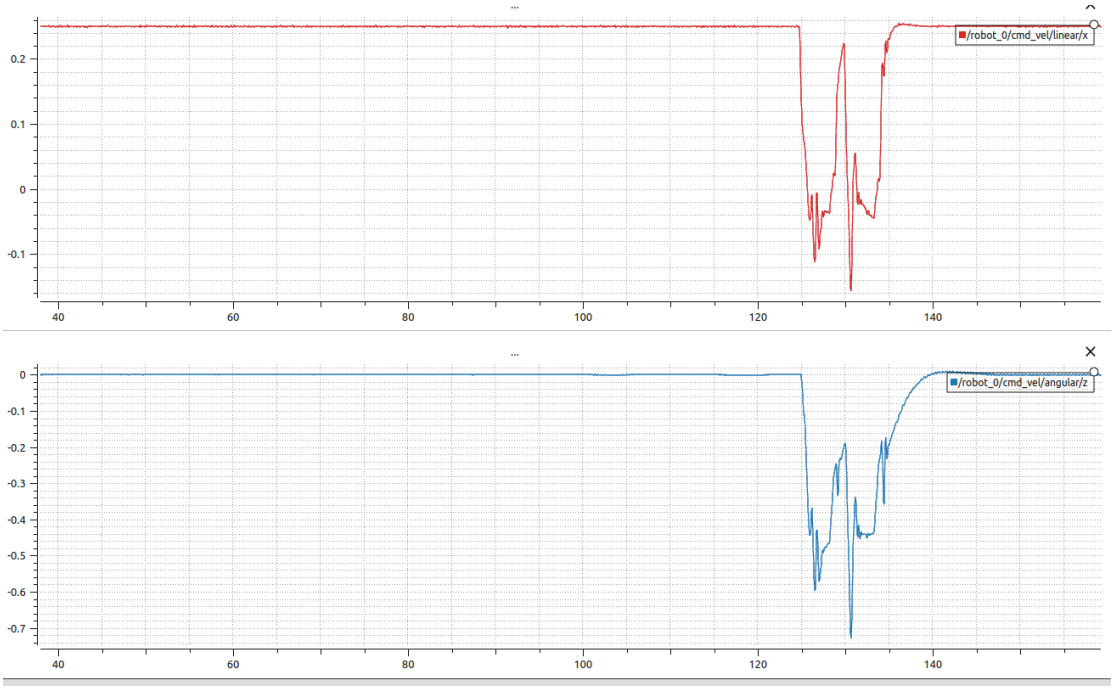
全覆盖跟踪中的差值处理以及常见控制方法研究

1. 对与全覆盖中采用的全覆盖的 pid 控制方法分析



全覆盖中在拐弯时的仿真速度和角度曲线

从图中可以看到速度会增到到超过设定的 0.85 的值，在拐弯时乖的角度并不是很丝滑。可以作为后面的优化项来处理。

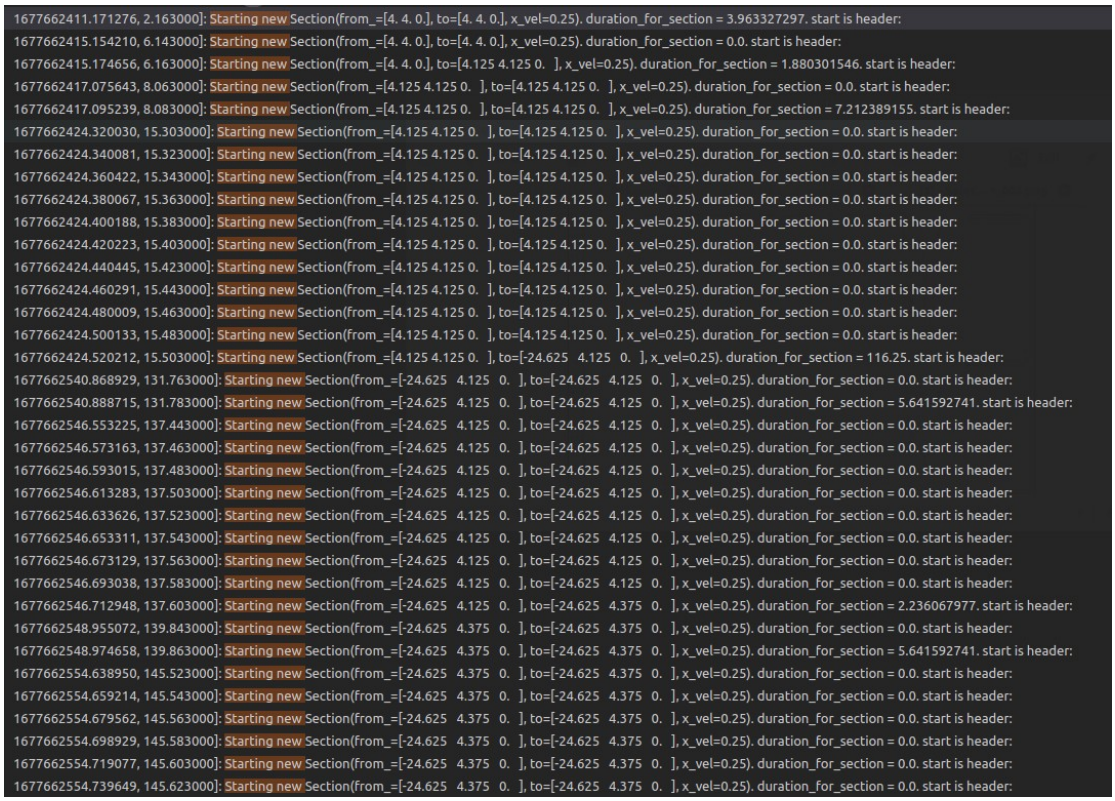


将速度设置为 0.25 时，同样在拐弯时拐的不是很丝滑，但是明显曲线不会有很大的超调，同时角度曲线也更加平滑。这里的速度会出现负值，说明还是有倒车现象发生的，从仿真中比较难观察到此现象。

2. 全覆盖中的差值处理

在全覆盖得到全局的 path 之后，会给当前车发一个要去跟踪的 pose 点，这个点中是如何处理的?

当得到一条 path 时，path/via 表示的是规划出来的整张图上面的路径，在拿到整体的 path 之后，根据频率在间隔一定时间之后，去调用_update_target 函数去更新车身前的目标点，通过触发事件(时间戳)来执行,如果是角度偏差，根据速度计算出一个持续时间



>>> duration_on_section = (current_time - section_start_time)

>>> duration_for_section = (length_of_section / target_(x_yaw)velocity)

>>> progress_on_section = (duration_on_section / duration_for_section)

>>> target_x = start_x + delta_x * progress_on_section

设定的速度值，最终会通过这 4 个公式来计算出 target_x，所以这里的 target_x 里面就含有了设定的速度值的信息，这样在发送给 pid 进行控制时就只需要发目标点的 pose 信息，通过 pid 控制这个 target 的 pose，便能够输出对应的 cmd_vel，使得下发给底盘的 cmd_vel 始终在设定的速度值附近。

根据 progress_on_section 的范围是 0 到 1，所以根据时间能够得到每一帧要跟随的目标点。

在代码中的指定 topic,”trajectory”发送 traj_point 给控制作为目标点，发送的这个 traj_point 里面是带有速度信息的，但是其实 controller 在接收到这个点时只对这个点的 pose 信息进行筛选，至于里面带的速度信息并没有使用。如下图所示：

```
[ INFO] [1677668177.409000479, 5764.267000000]: the waypoint is :time:
  data: 0
controller:
  data: 0
pose:
  header:
    seq: 24761
    stamp: 5764.263000000
    frame_id: map
  pose:
    position:
      x: 0.69375
      y: 10.875
      z: 0
    orientation:
      x: 0
      y: 0
      z: 0
      w: 1
velocity:
  linear:
    x: 0.25
    y: 0
    z: 0
  angular:
    x: 0
    y: 0
    z: 0
acceleration:
  linear:
    x: 0
    y: 0
    z: 0
  angular:
    x: 0
    y: 0
    z: 0
```

3. 运动控制和全覆盖代码中 pid 设计

运动控制的一些概念：

机器人的运动控制主要分为有三个方面，路径跟踪，轨迹跟踪 ，点位运动/点镇定，路径跟踪的话是不考虑时间约束的，轨迹跟踪的话需要考虑时间因素，其中轨迹跟踪的话是最复杂的，其中考虑时间约束，然后考虑的模型的话也要更加复杂一些。

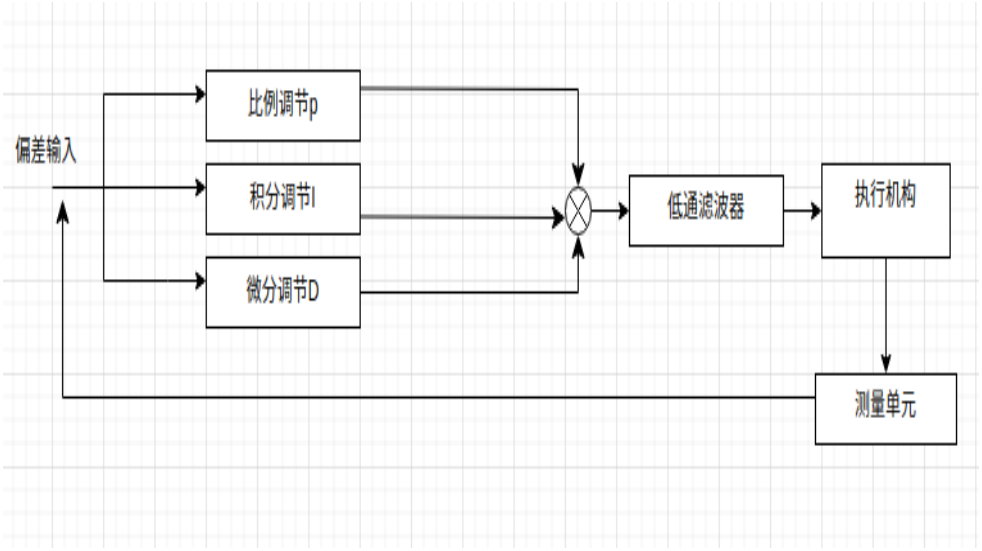
pure pursuit 的话其实就是一个路径跟踪的问题，以车的预瞄距离为半径，然后与路径的焦点为预瞄点，计算出一个 yaw 的值，之后从定位拿到车自身的姿态的 yaw 信息，然后做差值，计算出需要下发给底盘的 yaw,因为是路径跟踪，所以这里直接给底盘下发一个指定的线速度即可。这种方式相对来说比较常见，因为将线速度和 yaw 进行分开来控制，没有耦合发生。这类方法的缺点是：当偏差大于预瞄距离时无解，跟踪曲线时有稳态误差，曲率越大，残差越大，预瞄距离的选择，预瞄太远过于平滑，预瞄太近不稳定，速度越高越不稳定。

轨迹跟踪，轨迹跟踪是控制无人车沿着给定轨迹走，该轨迹定义在时间域上。跟踪过程中选择的目标点是有当前时刻的时标确定的，需要同时控制无人车的横向和纵向，以趋近该目标点。轨迹跟踪的效果是，车贴着给定的轨迹走，并且所用的时间趋近于参考轨迹的时长。

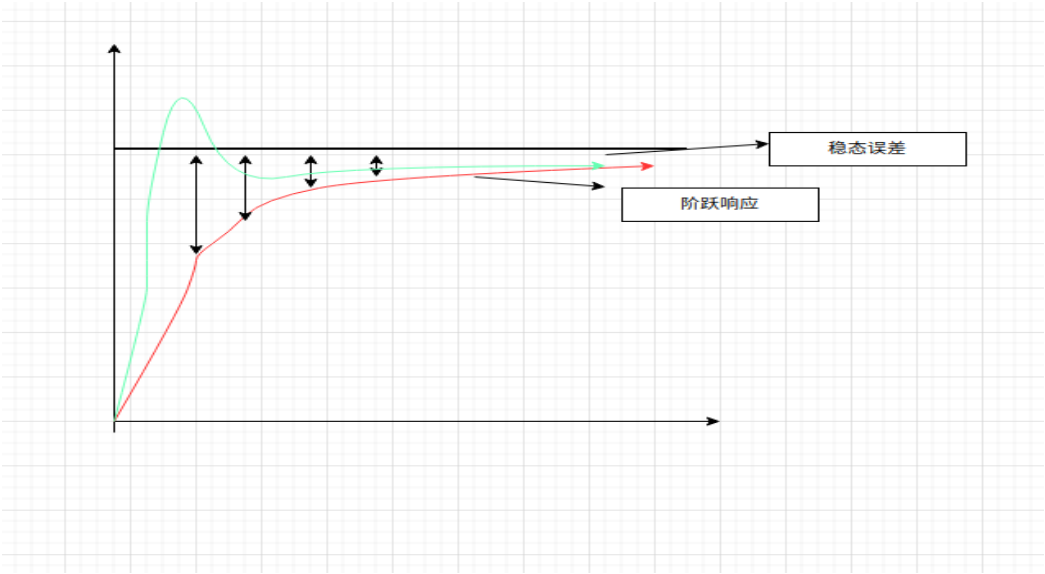
利用点镇定来做跟踪任务时，由于其路径是由点列组成，将前方距离最近的一个点设为目标点，在自车到达其邻域后切换到下一个目标点。与预瞄法类似，但是又有所不同，点镇定是横纵向同时控制，预瞄法只做横向控制。点镇定会同时约束车体抵达目标点的位置和方向，预瞄法只约束抵达位置。有两点因素是要考虑的，第一点是因为点镇定是横纵向同时控制，所以点镇定在追点过程中会调节线速度，每切换下一点，该调节会重复，容易造成速度波动。为了速度的平稳，将点列差值，加大密度。每次迭代时，插值生成一个更新的目标点，这样线速度可维持不变。第二点是由于点镇定对抵达目标点的方向也有约束，所以选择目标点时，距离取近一些也不会像预瞄法那样不稳定。

目前全覆盖这套代码中所用的 pid 是带有耦合的，将设定的速度值信带入到了下发给 pid 控制器中，其目标点信息中含有了速度信息，所以在设计 pid 控制器时就不用考虑速度的设计，按照位置的横向、纵向、yaw 来分别进行 pid 的控制之后做个滤波的处理，之后按照是否添加前馈处理从而添加新的元素给输出，最终将计算出的结果下发到底盘进行控制。

pid 中的基本原理介绍：



基本 PID 控制器的架构



阶跃响应曲线调节对应的参数

针对全覆盖中的源码的一些理解：

一、 pid 算法的输入 1.waypoint 2.根据 tf 得到当前车在 map 中的姿态信息 3.采样时间 4.pid 控制调试显示的 debug 信息。

二、 在获取到当前车的姿态之后，因为车是一个质点，但是车的质心距离车头是有一段距离的，所以这时想要计算，当前车距离下一个目标点的距离，应该是车头距离下一个目标点的距离。所以需要对车当前的位置进行更新，需要考虑车的质心到车头的距离 l。所以会有

newOrigin.setX(current.translation.x + l * cos(theta_rp));

newOrigin.setY(current.translation.y + l * sin(theta_rp));

三、 track_base_link_enabled 这个标志位的意思是用来表示的是否考虑车的质心到车头的距离如果考虑则目标点为：

newGoalOrigin.setX(goal.getOrigin().x() + l * cos(theta_goal));

newGoalOrigin.setY(goal.getOrigin().y() + l * sin(theta_goal));

四、 计算当前点和目标点之间的误差：

tf::Transform error = newCurrent.inverseTimes(newGoal);

得到当前车和目标点的位姿差值，对 3 个方向使用 pid 进行控制，分别是纵向、横向、和角度，构造一个大小为 3 的 vectory 用来滤波计算使用，在设计时，考虑了调试时需要的必要信息利用 pid_debug 来获取。

五、 纵向、横向、角度 的 pid 控制：

获取纵向积分、横向积分、角度积分值，然后对其进行限幅处理

之后对纵向、横向、角度基于截止频率做个敞亮的参数计算分贝得到

c_long、c_lat、c_ang

基于 error_long 和计算得出的 c_long 得出 filtered_error_long 的值计算公式为：

```
filtered_error_long.at(0) = (1 / (1 + c_long * c_long + 1.414 *
c_long)) *
(error_long.at(2) + 2 * error_long.at(1) + error_long.at(0) -
(c_long * c_long - 1.414 * c_long + 1) *
filtered_error_long.at(2) -
(-2 * c_long * c_long + 2) * filtered_error_long.at(1));
```

横向和角度的滤波处理是类似的

之后分别求 横向、纵向、以及角度偏差的倒数，做个滤波处理公式同上

之后计算纵向、横向、角度的比例项、积分项、微分项

在这个基础上在计算控制力时分了不同的情况进行控制力的计算：

分别是纵向反馈、增加了纵向前馈控制、横向反馈控制、增加了横向前馈控制、角度控制、增加了前馈角度控制,目前程序中默认除了前馈的角度控制没有开启外，其余的控制力的输出都是开启的，前馈控制的力的计算目前是屏蔽掉的，所以目前代码中并没有使用前馈控制。

对算出来的纵向、横向、以及角度力进行限制

其中几个重要的参数设置：

coupling_ang_long（bool）: 是否启用角度和纵向控制器之间的耦合

`dead_zone_yaw_error_cal(float)`: 角度控制器和纵向控制器之间航向角误差死区

max_yaw_error_cal：角度控制器和纵向控制器之间耦合的最大航向偏差

针对差速车，分别对线速度和角度进行赋值则得到最终下发给底盘的速度，驱动即可。