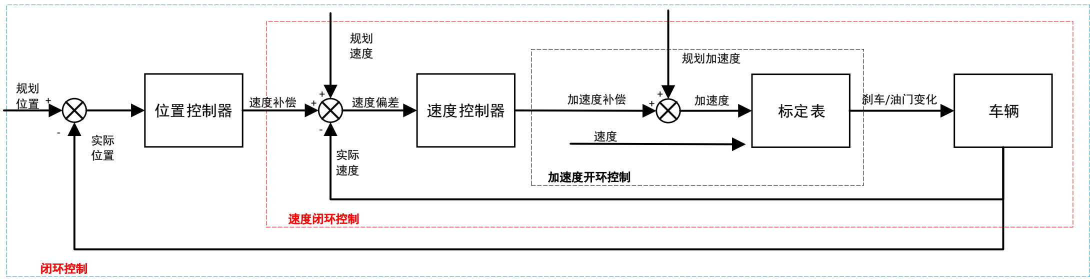
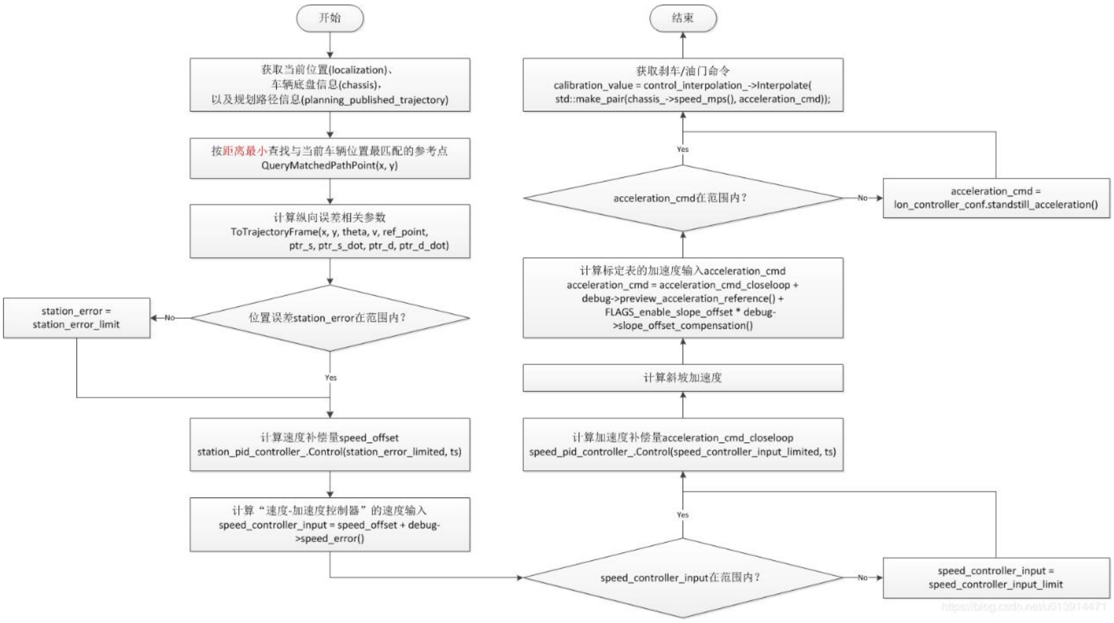
1. 纵向控制

纵向控制主要为速度控制，通过控制刹车、油门、档位等实现对车速的控制，对于自动挡车辆来说，控制对象其实就是刹车和油门。

Apollo 纵向控制如下图所示，主要是两个 PID 串级控制，外环是位置环，中间是速度环，最里面是加速度的开环控制



流程图如下：



1.1 计算误差：

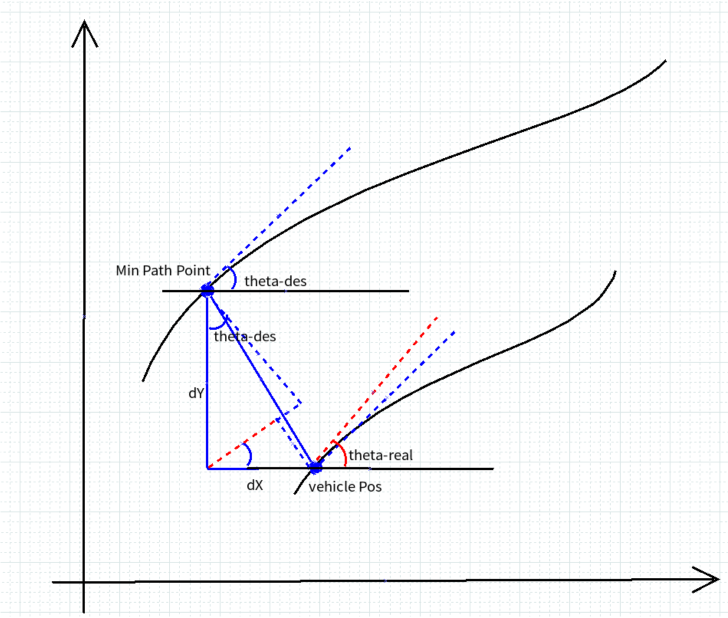
1. 位置误差

车辆的位置误差是在 frenet 坐标系下进行计算的，主要由 s、d 两个方向的误差，根据车辆当前位置与期望轨迹，计算得到距离车辆最近的期望点 MinPathPoint，并计算出该点的方向 θ_{des} 与曲率。

S 方向： $station\_error = dx * \cos\theta_{des} + dy * \sin\theta_{des}$

d 方向： $cross\_rd\_nd = \cos\theta_{des} * dy - \sin\theta_{des} * dx$

计算得到的位置误差： $station\_error = -(dx * \cos\theta_{des} + dy * \sin\theta_{des})$



2. 速度误差

$$speed\_error = V_{des} - V * \cos\Delta\theta / k$$

其中，Vdes 为期望车辆线速度，V 为当前车辆线速度，Δθ 为航向误差，k 为系数，即代码中的 one_minus_kappa_r_d

求得位移误差和速度误差后，结合“位移-速度闭环 PID 控制器”和“速度-加速度闭环 PID 控制器”，求得刹车/油门标定表的两个输入量：速度和加速度，利用插值计算在标定表中查找相应的控制命令值。

Apollo 所用的 PID 控制器就是一般的位置型 PID 控制器：

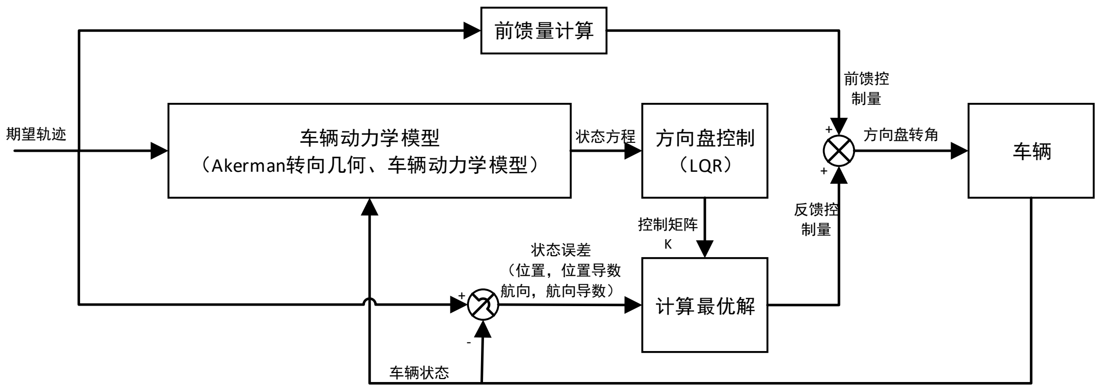
$$u(k) = K_P e(k) + K_I \sum_{i=0}^{k} e(i) + K_D[e(k) - e(k-1)] + u(0)$$

代码截图如下：

```cpp
// 计算横纵向误差
ComputeLongitudinalErrors(trajectory_analyzer_.get(), preview_time, debug);

double station_error_limit = lon_controller_conf.station_error_limit();
double station_error_limited = 0.0;
if (FLAGS_enable_speed_station_preview) {
  station_error_limited =
      common::math::Clamp(debug->preview_station_error(),
                          -station_error_limit, station_error_limit);
} else {
  station_error_limited = common::math::Clamp(
      debug->station_error(), -station_error_limit, station_error_limit);
}
// 配置PID参数
if (trajectory_message_->gear() == canbus::Chassis::GEAR_REVERSE) {
   station_pid_controller_.SetPID( lon_controller_conf.reverse_station_pid_conf());
   speed_pid_controller_.SetPID(lon_controller_conf.reverse_speed_pid_conf());
} else if (VehicleStateProvider::Instance()->linear_velocity() <=
           lon_controller_conf.switch_speed()) {
  speed_pid_controller_.SetPID(lon_controller_conf.low_speed_pid_conf());
} else {
  speed_pid_controller_.SetPID(lon_controller_conf.high_speed_pid_conf());
}
```

```cpp
// 位置PID控制环，主要是计算期望速度的偏置 speed_offset
double speed_offset = station_pid_controller_.Control(station_error_limited, ts);

double speed_controller_input = 0.0;
double speed_controller_input_limit = lon_controller_conf.speed_controller_input_limit();
double speed_controller_input_limited = 0.0;
if (FLAGS_enable_speed_station_preview) {
  speed_controller_input = speed_offset + debug->preview_speed_error();
} else {
  speed_controller_input = speed_offset + debug->speed_error();
}
speed_controller_input_limited =
 common::math::Clamp(speed_controller_input, -speed_controller_input_limit,
                       speed_controller_input_limit);

double acceleration_cmd_closeloop = 0.0;
// 速度PID控制环，主要是计算期望减速度 acceleration_cmd_closeloop
acceleration_cmd_closeloop = speed_pid_controller_.Control(speed_controller_input_limited, ts);

double slope_offset_compenstaion = digital_filter_pitch_angle_.Filter(
    GRA_ACC * std::sin(VehicleStateProvider::Instance()->pitch()));

if (isnan(slope_offset_compenstaion)) {
  slope_offset_compenstaion = 0;
}

debug->set_slope_offset_compensation(slope_offset_compenstaion);
// 计算期望减速度  期望减速度 = PID计算得到的减速度 + 轨迹规划得到的减速度 + 道路坡度的减速度补偿
double acceleration_cmd =
    acceleration_cmd_closeloop + debug->preview_acceleration_reference() +
    FLAGS_enable_slope_offset * debug->slope_offset_compensation();
debug->set_is_full_stop(false);
GetPathRemain(debug);
```

横向控制



前馈量计算
期望轨迹
车辆动力学模型
（Akerman转向几何、车辆动力学模型）
状态方程
方向盘控制
（LQR）
前馈控
制量
+
+
方向盘转角
车辆
反馈控
制量
控制矩阵
K
状态误差
（位置，位置导数
航向，航向导数）
+
-
计算最优解
车辆状态