【运动控制】Apollo6.0的lon_controller解析

隐匿潮水之中
其修远路漫，而上下求索。

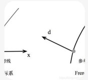
已关注

lon_controller解析

目录

对Apollo 6.0的纵向控制模块解析。

【运动控制】Apollo6.0的lon_controller解析_yanghq13的博客-CSDN博客
blog.csdn.net/weixin_44041199/article/detai



纵向控制算法的步骤：

1. 创建一个纵向控制器
2. 在文件 `control_config` 中添加纵向控制器的配置信息
3. 注册纵向控制器

1 创建纵向控制器

所有的控制器都必须继承基类 `Controller`，它定义了一组接口，以下是纵向控制器实现的实例：

```
class LonController : public Controller {
 public:
  /**
   * @brief constructor
   */
  LonController();

  /**
   * @brief destructor
   */
  virtual ~LonController();

  /**
   * @brief initialize Longitudinal Controller
   * @param control_conf control configurations
   * @return Status initialization status
   */
  common::Status Init(std::shared_ptr<DependencyInjector> injector,
                      const ControlConf *control_conf) override;

  /**
   * @brief compute brake / throttle values based on current vehicle st
   *        and target trajectory
   * @param localization vehicle location
```



```
   * @return Status computation status
```

```
   */
  common::Status ComputeControlCommand(
      const localization::LocalizationEstimate *localization,
      const canbus::Chassis *chassis, const planning::ADCTrajectory *tr
      control::ControlCommand *cmd) override;

  /**
   * @brief reset longitudinal controller
   * @return Status reset status
   */
  common::Status Reset() override;

  /**
   * @brief stop longitudinal controller
   */
  void Stop() override;

  /**
   * @brief longitudinal controller name
   * @return string controller name in string
   */
  std::string Name() const override;

 protected:
  void ComputeLongitudinalErrors(const TrajectoryAnalyzer *trajectory,
                                 const double preview_time, const doubl
                                 SimpleLongitudinalDebug *debug);

  void GetPathRemain(SimpleLongitudinalDebug *debug);

 private:
  void SetDigitalFilterPitchAngle(const LonControllerConf &lon_controll

  void LoadControlCalibrationTable(
      const LonControllerConf &lon_controller_conf);

  void SetDigitalFilter(double ts, double cutoff_freq,
                        common::DigitalFilter *digital_filter);

  void CloseLogFile();https://blog.csdn.net/weixin_44041199

  const localization::LocalizationEstimate *localization_ = nullptr;
  const canbus::Chassis *chassis_ = nullptr;

  std::unique_ptr<Interpolation2D> control_interpolation_;
  const planning::ADCTrajectory *trajectory_message_ = nullptr;
  std::unique_ptr<TrajectoryAnalyzer> trajecf21d198d172d63251f52687a56f

  std::string name_;
  bool controller_initialized_ = false;

  double previous_acceleration_ = 0.0;
  double previous_acceleration_reference_ = 0.0;

  PIDController speed_pid_controller_;
  PIDController station_pid_controller_;

  LeadlagController speed_leadlag_controller_;
```

```
  LeadlagController station_leadlag_controller_;

  FILE *speed_log_file_ = nullptr;

  common::DigitalFilter digital_filter_pitch_angle_;

  const ControlConf *control_conf_ = nullptr;

  // vehicle parameter
  common::VehicleParam vehicle_param_;

  std::shared_ptr<DependencyInjector> injector_;
};https://blog.csdn.net/weixin_44041199
}  // namespace control
}  // namespace apollo
```

下面来分别解析

1.1 构造函数

构造函数 `LonController` 判断 `FLAGS_enable_csv_debug` 是否成功， `FLAGS_enable_csv_debug` `control_gflags.cc` 里面定义的是 `True to write out csv debug file` ，即是否正确写出csv debug文件 `speed_log_file_` 。

```
LonController::LonController()
    : name_(ControlConf_ControllerType_Name(ControlConf::LON_CONTROLLEF
  if (FLAGS_enable_csv_debug) {
    time_t rawtime;
    char name_buffer[80];
    std::time(&rawtime);
    std::tm time_tm;
    localtime_r(&rawtime, &time_tm);
    strftime(name_buffer, 80, "/tmp/speed_log__%F_%H%M%S.csv", &time_tm
    speed_log_file_ = fopen(name_buffer, "w");
    if (speed_log_file_ == nullptr) {
      AERROR << "Fail to open file:" << name_buffer;
      FLAGS_enable_csv_debug = false;
    }
    if (speed_log_file_ != nullptr) {
      fprintf(speed_log_file_,
              "station_reference,"https://blog.csdn.net/weixin_44041199
              "station_error,"
              "station_error_limited,"
              "preview_station_error,"
              "speed_reference,"
              "speed_error,"
              "speed_error_limited,"
              "preview_speed_reference,"
              "preview_speed_error,"
              "preview_acceleration_reference,"
              "acceleration_cmd_closeloop,"
              "acceleration_cmd,"
              "acceleration_lookup,"
              "speed_lookup,"
              "calibration_value,"
              "throttle_cmd,"
              "brake_cmd,"
```

```
                "is_full_stop,"
                "\r\n");

      fflush(speed_log_file_);
    }
    AINFO << name_ << " used.";
  }
}
DEFINE_bool(enable_csv_debug, false, "True to write out csv debug file.
```

1.2 析构函数

析构函数里面只调用了一个 `CloseLogFile()` 函数，即关闭 `speed_log_file_`，

```
LonController::~LonController() { CloseLogFile(); }
```

`CloseLogFile()` 的函数实现如下

```
void LonController::CloseLogFile() {
  if (FLAGS_enable_csv_debug) {
    if (speed_log_file_ != nullptr) {
      fclose(speed_log_file_);
      speed_log_file_ = nullptr;
    }
  }
}
```

1.3 初始化函数

`LonController::Init` 进行初始化纵向控制器，调用 `control_conf` 控制参数配置，返回初始化状态 `Status` 。流程如下

- Input：控制算法参数配置 `control_conf`
- Output：初始化状态 `Status`
- 判断控制算法参数配置 `control_conf` ，若为空，放回初始化状态false；
- 读取 `control_conf.pb.txt` 中的 `Lon_controller_conf` 的参数，如下
- 时间 `ts` ；
- leadlag使能 `enable_leadlag` ；
- 执行station_pid_controller_.Init()，位置pid控制器初始化；
- 执行speed_pid_controller_.Init()，速度pid控制器初始化；
- 根据 `enable_leadlag` 来判断是否执行station_leadlag_controller_.Init()，超前滞后控制器初始化；
- 根据 `canbus_conf.pb.txt` ，执行vehicle_param_.CopyFrom()，配置车辆参数；
- 执行SetDigitalFilterPitchAngle()，设置俯仰角数字滤波器；
- 执行LoadControlCalibrationTable()，加载控制标定参数表；

初始化函数调用关系如下：

初始化函数主要实现功能：

- 判断初始化是否成功；
- 初始化位置PID和速度PID控制器；
- 设置俯仰角滤波器；
- 加载控制标定参数表。

```
Status LonController::Init(std::shared_ptr<DependencyInjector> injector
                           const ControlConf *control_conf) {
  control_conf_ = control_conf;
  if (control_conf_ == nullptr) {
    controller_initialized_ = false;
    AERROR << "get_longitudinal_param() nullptr";
    return Status(ErrorCode::CONTROL_INIT_ERROR,
                  "Failed to load LonController conf");
  }
  injector_ = injector;
  const LonControllerConf &lon_controller_conf =
      control_conf_->lon_controller_conf();
  double ts = lon_controller_conf.ts();
  bool enable_leadlag =
      lon_controller_conf.enable_reverse_leadlag_compensation();

  station_pid_controller_.Init(lon_controller_conf.station_pid_conf());
  speed_pid_controller_.Init(lon_controller_conf.low_speed_pid_conf());

  if (enable_leadlag) {
    station_leadlag_controller_.Init(
        lon_controller_conf.reverse_station_leadlag_conf(), ts);
    speed_leadlag_controller_.Init(
        lon_controller_conf.reverse_speed_leadlag_conf(), ts);
  }

  vehicle_param_.CopyFrom(
      common::VehicleConfigHelper::Instance()->GetConfig().vehicle_para

  SetDigitalFilterPitchAngle(lon_controller_conf);

  LoadControlCalibrationTable(lon_controller_conf);
  controller_initialized_ = true;

  return Status::OK();
}
```

PID控制器初始化

```
void PIDController::Init(const PidConf &pid_conf) {
  previous_error_ = 0.0;
  previous_output_ = 0.0;
  integral_ = 0.0;
  first_hit_ = true;
  integrator_enabled_ = pid_conf.integrator_enable();
  integrator_saturation_high_ =
      std::fabs(pid_conf.integrator_saturation_level());
  integrator_saturation_low_ =
      -std::fabs(pid_conf.integrator_saturation_level());
  integrator_saturation_status_ = 0;
  integrator_hold_ = false;
  output_saturation_high_ = std::fabs(pid_conf.output_saturation_level(
  output_saturation_low_ = -std::fabs(pid_conf.output_saturation_level(
  output_saturation_status_ = 0;
  SetPID(pid_conf);
}
```

超前滞后控制器初始化

```
void LeadlagController::Init(const LeadlagConf &leadlag_conf, const dou
  previous_output_ = 0.0;
  previous_innerstate_ = 0.0;
  innerstate_ = 0.0;
  innerstate_saturation_high_ =
      std::fabs(leadlag_conf.innerstate_saturation_level());
  innerstate_saturation_low_ =
      -std::fabs(leadlag_conf.innerstate_saturation_level());
  innerstate_saturation_status_ = 0;
  SetLeadlag(leadlag_conf);
  TransformC2d(dt);
}
```

车辆参数

```
void TrackedObject::CopyFrom(TrackedObjectPtr rhs, bool is_deep) {
  *this = *rhs;
  if (is_deep) {
    object_ptr = base::ObjectPool::Instance().Get();
    *object_ptr = *(rhs->object_ptr);
  } else {
    object_ptr = rhs->object_ptr;
  }
}
```

数字滤波器

```
void LonController::SetDigitalFilterPitchAngle(
    const LonControllerConf &lon_controller_conf) {
  double cutoff_freq =
      lon_controller_conf.pitch_angle_filter_conf().cutoff_freq();
  double ts = lon_controller_conf.ts();
  SetDigitalFilter(ts, cutoff_freq, &digital_filter_pitch_angle_);
}
void LonController::SetDigitalFilter(double ts, double cutoff_freq,
```

```
                                      common::DigitalFilter *digital_fil
  std::vector<double> denominators;
  std::vector<double> numerators;
  common::LpfCoefficients(ts, cutoff_freq, &denominators, &numerators);
  digital_filter->set_coefficients(denominators, numerators);
}
void LpfCoefficients(const double ts, const double cutoff_freq,
                     std::vector<double> *denominators,
                     std::vector<double> *numerators) {
  denominators->clear();
  numerators->clear();
  denominators->reserve(3);
  numerators->reserve(3);

  double wa = 2.0 * M_PI * cutoff_freq;  // Analog frequency in rad/s
  double alpha = wa * ts / 2.0;          // tan(Wd/2), Wd is discrete f
  double alpha_sqr = alpha * alpha;
  double tmp_term = std::sqrt(2.0) * alpha + alpha_sqr;
  double gain = alpha_sqr / (1.0 + tmp_term);

  denominators->push_back(1.0);
  denominators->push_back(2.0 * (alpha_sqr - 1.0) / (1.0 + tmp_term));
  denominators->push_back((1.0 - std::sqrt(2.0) * alpha + alpha_sqr) /
                          (1.0 + tmp_term));

  numerators->push_back(gain);
  numerators->push_back(2.0 * gain);
  numerators->push_back(gain);
}
```

加速度标定表

```
void LonController::LoadControlCalibrationTable(
    const LonControllerConf &lon_controller_conf) {
  const auto &control_table = lon_controller_conf.calibration_table();
  AINFO << "Control calibration table loaded";
  AINFO << "Control calibration table size is "
        << control_table.calibration_size();
  Interpolation2D::DataType xyz;
  for (const auto &calibration : control_table.calibration()) {
    xyz.push_back(std::make_tuple(calibration.speed(),
                                  calibration.acceleration(),
                                  calibration.command()));
  }
  control_interpolation_.reset(new Interpolation2D);
  ACHECK(control_interpolation_->Init(xyz))
      << "Fail to load control calibration table";
}
```

2 计算纵向控制指令

2.1 计算流程

计算纵向控制指令的流程如下：

- Input：定位信息 `localization::LocalizationEstimate *localization`，底盘信息 `canbus::Chassis *chassis`，规划信息 `planning::ADCTrajectory`
- Output：纵向控制指令（油门、刹车） `control::ControlCommand *cmd`

```
Status LonController::ComputeControlCommand(
    const localization::LocalizationEstimate *localization,
    const canbus::Chassis *chassis,
    const planning::ADCTrajectory *planning_published_trajectory,
    control::ControlCommand *cmd) {
```

1.获取当前位置，车辆底盘和路径规划信息；

```
localization_ = localization;
  chassis_ = chassis;
  trajectory_message_ = planning_published_trajectory;
```

2.判断标定表的指针 `control_interpolation_` 是否存在，若不存在则返回错误信息。

```
if (!control_interpolation_) {
    AERROR << "Fail to initialize calibration table.";
    return Status(ErrorCode::CONTROL_COMPUTE_ERROR,
                  "Fail to initialize calibration table.");
  }
```

3.如果轨迹规划算法未设定完成，或者轨迹规划算法序列不等于估计算法消息序列，则重新设置新的轨迹规划信息指针。

```
if (trajectory_analyzer_ == nullptr ||
      trajectory_analyzer_->seq_num() !=
          trajectory_message_->header().sequence_num()) {
    trajectory_analyzer_.reset(new TrajectoryAnalyzer(trajectory_messag
  }
```

4.根据 `control_conf.pb.txt` 文件中的 `Lon_controller_conf` 的参数配置信息对参数赋值，如下

- 时间 `ts`
- 预瞄视野 `preview_window`
- 使能超前滞后校正标志位 `enable_leadlag`

5.将 `control_cmd.proto` 中的debug信息指向纵向控制的debug信息。

```
const LonControllerConf &lon_controller_conf =
      control_conf_-代码详解>lon_controller_conf();
  auto debug = cmd->mutable_debug()->mutable_simple_lon_debug();
  debug->Clear();
  double brake_cmd = 0.0;
  double throttle_cmd = 0.0;
  double ts = lon_controller_conf.ts();
  double preview_time = lon_controller_conf.preview_window() * ts;
  bool enable_leadlag =   lon_controller_conf.enable_reverse_leadlag_co
```

6.判断预瞄时间（第4步中的 `preview_window` 计算）是否小于零，小于则返回错误信息。

```
if (preview_time < 0.0) {
    const auto error_msg =
        absl::StrCat("Preview time set as: ", preview_time, " less than
    AERROR << error_msg;
    return Status(ErrorCode::CONTROL_COMPUTE_ERROR, error_msg);
  }
```

7.调用 `ComputeLongitudinalErrors` 函数计算纵向误差。

```
ComputeLongitudinalErrors(trajectory_analyzer_.get(), preview_time, ts,
                            debug);
```

8.对位置做限幅处理，利用 `math::Clamp` 函数。

```
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_lim
  }
```

9.根据前进挡或倒挡设置位置PID和速度PID控制器

```
if (trajectory_message_->gear() == canbus::Chassis::GEAR_REVERSE) {
    station_pid_controller_.SetPID(
        lon_controller_conf.reverse_station_pid_conf());    speed_pid_c
    if (enable_leadlag) {
      station_leadlag_controller_.SetLeadlag(
          lon_controller_conf.reverse_station_leadlag_conf());
      speed_leadlag_controller_.SetLeadlag(
          lon_controller_conf.reverse_speed_leadlag_conf());
    }
  } else if (injector_->vehicle_state()->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_con
  }
```

10.根据位置误差进行速度PID控制，计算速度补偿值 `speed_offset` 。如果enable_leadlag为真，则计算超前滞后速度补偿值 `speed_offset` 。

```
double speed_offset =
      station_pid_controller_.Control(station_error_limited, ts);
  if (enable_leadlag) {
    speed_offset = station_leadlag_controller_.Control(speed_offset, ts
  }
```

11.计算速度输入，根据 纵向速度=速度补偿+速度偏差 进行计算，此处的车速偏差应该为参考点的车速 `s_dot_matched` 。

```
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();
  }
```

12.对车速做限幅处理，利用 `math::Clamp` 函数。

```
speed_controller_input_limited =
      common::math::Clamp(speed_controller_input, -speed_controller_inp
                          speed_controller_input_limit);
```

13.根据上一步处理的车速，通过速度PID控制器计算补偿加速度。

```
double acceleration_cmd_closeloop = 0.0;

  acceleration_cmd_closeloop =
      speed_pid_controller_.Control(speed_controller_input_limited, ts)
  debug->set_pid_saturation_status(
      speed_pid_controller_.IntegratorSaturationStatus());
  if (enable_leadlag) {
    acceleration_cmd_closeloop =
        speed_leadlag_controller_.Control(acceleration_cmd_closeloop, t
    debug->set_leadlag_saturation_status(
        speed_leadlag_controller_.InnerstateSaturationStatus());
  }
```

14.计算斜坡补偿加速度=重力加速度*车辆俯仰角，应该就是计算在坡度方向上的加速度分量。

```
double slope_offset_compensation = digital_filter_pitch_angle_.Filter(
      GRA_ACC * std::sin(injector_->vehicle_state()->pitch()));

  if (std::isnan(slope_offset_compensation)) {
    slope_offset_compensation = 0;
  }

  debug->set_slope_offset_compensation(slope_offset_compensation);
```

15.计算加速度=加速度补偿+预瞄加速度+坡度补偿加速度。

```
double acceleration_cmd =
      acceleration_cmd_closeloop + debug->preview_acceleration_referenc
      FLAGS_enable_slope_offset * debug->slope_offset_compensation();
```

16.计算剩余规划点的个数，当同时满足以下条件时，判断该点在车辆停止，设置加速度=最大停车加速度。

- 轨迹类型与ADCTrajectoryy::NORMAL相同
- 参考预瞄加速度小于停车最大加速度
- 参考预瞄速度小于停车最大abs速度
- 剩余规划点数量小于停车最大剩余规划点数量

```
debug->set_is_full_stop(false);
  GetPathRemain(debug);
  // At near-stop stage, replace the brake control command with the sta
  // acceleration if the former is even softer than the latter
  if ((trajectory_message_->trajectory_type() ==
       apollo::planning::ADCTrajectory::NORMAL) &&
      ((std::fabs(debug->preview_acceleration_reference()) <=
            control_conf_->max_acceleration_when_stopped() &&
        std::fabs(debug->preview_speed_reference()) <=
            vehicle_param_.max_abs_speed_when_stopped()) ||
       std::abs(debug->path_remain()) <
           control_conf_->max_path_remain_when_stopped())) {
    acceleration_cmd =
        (chassis->gear_location() == canbus::Chassis::GEAR_REVERSE)
            ? std::max(acceleration_cmd,
                       -lon_controller_conf.standstill_acceleration())
            : std::min(acceleration_cmd,
                       lon_controller_conf.standstill_acceleration());
    ADEBUG << "Stop location reached";
    debug->set_is_full_stop(true);
  }
```

获取剩余路径点

```
// TODO(all): Refactor and simplify
void LonController::GetPathRemain(SimpleLongitudinalDebug *debug) {
  int stop_index = 0;
  static constexpr double kSpeedThreshold = 1e-3;
  static constexpr double kForwardAccThreshold = -1e-2;
  static constexpr double kBackwardAccThreshold = 1e-1;
  static constexpr double kParkingSpeed = 0.1;

  if (trajectory_message_->gear() == canbus::Chassis::GEAR_DRIVE) {
    while (stop_index < trajectory_message_->trajectory_point_size()) {
      auto &current_trajectory_point =
          trajectory_message_->trajectory_point(stop_index);
      if (fabs(current_trajectory_point.v()) < kSpeedThreshold &&
          current_trajectory_point.a() > kForwardAccThreshold &&
          current_trajectory_point.a() < 0.0) {
        break;
      }
      ++stop_index;
    }
  } else {
    while (stop_index < trajectory_message_->trajectory_point_size()) {
      auto &current_trajectory_point =
          trajectory_message_->trajectory_point(stop_index);
      if (current_trajectory_point.v() < kSpeedThreshold &&
          current_trajectory_point.a() < kBackwardAccThreshold &&
          current_trajectory_point.a() > 0.0) {
        break;
      }
      ++stop_index;
    }
  }
  if (stop_index == trajectory_message_->trajectory_point_size()) {
    --stop_index;
```

```
    if (fabs(trajectory_message_->trajectory_point(stop_index).v()) <
        kParkingSpeed) {
      ADEBUG << "the last point is selected as parking point";
    } else {
      ADEBUG << "the last point found in path and speed > speed_deadzor
    }
  }
  debug->set_path_remain(
      trajectory_message_->trajectory_point(stop_index).path_point().s(
      debug->current_station());
}
```

17.计算油门和刹车的下限值。

```
double throttle_lowerbound =
      std::max(vehicle_param_.throttle_deadzone(),
               lon_controller_conf.throttle_minimum_action());
  double brake_lowerbound =
      std::max(vehicle_param_.brake_deadzone(),
               lon_controller_conf.brake_minimum_action());
```

18.获取油门和刹车指令。

```
double calibration_value = 0.0;
  double acceleration_lookup =
      (chassis->gear_location() == canbus::Chassis::GEAR_REVERSE)
          ? -acceleration_cmd
          : acceleration_cmd;
```

19.如果 `FLAGS_use_preview_speed_for_table` 为真，则使用预瞄速度进行查表。

```
if (FLAGS_use_preview_speed_for_table) {
    calibration_value = control_interpolation_->Interpolate(
        std::make_pair(debug->preview_speed_reference(), acceleration_l
  } else {
    calibration_value = control_interpolation_->Interpolate(
        std::make_pair(chassis_->speed_mps(), acceleration_lookup));
  }
DEFINE_bool(use_preview_speed_for_table, false,
            "True to use preview speed for table lookup");
```

20.油门指令和制动指令限幅。

```
if (acceleration_lookup >= 0) {
    if (calibration_value >= 0) {
      throttle_cmd = std::max(calibration_value, throttle_lowerbound);
    } else {
      throttle_cmd = throttle_lowerbound;
    }
    brake_cmd = 0.0;
  } else {
    throttle_cmd = 0.0;
    if (calibration_value >= 0) {
      brake_cmd = brake_lowerbound;
    } else {
```

```
      brake_cmd = std::max(-calibration_value, brake_lowerbound);
    }
  }
```

21.设置debug信息，如下

- 位置误差限幅 `station_error_limited`；
- 速度补偿 `speed_offset`；
- 速度控制输入限幅 `speed_controller_input_limited`；
- 加速度指令 `acceleration_cmd`；
- 油门指令 `throttle_cmd`；
- 刹车指令 `brake_cmd`；
- 加速度查表 `acceleration_lookup`；
- 速度查表 `chassis_->speed_mps()`；
- 参数标定表 `calibration_value`；
- 加速度指令闭环 `acceleration_cmd_closeloop`。

```
debug->set_station_error_limited(station_error_limited);
  debug->set_speed_offset(speed_offset);
  debug->set_speed_controller_input_limited(speed_controller_input_limi
  debug->set_acceleration_cmd(acceleration_cmd);
  debug->set_throttle_cmd(throttle_cmd);
  debug->set_brake_cmd(brake_cmd);
  debug->set_acceleration_lookup(acceleration_lookup);
  debug->set_speed_lookup(chassis_->speed_mps());
  debug->set_calibration_value(calibration_value);
  debug->set_acceleration_cmd_closeloop(acceleration_cmd_closeloop);
```

22.判断 `FLAGS_enable_csv_debug` 是否为真和 `speed_log_file_` 是否为空，打印debug信息。

```
if (FLAGS_enable_csv_debug && speed_log_file_ != nullptr) {
    fprintf(speed_log_file_,
            "%.6f, %.6f, %.6f, %.6f, %.6f, %.6f, %.6f, %.6f, %.6f, %.6f
            "%.6f, %.6f, %.6f, %.6f, %.6f, %.6f, %.6f, %d,\r\n",
            debug->station_reference(), debug->station_error(),
            station_error_limited, debug->preview_station_error(),
            debug->speed_reference(), debug->speed_error(),
            speed_controller_input_limited, debug->preview_speed_refere
            debug->preview_speed_error(),
            debug->preview_acceleration_reference(), acceleration_cmd_c
            acceleration_cmd, debug->acceleration_lookup(),
            debug->speed_lookup(), calibration_value, throttle_cmd, bra
            debug->is_full_stop());
  }
DEFINE_bool(enable_csv_debug, false, "True to write out csv debug file.
```

23.控制指令设置。

```
// if the car is driven by acceleration, disgard the cmd->throttle and
  cmd->set_throttle(throttle_cmd);
  cmd->set_brake(brake_cmd);
  cmd->set_acceleration(acceleration_cmd);
```

24.如果目前车速小于停车最大abs车速，或者当前档位 `gear_location` 是空挡 `GEAR_NEUTRAL`，则设置档位为规划的档位，否则设置为当前档位。

```
if (std::fabs(injector_->vehicle_state()->linear_velocity()) <=
          vehicle_param_.max_abs_speed_when_stopped() ||
      chassis->gear_location() == canbus::Chassis::GEAR_NEUTRAL) {
    cmd->set_gear_location(trajectory_message_->gear());
  } else {
    cmd->set_gear_location(chassis->gear_location());
  }

  return Status::OK();
}
```

2.2 计算纵向误差

函数 `ComputeLongitudinalErrors` 是纵向控制器计算的关键，有许多概念在不了解定义的情况下，难以自行进行理解。

- Input：轨迹规划器指针 `*trajectory_analyzer`，预瞄时间 `preview_time`，控制周期 `ts`，调试指针 `*debug`；
- Output： `void`。

```
void LonController::ComputeLongitudinalErrors(
    const TrajectoryAnalyzer *trajectory_analyzer, const double previev
    const double ts, SimpleLongitudinalDebug *debug){
```

1.在Frenet坐标系分析车辆运动，如下

- s：沿参考轨迹的纵向累积距离；
- s_dot：沿参考轨迹的纵向速度；
- d：横向距离 w.r.t. 参考轨迹；
- d_dot：横向距离变化率，即 dd/dt；

```
double s_matched = 0.0;
  double s_dot_matched = 0.0;
  double d_matched = 0.0;
  double d_dot_matched = 0.0;
```

2.根据距离最小，在规划点中选择与当前位置匹配点。

```
auto vehicle_state = injector_->vehicle_state();
  auto matched_point = trajectory_analyzer->QueryMatchedPathPoint(
      vehicle_state->x(), vehicle_state->y());
PathPoint TrajectoryAnalyzer::QueryMatchedPathPoint(const double x, cor
  CHECK_GT(trajectory_points_.size(), 0U);

  double d_min = PointDistanceSquare(trajectory_points_.front(), x, y);
  size_t index_min = 0;

  for (size_t i = 1; i < trajectory_points_.size(); ++i) {
    double d_temp = PointDistanceSquare(trajectory_points_[i], x, y);
    if (d_temp < d_min) {
      d_min = d_temp;
      index_min = i;
    }
  }
```

```
  size_t index_start = index_min == 0 ? index_min : index_min - 1;
  size_t index_end =
      index_min + 1 == trajectory_points_.size() ? index_min : index_mi

  const double kEpsilon = 0.001;
  if (index_start == index_end ||
      std::fabs(trajectory_points_[index_start].path_point().s() -
                trajectory_points_[index_end].path_point().s()) <= kEps
    return TrajectoryPointToPathPoint(trajectory_points_[index_start]);
  }

  return FindMinDistancePoint(trajectory_points_[index_start],
                              trajectory_points_[index_end], x, y);
}
// Squared distance from the point to (x, y).
double PointDistanceSquare(const TrajectoryPoint &point, const double x
                           const double y) {
  const double dx = point.path_point().x() - x;
  const double dy = point.path_point().y() - y;
  return dx * dx + dy * dy;
}
```

3.计算横纵向误差

```
trajectory_analyzer->ToTrajectoryFrame(
      vehicle_state->x(), vehicle_state->y(), vehicle_state->heading(),
      vehicle_state->linear_velocity(), matched_point, &s_matched,
      &s_dot_matched, &d_matched, &d_dot_matched);
void TrajectoryAnalyzer::ToTrajectoryFrame(const double x, const double
                                           const double theta, const do
                                           const PathPoint &ref_point,
                                           double *ptr_s, double *ptr_s
                                           double *ptr_d,
                                           double *ptr_d_dot) const {
  double dx = x - ref_point.x();
  double dy = y - ref_point.y();

  double cos_ref_theta = std::cos(ref_point.theta());
  double sin_ref_theta = std::sin(ref_point.theta());

  // the sin of diff angle between vector (cos_ref_theta, sin_ref_theta
  // (dx, dy)
  double cross_rd_nd = cos_ref_theta * dy - sin_ref_theta * dx;
  *ptr_d = cross_rd_nd;

  // the cos of diff angle between vector (cos_ref_theta, sin_ref_theta
  // (dx, dy)
  double dot_rd_nd = dx * cos_ref_theta + dy * sin_ref_theta;
  *ptr_s = ref_point.s() + dot_rd_nd;

  double delta_theta = theta - ref_point.theta();
  double cos_delta_theta = std::cos(delta_theta);
  double sin_delta_theta = std::sin(delta_theta);

  *ptr_d_dot = v * sin_delta_theta;

  double one_minus_kappa_r_d = 1 - ref_point.kappa() * (*ptr_d);
```

```
  if (one_minus_kappa_r_d <= 0.0) {
    AERROR << "TrajectoryAnalyzer::ToTrajectoryFrame "
              "found fatal reference and actual difference. "
              "Control output might be unstable:"
           << " ref_point.kappa:" << ref_point.kappa()
           << " ref_point.x:" << ref_point.x()
           << " ref_point.y:" << ref_point.y() << " car x:" << x
           << " car y:" << y << " *ptr_d:" << *ptr_d
           << " one_minus_kappa_r_d:" << one_minus_kappa_r_d;
    // currently set to a small value to avoid control crash.
    one_minus_kappa_r_d = 0.01;
  }

  *ptr_s_dot = v * cos_delta_theta / one_minus_kappa_r_d;
}
```

4.寻找时间上与目前车辆的时间最接近的规划点

```
double current_control_time = Time::Now().ToSecond();
  double preview_control_time = current_control_time + preview_time;

  TrajectoryPoint reference_point =
      trajectory_analyzer->QueryNearestPointByAbsoluteTime(
          current_control_time);
  TrajectoryPoint preview_point =
      trajectory_analyzer->QueryNearestPointByAbsoluteTime(
          preview_control_time);
TrajectoryPoint TrajectoryAnalyzer::QueryNearestPointByAbsoluteTime(
    const double t) const {
  return QueryNearestPointByRelativeTime(t - header_time_);
}
TrajectoryPoint TrajectoryAnalyzer::QueryNearestPointByRelativeTime(
    const double t) const {
  auto func_comp = [](const TrajectoryPoint &point,
                      const double relative_time) {
    return point.relative_time() < relative_time;
  };

  auto it_low = std::lower_bound(trajectory_points_.begin(),
                                 trajectory_points_.end(), t, func_comp

  if (it_low == trajectory_points_.begin()) {
    return trajectory_points_.front();
  }

  if (it_low == trajectory_points_.end()) {
    return trajectory_points_.back();
  }

  if (FLAGS_query_forward_time_point_only) {
    return *it_low;
  } else {
    auto it_lower = it_low - 1;
    if (it_low->relative_time() - t < t - it_lower->relative_time()) {
      return *it_low;
    }
    return *it_lower;
```

```
  }
}
```

5.设置debug信息

```
debug->mutable_current_matched_point()->mutable_path_point()->set_x(
      matched_point.x());
  debug->mutable_current_matched_point()->mutable_path_point()->set_y(
      matched_point.y());
  debug->mutable_current_reference_point()->mutable_path_point()->set_x
      reference_point.path_point().x());
  debug->mutable_current_reference_point()->mutable_path_point()->set_y
      reference_point.path_point().y());
  debug->mutable_preview_reference_point()->mutable_path_point()->set_x
      preview_point.path_point().x());
  debug->mutable_preview_reference_point()->mutable_path_point()->set_y
      preview_point.path_point().y());

  ADEBUG << "matched point:" << matched_point.DebugString();
  ADEBUG << "reference point:" << reference_point.DebugString();
  ADEBUG << "preview point:" << preview_point.DebugString();
```

6.计算相关参数偏差，并加入debug信息，如下

- 航向角偏差 `heading_error`；
- 纵向车速 `lon_speed`；
- 纵向加速度 `lon_acceleration`；
- 横向最小曲率偏差 `one_minus_kappa_lat_error`；
- 参考冲击度偏差 `jerk_reference`；
- 纵向冲击度 `lon_jerk`。

```
double heading_error = common::math::NormalizeAngle(vehicle_state->head
                                                    matched_point.the
  double lon_speed = vehicle_state->linear_velocity() * std::cos(headir
  double lon_acceleration =
      vehicle_state->linear_acceleration() * std::cos(heading_error);
  double one_minus_kappa_lat_error = 1 - reference_point.path_point().k
                                          vehicle_state->linear_velc
                                          std::sin(heading_error);

  debug->set_station_reference(reference_point.path_point().s());
  debug->set_current_station(s_matched);
  debug->set_station_error(reference_point.path_point().s() - s_matched
  debug->set_speed_reference(reference_point.v());
  debug->set_current_speed(lon_speed);
  debug->set_speed_error(reference_point.v() - s_dot_matched);
  debug->set_acceleration_reference(reference_point.a());
  debug->set_current_acceleration(lon_acceleration);
  debug->set_acceleration_error(reference_point.a() -
                                lon_acceleration / one_minus_kappa_lat_
  double jerk_reference =
      (debug->acceleration_reference() - previous_acceleration_referenc
  double lon_jerk =
      (debug->current_acceleration() - previous_acceleration_) / ts;
  debug->set_jerk_reference(jerk_reference);
  debug->set_current_jerk(lon_jerk);
```

```
  debug->set_jerk_error(jerk_reference - lon_jerk / one_minus_kappa_lat

  previous_acceleration_reference_ = debug->acceleration_reference();
  previous_acceleration_ = debug->current_acceleration();

  debug->set_preview_station_error(preview_point.path_point().s() - s_m
  debug->set_preview_speed_error(preview_point.v() - s_dot_matched);
  debug->set_preview_speed_reference(preview_point.v());
  debug->set_preview_acceleration_reference(preview_point.a());
}
```

综上， `ComputeLongitudinalErrors` 的功能如下：

- 求解与当前位置最匹配的参考点
- 计算纵向位置和纵向速度偏差

3 Frenet

3.1 Frenet坐标系

Frenet坐标系使用道路的中心线作为参考线，使用参考线的**切向向量**和**法线向量**建立坐标系。

如下图所示，Frenet坐标系与笛卡尔坐标系不同，它以车辆自身为原点，坐标轴相互垂直。其中

- 纵向S：沿着参考线的方向；
- 横向D：参考线的法向。

相比于笛卡尔坐标系，Frenet坐标系在车辆运动分析中更为方便。基于参考线的位置，可以使用纵向距离和横向距离来描述任意位置，同时纵向和横向的速度、加速度等信息也方便计算。

基于Frenet坐标系的轨迹规划方法，在高速时车道保持和换道过程中都具有很强的实用性。在不考虑臂章的情况下，通常使横向的冲击度(加加速度)最小化来提高换道时的平顺性。

3.2 Frenet资料

关于Frenet的学习可以参考wiki。

论文可以参考：Moritz Werling经典的《Optimal trajectory generation for dynamic street scenarios in a Frenét Frame》。

公式的推导主要有两种：

1. 通过向量关系进行推导. 可以参考：Cartesian 坐标系与Frenet坐标系的转换；
2. 通过运动学分析推导。 可以参考：Apollo Lattice Planner

3.3 公式推导

假定是参考线 $\vec{r}(s)$ （reference line）在弧长处s的位置， $\vec{x}$ 是当前车辆轨迹（trajectory）点，该向量一般采用笛卡尔坐标系表示($\vec{x}=[x, y, z]^T$ ，此处 z 忽略)。此处采用弧长 s 和横向偏移 l 来表示，即 $\vec{x}=\vec{x}(s, l)$ 。

令 θ_r ，$\overrightarrow{\mathbf{T}}_r$ ，$\mathbf{N}r$ 分别为当前参考线 $\vec{r}(s)$ 的方位角、单位切向量和单位法向量， θ_x ，$\overrightarrow{\mathbf{T}}_x$ ，$\overrightarrow{\mathbf{N}}_x$ 分别为当前轨迹点 $\vec{x}(s, l)$ 的方位角、单位切向量和单位法向量, 根据正交基的定义有:
$\overrightarrow{\mathbf{T}}r=[\cos\theta r \quad \sin\theta_r]^T$
$\overrightarrow{\mathbf{N}}r=[-\sin\theta r \quad \cos\theta_r]^T$
$\overrightarrow{\mathbf{T}}x=[\cos\theta x \quad \sin\theta_x]^T$
$\overrightarrow{\mathbf{N}}x=\left[-\sin\theta x \quad \cos\theta_x\right]^T$

第1步：求横纵向偏移，即沿参考轨迹的横向偏差 $l(s)$ 纵向偏差 $d(s)$

为了便于用图分析，以横向偏差计算为例，根据平面几何知识可知：

$$\vec{x}(s, l)=\vec{r}(s)+l(s)\overrightarrow{N_r}(s) \qquad (1)$$

对（1）变形得：

$$\vec{x}(s, l)-\vec{r}(s)=l(s)\overrightarrow{N_r}(s) \qquad (2)$$

考虑到 $\overrightarrow{N_r}(s)$ 为单位法向量，对（2）同左乘以 $\overrightarrow{N_r}(s)^T$ ，可得 $l(s)$ 的表达式：

$$\overrightarrow{N_r}(s)^T[\vec{x}(s, l)-\vec{r}(s)]=l(s)\overrightarrow{N_r}(s)^T\overrightarrow{N_r}(s) \qquad (3)$$

即：

$$l(s)=\overrightarrow{N_r}(s)^T[\vec{x}(s,l)-\vec{r}(s)] \tag{4}$$

为了方便处理，（4）转置不影响计算结果，将（4）转置得：

$$l(s)=[\vec{x}(s,l)-\vec{r}(s)]\overrightarrow{N_r}(s)^T \tag{5}$$

设 $\vec{x}(s,l)$ 和 $\vec{r}(s)$ 的笛卡尔坐标系为 (x,y) 和 (x_r,y_r) ，结合图中的 $\overrightarrow{N_r(s)}$ ，可得到第1步的横向偏差 $l(s)$ 表达式（7-1）：

$$l(s)=\left[\, x-x_r \quad y-y_r \,\right]\left[\, -sin\theta_r \quad cos\theta_r \,\right] \tag{6}$$

$$l(s)=(y-y_r)\cos\theta_r-(x-x_r)\sin\theta_r \tag{7-1}$$

按以上思路亦可得纵向偏差$d(s)$：

$$d(s)=(x-x_r)\cos\theta_r+(y-y_r)\sin\theta_r \tag{7-2}$$

第2步：求横向偏差变化率 $\dot{l}\,(s)$ ，即沿参考轨迹的横向速度

下文对时间 t 和弧长 s 的求导，为方便表示这两者的导数，在此提出一约定：设 var 为任意一个变量或向量，记 $v\dot{a}r=\frac{dvar}{dt}, v\ddot{a}r=\frac{dv\dot{a}r}{dt}$ 。

对表达式（5）求导，根据链式法则可得：

$$\dot{l}\,(s)=[\dot{\vec{x}}(s,l)-\dot{\vec{r}}(s)]^T\vec{N}r(s)+[\vec{x}(s,l)-\vec{r}(s)]^T\dot{\vec{N}}r(s) \tag{8}$$

为化简（8），逐次求（8）右边各项，根据单位切向量和单位法向量的定义可得：

$$\dot{\vec{x}}(s,l)=\frac{d\|\vec{x}(s,l)\|}{dt}\vec{T}x(s)=vx\vec{T}_x(s) \tag{9}$$

$$\dot{\vec{r}}(s)=\frac{d\|\vec{r}(s)\|}{dt}\vec{T}r(s)=\dot{s}\vec{T}r(s) \tag{10}$$

根据链式求导法则，有：

$$\dot{\vec{N}}r(s)=\frac{\vec{N}r(s)}{ds}\frac{ds}{dt} \tag{11}$$

又根据二维Frenrt-Serret公式，有：

$$\vec{N}r(s)'=-\kappa r\vec{T}r(s) \tag{12}$$

将(12)代入(11)可得：

$$\dot{\vec{N}}r(s)=-\kappa_r\dot{s}\vec{T}r(s) \tag{13}$$

将（2）（9）（10）（13）代入（8）得：

$$dotl(s)=\left[vx\vec{T}x(s)-\dot{s}\vec{T}r(s)\right]^T\vec{N}r(s)+l(s)\vec{N}r(s)^T\left(-\kappa_r\dot{s}\vec{T}r(s)\right) \tag{14}$$

因为单位切向量和单位法向量正交，于是有：

$$\vec{T}r(s)^T\vec{N}r(s)=0,\vec{N}r(s)^T\vec{T}r(s)=0 \tag{15}$$

根据（15）化简（14）可得：

$$\dot{l}\,(s)=vx\vec{T}x(s)^T\vec{N}r(s) \tag{16}$$

将 $\vec{T}x(s)$和$\vec{N}r(s)$ 代入（16），即可得横向偏差变化率 $\dot{l}\,(s)$：

$$\dot{l}\,(s)=v_x\left[-\cos\theta_x\sin\theta_r+\sin\theta_x\cos\theta_r\right]=v_x\sin\Delta\theta \tag{17}$$

式中， $\Delta\theta=\theta_x-\theta_r$ 。

第3步：求纵向偏差变化率 $\dot{s}$ ，即沿参考轨迹的的纵向速度

根据定义 $\dot{s}=\frac{ds}{dt}$ ，那么如何利用第1和第2步将结论计算 $\dot{s}$ 呢?

本方法是先推导 v_x 与 $\dot{s}$ 的关系，最后根据 $\dot{l}\,(s)$ 和 v_x ，计算得 $\dot{s}$ 。（前提是已知 v_x ）

根据 v_x 的定义有：

$$v_x=\frac{d\|\vec{x}(s,l)\|}{dt}=\left\|\frac{d\vec{x}(s,l)}{dt}\right\|=\|\dot{\vec{x}}(s,l)| \tag{18}$$

对（1）求导，变换得（20），再结合（18）可得 v_x 与 $\dot{s}$ 的关系可得

$$\dot{\vec{x}}(s,l)=\dot{\vec{r}}(s)+\dot{l}\,(s)\vec{N}r(s)+l(s)\dot{\vec{N}}r(s) \tag{19}$$

将式（10）（13）代入（19）可得：

$$\begin{aligned}\dot{\vec{x}}(s,l)&=\dot{s}\vec{T}(s)+\dot{l}\,(s)\vec{N}r(s)-l(s)\kappa r\dot{s}\vec{T}r(s)\\&=(1-l(s)\kappa r)\,\dot{s}\vec{T}r(s)+\dot{l}\,(s)\vec{N}r(s)\end{aligned} \tag{20}$$

$$\begin{aligned}v_x&=\|\dot{\vec{x}}(s,l)\|\\&=\sqrt{\dot{\vec{x}}(s,l)^T\dot{\vec{x}}(s,l)}\\&=\sqrt{\left[(1-l(s)\kappa_r)\,\dot{s}\right]^2+[\dot{l}\,(s)]^2}\end{aligned} \tag{21}$$

将(17)代入(21)得：

$$v_x=\sqrt{\left[(1-l(s)\kappa_r)\,\dot{s}\right]^2+(v_x\sin\Delta\theta)^2} \tag{22}$$

将（22）两边同时平方，然后将右边关于 v_x 的项移动左边得：

$$\begin{aligned}v_x^2&=\left[(1-l(s)\kappa_r)\,\dot{s}\right]^2+(v_x\sin\Delta\theta)^2\ v_x^2-v_x^2\sin^2\Delta\\\theta&=\left[(1-l(s)\kappa_r)\,\dot{s}\right]^2\ v_x^2\cos^2\Delta\\\theta&=\left[(1-l(s)\kappa_r)\,\dot{s}\right]^2\end{aligned} \tag{23}$$

假定车辆实际轨迹一直沿参考线附近运动，使得 $|\Delta\theta|<\pi/2,1-\kappa_r l(s)>0$ ，则(23)可变为：

$$v_x\cos\Delta\theta=1-l(s)\kappa_r\dot{s} \tag{24}$$

即可得纵向偏差变化率$\dot{s}$的表达式：

$$\dot{s}=\frac{v_x\cos\Delta\theta}{1-l(s)\kappa_r} \tag{25}$$

综上，由公式（7-2）和（25）可得

$$\begin{aligned}s_{matched}&=|\vec{r}(s)|+d(s)\\dots_{matched}&=\dot{s}\end{aligned} \tag{26}$$

4 参考资料

编辑于 2021-09-15 16:37

「真诚赞赏，手留余香」

赞赏

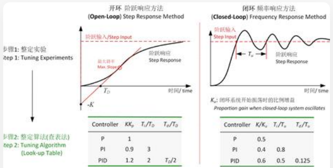
还没有人赞赏，快来当第一个赞赏的人吧！

控制算法 自动控制 控制系统

文章被以下专栏收录

自动驾驶
主要为学习自动驾驶技术的朋友们分享一些见解

推荐阅读



控制算法手记-自动整定方法初步

3ston... 发表于机电&自动...

控制理论学习笔记（1）——开闭环，LTI和传递函数

部分截图来自Brian Douglas的Classical Control Theory公开课，请支持原作者。https://www.youtube.com/playlist?list=PLUMWjy5jgHK1NC52DXXr... 本章对应教学视频的1-5节。 开

关右

如何在simulink中使用枚举量使工作更容易

渔晓民

从整车控制器VCU模型入门simulink（3）

杂谈精要

还没有评论

写下你的评论...