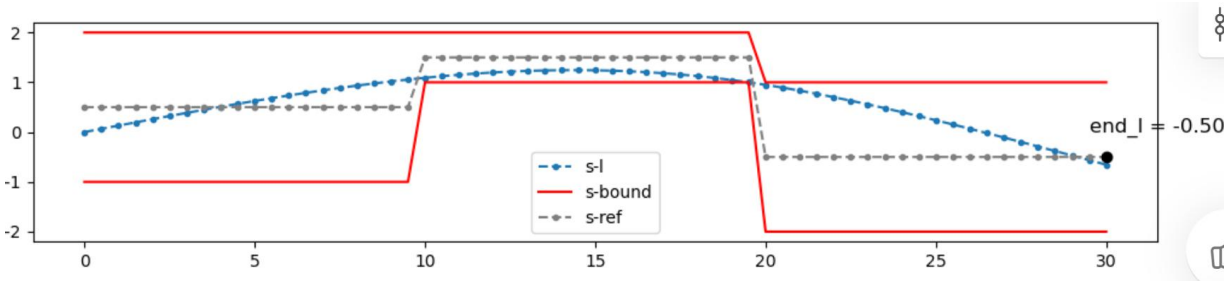
OSQP使用说明（在appollo planning 中用于optimization based path generation)

Wufan
Mail：fanwu3832@gmail.com
如果想更好了解在sl坐标系里的优化，参见上一份sl坐标转换分享分档。

1. 问题类型（path QP问题）：解决目标函数实二次型含有等式约束和不等式约束的大规模稀疏凸优化问题

在path优化里的具体表现



frenet坐标系下，沿固定距离△s将路径均匀采样

使用常三阶多项式连接各个点（每段曲线的三阶导数jerk为常量）

$$\begin{matrix} l_0 & \overset{\Delta s}{\rightarrow} & l_1 & \overset{\Delta s}{\rightarrow} & l_2 & \dots & l_{n-2} & \overset{\Delta s}{\rightarrow} & l_{n-1} \\ l_0' & & l_1' & & l_2' & & l_{n-2}' & & l_{n-1}' \\ l_0'' & & l_1'' & & l_2'' & & l_{n-2}'' & & l_{n-1}'' \end{matrix}$$

三阶导数是常量，那么有:

$$l'''_{i\rightarrow i+1} = \frac{l''_{i+1} - l''_i}{\Delta s}$$

积分得到:

$$l_{i+1}'' = \int_0^{\Delta s} l_{i->i+1}''' ds = l_i'' + l_{i->i+1}''' \cdot \Delta s$$

$$l_{i+1}' = l_i' + \int_0^{\Delta s} l''(s) ds = l_i' + l_i'' \cdot \Delta s + \frac{1}{2} l_{i->i+1}''' \cdot \Delta s^2$$

$$l_{i+1} = l_i + \int_0^{\Delta s} l'(s) ds$$

$$= l_i + (l_i)' \cdot \Delta s + \frac{1}{2} l_i'' \cdot s^2 + \frac{1}{6} l_{i->i+1}''' \cdot \Delta s^3$$

评价这条path的标准（如何构建Obj和constraints）：从现实世界到数学模型

1.1 无碰撞 l 要在path boundary 之内

在planning/tasks/piecewise_jerk_path/piecewise_jerk_path_optimizer.cc 的优化开始前会

接收到 path boundary decider 给出的path bondary(vector<path boundary point>)

1.2 舒适平滑 ：dl和ddl的变化不能太大

1.3 沿着中心线行驶（dist(l,中心线)^2）

1.4 有障碍物的情况：首先对应的path boundary会改变，其次可以加入dist(l,refline)^2 实现

远离障碍物的cost

2. Formulate problem

$$\begin{aligned} &\text{minmize } x^T P x + q^T x \\ &\text{subject to } l \le Ax \le u \end{aligned}$$

2.1 凸优化问题的定义：必须目标函数和约束都是凸函数
2.2 凸优化问题的规模（影响优化问题解的时间）：优化变量的个数，约束条件的个数
Path 优化里面，

优化的变量个数是选取的point 的个数*3 【l dl ddl】目前使用200个点，优化变量是600个

frenet坐标系下，沿固定距离△s将路径均匀采样
使用常三阶多项式连接各个点（每段曲线的三阶导数jerk为常量）

$$\begin{matrix} l_0 \\ l'_0 \\ l''_0 \end{matrix} \overset{\Delta s}{\rightarrow} \begin{matrix} l_1 \\ l'_1 \\ l''_1 \end{matrix} \overset{\Delta s}{\rightarrow} \begin{matrix} l_2 \\ l'_2 \\ l''_2 \end{matrix} \dots \begin{matrix} l_{n-2} \\ l'_{n-2} \\ l''_{n-2} \end{matrix} \overset{\Delta s}{\rightarrow} \begin{matrix} l_{n-1} \\ l'_{n-1} \\ l''_{n-1} \end{matrix}$$

优化问题的约束有;

A.起始点约束（x0，path优化里就是l0，需要自车的x0，y0，v0，a0+refline 提供的kappa

——r和dkappa_r）

B.dl（tan（heading）,ddl（Kappa） 的物理约束

C.l，dl，ddl的等式约束

D.l的物理约束（道路boundary）

3. Data flow

3.1对于每一个frame开始做优化的时候，整个优化的结构是不变的（不增加取样采点数，不增加约束的个数）

意思是在一个frame的优化是由{P，q，A，l，u}完全决定的

P：目标函数各个损失的权重，不改变权重的话，矩阵内容不变

q：偏置矩阵，和referencepoint 有关

A: 约束条件的矩阵系数（常数）

L，u：如果有障碍物的话就会改变

Summary：在不同frame（不同场景中主要改变的是l，u（和障碍物有关），q（ref point）

如果上下两个frame的l，u,ref point 优化就是同一个问题，solve的时间也会比较快

4. osqp solver 内部参数调整（path optimization）

具体的所有函数接口在osqp/osqp.h 头文件里，大家可以自己看，参数和函数的意义（为什么有这个参数以及为什么有这个函数都是和优化算法的原理有关，大家学要看论文，建议学习路径：凸优化->ADMM->数值优化->大规模稀疏数据优化 辅助理论：fix point 定理

Ref1（ADMM）：https://stanford.edu/~boyd/papers/pdf/admm_talk.pdf

Ref2（OSQP-Eigen的源码）：https://github.com/robotology/osqp-eigen

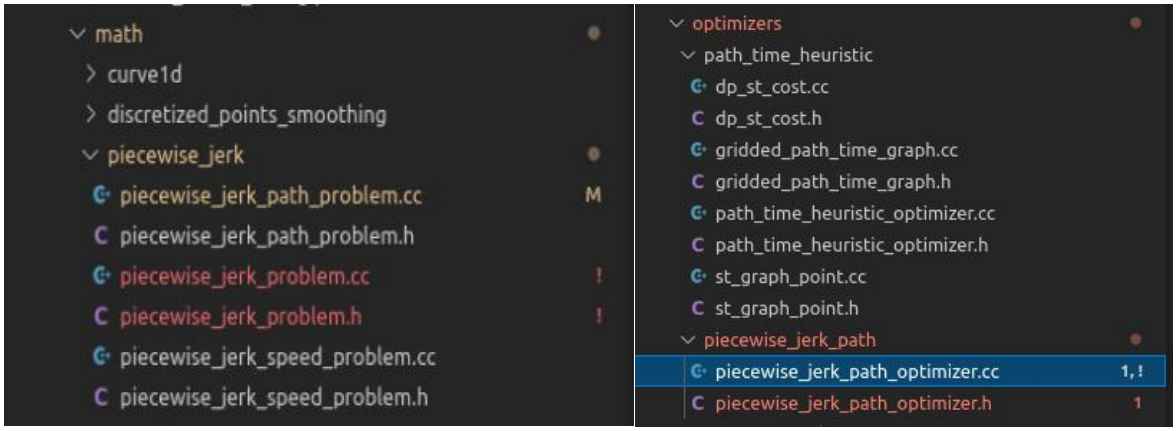
OSQP—Eigen核心代码

Data类：对C语言OSQP中的OSQPData(struct)进行包装

Setting类：对C语言OSQP中的OSQPSetting(struct)进行包装

Solver类：对C语言OSQP中的OSQPWorkspace(struct)进行包装

4.1 在path planning属于嵌入式优化，在piecewise_jerk_problem.cc/h 中定义了Data，Setting，和solver相关的接口函数。



4.2 planning/math/piecewise_jerk/piecewise_jerk_problem.cc/h

优化问题的父类：定义了优化问题所需要的数据（矩阵）和优化器的参数

优化问题的主框架：bool PiecewiseJerkProblem::Optimize(const int max_iter) {}

内部数学原理：

Algorithm 1

given initial values x^0, z^0, y^0 and parameters $\rho > 0$, $\sigma > 0$, $\alpha \in (0, 2)$

repeat

$(\tilde{x}^{k+1}, \nu^{k+1}) \leftarrow$ solve linear system $\begin{bmatrix} P + \sigma I & A^T \\ A & -\rho^{-1} I \end{bmatrix} \begin{bmatrix} \tilde{x}^{k+1} \\ \nu^{k+1} \end{bmatrix} = \begin{bmatrix} \sigma x^k - q \\ z^k - \rho^{-1} y^k \end{bmatrix}$

$\tilde{z}^{k+1} \leftarrow z^k + \rho^{-1}(\nu^{k+1} - y^k)$

$x^{k+1} \leftarrow \alpha \tilde{x}^{k+1} + (1 - \alpha) x^k$

$z^{k+1} \leftarrow \Pi\left(\alpha \tilde{z}^{k+1} + (1 - \alpha) z^k + \rho^{-1} y^k\right)$

$y^{k+1} \leftarrow y^k + \rho\left(\alpha \tilde{z}^{k+1} + (1 - \alpha) z^k - z^{k+1}\right)$

until termination criterion is satisfied

4.3 修改Setting

Ref （官方论文）：https://arxiv.org/pdf/1711.08013.pdf

Setting类除了所有的Set函数之外，还可以通过getSetting（）函数获取Setting类的指针

4.3.1

Main solver API¶ 和 setting 比较重要

A. c_int osqp_setup(OSQPWorkspace **workp, const OSQPData *data, const OSQPSettings *settings)¶

Parameters

workp – Solver workspace pointer

data – Problem data

settings – Solver settings

Returns

Exitflag for errors (0 if no errors)

B. c_int osqp_solve(OSQPWorkspace *work

求解二次规划

最终解算器信息存储在work->info结构中

Solution存储在work->solution结构中

如果问题基本上不可行，验证将存储在work->delta_y中

如果问题是双重不可行的，验证结果将存储在work->delta_x中

C. Warm start

优化器里面会check上一个frame和下一个frame的数据，并且会使用上一个frame的最优值作为下一个frame的warm_start，所以在没有障碍物的直路上行驶，迭代次数只有两次。

D. 预处理和参数选择·

遇到ill-condition的matrix（判断优化里线性方程组的解的性质，优化第一步），主要由矩阵的条数决定

预处理相关方法：Nocedal, J., Wright, S.J.: Numerical Optimization. Springer Series in Operations Research and Financial Engineering. Springer, Berlin (2006（书里第七章）

QPs选择算法参数：

选择σ和α参数：σ是一个正则化项，用于确保Algorithm 1，step 3的唯一解总是存在的，即使P有一个或多个零本征值-是的。在对P进行缩放以使其条件数最小化后，我们选择σ为尽可能小的保持数值稳定性而不减慢算法。我们开始默认值为σ=10−6. α要在[1.5,1.8]在提议的方法中，默认值设置为α=1.6。完整的超参数选择范围和默认值如下：

Argument	Description	Allowed values	Default value
rho	ADMM rho step	0 < rho	0.1
sigma	ADMM sigma step	0 < sigma	0.000001
max_iter	Maximum number of iterations	0 < max_iter (integer)	4000
eps_abs *	Absolute tolerance	0 <= eps_abs	0.001
eps_rel *	Relative tolerance	0 <= eps_rel	0.001
eps_prim_inf *	Primal infeasibility tolerance	0 <= eps_prim_inf	0.0001
eps_dual_inf *	Dual infeasibility tolerance	0 <= eps_dual_inf	0.0001
alpha *	ADMM overrelaxation parameter	0 < alpha < 2	1.6
linsys_solver	Linear systems solver type		qdldl
delta *	Polishing regularization parameter	0 < delta	0.000001
polish *	Perform polishing	True/False	FALSE
polish_refine_iter *	Refinement iterations in polish	0 < polish_refine_iter (integer)	3
warm_start *	Perform warm starting	True/False	TRUE

例子planning/math/smoothing_spline/osqp_spline_2d_solver.cc

```
// Define Solver settings as default
osqp_set_default_settings(settings);
settings->alpha = 1.0; // Change alpha parameter
settings->eps_abs = 1.0e-05;
settings->eps_rel = 1.0e-05;
settings->max_iter = 5000;
settings->polish = true;
settings->verbose = FLAGS_enable_osqp_debug;
```

E. Polish（原理在论文的地搜部分）

为了提高解的精度和active constraints有关，在piecewise_jerk_path_problem.cc里可以选择

关掉

const　double& osqp_polish_time=osqp_work->info->polish_time;

可以获得polish的具体时间

```
OSQPSettings* PiecewiseJerkPathProblem::SolverDefaultSettings() {
  // Define Solver default settings
  OSQPSettings* settings =
      reinterpret_cast<OSQPSettings*>(c_malloc(sizeof(OSQPSettings)));
  osqp_set_default_settings(settings);
  settings->polish = true;
  settings->verbose = FLAGS_enable_osqp_debug;
  settings->scaled_termination = true;
  //settings->time_limit = FLAGS_max_osqp_time;
  //MLOG(PLANNING, INFO) << "set path osqp time_limit: " << FLAGS_max_osqp_time;
  return settings;
}
```

4.3.2
Status values¶

如果改了一些最大迭代次数或者运行时间（为了强行在允许的时间范围内得到结果，不一定

是最优的）

主要在planning/math/piecewise_jerk/piecewise_jerk_problem.cc"改完出现了一些不在if

-elseL里的情况导致程序中止，要把对应的值补上去。

```
if (status < 0 || (status != 1 && status != 2 )) {
  MLOG(PLANNING, ERROR) << "failed optimization status:\t" << osqp_work->info->status;
  osqp_cleanup(osqp_work);
  FreeData(data);
  c_free(settings);
  return false;
} else if (osqp_work->solution == nullptr) {
  MLOG(PLANNING, ERROR) << "The solution from OSQP is nullptr";
  osqp_cleanup(osqp_work);
  FreeData(data);
  c_free(settings);
  return false;
}
```

Status	Constant	Value
solved	OSQP_SOLVED	1
solved inaccurate	OSQP_SOLVED_INACCURATE	2
maximum iterations reached	OSQP_MAX_ITER_REACHED	-2
primal infeasible	OSQP_PRIMAL_INFEASIBLE	-3
primal infeasible inaccurate	OSQP_PRIMAL_INFEASIBLE_INACCURATE	3
dual infeasible	OSQP_DUAL_INFEASIBLE	-4
dual infeasible inaccurate	OSQP_DUAL_INFEASIBLE_INACCURATE	4
interrupted by user	OSQP_SIGINT	-5
run time limit reached	OSQP_TIME_LIMIT_REACHED	-6
unsolved	OSQP_UNSOLVED	-10
problem non convex	OSQP_NON_CVX	-7

4.3.2
Solver errors

Errors	Constant	Value
Data validation	OSQP_DATA_VALIDATION_ERROR	1
Settings validation	OSQP_SETTINGS_VALIDATION_ERROR	2
Linear system solver loading	OSQP_LINSYS_SOLVER_LOAD_ERROR	3
Linear system solver initialization	OSQP_LINSYS_SOLVER_INIT_ERROR	4
Non convex problem	OSQP_NONCVX_ERROR	5
Memory allocation	OSQP_MEM_ALLOC_ERROR	6
Workspace not initialized	OSQP_WORKSPACE_NOT_INIT	7

4.4 调试方法：

修改setting的单个变量，打log/.txt

这部分都是优化效果的指标（时间加精度），log里会打/还有一个txt文件可以直接

导到excel里看（@吕迪）

```
// the information in the osqp solver
//number of iterations taken
osqp_iter = osqp_work->info->iter;
//primal objective
osqp_obj_val=osqp_work->info-> obj_val;
//time taken for setup phase (seconds)
osqp_setup_time=osqp_work->info->setup_time;
osqp_solve_time=osqp_work->info->solve_time;
osqp_polish_time=osqp_work->info->polish_time;
MLOG(PLANNING, INFO) << " osqp_iter = "
                     << osqp_iter ;
MLOG(PLANNING, INFO) << "osqp_obj_val = "
                     << osqp_obj_val ;
MLOG(PLANNING, INFO) << "osqp_setup_time = "
                     <<osqp_setup_time * 1000 << "ms";
MLOG(PLANNING, INFO) << "osqp_solve_time = "
                     << osqp_solve_time * 1000 << "ms";
MLOG(PLANNING, INFO) << "osqp_polish_time = "
                     << osqp_polish_time * 1000 << "ms";

auto status = osqp_work->info->status_val;
```

5. osqp 求解器 的可替代求解器

GUROBI（收费的）

MOSEK
ECOS
qpOASES https://github.com/coin-or/qpOASES

COPT （中国人自己写的，杉树科技开发的，4.0 版本可以解QP，不知道收不收费）

6. ipopt求解器 的可替代求解器（SQP）

听不懂就让它过去吧，反正我先说了，就是osqp是一类基于ADMM+矩阵代数的高级优化器，不用算梯度（非常nice），ipopt还是比较传统的依赖梯度和嗨森矩阵的方法（缺陷是依赖函数梯度和嗨森矩阵的精度和计算损失，主流的QP都不用算这些，所以模型不准确的时候一般也能很快地有结果，SQP挺好用的，不过比较费脑子（0-0，只有特定的人能做）OSQP在这里性能被限制了，没有发挥出应该的优势

SQP ： https://www.math.uh.edu/~rohop/fall_06/Chapter4.pdf

SQP solver：https://github.com/nuft/sqp_solver 通过书里的例子把nonliear 的优化变成很多的QP，然后调OSQP解（这个我没看到自动转换的，可能需要自手写）

SNOPT7：https://ccom.ucsd.edu/~optimizers/docs/snopt/ （可以直接换的求解器，论文里说比ipopt好），有C++interface