EGO-Planner：一种无需 ESDF（轻量且高效）的基于梯度的局部路径规划方法

原创 世界仙境与冷酷尽头 已于 2022-10-05 09:06:37 修改 阅读量7.1k 收藏 151 点赞数 28
分类专栏： 算法 文章标签： 人工智能

文章目录

0 摘要

基于梯度的规划器广泛用于四旋翼的局部路径规划，其中欧几里德距离场 (ESDF) 对于评估梯度大小和方向至关重要。然而，计算这样一个字段有余，因为轨迹优化过程只覆盖了 ESDF 更新范围的非常有限的子空间。在本文中，提出了一种基于无 ESDF 梯度的规划框架，显著减少了计算时间改进是惩罚函数中的碰撞项是通过将碰撞轨迹与无碰撞引导路径进行比较来制定的。只有当轨迹碰到新的障碍物时，才会存储生成的障碍物信息，只提取必要的障碍物信息。然后，如果违反动态可行性，我们会延长时间分配，引入各向异性曲线拟合算法，在保持原始形状的同时调整轨迹的高

1 概览

1.1 传统方法

- 传统方法一般是采用基于梯度的规划器，但是会依靠预先构建的 $ESDF$ 图来评估梯度的大小和方向，并使用数值优化来生成**局部最优解**。但是 $ESDF$ 图很困难，$ESDF$ 计算占用了执行本地规划的总时间的70%，所以说，构建 $ESDF$ 成为了基于梯度的规划器的瓶颈。

1.2 如何构建 $ESDF$？

- 方法可以分为增量式的全部更新，以及批量的局部计算，然而这两种方法都没有关注**轨迹本身**。因此，过多的计算花费在计算对规划没有贡献的上。换句话说，当前基于 $ESDF$ 的方法不能**单独和直接地为轨迹优化服务**。如下图所示，对于一般的自主导航场景，期望无人机在本地避免碰覆盖 ESDF 更新范围的有限空间。在实践中，虽然一些手工规则可以决定一个很小的 ESDF 范围，但它们缺乏理论上的合理性，并且仍然会导致算

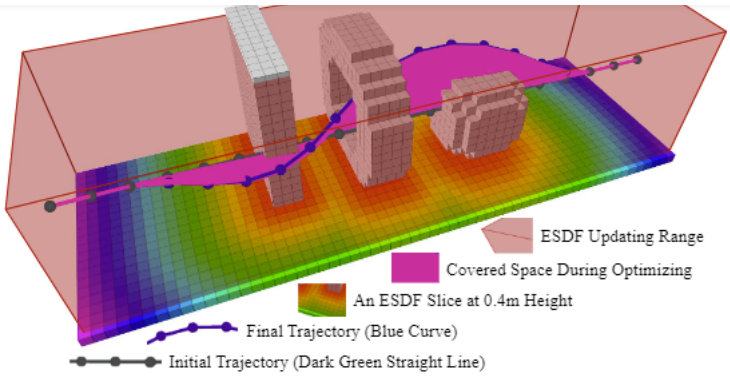


Fig. 1: Trajectory during optimizing just covers a very limited space of the ESDF updating range.

- 如图，优化过程中的轨迹只覆盖了 ESDF 更新范围的非常有限的空间
- $ESDF$ 更新的空间就是被**粉色**墙面包围的空间
- 但是轨迹优化期间覆盖的空间（如上图**紫色**的部分）
- 所以轨迹仅覆盖 $ESDF$ 更新范围的有限空间，在实践中，虽然一些手工规则可以决定一个很小的 ESDF 范围，但它们缺乏理论上的合理性，并致不必要的计算

1.3 $EGO-Planner$ 介绍

- $EGO-Planner$ 是一个无 $ESDF$ 的基于梯度的局部路径规划框架
- $EGO-Planner$ 由基于梯度的样条优化器和后细化程序组成

1.3.1 方法步骤

- 首先使用平滑度，碰撞以及动态可行性优化轨迹
- 与预先计算的 $ESDF$ 的传统方法不同，通过将障碍物内的轨迹与引导的无碰撞轨迹进行比较对比，来对碰撞成本进行建模
- 之后将力投射到**碰撞轨迹**上并生成**估计的梯度**以将**轨迹包裹在障碍物之外**
- 在优化过程中，轨迹会在附近的障碍物之间**反弹几次**，**最终终止于安全区域**
- 这样，我们只在**必要时计算梯度**，避免在与局部轨迹无关的区域计算$ESDF$
- 如果生成的轨迹违反动态限制，这通常是由**不合理的时间分配**引起的，则激活细化过程
- 在细化期间，当超出限制时**重新分配轨迹时间。**随着时间分配的扩大，生成了一种新的B样条，它在平衡可行性和拟合精度的同时，拟合了之前的行
- 为了提高鲁棒性，**拟合精度采用各向异性建模**，在轴向和径向上有不同的惩罚

2 相关工作

2.1 基于梯度的运动规划

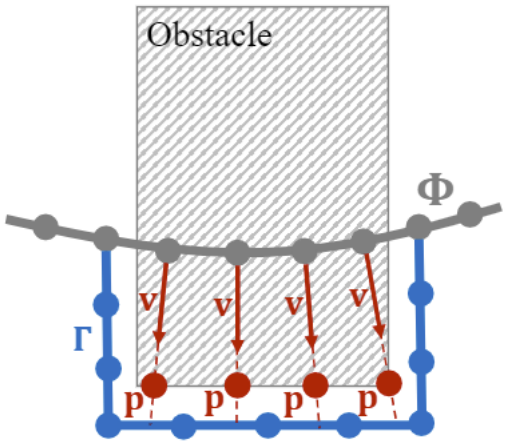
- 基于梯度的运动规划是无人机**局部轨迹生成**的主流，这是一种无约束的非线性优化
- 许多框架利用其丰富的梯度信息，直接优化配置空间的轨迹
- 还有一种连续时间多项式的轨迹优化方法，但是，势函数积分导致了很大的计算代价
- 还有就是老朋友 $B-Spline$的参数化，利用其凸包特性
- 通过寻找无碰撞路径（通常可由采样的路径规划找出，比如$Kinodynamic\ RRT*$），将无碰撞的初始路径作为前端，成功率显著提高
- 当初始无碰撞路径的生成考虑动力学约束的同时，性能会进一步提高
- 结合感知意识，鲁棒性提升明显
- $ESDF$ 在评估与附近障碍物的梯度大小和方向的距离方面起着至关重要的作用

2.2 $ESDF$

成方法，即 $Voxblox$ 和 $FIESTA$。尽管这些方法在动态更新情况下非常有效，但生成的 $ESDF$ 几乎总是包含可能根本不会在规划过程中使
信息。如上图所示，该轨迹仅扫过整个 $ESDF$ 更新范围的非常有限的子空间。因此，设计一种更智能、更轻量级的方法，而不是维护整个领域
的。

2.3 碰撞避免

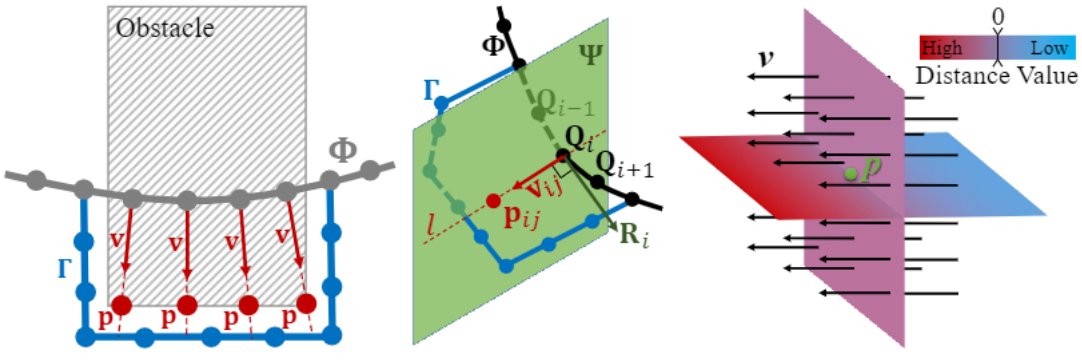
- 决策变量是 B 样条曲线的控制点 Q。每个 Q 都独立地拥有自己的环境信息。一开始，生成了一条满足约束但是不考虑障碍物的 B 样条曲线 Φ，
对于在迭代中检测到的每个碰撞段，用一些算法（比如A*，RRT*等全局规划算法）生成一条无碰撞路径Γ。对于发生碰撞的线段的每个控制点Q
成一个在障碍物表面的定位点p_{ij}，且对应一个排斥方向向量v_{ij}，v_{ij}与向量$Q_i p_{ij}$同向，这里不应该是相等，因为这个v_{ij}只是一个单位方向向量
图中的红色虚线与实线即可。每一对p, v都对应一个特定的控制点Q_i，这点也能从图中看出来，如下图所示。



(a) {p, v} Pairs

- 用 $i \in N+$ 表示控制点的索引，$j \in N$ 表示 $\{p, v\}$ 对的索引。请注意，每个 $\{p, v\}$ 对只属于一个特定的控制点。为简洁起见，省略了下标 ij 而
歧义。则从 Q_i 到第 j 个障碍物的障碍物距离定义为

- p, v是怎么生成的?
- 穿过障碍物的轨迹 Φ 为控制点 Q 生成几个 (p, v) 对
- p 是障碍物表面上的点（前面说了p是由控制点Q生成的），v 是从控制点Q指向 p 的单位向量)
- 垂直于切向量 R_i （这个切向量，仔细看，就是之前用 $A*$ 生成的那条的不考虑碰撞的曲线的切线）的平面 Ψ 与 Γ （Γ是那一条无碰撞路径）相
一条线 l，从中确定 p, v 对。 距离场定义 $d_{ij} = (Q_i - p_{ij}) \cdot v_{ij}$ 的切片可视化。颜色表示距离，箭头是等于 v 的相同梯度。 p 在平面上（障碍
的点，如图 c ）。



(a) {p, v} Pairs (b) p, v Generation (c) Distance Field of A {p, v} Pair

Fig. 3: a) A trajectory Φ passing through an obstacle generates several $\{\mathbf{p}, \mathbf{v}\}$ pairs for control points. $\mathbf{p}$ are the points at the obstacle surface and $\mathbf{v}$ are unit vectors pointing from control points to $\mathbf{p}$. b) A plane Ψ which is perpendicular to a tangent vector $\mathbf{R}_i$ intersects Γ forming a line l, from which a $\{\mathbf{p}, \mathbf{v}\}$ pair is determined. c) Slice visualization of distance field definition $d_{ij} = (\mathbf{Q}_i - \mathbf{p}_{ij}) \cdot \mathbf{v}_{ij}$. The color indicates the distance and the arrows a

Algorithm 1 CheckAndAddObstacleInfo

Notation: Environment $\mathcal{E}$, Control Points Struct $\mathbf{Q}$, Anchor Points $\mathbf{p}$, Repulsive Direction Vector $\mathbf{v}$, Colliding Segments $\mathbf{S}$
Input: $\mathcal{E}$, $\mathbf{Q}$
for $\mathbf{Q}_i$ in $\mathbf{Q}$ **do**
  if FindConsecutiveCollidingSegment($\mathbf{Q}_i$) **then**
    $\mathbf{S}$.push_back(GetCollisionSegment())
  end if
end for
for $\mathbf{S}_i$ in $\mathbf{S}$ **do**
  $\mathbf{\Gamma} \leftarrow$ PathSearch($\mathcal{E}$, $\mathbf{S_i}$)
  for $\mathbf{S}_i$.begin $\leq j \leq \mathbf{S}_i$.end **do**
    $\{\mathbf{p}, \mathbf{v}\} \leftarrow$ Find_p_v_Pairs($\mathbf{Q}_j, \mathbf{\Gamma}$)
    $\mathbf{Q}_j$.push_back($\{\mathbf{p}, \mathbf{v}\}$)
  end for
end for

伪代码解析：

1. $FindConsecutiveCollidingSegment(Q)$： 找到与控制点 Q 发生碰撞的障碍物，并判断其是否存在
2. $GetCollisionSegment()$：提取出与控制点 Q 发生碰撞的障碍物
3. $.push\_back()$：将障碍物添加入障碍物集合 S 中
4. $PathSearch()$：针对障碍物生成一个无碰撞的路径
5. $Find\_p\_v\_Pairs(Q, path)$：就是根据控制点与无碰撞路径确定匹配的(p, v)对

- Q_i 到第 j 个障碍物的距离如下，需要注意单位向量 v 在第一次生成后就不会再次发生改变，所以 d_{ij} 的值是分正负的，$d_{ij} = (Q_i - p_{ij}) \cdot v_{ij}$
- 通过以上的那个公式，我们就可以利用其来判断这个控制点 Q_i
- 因为之前说了，在**老障碍物**当中 v_{ij} 与向量 $Q_i p_{ij}$ 同向，那么向量 $Q_i p_{ij}$ 就是 $-(Q_i - p_{ij})$，是这个与 v_{ij} 同向，而 $(Q_i - p_{ij})$ 肯定就是与了，如果不是新发现的障碍物，那么 $d_{ij} = (Q_i - p_{ij}) \cdot v_{ij}$ 算出来肯定是 $d_{ij} < 0$ （**两者反向**），而对于**新发现的障碍物**，肯定就是 $d_{ij} = ($ $p_{ij}) \cdot v_{ij}$，算出来为 $d_{ij} > 0$

> 综上，为了防止轨迹被拉出当前障碍物前，为了避免在前几次迭代过程中轨迹从当前障碍物逃逸之前产生重复的 $\{p, v\}$ 对，迭代过程中反复生成对，判断是否为新障碍物的标准是：如果控制点Q_i 处于障碍物中时，并且对于当前得到的所有障碍物 j 满足 $d_{ij} > 0$，则该障碍物为**新发现**的障从而只计算影响轨迹的障碍物信息，减少运行时间。

2.3.2 碰撞力估计小结

> 为了将**必要的*环境意识**融入当地的规划中，需要明确地构建一个目标函数(**设计基于梯度的轨迹优化器**)，使轨道远离障碍。$ESDF$ 提供了这种至的碰撞信息，但代价是**沉重的计算负担**。此外，如图 2 所示，由于 $ESDF$ 反馈的错误信息不足，基于 $ESDF$ 的规划者很容易陷入**局部最小值**脱障碍。为了避免这种情况，额外的前端总是需要提供一个无碰撞的初始轨迹。由于明确设计的斥力对于不同的任务和环境都是相当有效的，所以在**提供避免碰撞**的重要信息方面优于 $ESDF$ 。

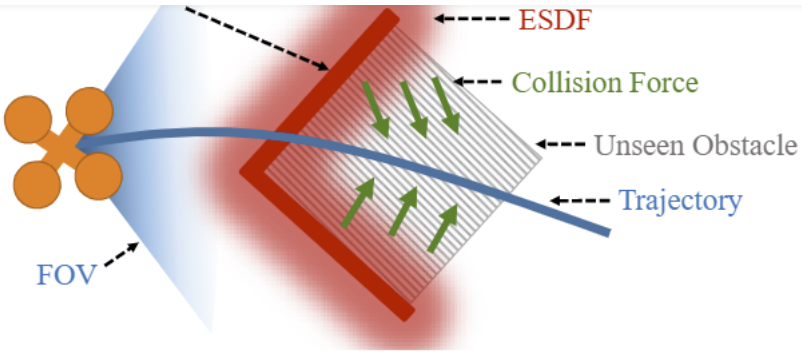


Fig. 2: The trajectory gets stuck into a local minimum, which is very common since the camera has no vision of the back of the obstacle.

3 基于梯度的轨迹优化器

3.1 建模

- 本文使用均匀 B 样条曲线 Φ 来表示轨迹，其阶数为 p_b，均匀 B 样条曲线的每个节点有相同的时间间隔 $\{t_1, t_2, ..., t_M\}$，每个节点与其父节点间间隔相同，为：$\triangle_t = t_{m+1} - t_m$
- B 样条曲线的第一个性质也即凸包性质表明，某段曲线只与 $p_b + 1$ 个连续的控制点（$\mathrm{Q} \in \mathbb{R}^3$）有关，并且曲线被包含在这些点构成的凸包内，条的 (t_i, t_{i+1}) 内的跨度位于由 $\{Q_i - p_b, Q_{i-p_b+1}, ..., Q_i\}$ 形成的凸包内
- B 样条曲线的第二个性质表面，B 样条曲线的 k 阶导数仍然是 $p_b - k$ 阶 B 样条曲线，由于时间间隔 $\triangle_t$ 是相同的，轨迹 Φ 的一阶（V_i）、二）、三阶（J_i）导数的控制点分别为：

$$\mathbf{V}_i = \frac{\mathbf{Q}_{i+1} - \mathbf{Q}_i}{\Delta t}, \mathbf{A}_i = \frac{\mathbf{V}_{i+1} - \mathbf{V}_i}{\Delta t}, \mathbf{J}_i = \frac{\mathbf{A}_{i+1} - \mathbf{A}_i}{\Delta t}$$

- 其实大体上的框架还是 $Fast - Planner$的框架，$HKUST$还是牛的哇
- 再根据 UAV 的微分平坦特性，需要降低要规划的变量，优化问题被重新定义为：

$$\min_{\mathrm{Q}} J = \lambda_s J_s + \lambda_c J_c + \lambda_d J_d$$

> 微分平坦特性，在做轨迹规划的过程中不可能对 UAV 12 维的全维度空间进行规划，这是非常复杂的，但可以找到一个**平坦输出**的空间，**这个空个维度的变量**，位置x, y, z和偏航角 Ψ，**剩下所有的状态都能用这四个变量和其有限阶导数的代数组合所表示**，因此在做无人机轨迹规划的时候只**这四个变量**进行规划就可以了。

- 是不是很熟悉？这其实就是在贝塞尔曲线那里的三个惩罚项函数
- J_s 为**光滑项**惩罚，J_c 为**碰撞项**惩罚，J_d 为**动力学可行项**惩罚， λ 为**惩罚项的权值**

3.1.1 光滑项惩罚

- 在 $B - Spline$ 轨迹优化当中被提出，可以看看我之前的 $Blog$，有提到过。光滑性惩罚被公式化为轨迹参数(**加速度、加加速度等**)的平方导数**分**。由于 B 样条曲线的凸包性质，只要最小化轨迹 Φ 的二阶和三阶控制点的平方和就能够有效地减小加速度和加加速度的平方和（类似 $minim$ $snap$，这东西真是经典，哪里都用到了），公式如下：

$$J_s = \sum_{i=1}^{N_c-2} \|A_i\|_2^2 + \sum_{i=1}^{N_c-3} \|J_i\|_2^2$$

3.1.2 碰撞项惩罚

- **碰撞惩罚使控制点远离障碍物**，这是通过采用**安全间隙和惩罚控制点** $d_{ij} < s_f$ 来实现的。为了进一步优化，构造了一个**二次连续可微惩罚函数**，d_{ij} 的减小而抑制其斜率，从而得到分段函数：

$$j_c(c,f) = \begin{cases} c_{ij}^3, & 0 \le c_{ij} \le s_f \\ 3s_f c_{ij}^2 - 3s_f^2 c_{ij} + s_f^3, & c_{ij} > s_f \end{cases}$$

$$c_{ij} = s_f - d_{ij}$$

- 对所有控制点的惩罚求和得到总的碰撞项惩罚：

$$J_c = \sum_{i=1}^{N_c} j_c(Q_i)$$

- 相比于传统用**三线性插值**的方法求碰撞项的梯度，**直接计算二次连续可微惩罚函数**对 Q_i 的导数来得到梯度：

$$\frac{\partial j_c}{\partial Q_i} = \sum_{i=1}^{N_c}\sum_{j=1}^{N_p} \begin{cases} 0, & c_{ij} \le 0 \\ -3c_{ij}^2, & 0 \le c_{ij} \le s_f \\ -6s_f c_{ij} + 3s_f^2, & c_{ij} > s_f \end{cases}$$

3.1.3 可行项惩罚

- 通过限制轨迹在每一维上的**高阶导数**来保证其可行性。由于凸包的性质，对控制点的导数进行约束足以约束整个 B 样条。$F()$ 为为每个维度（速 $Jerk$）的高阶导数构造的惩罚函数：

$$J_d = \sum_{i=1}^{N_c-1} \omega_v F(V_i) + \sum_{i=1}^{N_c-2} \omega_a F(A_i) + \sum_{i=1}^{N_c-3} \omega_j F(J_i)$$

$$F(C) = \sum_{r=x,y,z} f(c_r)$$

$$f(c_r) = \begin{cases} a_1 c_r^2 + b_1 c_r + c_1, & c_r < -c_j \\ (-\lambda c_m - c_r)^3, & -c_j < c_r < -\lambda c_m \\ 0, & -\lambda c_m \le c_r \le \lambda c_m \\ (c_r - \lambda c_m)^3, & \lambda c_m < c_r < c_j \\ a_2 c_r^2 + b_2 c_r + c_2, & c_r > c_j \end{cases}$$

> 可以看出问题建模中应用的惩罚函数 全部是多项式和，这有利于降低求解最优化问题的复杂度。（轻量化算法）

3.2 最优化 解法（数值优化）

- **目标函数**：

$$\min_{Q} J = \lambda_s J_s + \lambda_c J_c + \lambda_d J_d$$

会随着**新障碍物**的加入而不断改变，这就要求求解器能够快速重启并求解，并且目标函数主要由二次项组成，所以 $Hessian$（黑塞矩阵）信息能 敛速度。但得到精确的 $Hessian$ 消耗大量计算机资源。所以使用拟牛顿法（$quasi-Newton\ methods$）从梯度信息中来近似计算 $Hessia$

- 在对比了$Barzilai-Borwein\ method$、$truncated\ Newton\ method$和$L-BFGS\ methodh$后发现，$L-BFGS$表现最好，平衡了 和逆$Hessian$ 估计的准确性。$L-BFGS$ 从以前的目标函数评估近似 $Hessian$，但需要一系列的迭代，以达到一个相对准确的估计

4 时间重新分配以及轨迹进一步优化

- 在优化之前进行准确的时间分配是不合理的，因为那时候还不知道关于最终轨迹的信息，所以，一个额外的时间重新分配程序对于确保动态可行 要的
- 之前都是将轨迹参数化为非均匀 B 样

$an\ anisotropic\ curve\ fitting\ method$）使 Φ_f 在保持形状与 Φ_s 几乎相同的导数形状的同时（因为 Φ_s 是安全轨迹，所以形状必须差不多
够自由地优化其控制点，以满足高阶导数的约束

4.1 具体步骤

- 首先像 $Fast-Planner$ 所做的那样，计算极限超标率（超过限制的最大的比例，下标 m 表示限制的最大值）：

$$r_e = \max\left\{|\mathbf{V}_{i,r}/v_m|, \sqrt{|\mathbf{A}_{j,r}/a_m|}, \sqrt[3]{|\mathbf{J}_{k,r}/j_m|}, 1\right\}$$

- r_e 表明相对于 Φ_s 来说，Φ_f 需要分配多少时间
- V_i，A_j 和 J_k 分别是与 $\triangle t$ 的一次、二次、和三次成反比，通过与时间的反比关系可以降低速度及其导数
- 得到了 Φ_f 的新时间间隔为：

$$\Delta t' = r_e \Delta t$$

- 通过求解一个如下的**闭式的最小二乘问题**，在速度及其导数的约束下初始生成时间跨度为 $\Delta t'$ 的轨迹 Φ_f，同时保持与 Φ_s 相同的形状和控制点
新计算光滑项惩罚和可行性项惩罚得到新的目标函数：

$$\min_{\mathbf{Q}} J' = \lambda_s J_s + \lambda_d J_d + \lambda_f J_f$$

- λ_f 是适应度（**拟合**）项的权重，J_f 被定义为从 $\Phi_f(\alpha T\prime)$ 到 $\Phi_s(\alpha T)$ 各向异性位移的积分（$\alpha \in [0,\ 1]$），其中 T 和 $T\prime$ 为轨迹 Φ_s 和 Φ_f
间
- 由于拟合的曲线 Φ_s 已经是无碰撞的（刚刚说过了 Φ_s 是安全轨迹），所以对于两条曲线，用带有**低权重的轴向位移** d_a 来**放宽光滑调整限制**，用
径向位移 d_r 来**防止碰撞**，如下图所示，

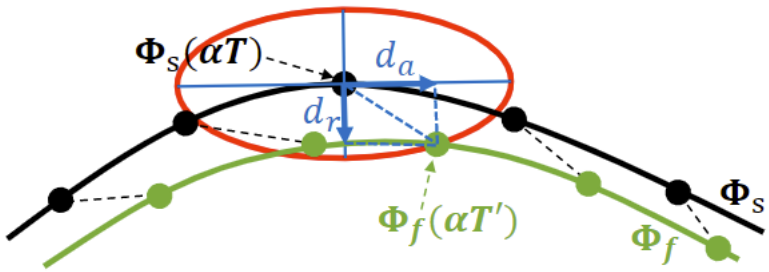


Fig. 5: Optimizing trajectory Φ_f to fit trajectory Φ_s while adjusting smoothness and feasibility. Black and green dots are sample points on the trajectory. The displacement between $\Phi_f(\alpha T')$ and $\Phi_s(\alpha T)$ breaks down into d_a and d_r along two ellipse principal axes. Points at the red ellipse surface produce identical penalties.

- 使用**球状度量**（好像在老师的某篇文章中也提到过，具体的我忘了）来使在**同一球体表面的位移**产生相同的惩罚。(关于径向位移和轴向位移应该
中了解，目前我认为轴向位移为该点的切线方向，而径向方向为该切线的垂线方向，两者计算公式如下：)

$$d_a = (\Phi_f - \Phi_s) \cdot \frac{\dot{\Phi}_s}{\left\|\dot{\Phi}_s\right\|}$$

$$d_r = \left\| (\Phi_f - \Phi_s) \times \frac{\dot{\Phi}_s}{\left\|\dot{\Phi}_s\right\|} \right\|$$

- 用于度量 $\Phi_f(\alpha T\prime)$ 惩罚大小的椭圆体是一个以 $\Phi_f(\alpha T\prime)$ 为中心的椭圆，其半长轴长度为 a、其半短轴长度为 b。则轴向位移 d_a 和径向位移 d

$$d_r\left(\alpha T'\right)=\left\|\left(\Phi_f\left(\alpha T'\right)-\Phi_s(\alpha T)\right)\times\frac{\dot{\Phi}_s(\alpha T)}{\left\|\dot{\Phi}_s(\alpha T)\right\|}\right\|$$

- 则拟合项可以表示为：

$$J_f=\int_0^1\left[\frac{d_a\left(\alpha T'\right)^2}{a^2}+\frac{d_r\left(\alpha T'\right)^2}{b^2}\right]\mathrm{d}\alpha$$

- 其中 a 和 b 分别是椭圆的半长轴和半短轴，径向位移对应的半短轴 b 使径向位移的惩罚权重增大以防止防止碰撞
- 上式拟合项被离散化为有限个数的点 $\Phi_f\left(\alpha\Delta t\prime\right)$ 和 $\Phi_s\left(\alpha\Delta t\right)$
- 其中，$\mathrm{K}\in\mathbb{R}$，$0\leq k\leq[T/\triangle t]$

5 实验结果

5.1 算法框架

Algorithm 2 Rebound Planning

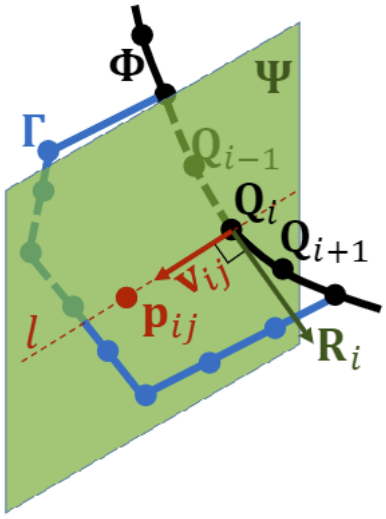
Notation: Goal $\mathcal{G}$, Environment $\mathcal{E}$, Control Point Struct $\mathbf{Q}$, Penalty J, Gradient $\mathbf{G}$
Initialize: $\mathbf{Q}\leftarrow$ FindInit($\mathbf{Q}_{last}$, $\mathcal{G}$)
while $\neg$ IsCollisionFree($\mathcal{E}$, $\mathbf{Q}$) **do**
    CheckAndAddObstacleInfo($\mathcal{E}$, $\mathbf{Q}$)
    (J, $\mathbf{G}$) $\leftarrow$ EvaluatePenalty($\mathbf{Q}$)
    $\mathbf{Q}\leftarrow$ OneStepOptimize(J, $\mathbf{G}$)
end while
if $\neg$ IsFeasible($\mathbf{Q}$) **then**
    $\mathbf{Q}\leftarrow$ ReAllocateTime($\mathbf{Q}$)
    $\mathbf{Q}\leftarrow$ CurveFittingOptimize($\mathbf{Q}$)
end if
return $\mathbf{Q}$

- **代码解析**
- $FindInit(Q,G)$：生成一条满足终端约束但不考虑障碍物的 B 样条曲线 Φ 对应的控制点，对应初始化步骤
- $IsCollisionFree(E,Q)$：判断控制点是否在环境中是无碰撞的，有碰撞时输出 $false$，无碰撞时输出 $true$
- $CheckAndAddObstacleInfo(E,Q)$：检测控制点所在障碍物，并添加障碍物信息($\{p,v\}$对以及距离场)
- $EvaluatePenalty(Q)$：根据问题建模构造控制点相应惩罚项 J 以及对应的梯度（这个惩罚项就是那三个惩罚，光滑项惩罚，碰撞项惩罚，可
罚）
- $OneStepOptimize(J,G)$：求解惩罚项的最小化问题，即**最优化求解**，从而得到满足惩罚项的最小化的控制点位置，即完成第一步的轨迹优
- $IsFeasible(Q)$：判断由控制点 Q 决定的轨迹是否可行(主要是速度以及其多阶导数否超过限制最大值)
- $ReAllocateTime(Q)$：通过重新分配由控制点 Q 决定的轨迹的时间降低速度以及其多阶导数，使其满足各类速度约束
- $CurveFittingOptimize(Q)$：将之前的惩罚中碰撞项替换为曲线拟合项，求解惩罚项最优化问题。使其在满足新时间间隔的前提下，拟合由
构成的轨迹得到新轨迹，在继承旧轨迹

- 设置B样条曲线的阶数 $p_b = 3$，控制点的个数 $N_c = 25$个左右，具体由规划预期距离（大约$7m$）和初始的邻近点间距（大约$0.3m$）决定。这经验通过平衡了问题的复杂度和自由度而得到
- 因为根据 B 样条曲线的性质，一个控制点只影响周围的轨迹，所以算法的时间复杂度为 $O(N_c)$
- $L-BFGS$ 的复杂性在相同的相对公差上也是线性的($The\ complexity\ of\ L-BFGS\ is\ also\ linear\ on\ the\ same\ relative\ toleran$
- 在无碰撞路径搜索中，我们采用 A 星（$A*$）算法进行轨迹优化，而它生成的轨迹 Γ 常常贴着障碍物。因此，我们可以直接在 A 星算法生成的轨择定位点（$anchor\ point$）p 而不用搜索障碍物的表面(这里才真正解释出了一开始所说采用 A 星算法的作用)。对于图 $3b$ 中定义的向量R_i，样条参数化的性质，可以推导出：

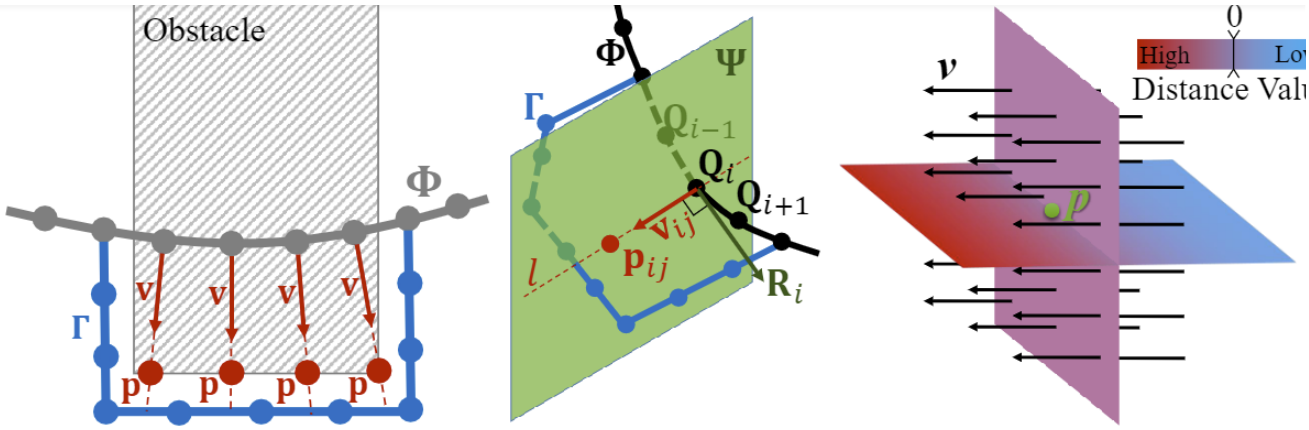
$$\mathbf{R}_i = \frac{\mathbf{Q}_{i+1} - \mathbf{Q}_{i-1}}{2\Delta t}$$

- 这里的R_i是下图中确定 $\{p, v\}$ 对的关键



(b) p, v Generation

读到这里，我们再看看论文中的图3：
第一步：根据生成一条满足终端约束但不考虑障碍物的 B 样条曲线 Φ，依靠 A 星算法生成的轨迹 Γ
第二步：根据上述的 R_i 公式通过 B 样条曲线的控制点计算出向量 R_i，再做出垂直于 R_i 的平面 Ψ，平面 Ψ 与依靠 A 星算法生成的轨迹 Γ 相点（$anchor\ point$）p，连接对应的定位点（$anchor\ point$）p 与控制点 Q，才得到直线 l，而向量 v 是向量 l 对应的由起点控制点 Q 到终点对应的**单位向量**。(v 可能是生成以后不会再变化的)。到这里才生成了 $\{p, v\}$ 对
Q：关于 $\{p, v\}$ 对过程中，为什么将定位点 p 定位到 A 星算法生成的贴着障碍物的轨迹上，而不直接定位在障碍物表面？
A：首先 $A*$ 算法生成的轨迹肯定是安全无碰撞的轨迹，其次直接定位到障碍物表面的话，因为这是仿真，难免会产生误差，使得与现实世界所得符，再者使用 $A*$ 算法生成的轨迹定位的算力比直接定位在障碍物表面的要**低**，所以不管是为了留有稳定的裕量还是为了降低计算的算力，还是选 $A*$ 算法进行定位。



(a) $\{\mathbf{p}, \mathbf{v}\}$ Pairs (b) $\mathbf{p}, \mathbf{v}$ Generation (c) Distance Field of A $\{\mathbf{p}, \mathbf{v}\}$ Pair

Fig. 3: a) A trajectory $\mathbf{\Phi}$ passing through an obstacle generates several $\{\mathbf{p}, \mathbf{v}\}$ pairs for control points. $\mathbf{p}$ are the points at the surface and $\mathbf{v}$ are unit vectors pointing from control points to $\mathbf{p}$. b) A plane $\mathbf{\Psi}$ which is perpendicular to a tangent vector $\mathbf{R}_i$ int forming a line l, from which a $\{\mathbf{p}, \mathbf{v}\}$ pair is determined. c) Slice visualization of distance field definition $d_{ij} = (\mathbf{Q}_i - \mathbf{p}_{ij}) \cdot \mathbf{v}_{ij}$. indicates the distance and the arrows are identical gradients equal to $\mathbf{v}$. $\mathbf{p}$ is at the zero distance plane.

6 总结

> 该方法仍然存在一些缺陷，即 $A*$ 算法搜索引入的局部最小值和统一时间重新分配引入的保守轨迹。因此，我们将致力于执行拓扑规划，以逃避局值，并重新制定问题，以生成接近最优的轨迹。规划器为静态环境设计，无需处理缓慢移动的障碍物（低于$0.5m/s$）。在未来，我们将通过移动和拓扑规划来研究动态环境导航。