凸优化天梯报告

王振宇 2020310754
傅星运 2020310901
王骁 2020312680

一． 问题建模与分析

a) 问题建模

近年来，扫地机器人拥有极高的热度，也引发了研究人员的兴趣，如何构建空间地图，如何进行导航，如何寻找到垃圾位置，以及如何规划最短的清扫路径，都是值得进一步研究的科学问题。本次天梯就是一个以自动寻路清扫垃圾为背景的路径规划问题。本题的地图如下图所示，共有 14 个垃圾点，同时还有 A、B、C、D 四个障碍物，对于红色的清扫小车，需要找到一条最短的路径，使得小车可以经过所有垃圾点，同时避开所有障碍物。

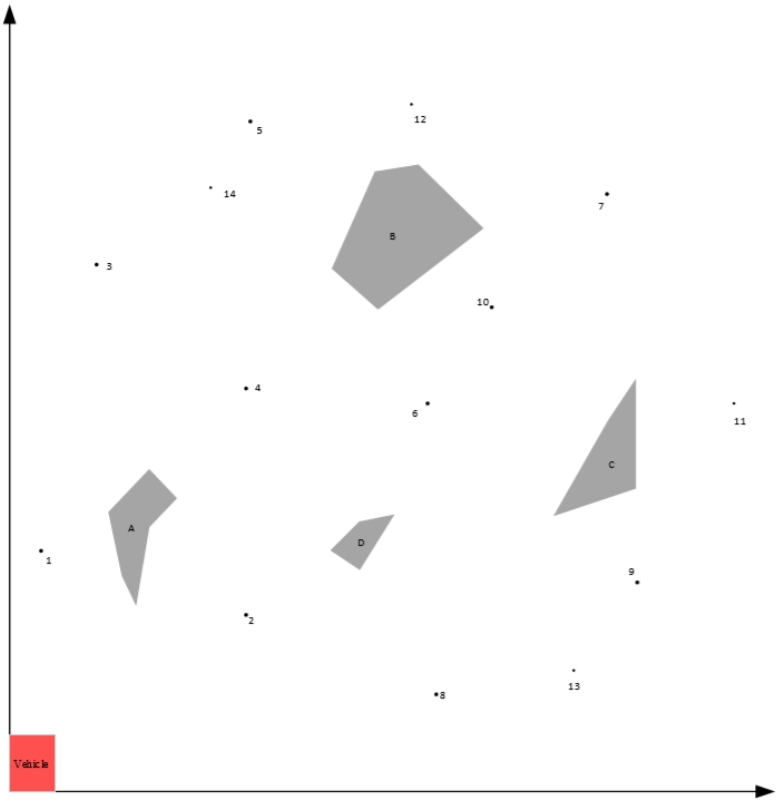

图 1 原问题垃圾点与障碍物位置图

将问题进行建模，可以转化为带约束的优化问题。在本题中，进行路径规划时的采样点需要少于 100，第 k 个采样点的状态表示为$z_k = [x_k, y_k \theta_k]^T$，小车中心的位置表示为$P_k = [x_k, y_k]^T$。目标函数为

$$\min J(z) = \sum_{k=0}^{n-1} \sqrt{(x_{k+1} - x_k)^2 + (y_{k+1} - y_k)^2},$$

约束条件共有三项，小车路径必须覆盖所有垃圾点，小车路径不能与任何障碍物接触，以及每两个相邻采样点之间的连线的夹角小于0.2π。对于第一个约束，可以将约束条件写为

$$\forall m_i, \exists j, s.t. m_i \in V_j$$

这里m_i代表第 i 个垃圾点，V_j代表以第 j 个采样点为中心的矩形，矩形是一个凸多边形，对于单个点是否在凸多边形内的判断是一个凸集，但覆盖每个垃圾进行搜索并不是凸集。第二个约束需要判断采样点形成的路径是否与障碍相交，为了避免碰撞，计算路径上的每一条直线$(x_i, y_i) - (x_{i+1}, y_{i+1})$与障碍物某一条边$(a_j, b_j) - (a_{j+1}, b_{j+1})$的交点坐标。经过计算，交点的 x 坐标可以写为如下形式：

$$\mathrm{x}' = \frac{(x_{i+1} - x_i)(y_i x_{i+1} - y_{i+1} x_i) - (a_{i+1} - a_i)(b_i a_{i+1} - b_{i+1} a_i)}{((x_{i+1} - x_i)(b_{i+1} - b_i) - (y_{i+1} - y_i)(a_{i+1} - a_i))}$$

若满足x'在$[x_i, x_{i+1}]$之外或$[a_i, a_{i+1}]$之外，则直线并没有相交，及若 $(x_i - x')(x_{i+1} - x') > 0$或$(a_i - x')(a_{i+1} - x') > 0$, 则路线满足不与障碍物相交的约束。这个约束不是凸集，因为如果两个采样点均在障碍外，两点连线仍然可能处在障碍内。第三个约束可以用公式表示为

$$\alpha(P_k - P_{k-1}, P_{k+1} - P_k) \leq \alpha_{max}$$

这个约束是为了保证小车有足够空间进行转向。尽管该问题的模型中需要优化的目标函数是凸函数，但是由于存在非凸的约束，最终的优化问题并不能视作凸优化问题。

b) 问题分析

由于该问题不是凸优化问题，因此不能采用 cvx 等凸优化工具包进行求解，讨论后选取 pytorch 进行求解，对于约束以惩罚函数的形式给出。在最初进行分析时，将采样点作为优化的目标变量进行求解，发现易陷入局部最优解。由于约束条件非凸，当需要优化的目标变量较多时，很难达到收敛，同时以采样点为优化变量，初始化的采样点对结果有很大影响，若初始采样点靠近局部最优解，惩罚函数难以控制为 0，例如某个障碍物下方的路径是最优解，但初始化将采样点设置在了障碍物上方，优化过程中一旦采样点向下搜索，就会进入障碍物，导致惩罚函数项增大，只能回到障碍物上方，因此这个方法很难优化到全局最优解。如下图所示，这种优化方法会在障碍物附近绕圈，导致路径不断增加，因此这个方法并不可行。这个问题中存在很多局部最优解，因此使用 pytorch 对采样点的优化很容易陷入局部最优解当中。

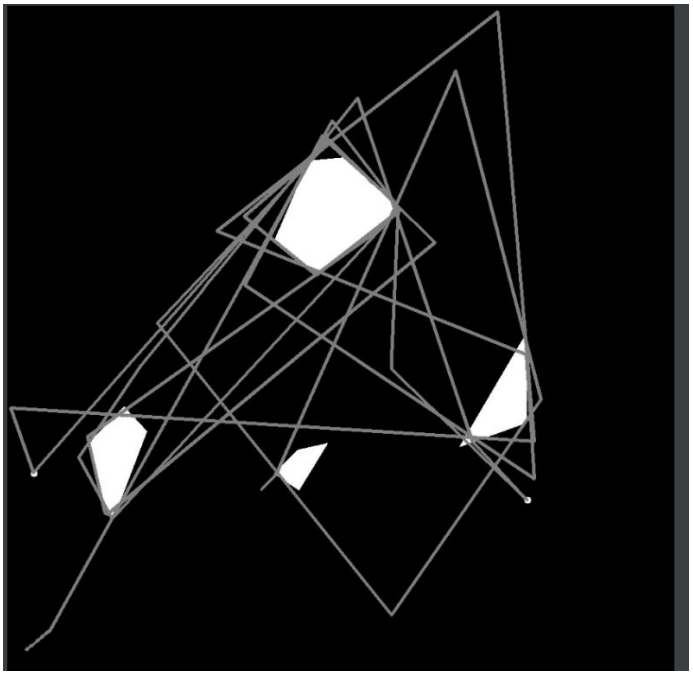

图 2 陷入局部最优解示意图

鉴于对采样点进行优化的思路并不可行，尝试将复杂的问题简单化，分成两个部分分别求解，即首先对每两个点之间的最短距离进行计算，即两点间的避障距离，对于任意两点在不碰到障碍的前提下最短距离是多少可以计算得到，再以这个距离为基础构造一个类似旅行商问题的求解问题，对路径进行规划，求解得到清扫垃圾点的顺序，这样就得到了最终的最短路径。尽管第一部分的问题仍然存在非凸性约束，可能陷入局部最优解，但局部最优解的数量相比合并求解大大减少，距离全局最优解的距离也会减小。接下来的部分将会从算法设计角度详细介绍如何求解两点间的避障距离及如何进行路径规划与路径优化。

二． 算法设计

a) 距离估计

i. 优化问题

进行路径规划前，首先求解两点之间的路程。由于两点之间的路程并非直线可达，受障碍物影响，同样是一个约束优化问题，然而问题规模大幅度减小，因此更容易求解得到最优解。

由于需要对包括出发点在内的 15 个点进行路程求解，考虑到算法效率的问题，我们考虑的是简化的问题：即假设小车的面积为 0 时，并且不考虑路径上的转角约束时，考虑避障要求，计算两点之间的最短距离。

也就是说我们要求解的是下面的优化问题，给定初始点(x_0, y_0)和终止点(x_t, y_t)，求解一系列采样点(x_i, y_i)，在满足避免碰撞的约束下，最小化路程的总和。我们要优化的目标函数可以写成如下形式：

$$\min_{x_i, y_i} \sum_{i=1}^{t} \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}$$

为了避免碰撞，计算路径上的每一条直线$(x_i, y_i) - (x_{i+1}, y_{i+1})$与障碍物某一条边$(a_j, b_j) - (a_{j+1}, b_{j+1})$的交点坐标。经过计算，交点的 x 坐标可以写为如下形式：

$$\mathrm{x}' = \frac{(x_{i+1} - x_i)(y_i x_{i+1} - y_{i+1} x_i) - (a_{i+1} - a_i)(b_i a_{i+1} - b_{i+1} a_i)}{((x_{i+1} - x_i)(b_{i+1} - b_i) - (y_{i+1} - y_i)(a_{i+1} - a_i))}$$

若满足x'在$[x_i, x_{i+1}]$之外或$[a_i, a_{i+1}]$之外，则直线并没有相交，及若$(x_i - x')(x_{i+1} - x') > 0$或$(a_i - x')(a_{i+1} - x') > 0$，则路线满足不遇障碍物相交的约束。

然而$-(x_i - x')(x_{i+1} - x')$并非凸函数，因此约束集并非为凸集。在处理时使用凸优化算法容易求解得到局部最优解，然而由于此子问题相比原问题进行了简化，仅仅存在少量的几个局部最优解。因此能够使用一些非凸优化算法更容易的得到全局最优解。

ii. 优化算法

本部分的求解使用 matlab 的 global optimization toolbox 进行求解。该工具包中包含了几种全局优化求解器，包括 GlobalSearch、MultiStart、patternsearch、surrogateopt、particleswarm、ga、simulannealbnd 等求解器。

其中 MultiStart 求解器从多个初始点出发通过基于梯度的方法进行迭代更新，由于初始点的选取具有随机性，因此其结果具有一定的随机性。该方法能够快速的收敛到全局最优解。Patternsearch 是可以求解有界约束问题的模式搜索求解器，是不基于梯度的优化算法。相对而言搜索速度更慢，能够证明该算法能够收敛到某个局部最优解。Surrogateopt 对于有界约束问题被证明能够搜索到全局最优解，但是求解速度比较慢，对于目标函数计算复杂度的情况下具有广泛的应用。Particleswarm 是一种粒子群求解器，相对而言算法思路更加直观，类似于蒙特卡洛

方法基于仿真实验进行优化的思路，而不需要依赖于梯度信息。然而该方法没有收敛性证明。Ga 表示使用遗传算法进行求解，同粒子群算法类似不需要基于梯度信息，对于某些整数规划问题具有更好的效果，但是同样也没有收敛性证明。Simulannealbnd 是一种模拟退火求解器，被证明可以收敛到全局最优解，但是并不能求解形式多样的约束问题，仅仅支持对变量进行上下界约束。

我们在实验过程中，分别使用每一种求解器进行过求解，从实验效果来说，globalsearch 能够达到最好的效果。Globalsearch 在 MultiStart 求解器的基础上引入了启发性的信息，因此可以减小部分不必要的初始点，并且在本部分中较简单的优化问题中也可以很快的收敛到最优解。

（我们也考虑过直接使用能够收敛到全局最优解的全局搜索算法求解该问题，然而由于计算十分耗时，因此最终并没有采用该方法）

通过上一部分中的约束形式我们可以给出约束函数 nlinconst(x)和目标函数 cost(x)，之后 solver.m 中使用 global optimization toolbox 进行距离的计算；计算的结果绘制为表格，作为接下来进行路径规划中垃圾与垃圾之间的距离。

b) 路径规划

有了点与点之间的距离，则可以通过规划的方法计算得到。然而这个问题与旅行商问题类似，直接便利所有路径之后使用深度优先搜索或者广度优先搜索得到解的过程，是一个 n！复杂度的问题，也就是说要进行的运算在 15！次，这是难以实现的，因此借鉴旅行商问题的求解过程，使用动态规划的方法进行求解。

具体来说，我们求解的问题是从原点出发，找到一个最优的遍历全部节点的路径能够使得总的路径最短。

首先我们需要选取一个合理的状态表示能够完全描述小车的性质。选取当前所在的节点 p 和小车已经经过的节点集合 V 作为状态，则该状态满足马尔科夫性。即状态中包含了问题所需的全部信息，小车处于路径中的什么位置可以仅仅用当前状态来决定，而不需要过去的状态。因此，该问题可以使用动态规划的方法进行求解。

该马尔科夫决策问题的 bellman 等式可以写为如下形式。

$$\mathrm{c}(\mathrm{p}',\mathrm{V}') = \min_{p,V} c(p,V) + d(p,p')$$

其中$\mathrm{c}(\mathrm{p}',\mathrm{V}')$表示从原点出发前往状态$(\mathrm{p}',\mathrm{V}')$所需要的路程之和，$d(p,p')$为从$p$节点到$p'$节点所需的路径，即（a）部分所计算的内容。$\mathrm{V}'$为状态为$(p,V)$的情况下，下一个节点前往$p'$时，新的经过节点集合，容易看出$\mathrm{V}' = \mathrm{V} \cup \{\mathrm{p}\}$。因此，贝尔曼方程可以写为如下形式：

$$\mathrm{c}(\mathrm{p}',\mathrm{V}') = \min_{p} c(p,V'/\{p\}) + d(p,p')$$

使用该式进行迭代，即可计算出最优路径。

动态规划问题的迭代求解速度缓慢，然而由于可以看到该问题为一个分阶段的问题。因此每次进行贝尔曼方程迭代的过程并不需要遍历所有状态，而只需要在当前 stage（由 V 中的元素数量决定）中进行迭代即可。因此该问题的求解同样非常迅速。

c) 路线优化

i. 优化思路

经过（b）过程我们已知了扫到垃圾的顺序，然而（a）过程的求解为了计算效率二进行了大量的简化操作，因此路程计算并不准确。本部分希望考虑除去转角约束外的其他约束，同时最小化小车的路程。

由于（b）中确定了小车的路径，因此优化问题更为简单。之前的约束为“对于任意

垃圾 i，存在某个采样点覆盖垃圾 i”，而经过（b）的计算之后，我们可以把问题修改为更容易求解的形式，比如“第 5i 个采样点会覆盖垃圾 i”，此时原来关于覆盖所有垃圾的非凸约束就变成了凸约束。

关于这个凸优化问题，我们采用 Adam 算法求解。首先需要将约束转变为惩罚函数。

关于避障问题的处理，最简单的处理方式是当与障碍物与路线相交时在原问题基础上附加惩罚为 1，当障碍物与路线不想交时在原问题基础上附加惩罚为 1（为了训练顺利进行惩罚项前需要设定惩罚系数λ）。

然而这样处理的优化目标并无法达到避障的效果，因为 0-1 函数的导数处处为 0，因此使用基于梯度的方法并不能有效的求解。也可以考虑使用类似使用障碍函数求解约束优化问题的方法，然而由于碰撞函数表示形式复杂，且其中的逻辑运算物理意义并不明确，因此在本部分放弃使用判断线段与多边形相交的方法。

因此更改为更加直观的方式，即在直线上再次进行采样，使用采样点与多边形的位置关系来描述直线与多边形的位置关系。若采样点均在多边形外，则直线与多边形不想交，若采样点在多边形内，则用采样点到各个边的距离的乘积描述采样点距离多边形外的距离，如下图中的$x_1x_2x_3x_4x_5$表示中间一点距离多边形外的距离：

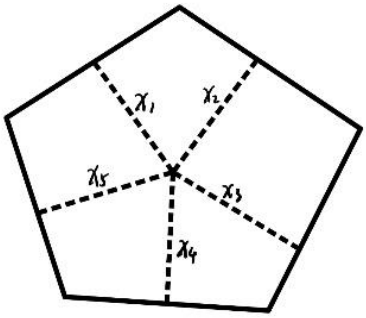

图 3 碰撞惩罚函数示意图

具体的优化算法采用 pycharm+Adam 优化器进行。Adam 算法为一种自适应优化器，结合了 Adagrad 与 RMSProp 算法的优点，对于梯度的一阶矩估计和二阶矩估计进行综合考虑，自动调整学习率，对于优化问题存在很大噪声或者梯度稀疏的问题都能够很好地进行求解。

值得一提的是，由于我们始终将小车视为质点，如果如此优化势必导致小车与障碍物的碰撞。因此在进行碰撞检测时，我们将距离 x 留取一定的裕度，如上图在实际计算过程中会留出小车半宽度 h 的距离，及$(x_1+h)(x_2+h)(x_3+h)(x_4+h)(x_5+h)$，此时可保证小车宽度为 2h 的情况之下仍旧不与障碍物相碰撞。

在优化目标中，除去坐标(x,y)，仍有一项θ项未进行处理，由于我们在考虑小车大小时直接用宽度处理，事实上默认小车朝向与小车运行方向相同，即$\theta_i = \arctan\left(\frac{y_{i+1}-y_i}{x_{i+1}-x_i}\right)$。

ii. 优化算法

Adam 算法的更新过程如下所示：

1. 首先计算 t 时刻的梯度

$$g_t = \nabla J(\theta_{t-1})$$

2. 计算梯度的指数移动平均数

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

3. 计算梯度平方的指数移动平均值

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

4. 矫正：

$$\hat{m}_t = m_t/(1-\beta_1^t)$$
$$\hat{v}_t = v_t/(1-\beta_2^t)$$

5. 参数更新

$$\theta_t = \theta_{t-1} - \frac{\alpha \hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}$$

d) 角度微调

对路线进行优化后，基本得到了最短路径，但是在之前的计算中未考虑到角度的约束条件，由于采样点数量较多，可以通过插值的方法进行微调，以满足角度的约束条件。相邻采样点之间连成的直线形成的角度可以用两向量的夹角来表达，即

$$cos\theta = \frac{(P_k - P_{k-1}, P_{k+1} - P_k)}{|P_k - P_{k-1}|\,|P_{k+1} - P_k|}$$

如果夹角大于0.2π，可以进行插值微调，即在两条直线间插入更多的采样点，采样点的选取以和前一条直线夹角为0.2π作为方向，为了尽可能减小总距离，将步长选取为0.05 m，完成了角度的插值。这里的角度微调事实上就是在转角过大的地方以接近原地转向的方式进行运动，前文提到对于避障已经考虑了小车大小的影响，因此旋转过程中不会碰到障碍物。从实际情况来看，在垃圾点附近，由于小车有一定面积，小车只需要微小的旋转就可以清扫到垃圾，最终的路径距离应该小于本计算方法中得到的距离。由于时间有限，对小车的大小的更细节的考虑没有进行进一步研究。对于小车转角的插值微调的代码如下所示：

三． 实验结果

a) 距离估计：首先展示 matlab 的 global optimization toolbox 的求解效果。这里展示使用 global search 求解的点 10 与点 14 之间的最短路径（中间采样点数为 2）。

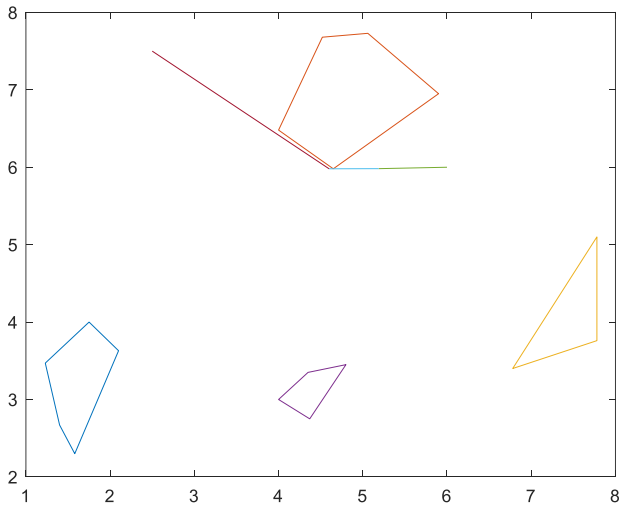

图 4 Global optimization toolbox 求解结果示意图

最终计算的距离可以用邻接矩阵表示：

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0.00000	2.65250	3.20320	6.20380	5.35250	8.40740	6.63020	10.01630	5.08650	7.84460	8.03640	9.84590	9.43110	6.81280	7.48540
1	2.65250	0.00000	2.69920	3.56930	3.20160	5.90340	5.25900	8.27700	5.25610	7.64000	6.39780	10.46470	7.17010	6.87370	4.96590
2	3.20320	2.69920	0.00000	4.66150	2.80000	6.10080	3.47130	6.87680	2.60000	4.91630	4.90410	7.67290	6.86320	4.15930	5.31530
3	6.20380	3.56930	4.66150	0.00000	2.53110	2.61730	4.43850	6.70790	6.76240	7.75240	4.93890	8.08200	4.38290	7.73370	1.72050
4	5.35250	3.20160	2.80000	2.53110	0.00000	3.30150	2.30870	5.10000	4.49540	5.45620	3.25000	6.15500	4.08820	5.39070	2.53180
5	8.40740	5.90340	6.10080	2.61730	3.30150	0.00000	4.14310	4.49110	7.46320	7.59290	4.31550	7.07410	2.01000	7.89050	0.94340
6	6.63020	5.25900	3.47130	4.43850	2.30870	4.14310	0.00000	3.40590	3.60140	3.40760	1.44220	3.88490	4.10740	3.75900	3.81840
7	10.01630	8.27700	6.87680	6.70790	5.10000	4.49110	3.40590	0.00000	6.45600	4.83540	1.97990	3.05290	2.64010	5.97010	4.93910
8	5.08650	5.25610	2.60000	6.76240	4.49540	7.46320	3.60140	6.45600	0.00000	2.86530	4.85080	5.16240	7.62470	1.72630	6.89420
9	7.84460	7.64000	4.91630	7.75240	5.45620	7.59290	3.40760	4.83540	2.86530	0.00000	4.21080	2.50600	7.31870	1.36010	7.22990
10	8.03640	6.39780	4.90410	4.93890	3.25000	4.31550	1.44220	1.97990	4.85080	4.21080	0.00000	3.23110	2.75490	4.60980	4.01970
11	9.84590	10.46470	7.67290	8.08200	6.15500	7.07410	3.88490	3.05290	5.16240	2.50600	3.23110	0.00000	5.44890	3.85880	7.24070
12	9.43110	7.17010	6.86320	4.38290	4.08820	2.01000	4.10740	2.64010	7.62470	7.31870	2.75490	5.44890	0.00000	7.40130	2.69260
13	6.81280	6.87370	4.15930	7.73370	5.39070	7.89050	3.75900	5.97010	1.72630	1.36010	4.60980	3.85880	7.40130	0.00000	7.50000
14	7.48540	4.96590	5.31530	1.72050	2.53180	0.94340	3.81840	4.93910	6.89420	7.22990	4.01970	7.24070	2.69260	7.50000	0.00000

图 5 距离估计得到邻接矩阵

b) 路径规划结果：经过动态规划求得的扫垃圾的顺序为[1, 2, 8, 13, 9, 11, 7, 10, 6, 4, 3, 14, 5, 12]

c) 路线优化过程中，待优化函数的 loss 随训练过程的变化如下图所示：

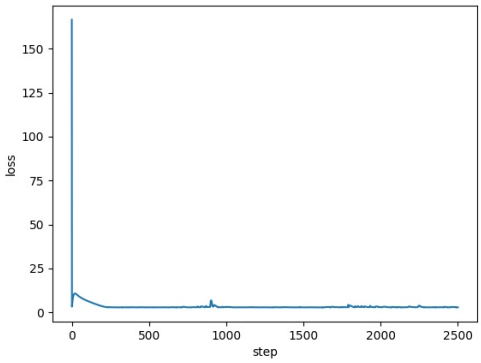

图 6 优化目标训练过程

d) 在不考虑避障与角度约束时时，使用动态规划求解的最佳路径如下图所示

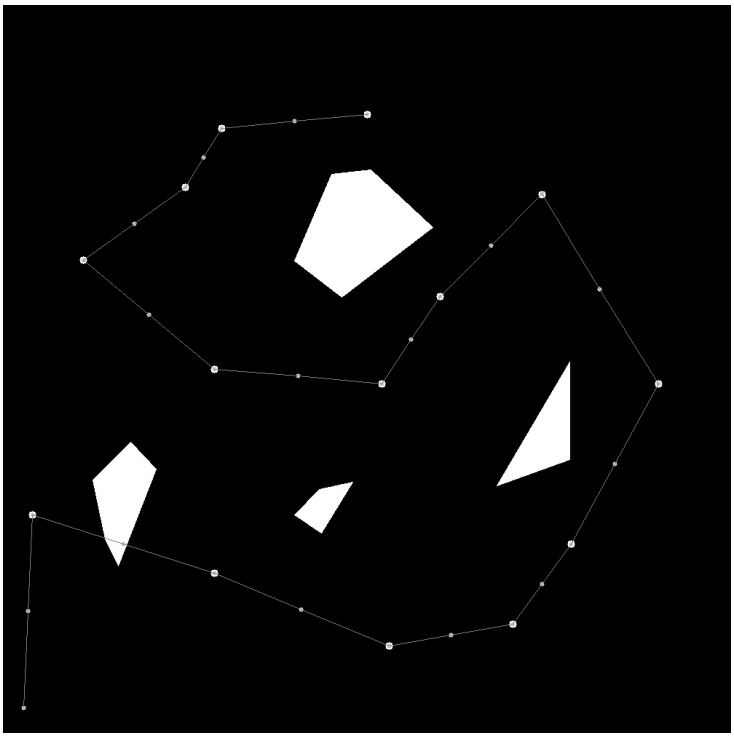

图 7 动态规划效果图

e) 考虑避障约束后，计算得到的最佳路径如下图所示

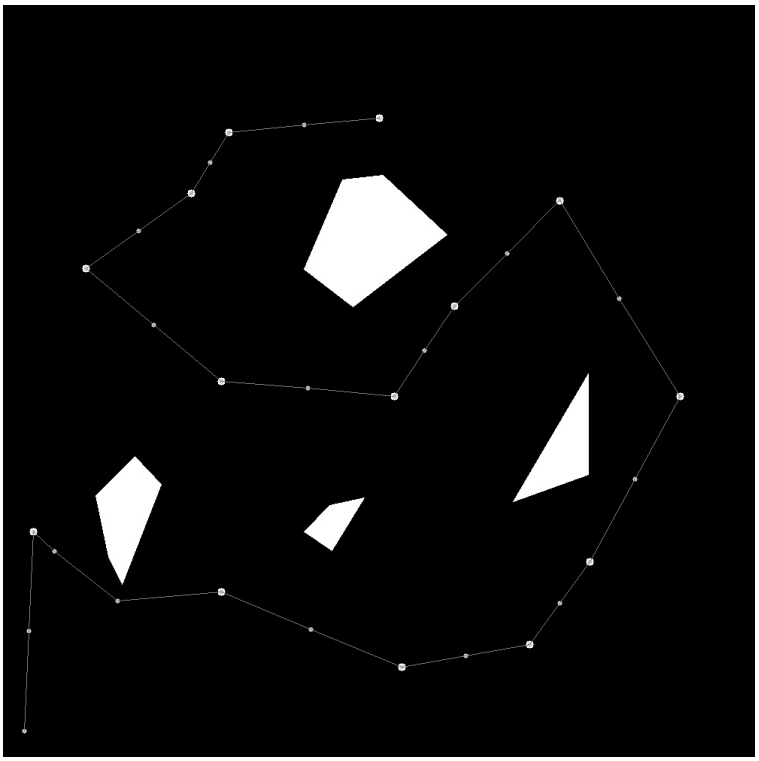

图 8 避障功能效果图

f) 最终考虑角度约束进行角度微调之后，计算得到的最优路径图如下图所示,该路径的长度为 33.0568 m。最优点的坐标保存在 result.txt 中。

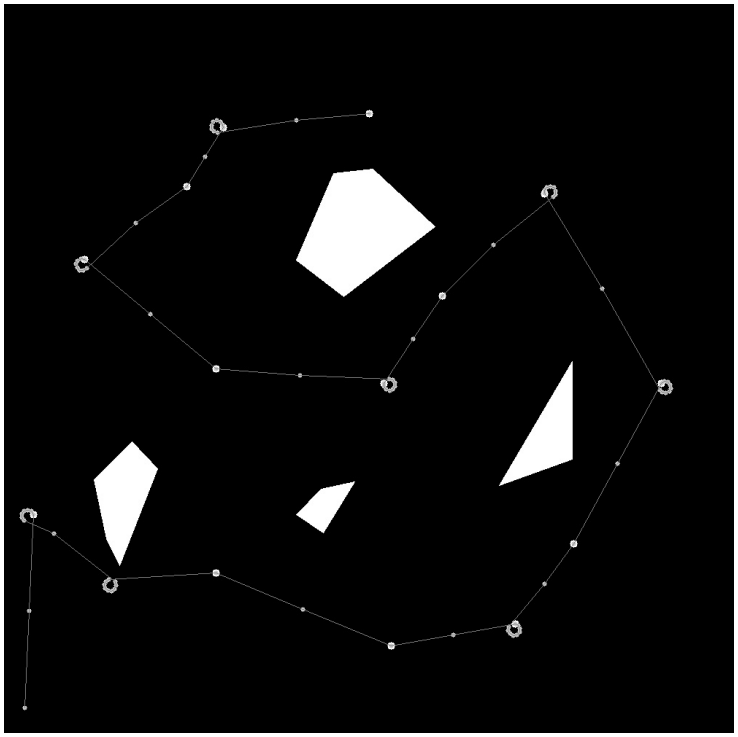

图 9 最优路径效果图

四． 实验总结

本次实验完成的是一个扫地机器人的仿真优化任务。直接进行连续的路径计算是困难的，因此将问题离散化，使用 99 个采样节点对问题进行离散化，优化这 99 个采样节点就可以近似的求得原问题的最优解。然而离散化后的优化问题仍然

尽管路径规划任务是一个凸目标函数，但是由于扫地机器人需要遍历所有垃圾节点并且满足避障要求，而这两个约束导致问题的可行集并非凸集。因此直接使用凸优化算法容易陷入局部最优解。事实上我们在实验的过程中也确实深受局部最优解的困扰。

因此我们考虑将非凸优化进行分层简化，首先在局部计算两点之间考虑避障情况下的路程，尽管该问题仍然非凸，然而经过简化后局部最优解数量较小，比较容易通过非凸优化算法求解。之后通过动态规划算法在进行路径规划，利用之前计算的路径对于小车扫垃圾的顺序进行规划。通过前两步能够计算出一个接近全局最优解的解。之后从这个近似最优解出发，使用 Adam 算法基于梯度信息进行优化，虽然理论上仍然会收敛到一个局部最优解，但是由于 Adam 优化器开始于一个接近全局最优解的解，因此更容易收敛到全局最优解。最后考虑我们之前一直忽视的角度约束，由于角度约束在该问题建模下可以非常轻松的满足，极限情况下类似于在两个采样点之间车子的坐标不变，而行驶角度旋转0.2π，即之前计算出的最优解事实上也是考虑了角度约束后的理论最优解。然而为了结果更加美观，我们采取手动插值的方式，绘制出了图 9 的结果。

由于本课程学习的是凸优化算法，对于非凸问题的处理涉及较少，在完成实验时，也能够感受到凸优化算法在面对非凸问题时极容易陷入局部最优解，即使是使用非凸的全局优化算法，在面对如此复杂的问题时，也容易产生局部最优解或者计算开销大的问题。

另外也能够看到对于复杂的优化问题，如果能够将问题模块化、分层化或者阶段化，则非常有利于优化问题的求解，能够大幅度缩减运算开销并且更容易收敛到全局最优解。