**摘要**

在工业化4.0时代到来之际，物联网背景下的自动驾驶也逐渐受到更多的

关注。无人车在遇障后做转弯时的轨迹规划问题也是业界内一大难题。本文通过界定分析法、算法等在已知的地图模型和无人车运动模型的基础上，建立了无人车运动的轨迹模型。

针对问题一：在交会几何和预测观测车辆行驶轨迹的前提下，我们构建了一个基于约束条件的轨迹模型。利用已有的控制算法即以点线式车辆-道路模型进行设计，忽略车辆与道路形状之间的实际差异，利用参数化曲线表示道路轨迹，车辆根据参数化曲线以及自身状态做出转向动作。由题设条件可知，我们采用界定分析法确定车辆行驶的轨迹区域，选取加速度、最大曲率、安全距离等作为约束条件，在已有的地图模型基础上建立轨迹预测模型[1]，并运用MATLAB中的Automated Driving Toolbox进行仿真。

针对问题二：在问题一的基础上，已知曲率决定转弯半径，方向盘转角决定的是车转弯时轨迹半径的极限值。而车在转弯时，是否需要倒车的条件是由路宽、车距、最小曲率半径所决定的。这里，我们采用路径规划算法对轨迹进行预测，寻找其界限值。由题设条件可知，临界值由安全距离、最小半径、车宽三者共同决定。根据经验可知，当最小半径都不满足一次性转弯时，那么就必须倒车。

针对问题三：通过寻找约束条件进行非线性规划建模[2]，并采用算法实现其路径规划。依据题干条件设定两个场景，场景一为仅存在F或G障碍物；场景二为障碍物F、G同时都存在。利用构建的无人车运动的非完整约束模型和行驶过程中的速度约束条件，并基于障碍项、曲率项、光滑项、场函数构建损失函数，并将其作为优化对象求解轨迹模型。

针对问题四：利用问题二中得到的约束条件和附件数据计算出最小转弯半径，用AutoCAD建立地图模型。在此基础上，进一步判断当道路中存在人行道时，求解无人车不压线通过的临界值。通过路径规划法计算出无人车在满足安全距离、交规等约束条件下的最短路径，使用MATLAB进行仿真得到轨迹模型。基于问题三的优化模型和搜索方式，重置地图坐标。再次采用算法完成路径规划，并结合MATLAB 中的Automated Driving Toolbox[3]得到仿真示意图。

针对问题五：依据题设条件，我们将来往车辆与无人车的速度进行归一化处理，即将来往车辆（动态障碍）与无人车间的速度差通过补偿，使其与动态障碍物处以一种相对静止的状态。此时，无人车的搜索路径便取决于距离障碍物的远近，由此建立障碍物位置变化约束方程。通过使用Automated Driving Toolbox标定障碍物与无人车的自由速度差，基于新增约束条件和吸引势进行[4]搜索路径。

针对问题六：基于上述五个模型且从算法复杂度和耗时率两方面考虑，问题三、四、五的模型中均采用了算法，在定义损失函数时添加了转角改变、是否倒车、方向盘行驶方向等参数，同时也增加了算法的运算量。在无人车的实际避障中，往往会采用2种或多种算法相结合，通过互补来提高算法效率。

**关键词**：MATLAB；算法；路径规划法；

**一、问题重述**

假设无人车为四轮乘用车，采用前轮转向后轮驱动；车身可认为是一个矩形，车长 5 米，车宽 2 米，轴距 2.8 米；方向盘最大转角 470°，方向盘与前轮转角的传动比为 16：1（方向盘每转动16°，前轮转动 1°），方向盘最大转速为 400°/s；最大油门加速 度 ，极限刹车加速度。

**问题 1**：如图 1 所示场景，无人车（用橙色矩形表示）正在最左侧车道准备调头，存在三个对向车道。附件数据文件中，调头区域边界以红色 边框 A 表示，两条对向车道线以 B、C 表示（均为路面上的虚线），中间的 隔离障碍物以 D 表示。无人车起始位置已经在附件数据文件中给出，当无 人车身触碰到对向的调头区域边界（图 1 中绿色部分）时，认为调头完成。

请阐述控制点位置，解释其理由；基于此建立无人车调头的数学模型，并

给出一个合理可行的算法设计，给出调头轨迹。

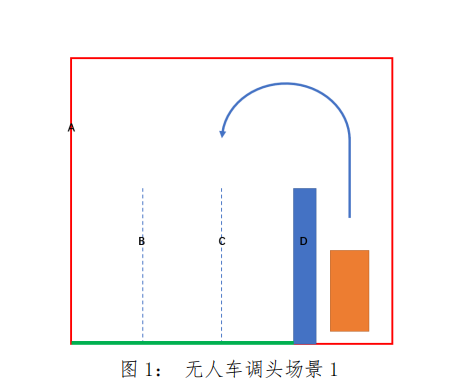


图 1无人车调头场景1

**问题 2**：当调头区域狭窄时，判断什么样的场景下无人车能够在不倒车的情况下完成调头，什么情况需要至少一次倒车才能通过。建立相应的无人车调头的 数学模型，给出合理的算法设计，并给出仿真结果。

**问题 3**：如果道路上还存在其它静止障碍物（附件数据文件中以 F 和 G 表示），当仅存在 F、仅存在 G、或二者都存在时，建立相应的无人车调头的数学模型，给出合理的算法设计，并给出调头轨迹，你的算法应明确如何进行避障？

**问题 4**：按交规规定，如果道路上存在人行横道即斑马线，其所占区域在附件数据文件中以 E 表示，则无人车需要尽可能在不压人行横道的情况下通过，如果无法做到，则需要越过（后轮越过）人行横道后再开始调头，禁止骑人行横道调头。其中标识“√”的为可行通行方式，标识“×”为骑人行道调头方式。建立相应的无人车调头的数学模型，给出合理的算法设计和相应的调头轨迹；当斑马线和障碍物同时存在时，你的模型和算法如何同时满足交规并进行避障绕行？

**问题 5：**无人车在实际路况行驶中进行调头时，道路中的障碍物通常是处于移动状态，如对向来车等，针对问题三两种场景，假设图中标识的障碍物位置为无人车处在起始时刻的初始位置，障碍物以一定的速度移动，假设只考虑与道路平行方向移动，请建立更一般的无人车调头轨迹规划模型并给出相应算法设计和无人车调头策略，给出仿真结果。

**问题 6：**为保证无人驾驶的安全性，无人车的轨迹规划算法必须拥有尽可能高的求解成功率（应倾向使用更收敛更稳定，失败率更低的计算方法）同时为了能对路况进行快速反应，需要以尽可能高的频率进行计算（计算复杂度尽量低）。你的算法如何在求解成功率和求解耗时这两方面优化？

**二、问题分析**

在交会几何和预测观测车辆行驶轨迹的前提下，我们构建了一个基于约束条件的轨迹模型。现有的控制算法是以点线式车辆-道路模型进行设计，忽略车辆与道路形状之间的实际差异，利用参数化曲线[5]表示道路轨迹，车辆根据参数化曲线以及自身状态做出转向动作。

针对问题一：在基于交会几何和预测观测车辆行驶轨迹的前提下，结合车辆行驶动力学模型和转弯时的侧向动力模型，我们构建了一个基于约束条件的轨迹模型。其中，车辆侧向受力时，从车辆侧向、横摆两个自由度进行运动。而且车辆在转弯时，需考虑转弯半径、速度，安全距离等因素。由题设条件可知：加速度、车宽、车道宽度、传动比等为定值，所以我们采用界定分析法确定车辆行驶的轨迹区域，选取加速度、最大曲率、安全距离等作为约束条件，在已有的地图模型基础上建立轨迹预测模型，并用MATLAB中的Automated Driving Toolbox进行仿真。

如下图所示，只要运动轨迹满足界定区域1、3之间就可以完成避障且实现转弯。

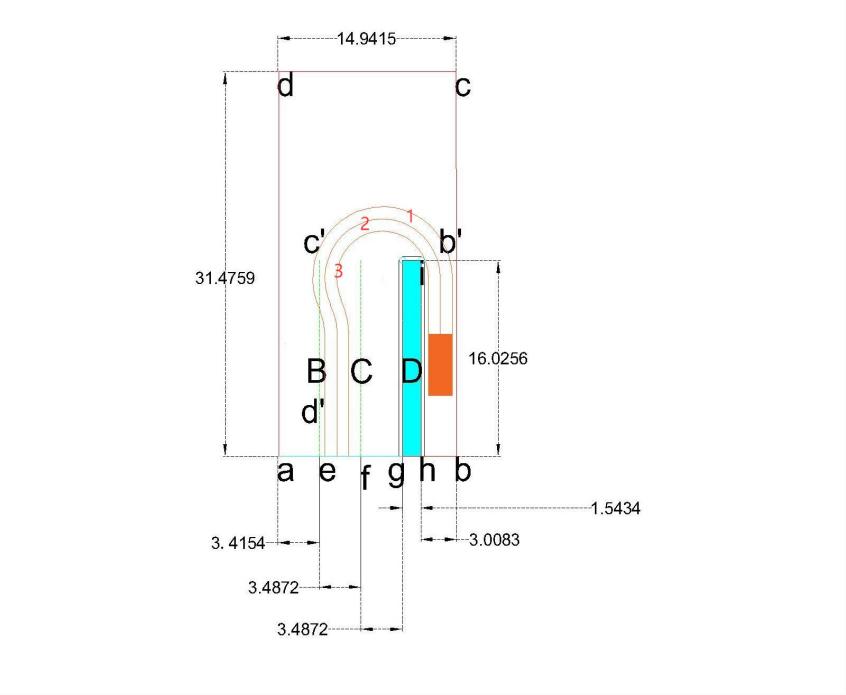


图 2问题一轨迹示意图

针对问题二：在问题一的基础上，已知曲率决定转弯半径，方向盘转角决定的是车转弯时轨迹半径的极限值。而车在转弯时，是否需要倒车的条件是由路宽、车距、最小曲率半径所决定的。这里，我们采用路径规划算法对轨迹进行预测，寻找其界限值。由题设条件可知，临界值由安全距离、最小半径、车宽三者共同决定。根据经验可知，当最小半径都不满足一次性转弯时，那么就必须倒车。同时，也将最小半径设为界定值。轨迹示意图如下所示：

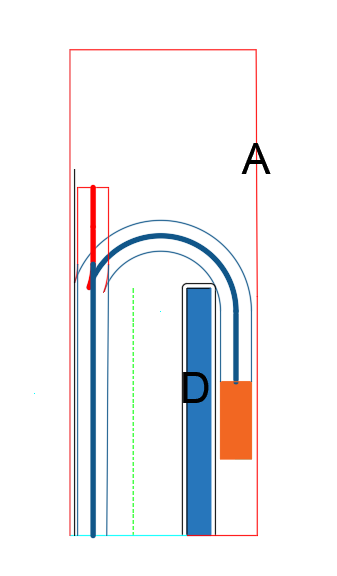


图 3问题二轨迹示意图

针对问题三：依据题干条件设定两个场景，场景一为仅存在F或G障碍物；场景二为障碍物F、G同时都存在。依据动力学原理，建立无人车运动的非完整约束模型和行驶过程中的速度约束方程[6]。基于障碍项、曲率项、光滑项、场函数构建损失函数，并将其作为优化对象。通过寻找约束条件进行非线性规划建模，并采用算法实现其路径规划。构建模型如下：

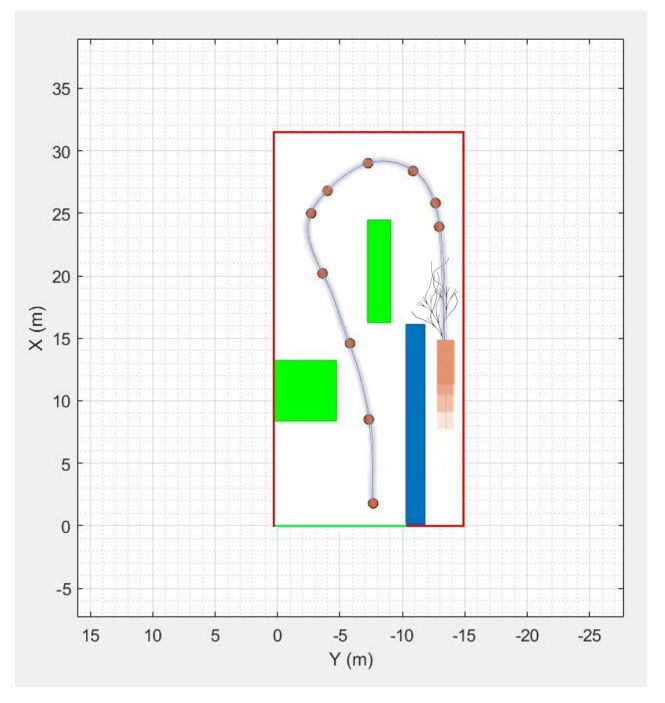


图 4 场景二障碍物F、G都存在时

针对问题四：利用问题二中得到的约束条件和附件数据计算出最小转弯半径，用AutoCAD建立地图模型。在此基础上，进一步判断当道路中存在人行道时，求解无人车不压线通过的临界值。通过路径规划法计算出无人车在满足安全距离、交规等约束条件下的最短路径，使用MATLAB进行仿真得到轨迹模型。基于问题三的优化模型和搜索方式，重置地图坐标。再次采用算法完成路径规划，并结合得到仿真示意图，结果如下：

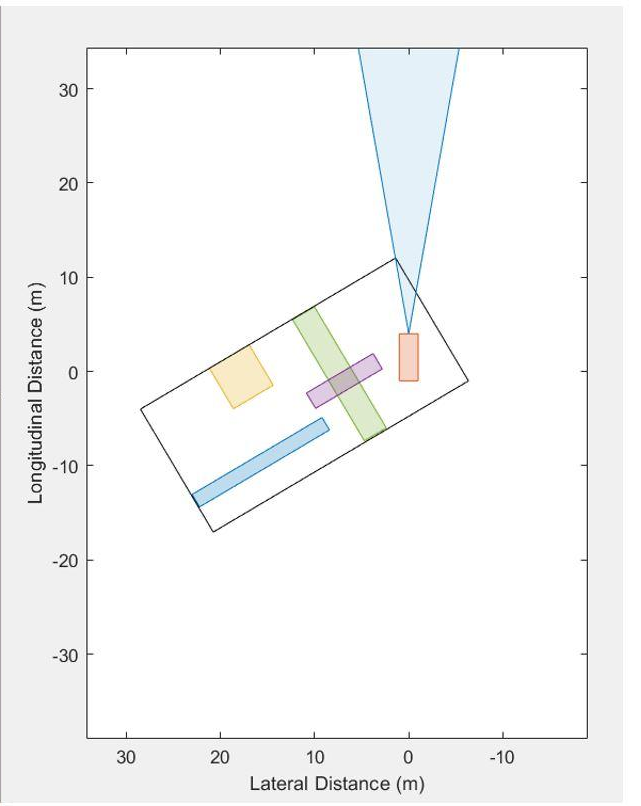


图 5 轨迹三仿真图一

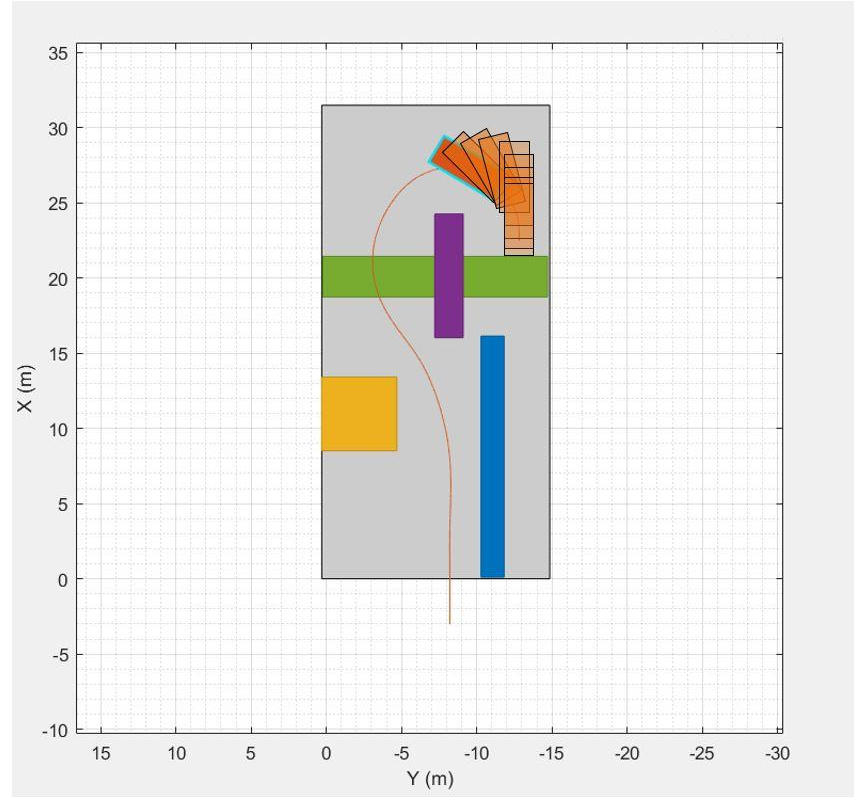


图 6 轨迹三仿真图一

针对问题五：无人车避障主要是利用先进的传感技术来增强无人车的感知能力，将感知系统获取的车速、位置等实时信息反馈给系统，同时根据路况与车流的综合信息判断和分析潜在的安全隐患。依据题设条件，我们将来往车辆与无人车的速度进行归一化处理，即将来往车辆（动态障碍）与无人车间的速度差通过补偿，使其与动态障碍物处以一种相对静止的状态。此时，无人车的搜索路径便取决于距离障碍物的远近，由此建立障碍物位置变化约束方程。通过使用标定障碍物与无人车的自由速度差[7]，基于新增约束条件和吸引势进行搜索,得到轨迹如下图所示：

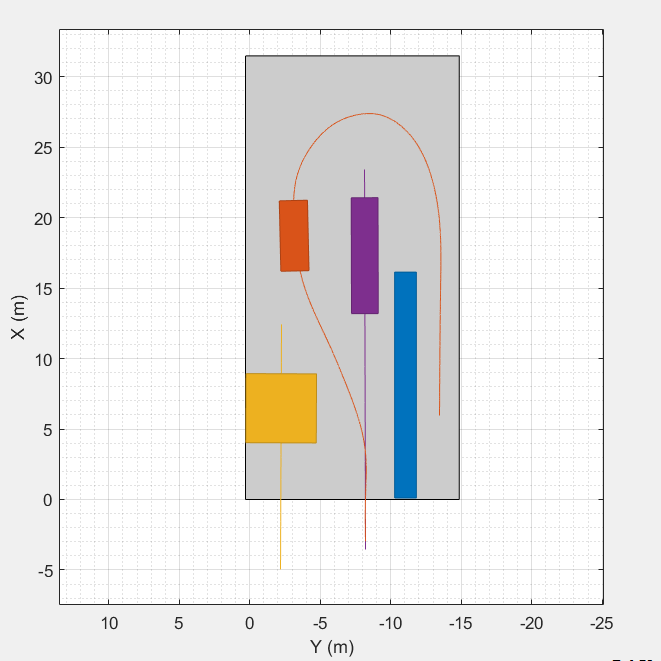


图 7 动态避障轨迹仿真图

针对问题六：结合以上五个模型以及从算法复杂度和耗时率两方面考虑，算法在一定程度上可以很好的解决自动驾驶车辆避障的问题，但也有其优点和局限性。在实际场景应用中，往往采用多种算法相结合的方式来实现避障。

**三、模型假设**

（1）无人车在行驶时，忽略一切意外情况

（2）假设路面平缓无陡坡，忽略风、地球等其他因素影响

（3）轨迹线的最大曲率应尽量不高于 0.205，禁止高于 0.21；2

（4）按轨迹行驶时，在调头完成之前，无人车车身任何点不得与任何障碍物者

调头区域边界发生碰撞，且与障碍物至少保留一个最小安全距离，一般不小于30cm。

（5）问题三模型中，假设无人车具有自我感知和定位能力

**四、定义符号说明**

|  |  |
| --- | --- |
| 符号 | 说明 |
|  | 左轮转角 |
|  | 右轮转角 |
| **l** | 前后轴距 |
| W | 表示前轮间距 |
|  | 控制点的转弯半径 |
|  | 方向盘夹角 |
| w | 车身车宽 |
| s | 表示时间内车身走过的距离 |
|  | 路径节点坐标 |
|  | 最近障碍物坐标 |
|  | 障碍物阈值 |
|  | 障碍物项 |
|  | 吸引势增益函数 |
|  | 吸引势纵向距离 |
|  | 吸引势横向距离 |
|  | 光滑度项 |
|  | Voronoi场函数下，路径远离障碍物的程度 |
|  | 为控制场的衰减率 |
|  | 路径节点到最近的GVD的长度 |
|  | 曲率项 |

**五、模型的建立与求解**

正常行驶状态下的车是以非完整的约束系统为主，在水平地面进行运动。结合无人车动力学模型和运动模型，建立坐标系以及受力和运动分析的前提下，从车辆行驶控制点以及常数变化的角度进行完善，以实现相关参数的受力分析和轨迹规划。

**5.1问题一的模型**

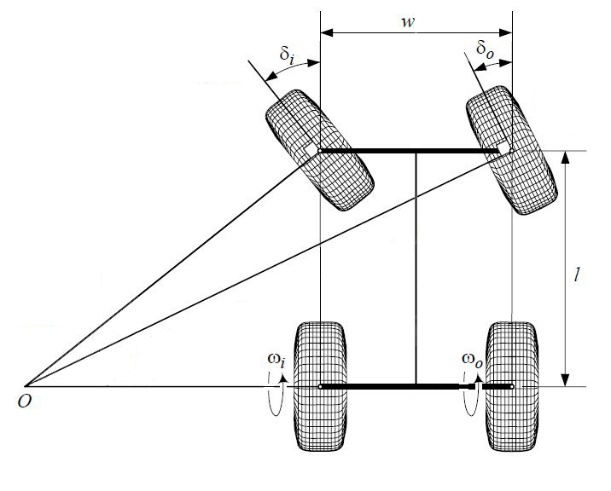
针对问题一，在基于交会几何和预测观测车辆行驶轨迹的前提下，结合车辆行驶动力学模型和转弯时的侧向动力模型，我们构建了一个基于约束条件的轨迹模型。其中，车辆侧向受力时，从车辆侧向、横摆两个自由度[8]进行运动。而且车辆在转弯时，需考虑转弯半径、速度，安全距离等因素。由题设条件可知：加速度、车宽、车道宽度、传动比等为定值，所以我们采用界定分析法确定车辆行驶的轨迹区域，选取加速度、最大曲率、安全距离等作为约束条件。通过动力学方程推导，选取两后轮连线的中点作为控制点。在已有的地图模型基础上建立轨迹预测模型，并用MATLAB中的Automated Driving Toolbox进行仿真。

其中无人车在运动时，需满足以下条件：

1. 转向条件

保证无人车在转弯过程中，车轮与地面之间是纯滚动，需符合Ackerman转向方程： （1）

（表示左轮转角，表示右轮转角，l表示前后轴距，w表示前轮间距）



**图 8 车轮侧向受力图**

（2）最小转弯半径

后轮连续驱动且前轮以最大偏角行驶时无人车所走出的轨迹半径，该位置处对应方向盘也处于极限位置（方向盘转角范围470°，在16:1的传动比下对应前轮摆角范围29.375°,用i表示方向盘与前轮转角的传动比，表示左前轮转角，表示方向盘转角，因此有：=）

（3）转弯几何原理

绘制出无人车转向过程中前轮摆角、车身结构和两后轮中心处轨迹半径以及整车重心处轨迹半径的几何关系，则有

转向速度与转向加速度对转弯的影响：

a = = = （2）

基于上述条件，选取无人车两后轮连线的中点作为控制点，理由如下：

绘制无人车转弯过程如图1所示，将无人车后轮中点设为，根据图中几何关系

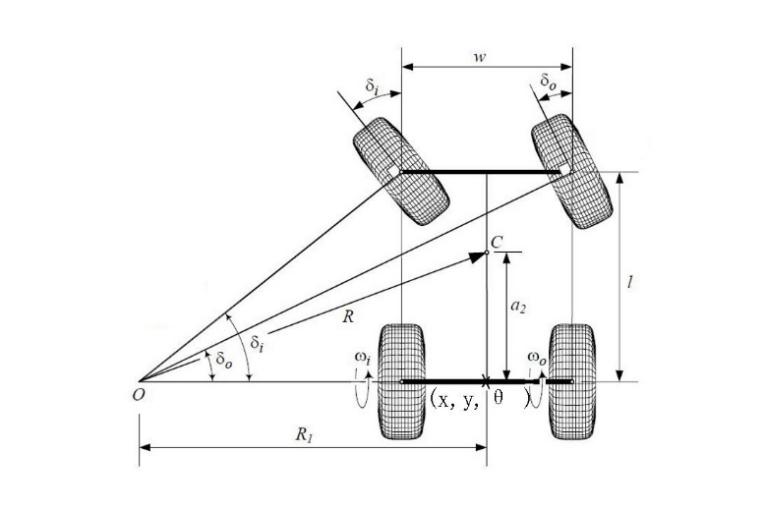


图 9 无人车转弯示意图

有：

（3）

式中：为无人车前后轴距，为控制点的转弯半径，为右侧前轮转角,为无人车车宽

又前轮摆角与方向盘转角满足传动关系：

（4）

式中：为无人车前轮平均转角，为方向盘转角，为无人车方向盘与前轮平均转角的传动比

为保证无人车在转弯过程中，车轮与地面之间是纯滚动，两前车轮转角需符合Ackerman转向方程，如图10所示：

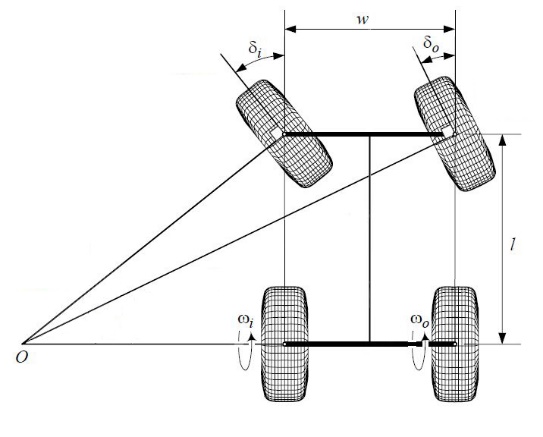


图 10

（5）

式中：表示左前轮转角，表示右前轮转角

因此内外侧轮胎的转向半径不同，联立公式（1）和公式（3）得：

（6）

（7）

则前轮平均转角

（8）

综上无人车行驶任意时刻的位置都可以用点的位置坐标和该点车身与水平方向的夹角来表示，因此选取为控制点。

取为，则有：

（9）

（10）

（11）

式中：、为点位置坐标，为车身与水平方向的夹角，为无人车行驶速度，为方向盘转角。

选取Q点为控制点后，采用界定分析法来确定轨迹区域，即给出所有约束条件则在非约束区域就是无人车运动轨迹。

（12）

结合上述约束条件以及动力学方程，由MATLAB中的Automated Driving Toolbox进行仿真，我们得到无人车避障转弯的运动轨迹如下图所示：

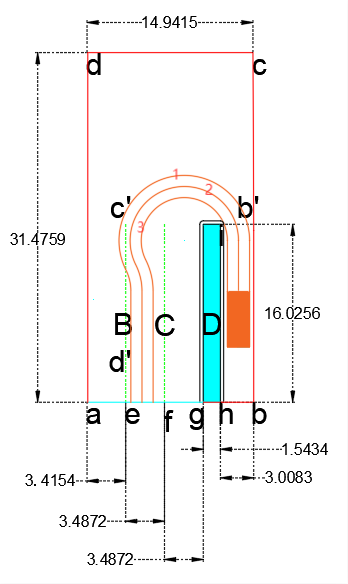


图 11 运动轨迹示意图

如上图所示，运动轨迹介于1、3之间都是满足要求的。

**5.2问题二的模型**

在问题一的基础上，已知曲率决定转弯半径，方向盘转角决定的是车转弯时轨迹半径的极限值。而车在转弯时，是否需要倒车的条件是由路宽、车距、最小曲率半径所决定的。这里，我们采用路径规划算法对轨迹进行预测，寻找其界限值。由题设条件可知，临界值由安全距离、最小半径、车宽三者共同决定。根据经验可知，当最小半径都不满足一次性转弯时，那么就必须倒车。同时，也将其设为界定值，如下图所示：

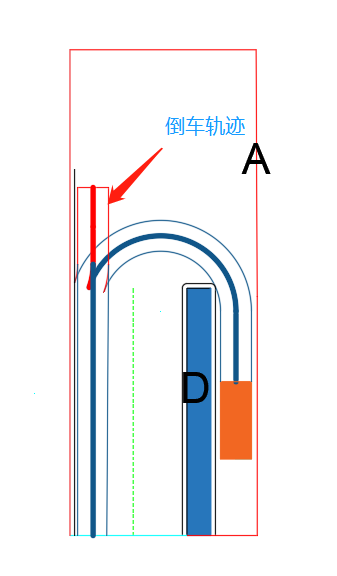


图 12 倒车临界示意图

安全距离、最小半径、初始位置时车与路右边侧距离三者之间共同构成约束条件。

+×2+ （13）

通过计算得到临界值为12.4333367 米，仿真图如上图所示。

**5.3问题三的模型**

通过寻找约束条件进行非线性规划建模，并采用算法实现其路径规划。依据题干条件设定两个场景，场景一为仅存在F或G障碍物；场景二为障碍物F、G同时都存在。由动力学原理，建立无人车运动的非完整约束模型[9]和行驶过程中的速度约束方程。基于障碍项、曲率项、光滑项、场函数构建损失函数，并将其作为优化对象。结合上述二者并利用算法使其能够在未知环境中自动搜索且规划路径。

**5.3.1 算法**

算法主要解决的是在无人车由充分感知和定位能力，则能够在线重新规划生成障碍物地图，并且能够在未知环境中行驶。相较于同类算法，该算法生成的路径光滑且能够满足无人车运动时的非完整约束。其优点在于能够在连续坐标下进行启发式搜索，并能够利用共轭梯度下降法[11]对生成的路径进一步改进，使其至少满足局部最优。

在只考虑无人车的非完整约束而不考虑障碍物和只考虑障碍物而不考虑无人车的非完整约束条件下分别构建启发函数，并求解上述二者的最大值建立损失函数。hybrid A\* 是在与网格精度一致的前提下（对应某一小段时间）使用三种控制动作：最大左转，最大右转，不转向来生成路径，因此该路径是一些受车辆转弯半径约束的圆弧和直线。具体算法实现流程如下图所示：



图13 算法流程示意图

**5.3.2问题三的模型**

由动力学原理，建立无人车运动的非完整约束模型和行驶过程中的速度约束方程。基于障碍项、曲率项、光滑项、场函数构建损失函数，并将其作为优化对象。结合上述二者并利用算法使其能够在未知环境中自动搜索且规划路径。具体实现步骤如下：

取时间微量为,则在微小时间内，可以认为无人车前进近似于延车身方向移动，则有：

（14）

（15）

（16）

联立以上三式可以推出：

=0 （17）

用表示时间内车身走过的距离，则有：

（18）

（19）

联立公式（14）和公式（15）得：

（20）

公式（16）对时间求导得：

（21）

即：

= （22）

综上无人车运动模型为：

（23）

因此对于该运动模型来说，只要给定某个时刻的加速度以及以及方向盘转角，系统就可以得到车辆在该时刻的坐标、方向、无人车行驶速度等状态信息。

公式（16）对时间求导得：

（24）

即：

= （25）

综上无人车运动模型为：

（26）

因此对于该运动模型来说，只要给定某个时刻的加速度以及以及方向盘转角，系统就可以得到车辆在该时刻的坐标、车身方向角、无人车行驶速度等状态信息。由此建立无人车的运动非完整模型。

无人车运动约束方程：由于无人车在行进过程，为了获得较好的驾车体验，应该防止无人车进行左右平移，即将垂直与无人车前进方向的速度限制为0，如图3所示：

将图3中向坐标X、Y分解有：

（27）

（28）

联立公式（20）和公式（21）得：

（29）

因此无人车行驶的速度约束条件为：

（30）

设计路径损失函数：当无人车通过较窄路段时，传统的人工势场法在狭窄路段会生成无人车无法通过的高势场，因此本文采用Voronoi势场函数，其场的值会随着导航中所有的可行空间的大小成比例缩放。Voronoi场函数为：

（30）

式中：代表路径节点到最近障碍物的长度，代表路径节点到最近的GVD的长度，为控制场的衰减率

规定：时，场值为0；场函数连续且其区间为[0,1],、不同时为0；在障碍物附近时场值最大，在GVD边上场值最小。

建立目标函数：考虑到题目中要求的安全性、舒适性、瞬时曲率上限等要求，建立如下损失目标函数：

（31）

表示障碍物项：

（32）

式中为路径节点坐标，为最近障碍物坐标，为障碍物阈值

从公式（26）可以看出，当节点靠近障碍物时，的值会减小，但该差值的二次方是增大的，从而放大了安全性的效果。

表示曲率项，由于从舒适性上考虑无人车在行驶中的每个节点的瞬时曲率都不能高于0.205，于是有：

（33）

（34）

（35）

综上：

（36）

表示光滑度项，用相邻两个节点的位移差的平方来度量，无人车行驶不平滑时，这一项将损失值赋给非均匀变化的节点以保证路径的光滑。

（37）

表示Voronoi场函数下，路径远离障碍物的程度。

（38）

以公式（25）为损失目标函数，公式（23）和公式（5.11）为约束条件，进行非线性规划建模。

表1无人车参数

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 结构参数 | 数值大小 | 约束参数 | 数值大小 | 约束参数 | 数值大小 |
| 车长 | 5m | 方向盘极限转角 | 470° | 加速度 | -5a3 |
| 车宽 | 2m | 最大曲率 | 0.205m-1 | 前轮极限转角 | 28.44° |
| 轴距 | 2.8m | 安全阈值 | 0.3m |  |  |

依据题设要求，分别设定两个场景，场景一为仅存在障碍物F或G；场景二为障碍物F、G同时存在。

在中建立地图模型，具体坐标（A：外轮廓，B：左侧车道线，C：右侧车道线，D：人行道，E、F、G对应三个障碍物）如表2、3所示：

表2 地图模型坐标点

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| A | | B | | C | | D | |
| x | y | x | y | x | y | x | y |
| 14.9415 | 31.4759 | 3.4154 | 16.0256 | 10.3898 | 16.0256 | 11.9332 | 16.0256 |
| 0 | 31.4759 | 3.4154 | 0 | 10.3898 | 0 | 10.3898 | 16.0256 |
| 0 | 0 |  |  |  |  | 10.3898 | 0 |
| 14.9415 | 0 |  |  |  |  | 11.9332 | 0 |

表3 地图模型坐标点

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| E | | F | | G | | 初始位置 | |
| x | y | x | y | x | y | x | y |
| 14.9415 | 21.1625 | 4.9998 | 13.3613 | 9.1994 | 24.2479 | 13.5661333 | 4.9640187 |
| 0 | 21.1625 | 0 | 13.3613 | 7.2996 | 24.2479 |  |  |
| 0 | 18.4633 | 0 | 8.4613 | 7.2996 | 16.0256 |  |  |
| 14.9415 | 18.4633 | 4.9998 | 8.4613 | 9.1994 | 16.0256 |  |  |

然后利用算法不断生成搜索树进行扩展，对不满足碰撞检验的扩展进行剔除。

针对场景一：当仅存在F障碍物时令G=NAN，D=NAN；当仅存在G障碍物时令F=NAN，D=NAN。利用MATLAB Automated Driving Toolbox仿真得到模拟轨迹图分别如下图所示：

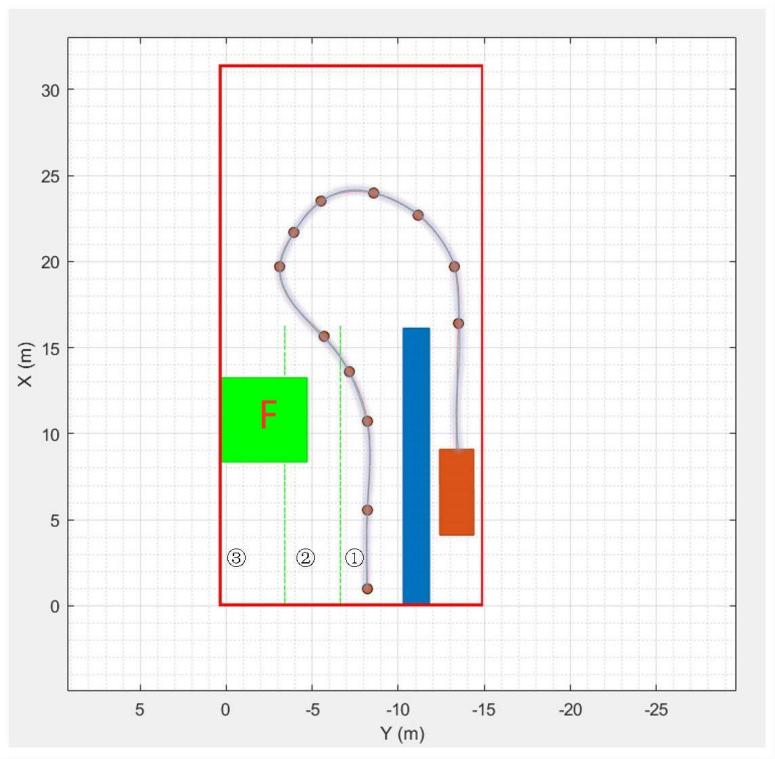


图 14场景一存在障碍物F时

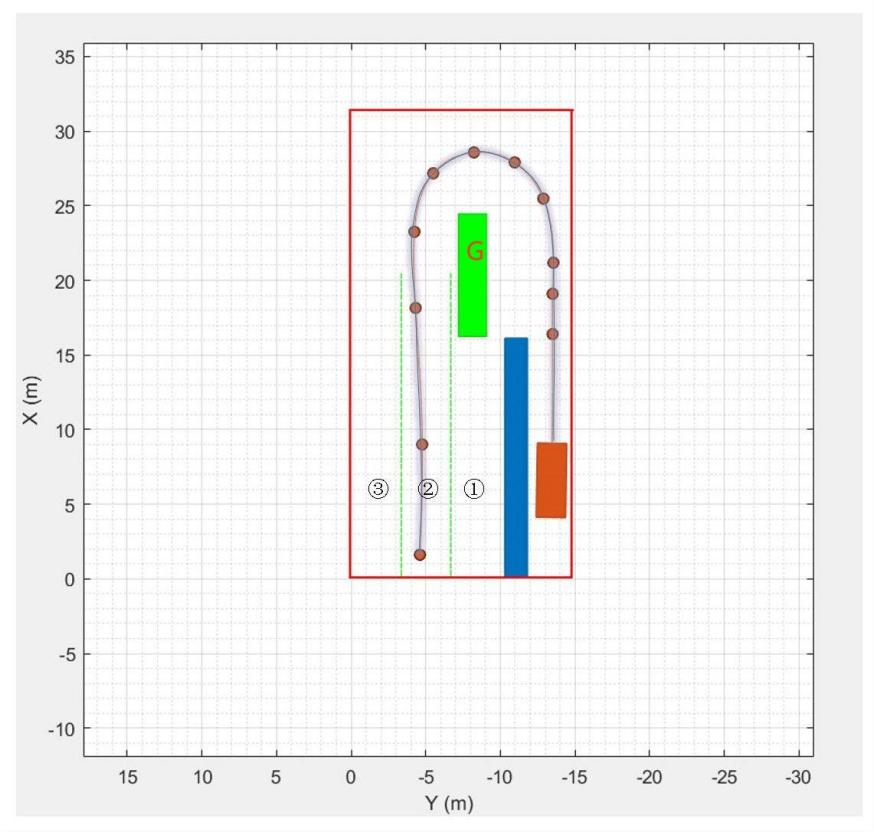


图15 场景一障碍物G存在时

针对场景二：当F、G两者都存在时，仅令D=NAN，将F、G和其余数据按照表2进行完整输入，将控制空间离散化后，以车辆的方向盘转角和方向以及车辆当前姿态为控制输入。利用MATLAB Automated Driving Toolbox仿真得到模拟轨迹图如下图所示：

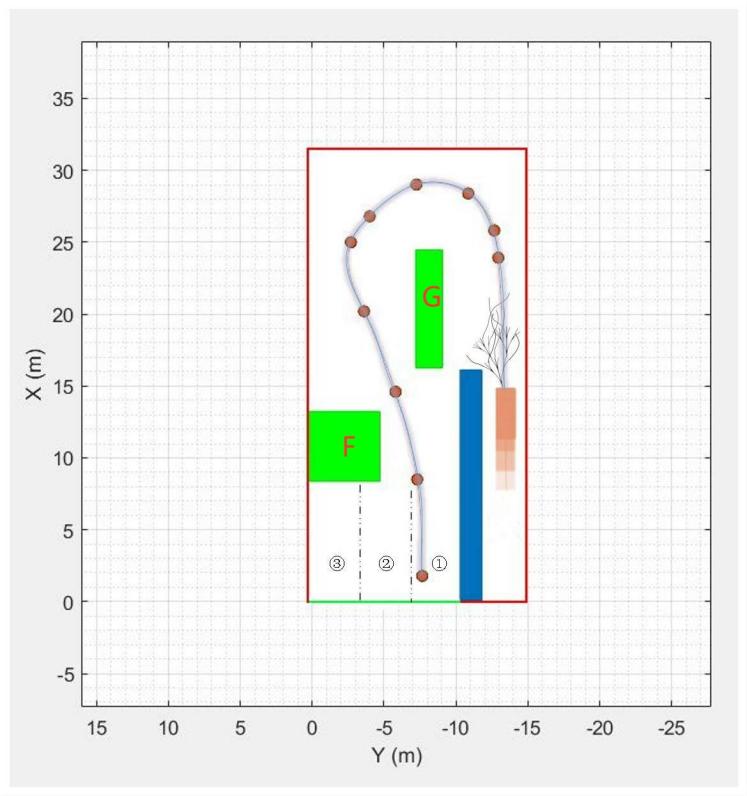


图 11场景2轨迹示意图

从仿真结果来看，障碍物的存在所导致结果是无人车转弯后所走的车道不同，当仅存在F时，无人车转弯后走①车道，当仅存在G时，无人车转弯后走②车道，当F和G都存在时，无人车转弯后走①车道，但存在压线风险，需要无人车离近F和G障碍物后再实时调整轨迹。

**5.4问题四的模型**

针对问题四：利用问题二中得到的约束条件和附件数据计算出最小转弯半径，用AutoCAD建立地图模型。在此基础上，进一步判断当道路中存在人行道时，求解无人车不压线通过的临界值。通过路径规划法计算出无人车在满足安全距离、交规等约束条件下的最短路径，使用MATLAB进行仿真得到轨迹模型。利用问题三得到的优化模型和搜索方式，重置地图坐标。再次采用算法完成路径规划，并结合得到仿真示意图。

根据原附件数据，计算出调头区域、障碍物、安全区域等尺寸，根据公式-，得到无人车左后点的最小转弯半径为3.879m。为方便说明，在AutoCAD中按照相关尺寸作图如图17。

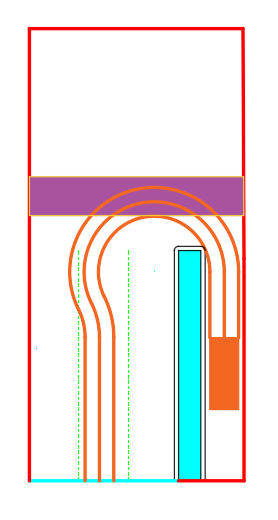


图 17

以无人车左后角为起点，竖直延长出一条直线，根据无人车运动中轨迹必须连续的条件，无人车走出的圆应当和直线相切，这样的轨迹有无数条。将圆轨迹与直线轨迹在相切的前提下自上而下平移直至与障碍物D的安全距离线相交，便是无人车所能走的最短路径。同时在无人车前进过程中，由于前轮摆角不能突变，无人车曲率的回转中心也在发生变化，曲率变化情况如图18所示，

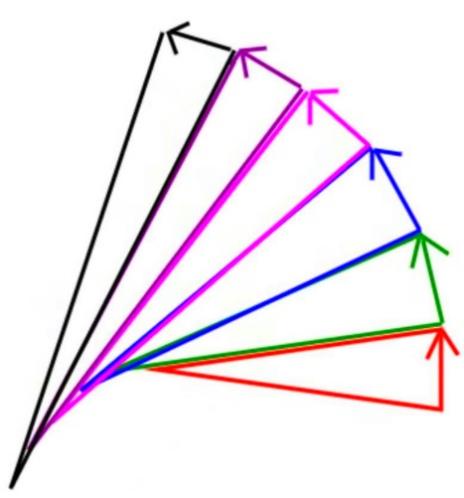


图 18 曲率变化图

根据AutoCAD设计出的路况以及问题二中的无人车运动方程和转角约束条件可以得出，不压人行横道条件下，约束轨迹区域小于车宽要求，无人车不可能曲率连续的条件下从人行道下方完成调头，所以无人车只能越过人行横道后再开始调头。

基于上述分析，无人车需要后轮越过人行横道才能开始调头。结合问题三所给出的优化模型和搜索方式，重置无人车坐标的相关参数，将无人车起始点坐标重新定义为控制点刚刚穿过人行道的临界点。利用进行仿真，继续采用算法进行搜索。

基于反转法的实时路况反馈如下图所示：

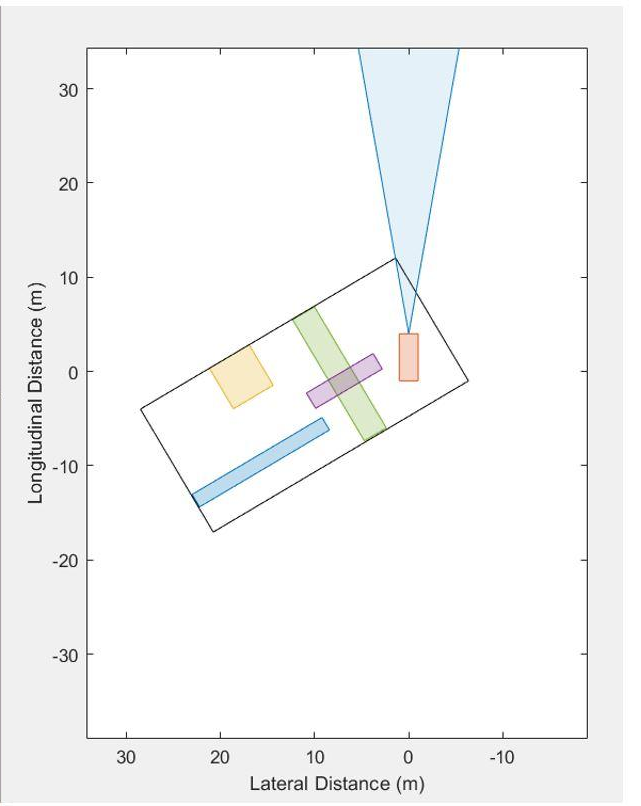


图 19实时路况反馈图

无人车后轮越过人行道调头时的轨迹：

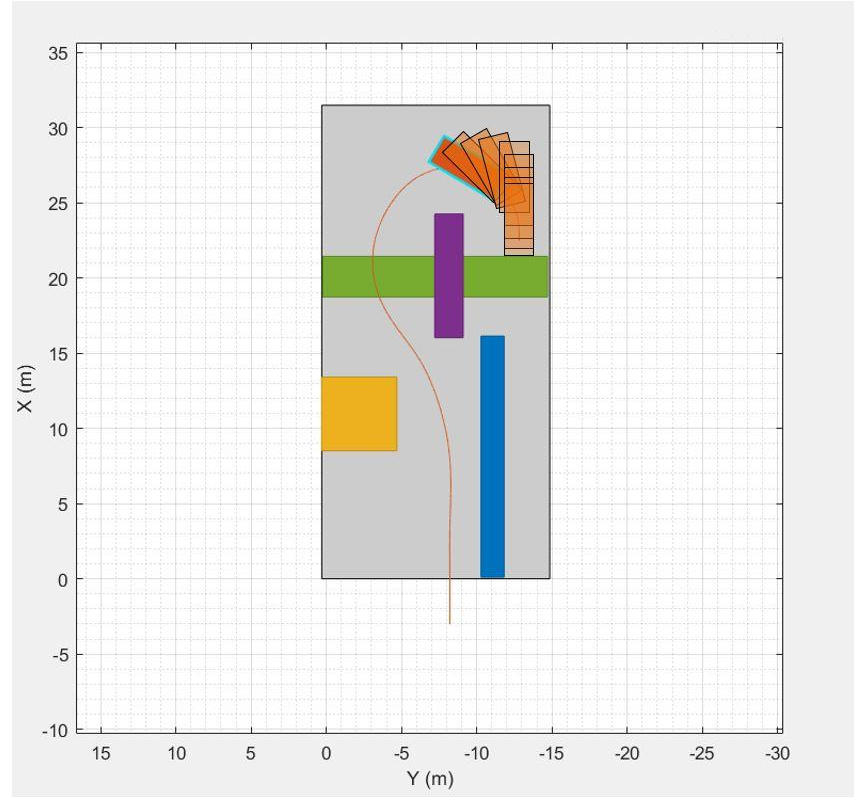


图 20 调头轨迹示意图

**5.5问题五的模型**

无人车避障主要是利用先进的传感技术来增强无人车的感知能力，将感知系统获取的车速、位置等实时信息反馈给系统，同时根据路况与车流的综合信息判断和分析潜在的安全隐患。依据题设条件，我们将来往车辆与无人车的速度进行归一化处理，即将来往车辆（动态障碍）与无人车间的速度差通过补偿，使其与动态障碍物处以一种相对静止的状态。此时，无人车的搜索路径便取决于距离障碍物的远近，由此建立障碍物位置变化约束方程。通过使用 标定障碍物与无人车的自由速度差，基于新增约束条件和吸引势进行搜索,得到路径轨迹。

当无人车在调头时，基于反转法假设无人车在存在障碍物F、G以一定速度进行移动时为静止状态。在实际生活中，全部处于对向车道的F、G以及调头完成的无人车都会同向行驶。定义F、G障碍物的计算移动速度为它们实际速度与无人车的速度差。这时，无人车的搜索路径便完全取决于距离障碍物的远近，由此就可以转化为问题三中的静态障碍物添加障碍物位置变化约束方程后的轨迹搜索问题，原理如图21所示。

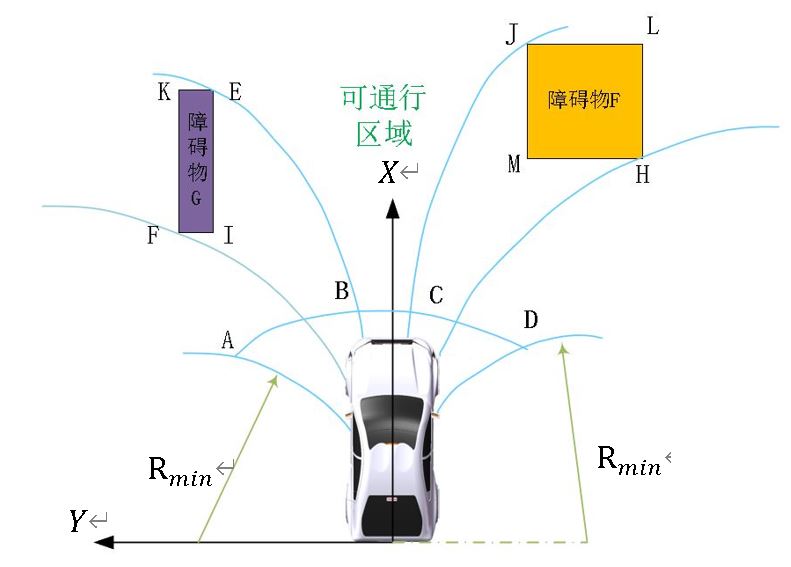


图 21 基于反转法的障碍物搜索原理图

从图9可以看出任何障碍物的边界轮廓都内接与控制点向四周发射的圆弧中，则有：

设和分别构成第i个障碍物的边界左右圆弧曲线，则有如下约束方程：

（39）

（40）

（41）

（42）

（43）

基于上述关系构建GAUSSIAN函数人工势场：

（44）

式中：为吸引势增益系数，和反应了吸引势在纵向和横向的作用距离

在中标定障碍物与无人车的自由速度差，基于新增约束和吸引势进行搜索，轨迹路径如图22、23所示，势场关系如图24所示。

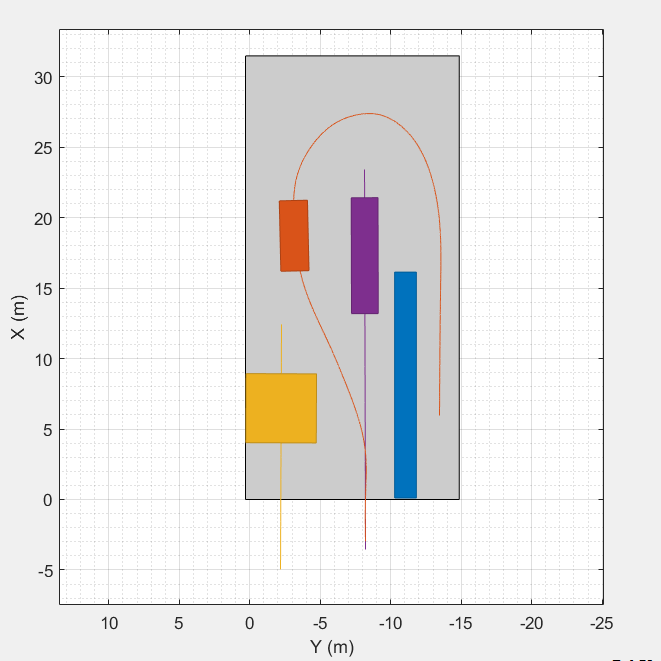


图22 轨迹路径示意图

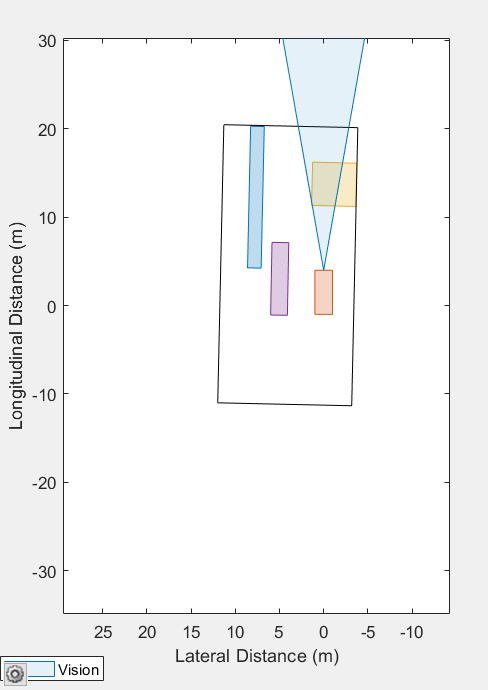


图 23 轨迹路径示意图

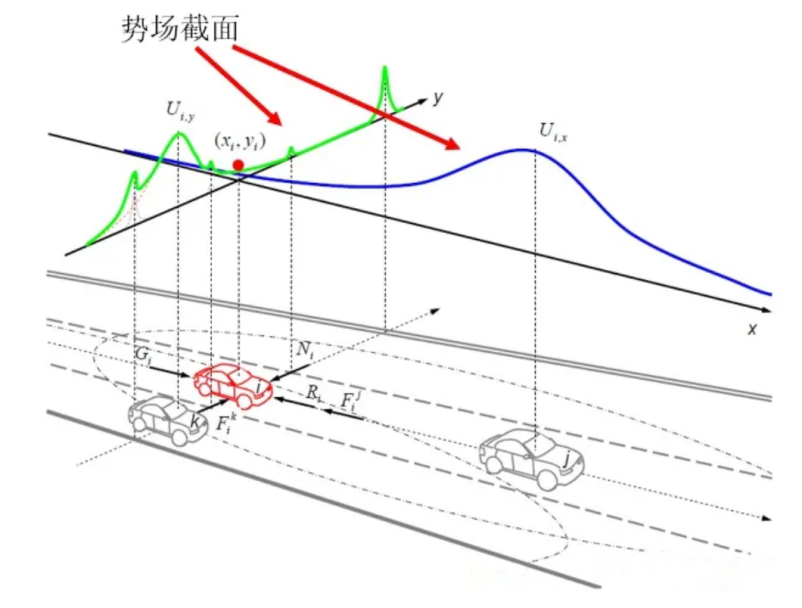


图 24 势场关系图

从仿真结果来看，通过反转法可以将动态障碍物以速度差的形式转化为静态障碍物的避障和路径规划问题。通过势场感知，最终以速度的和加速度的变化调整无人车的位置和姿态，实现障碍物的规避。

场景模拟如下图所示：



图 25 场景模拟图

**5.6问题六的模型**

基于上述五个模型且从算法复杂度和耗时率两方面考虑，问题三、四、五的模型中均采用了算法，在定义损失函数时添加了转角改变、是否倒车、方向盘行驶方向等参数，同时也增加了算法的运算量。在无人车的实际避障中，往往会采用2种或多种算法相结合，通过互补来提高算法效率。

**5.6.1智能优化算法**

智能优化算法[12]一般建立在生物智能或物理现象基础上的随机搜索算法，包括模糊逻辑算法、遗传算法、快速扩展随机数、蚁群算法等。

模糊逻辑算法主要是根据人类驾驶经验，设计出一个模糊控制规则库。将传感器获得的信息作为输入，经过模糊推理后得出车辆的运动学模型。但模糊规则往往是人们通过经验预先设定好的，所以存在无法学习和灵活性差的缺点，且模糊规则数量随着输入的增多呈指数型增长。

**5.6.2问题六的模型**

基于上述分析，通过采用分层分级的思想，将避障过程分为无人车绕开障碍物过程和车辆趋向目标过程的新方法，分别设计了一个避障行为控制器和目标趋向控制器，减少了算法的计算量也在一定程度上提高了运算速度。

在车辆的实际避障中，往往采用2种或多种算法相结合，通过互补来提高效率。

**六、模型的评价**

模型的优点：

（1）模型构建合理准确，基于动力学和几何交汇构建地图模型，并利用MATLAB中的Automated Driving Toolbox进行仿真得到的轨迹图准确可靠。

（2）本文采用算法，该算法将状态空间通过确定的方式离散化，然后利用启发式搜索算法搜索可行解甚至是最优解，是一种具体的、行之有效的、实用价值很高的常用路径规划算法。

（3）算法在总体上较为合理清晰。通过利用已有的函数和仿真测试构建的轨迹模型具有很强的鲁棒性。

（4）从算法路径寻优能力、算法实时性及算法复杂度等方面来看，本文的模型具有很强的实用性。

**七、参考文献**

1. 沈继红，数学建模精解精练[M] ，哈尔滨：哈尔滨工程大学出版社，2007
2. 龚晓岚，数学建模[M]，哈尔滨：哈尔滨：哈尔滨工业大学出版社，2012
3. 潘峰,鲍泓. 强化学习的自动驾驶控制技术研究进展[J]. 中国图象图形学报,2021,26(1):28-35.
4. 陈方樱，沈思，数据分析方法及SPSS应用，北京：科学出版社，2016
5. 杨朝阳,陈恳,彭丽. 基于模型预测控制无人驾驶车辆避障轨迹跟踪器设计[J]. 湖北汽车工业学院学报,2020,34(1):11-15,20. DOI:10.3969/j.issn.1008-5483.2020.01.003.
6. 李文越. 无人驾驶汽车自动寻路与动态避障研究[D]. 陕西:长安大学,2018.
7. 陈浩,喻厚宇. 基于势场搜索的无人车动态避障路径规划算法研究[J]. 北京汽车,2019(4):1-5,31. DOI:10.14175/j.issn.1002-4581.2019.04.001.
8. 方朋朋,杨家富,施杨洋,等. 基于梯度下降法和改进人工势场法的无人车避障方法[J]. 制造业自动化,2018,40(11):81-84. DOI:10.3969/j.issn.1009-0134.2018.11.021.
9. 陈晶,白丽丽. 基于人工智能的汽车无人驾驶避障模块的策略研究[J]. 时代汽车,2018(6):43-44. DOI:10.3969/j.issn.1672-9668.2018.06.016.
10. 任巢康. 基于模型预测控制的无人车轨迹跟踪与避障控制[D]. 浙江:浙江大学,2020.
11. 张高明. 室外无人车自主导航方法研究[D]. 浙江:浙江大学,2017.
12. 刘创. 无人驾驶车辆多目标容错跟踪与轨迹预测研究[D]. 浙江:浙江大学,2019.

1. **附录**

部分代码如下所示：

function [allData, scenario, sensor] = ds\_\_4()

%ds\_\_4 - Returns sensor detections

% allData = ds\_\_4 returns sensor detections in a structure

% with time for an internally defined scenario and sensor suite.

%

% [allData, scenario, sensors] = ds\_\_4 optionally returns

% the drivingScenario and detection generator objects.

% Generated by MATLAB(R) 9.8 (R2020a) and Automated Driving Toolbox 3.1 (R2020a).

% Generated on: 18-Apr-2021 12:13:54

% Create the drivingScenario object and ego car

[scenario, egoVehicle] = createDrivingScenario;

% Create all the sensors

sensor = createSensor(scenario);

allData = struct('Time', {}, 'ActorPoses', {}, 'ObjectDetections', {}, 'LaneDetections', {});

running = true;

while running

% Generate the target poses of all actors relative to the ego vehicle

poses = targetPoses(egoVehicle);

time = scenario.SimulationTime;

% Generate detections for the sensor

laneDetections = [];

[objectDetections, numObjects, isValidTime] = sensor(poses, time);

objectDetections = objectDetections(1:numObjects);

% Aggregate all detections into a structure for later use

if isValidTime

allData(end + 1) = struct( ...

'Time', scenario.SimulationTime, ...

'ActorPoses', actorPoses(scenario), ...

'ObjectDetections', {objectDetections}, ...

'LaneDetections', {laneDetections}); %#ok<AGROW>

end

% Advance the scenario one time step and exit the loop if the scenario is complete

running = advance(scenario);

end

% Restart the driving scenario to return the actors to their initial positions.

restart(scenario);

% Release the sensor object so it can be used again.

release(sensor);

%%%%%%%%%%%%%%%%%%%%

% Helper functions %

%%%%%%%%%%%%%%%%%%%%

% Units used in createSensors and createDrivingScenario

% Distance/Position - meters

% Speed - meters/second

% Angles - degrees

% RCS Pattern - dBsm

function sensor = createSensor(scenario)

% createSensors Returns all sensor objects to generate detections

% Assign into each sensor the physical and radar profiles for all actors

profiles = actorProfiles(scenario);

sensor = visionDetectionGenerator('SensorIndex', 1, ...

'SensorLocation', [4 0], ...

'MaxRange', 100, ...

'DetectorOutput', 'Objects only', ...

'Intrinsics', cameraIntrinsics([1814.81018227767 1814.81018227767],[320 240],[480 640]), ...

'ActorProfiles', profiles);

function [scenario, egoVehicle] = createDrivingScenario

% createDrivingScenario Returns the drivingScenario defined in the Designer

% Construct a drivingScenario object.

scenario = drivingScenario;

% Add all road segments

roadCenters = [31.476 -5 0;

0 -5 0];

laneSpecification = lanespec(3, 'Width', 3.4872);

road(scenario, roadCenters, 'Lanes', laneSpecification);

roadCenters = [31.476 -12.507 0;

0 -12.507 0];

laneSpecification = lanespec(1, 'Width', 4.5517);

road(scenario, roadCenters, 'Lanes', laneSpecification);

% Add the actors

actor(scenario, ...

'ClassID', 5, ...

'Length', 16.0256, ...

'Width', 1.5434, ...

'Height', 0.8, ...

'Position', [8.12 -11.05 0.01], ...

'PlotColor', [0 114 189] / 255);

actor(scenario, ...

'ClassID', 5, ...

'Length', 4.9, ...

'Width', 4.9998, ...

'Height', 0.8, ...

'Position', [-171.8 46.5 0], ...

'PlotColor', [0 255 0] / 255);

actor(scenario, ...

'ClassID', 5, ...

'Length', 8.2223, ...

'Width', 1.8998, ...

'Height', 0.8, ...

'Position', [-169.7 -11 0], ...

'PlotColor', [0 255 0] / 255);

% Add the ego vehicle

egoVehicle = vehicle(scenario, ...

'ClassID', 1, ...

'Length', 5, ...

'Width', 2, ...

'Position', [5.89 -13.39 0.01], ...

'PlotColor', [0.85 0.325 0.098]);

waypoints = [5.89 -13.39 0.01;

13.62 -13.46 0.01;

16.25 -13.46 0.01;

19.51 -13.11 0.01;

26.74 -10.41 0.01;

27.31 -9.14 0.01;

27.31 -7.72 0.01;

26.67 -6.02 0.01;

25.18 -4.46 0.01;

22.84 -3.32 0.01;

20.36 -3.11 0.01;

17.81 -3.96 0.01;

14.62 -6.16 0.01;

11.99 -7.36 0.01;

7.95 -8.21 0.01;

3.27 -8.21 0.01;

0.15 -8.21 0.01;

-3.04 -8.21 0];

speed = 30;

trajectory(egoVehicle, waypoints, speed);

vehicle(scenario, ...

'ClassID', 1, ...

'Length', 4.9, ...

'Width', 4.9998, ...

'Position', [9.51 -2.19 0.01], ...

'PlotColor', [0.929 0.694 0.125]);

vehicle(scenario, ...

'ClassID', 1, ...

'Length', 8.2223, ...

'Width', 1.8998, ...

'Position', [17.03 -8.15 0.01], ...

'PlotColor', [0.494 0.184 0.556]);

function [allData, scenario, sensor] = ds\_\_4()

%ds\_\_4 - Returns sensor detections

% allData = ds\_\_4 returns sensor detections in a structure

% with time for an internally defined scenario and sensor suite.

%

% [allData, scenario, sensors] = ds\_\_4 optionally returns

% the drivingScenario and detection generator objects.

% Generated by MATLAB(R) 9.8 (R2020a) and Automated Driving Toolbox 3.1 (R2020a).

% Generated on: 18-Apr-2021 12:16:07

% Create the drivingScenario object and ego car

[scenario, egoVehicle] = createDrivingScenario;

% Create all the sensors

sensor = createSensor(scenario);

allData = struct('Time', {}, 'ActorPoses', {}, 'ObjectDetections', {}, 'LaneDetections', {});

running = true;

while running

% Generate the target poses of all actors relative to the ego vehicle

poses = targetPoses(egoVehicle);

time = scenario.SimulationTime;

% Generate detections for the sensor

laneDetections = [];

[objectDetections, numObjects, isValidTime] = sensor(poses, time);

objectDetections = objectDetections(1:numObjects);

% Aggregate all detections into a structure for later use

if isValidTime

allData(end + 1) = struct( ...

'Time', scenario.SimulationTime, ...

'ActorPoses', actorPoses(scenario), ...

'ObjectDetections', {objectDetections}, ...

'LaneDetections', {laneDetections}); %#ok<AGROW>

end

% Advance the scenario one time step and exit the loop if the scenario is complete

running = advance(scenario);

end

% Restart the driving scenario to return the actors to their initial positions.

restart(scenario);

% Release the sensor object so it can be used again.

release(sensor);

%%%%%%%%%%%%%%%%%%%%

% Helper functions %

%%%%%%%%%%%%%%%%%%%%

% Units used in createSensors and createDrivingScenario

% Distance/Position - meters

% Speed - meters/second

% Angles - degrees

% RCS Pattern - dBsm

function sensor = createSensor(scenario)

% createSensors Returns all sensor objects to generate detections

% Assign into each sensor the physical and radar profiles for all actors

profiles = actorProfiles(scenario);

sensor = visionDetectionGenerator('SensorIndex', 1, ...

'SensorLocation', [4 0], ...

'MaxRange', 100, ...

'DetectorOutput', 'Objects only', ...

'Intrinsics', cameraIntrinsics([1814.81018227767 1814.81018227767],[320 240],[480 640]), ...

'ActorProfiles', profiles);

function [scenario, egoVehicle] = createDrivingScenario

% createDrivingScenario Returns the drivingScenario defined in the Designer

% Construct a drivingScenario object.

scenario = drivingScenario;

% Add all road segments

roadCenters = [31.476 -5 0;

0 -5 0];

laneSpecification = lanespec(3, 'Width', 3.4872);

road(scenario, roadCenters, 'Lanes', laneSpecification);

roadCenters = [31.476 -12.507 0;

0 -12.507 0];

laneSpecification = lanespec(1, 'Width', 4.5517);

road(scenario, roadCenters, 'Lanes', laneSpecification);

% Add the actors

actor(scenario, ...

'ClassID', 5, ...

'Length', 16.0256, ...

'Width', 1.5434, ...

'Height', 0.8, ...

'Position', [8.12 -11.05 0.01], ...

'PlotColor', [0 114 189] / 255);

actor(scenario, ...

'ClassID', 5, ...

'Length', 4.9, ...

'Width', 4.9998, ...

'Height', 0.8, ...

'Position', [-171.8 46.5 0], ...

'PlotColor', [0 255 0] / 255);

actor(scenario, ...

'ClassID', 5, ...

'Length', 8.2223, ...

'Width', 1.8998, ...

'Height', 0.8, ...

'Position', [-169.7 -11 0], ...

'PlotColor', [0 255 0] / 255);

% Add the ego vehicle

egoVehicle = vehicle(scenario, ...

'ClassID', 1, ...

'Length', 5, ...

'Width', 2, ...

'Position', [5.89 -13.39 0.01], ...

'PlotColor', [0.85 0.325 0.098]);

waypoints = [5.89 -13.39 0.01;

13.62 -13.46 0.01;

16.25 -13.46 0.01;

19.51 -13.11 0.01;

26.74 -10.41 0.01;

27.31 -9.14 0.01;

27.31 -7.72 0.01;

26.67 -6.02 0.01;

25.18 -4.46 0.01;

22.84 -3.32 0.01;

20.36 -3.11 0.01;

17.81 -3.96 0.01;

12.91 -5.23 0.01;

10.79 -5.45 0.01;

7.52 -5.8 0.01;

4.54 -6.09 0.01;

2.2 -6.23 0.01;

-0.99 -6.16 0];

speed = 30;

trajectory(egoVehicle, waypoints, speed);

vehicle(scenario, ...

'ClassID', 1, ...

'Length', 8.2223, ...

'Width', 1.8998, ...

'Position', [17.03 -8.15 0.01], ...

'PlotColor', [0.494 0.184 0.556]);

function [allData, scenario, sensor] = ds\_\_4()

%ds\_\_4 - Returns sensor detections

% allData = ds\_\_4 returns sensor detections in a structure

% with time for an internally defined scenario and sensor suite.

%

% [allData, scenario, sensors] = ds\_\_4 optionally returns

% the drivingScenario and detection generator objects.

% Generated by MATLAB(R) 9.8 (R2020a) and Automated Driving Toolbox 3.1 (R2020a).

% Generated on: 18-Apr-2021 12:20:07

% Create the drivingScenario object and ego car

[scenario, egoVehicle] = createDrivingScenario;

% Create all the sensors

sensor = createSensor(scenario);

allData = struct('Time', {}, 'ActorPoses', {}, 'ObjectDetections', {}, 'LaneDetections', {});

running = true;

while running

% Generate the target poses of all actors relative to the ego vehicle

poses = targetPoses(egoVehicle);

time = scenario.SimulationTime;

% Generate detections for the sensor

laneDetections = [];

[objectDetections, numObjects, isValidTime] = sensor(poses, time);

objectDetections = objectDetections(1:numObjects);

% Aggregate all detections into a structure for later use

if isValidTime

allData(end + 1) = struct( ...

'Time', scenario.SimulationTime, ...

'ActorPoses', actorPoses(scenario), ...

'ObjectDetections', {objectDetections}, ...

'LaneDetections', {laneDetections}); %#ok<AGROW>

end

% Advance the scenario one time step and exit the loop if the scenario is complete

running = advance(scenario);

end

% Restart the driving scenario to return the actors to their initial positions.

restart(scenario);

% Release the sensor object so it can be used again.

release(sensor);

%%%%%%%%%%%%%%%%%%%%

% Helper functions %

%%%%%%%%%%%%%%%%%%%%

% Units used in createSensors and createDrivingScenario

% Distance/Position - meters

% Speed - meters/second

% Angles - degrees

% RCS Pattern - dBsm

function sensor = createSensor(scenario)

% createSensors Returns all sensor objects to generate detections

% Assign into each sensor the physical and radar profiles for all actors

profiles = actorProfiles(scenario);

sensor = visionDetectionGenerator('SensorIndex', 1, ...

'SensorLocation', [4 0], ...

'MaxRange', 100, ...

'DetectorOutput', 'Objects only', ...

'Intrinsics', cameraIntrinsics([1814.81018227767 1814.81018227767],[320 240],[480 640]), ...

'ActorProfiles', profiles);

function [scenario, egoVehicle] = createDrivingScenario

% createDrivingScenario Returns the drivingScenario defined in the Designer

% Construct a drivingScenario object.

scenario = drivingScenario;

% Add all road segments

roadCenters = [31.476 -5 0;

0 -5 0];

laneSpecification = lanespec(3, 'Width', 3.4872);

road(scenario, roadCenters, 'Lanes', laneSpecification);

roadCenters = [31.476 -12.507 0;

0 -12.507 0];

laneSpecification = lanespec(1, 'Width', 4.5517);

road(scenario, roadCenters, 'Lanes', laneSpecification);

% Add the actors

actor(scenario, ...

'ClassID', 5, ...

'Length', 16.0256, ...

'Width', 1.5434, ...

'Height', 0.8, ...

'Position', [8.12 -11.05 0.01], ...

'PlotColor', [0 114 189] / 255);

actor(scenario, ...

'ClassID', 5, ...

'Length', 4.9, ...

'Width', 4.9998, ...

'Height', 0.8, ...

'Position', [-171.8 46.5 0], ...

'PlotColor', [0 255 0] / 255);

actor(scenario, ...

'ClassID', 5, ...

'Length', 8.2223, ...

'Width', 1.8998, ...

'Height', 0.8, ...

'Position', [-169.7 -11 0], ...

'PlotColor', [0 255 0] / 255);

% Add the ego vehicle

egoVehicle = vehicle(scenario, ...

'ClassID', 1, ...

'Length', 5, ...

'Width', 2, ...

'Position', [5.89 -13.39 0.01], ...

'PlotColor', [0.85 0.325 0.098]);

waypoints = [5.89 -13.39 0.01;

13.62 -13.39 0.01;

16.18 -13.32 0.01;

18.38 -12.97 0.01;

20.36 -11.9 0.01;

21.29 -10.77 0.01;

22.28 -8.5 0.01;

22 -7.01 0.01;

20.72 -4.95 0.01;

19.23 -4.38 0.01;

17.38 -4.74 0.01;

16.18 -6.02 0.01;

14.69 -7.08 0.01;

11.99 -7.79 0.01;

7.95 -8.21 0.01;

3.27 -8.21 0.01;

0.15 -8.21 0.01;

-3.04 -8.21 0];

speed = 30;

trajectory(egoVehicle, waypoints, speed);

vehicle(scenario, ...

'ClassID', 1, ...

'Length', 4.9, ...

'Width', 4.9998, ...

'Position', [9.51 -2.19 0.01], ...

'PlotColor', [0.929 0.694 0.125]);