队伍编号	MC2203201
题号	C

无人车自动泊车中的路径规划研究

摘 要

针对无人车自动规划路线停车问题,为了能快速而又准确的停放好所有的无人车，就要涉及到无人车的自动泊车问题，无人车的泊车问题在实际作业中有着非常广泛的应用。本文对无人车的路线规划进行分析，先讨论一辆无人车的泊车方案，再对多辆无人车的泊车以及不同的情况进行路径规划，建立出无人车路径规划模型并设计出合适的无人车停车算法。

针对问题一，我们根据无人车的相关参数在忽略影响因素有汽车特性、行驶速度、道路特性、稳定性等后提出了无人车直线运动模型以及转弯模型，根据题意，通过阿克曼几何关系求出了无人车的最小转弯半径为$R_{min}=4.9946m$,根据物理知识以及数学知识求出无人车在$V_{初}=0m/s$的情况下无人车加速到$20km/h$需要的最短路程为$1.85185m$,当车速达到$20km/h$时，曲率的变化率可以描述为$k=\left|\frac{\frac{\pi}{2}-\varphi-\delta}{S}\right|$。

针对问题二，在面对无人车的泊车问题，我们采用第一问就已经提出的模型，对无人车的运动状态进行分析就可以得出无人车运动学模型，再以上述模型为基础设计出了无人车在停车场内移动过程中的转弯模型以及直线模型，最后就是停车模型，我们根据不同的停车位分类出了“垂直泊车策略”、“平行泊车策略”以及“倾斜泊车策略”，最后通过$matlab$和$python$代码设计出了三种不同的停车位的相关算法，导出了相应数据，最后用这些算法和数据得到了三种针对 10 号停车位、82 号停车位以及 31 号停车位的 gif 动图。

针对问题三，在给定位置的无人车中，我们是通过$V_{车}$来左右无人车在面对复杂的路径问题上的选择，事实上，只要对无人车的速度进行合理的约束就可以对无人车的路径规划进行控制。

针对问题四，面对多辆无人车的泊车问题，我们先是设计出了流程图来帮助我们梳理多辆无人车的相关流程问题，我们又通过 HCA*算法对多辆无人车的泊车问题进行优化，得到模型多无人车路径规划模型。

最后，我们评价了四个问题中模型的优缺点，并给出了改进的思路。由于大量运算是基于物理知识以及基础数学，以上模型能够很好的对不同数量的无人车以及不同位置的无人车，具有较好的通用性。

关键词： 阿克曼几何模型； 无人车运动模型； 无人车泊车模型； 路径规划； HCA*算法；

目录

一、 问题重述

1.1 问题背景

本题的目的是实现在停车场中用无人乘用车自动泊车的功能。无人车即阿克曼结构的乘用车，前轮转向，后轮驱动，车长 4.9m，车宽 8m；车子轴距 2.8m，轮间距 1.7m；最大油门加速度为 $3.0m/s^2$，极限最大减速度为-$6.0m/s^2$，加加速度不超过 $20.0m/s^3$ 为宜；方向盘最大转角 470°，方向盘与前轮转角的传动比为 16∶1（方向盘转动 16°，前轮转动 1°），方向盘最大转速为 $400°/s$。停车位有三种类型：垂直停车位、平行停车位及倾斜停车位。在停车场平面图中，白色斜线区域为禁行区域，黄色横线区域为减速带。黄色减速带前后 5m，车辆行驶速度不超过 10km/h。

泊车过程的核心问题是使无人车识别出最优目标停车位，且快速安全到达；时间应尽可能短，前进车速不超过 20km/h，倒车车速不超过 10km/h，在减速带前后 5m 范围内车速不超过 10km/h，轨迹和速度都尽可能平滑，假设控制点为位于后轴中心上，解决以下四个问题。

问题 1，根据给定参数计算车辆最小转弯半径，并计算车辆沿直线行驶，到达最大加加速度 $20m/s^3$ 的最短时间，及探讨当车速 20km/h 时，从直线行驶进入转弯的路径上的曲率相对路径长度的变化率大小有何限制。问题 2，以无人车初始位置为车库入口，建立无人车泊车的数学模型，给出从初始位置到指定停车位的泊车轨迹，画出可视化轨迹图。其中，对红色禁停的车位不能发生冲撞情况。问题 3，根据所指示的初始位置及停车位状况，建立泊车模型，计算出最优停车位，求活动轨迹；试建立通用模型，设计适应车库任意停车位被占用的算法，考虑算法的复杂性。问题 4，为无人车建立泊车模型，求出从指定状态下到最优停车位的行驶轨迹仿真结果。

二、 问题分析

2.1 问题一分析

问题一要求的是最小转弯半径。最小转弯半径的影响因素主要有汽车特性、行驶速度、道路特性、稳定性等。无人汽车由于在泊车过程中转弯行驶轨迹简单，且行驶速度较为平缓，所以其最小转弯半径主要与自身参数相关。建立坐标系，通过初始位置的坐标与方向，及可求的无人车方向最小转角、无人车自身参数可以计算出最小转弯半径。针对无人车的运动状态，我们又提出无人车的直线运动模型以及转弯模型。

当车辆最大加加速度为 $20m/s^3$ 时，则对其求积分，得出汽车速度随时间变化的和变化的关系，然后对不同情况进行分析讨论，求得在不同情形下的最短路径；当车速为 20km/h 时，按照曲率计算公式即可求出曲率。

2.2 问题二分析

从车辆初始位置到不同泊车位之间的路程进行分段处理，使得无人车驶向不同位置的行驶时间最短。对于只需要直线行驶的无人车，受路段自身速度的约束以及无人车自身性能参数的约束；对转弯车辆而言，需要在此基础上加入路况影响的约束条件。当无人车转弯半径过大时，车辆外边缘极易受到碰撞；当转弯半径过小时，车辆内侧也容易遭受磕碰。

由此可见，如何选择车辆的驶入位置和转弯半径是本题的核心。以无人车转弯的圆心为坐标原点，利用问题一所建立的模型分别建立在不同情况下无人车行驶路径的极坐标方程，与道路障碍物相对比，即可求出各弯道的最优行驶轨迹。

2.3 问题三分析

问题三要计算出最优停车位，则要得到泊车过程中所需要时间即明确在整个行驶过程中需要经过的路段和路程所对应的时间。同时要注意汽车在倒车过程中的转弯角度以此来确定在不同倒车情况下的倒车时长。通过系统的将无人车在直线、弯道及倒库各环节的时间汇总，以此为目标函数，在已知的空余车位上寻求最优解即可得到最优停车方案。

2.4 问题四分析

问题四要对一小时内 30 辆无人车最优的行驶路径进行仿真，首先得明确未被占用的车位集合，根据问题三的模型得到当前车辆在进入停车最优车位的选择。

三、 模型假设

假设 1：无人车从开始行驶至停到有效车位的过程中没有其他车辆行驶。

假设 2：停车场路面较为理想没有明显的起伏和障碍物。

假设 3：无人车自身整体理想没有突发性故障。

假设 4：无人车与路面、空气间的摩擦阻力忽略不计。

四、 符号说明

表 1　本文所用的主要符号以及其描述和单位

符号	描述	符号	描述
$a(t)$	无人车任意时间内的加速度	a_0	无人车初始的加速度
$G(t)$	无人车加加速度	t	无人车行驶时间
$v(t)$	无人车任意时间内的速度	$S(t)$	无人车任意时间内的行驶路程
R_{min}	最小转弯半径	L	无人车车长
θ_{max}	前轮最大转角	$\triangle\theta$	角度改变量
S	无人车转弯过程中最大行驶路程	δ	无人车转动最大角度
ψ	航向角	v	车速
δ_f	等效前轮转角	L_z	无人车轴距
W	无人车车宽	θ	问题二初始位置航向角
S_1	泊车起始点	S_2	第一次停车位置
V	无人车车速	b	x 轴方向的最大值
φ	无人车倒车入库时车头朝向角度	θ	问题三无人车车头朝向角度

五、 模型建立与求解

5.1 问题一的模型建立

根据题意以及对车辆相关知识的了解，针对无人车的运动方式可以将无人车的运动方式分为两个种类。

1. 一是直线行驶；
2. 二是弯道行驶。

接下来分别对其两种不同的运动模型进行模型的建立。

5.1.1 无人车直线运动模型

在物理学上常常将可以忽略运动物体形状的物体运动将物体看成一个质点，根据题意可以建立一个无人车在沿着一条直线方向的运动方程，可以假设无人车在运动时与地

面的摩擦、滑动、空气阻力等都忽略掉，则可以得出无人车在直线上行驶的路程与时间的关系，根据题目的描述以及查阅资料，假设汽车的初始速度为 V_0，初始的加速度为 a_0，加加速度为 $G(t)$，则在任意的时间内的加速度 $a(t)$ 为

$$a(t) = a_0 + \int_0^t G(t)dt \tag{1}$$

无人小车在任意时间的速度 $v(t)$ 为

$$v(t) = v_0 + \int_0^t a(t)dt \tag{2}$$

无人小车在任意时间的行驶的路程 $S(t)$ 为

$$S(t) = S_0 + \int_0^t v(t)dt \tag{3}$$

5.1.2 无人车转弯模型

研究车辆转动最小半径可以采用阿克曼转向几何原理，即当车辆转弯时，每一个车轮绕着同一个中心点转动，前后轮之间没有相对转角，保证轮胎与地面间光滑且处于摩擦力最小的纯滚动状态，假设即当车辆前轮和后轮之间的内轮差变化不大时，汽车低速转弯时，离心力、侧抗力、侧滑角均可可忽略不计，即认为车辆前后轮的运动为无侧滑的旋转（转向）运动。

无人车行驶轨迹较为简单，其最小转弯半径计算模型也相对简单，最小转弯半径 R_{min} 主要与车长 L, 前车轮最大转角 θ_{max} 有关，针对最大转角 θ_{max} 我们可以看成转弯在一瞬间完成，通过查阅资料可得到该转弯半径计算公式为

$$R_{min} = \frac{L}{2Sin\theta_{max}} \tag{4}$$

而对于从无人车沿直线行驶状态开始转弯，路径上的曲率相对路径长度的变化率大小可以由图 1

假设在转弯的时候，速度恒定及 $V(t) \equiv 20km/h$ 由上所知，最小转弯半径为 R_{min}, 设角度改变量为 $\triangle\theta$, 由图 1 中的几何关系可知

$$\begin{cases} \varphi = \frac{S}{\triangle\theta} \\ \triangle\theta = \angle AFE - \angle ABD \\ \angle ABD = \varphi + \delta \end{cases} \tag{5}$$

有曲率公式可知:

$$k = \left| \frac{\triangle\theta}{S} \right| \tag{6}$$

5.2 问题一模型求解

根据题目意思可以对无人车的直线运动进行约束,并假设在 t_1 时刻速度达到 $\frac{200}{36}m/s$,并联立 (1) 和 (2) 公式得到下列方程组为

$$\begin{cases} v(t) = v_0 + \int_t^0 [a_0 + \int_t^0 G(t)dt]dt \\ \text{s.t. } G(t) \leq 20m/s^3 \\ v(t_1) = \frac{200}{36}m/s \end{cases} \tag{7}$$

对初始条件进行分类讨论，若 $a_0 = 0$，$v_0 = 0$ 的情况下，$G(t)$ 取最大值时距离最短，就可以得出方程 $G(t) = 20m/s^3$，根据公式 (1) 可以推出关系式

$$a(t) = 20t \tag{8}$$

根据上述关系式以及公式 (2) 接着可以推出

$$v(t) = 10t \tag{9}$$

根据上述关系式以及公式 (3) 接着可以推出

$$S(t) = \frac{10}{3}t^2 \tag{10}$$

根据初始条件 $v(t_1) = 10t^2 = \frac{200}{36}$ 可以求出 $t_1 = 0.745356s$，将 $t_1 = 0.745356s$ 带入公式 (8) 中可到 $S(t_1) = 1.85185m$

我们已知车长 $L = 4.9m$，方向盘最大转角 $470°$，方向盘与前轮转角的传动比为 16:1，则前轮最大偏转角 $\theta_{max} = \frac{470°}{16}$ 其带入最小转弯半径公式 (4) 中，可得:

$$R_{min} = \frac{L}{2Sin\theta_{max}} = 4.9946m \tag{11}$$

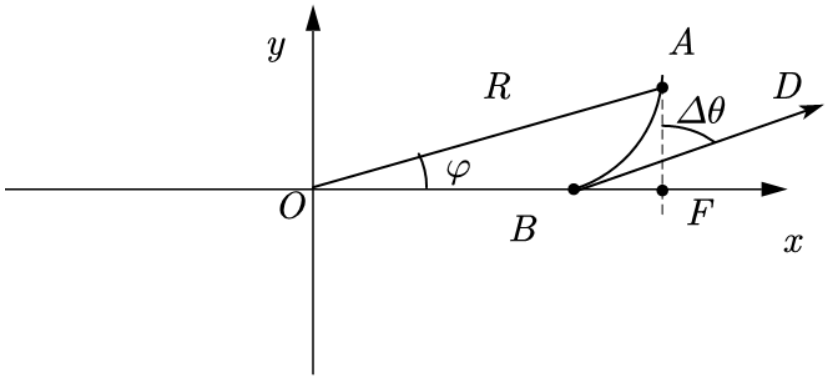


图 1　无人车转弯平面几何图

由图一可看出 $\angle AFE = 90°$ 所以由公式 (5) 可知:$\triangle\theta = \frac{\pi}{2} - \varphi - \theta$，将公式 (5) 化简得:

$$k = \left| \frac{\frac{\pi}{2} - \varphi - \delta}{S} \right| \tag{12}$$

5.3 问题二的模型建立

表 2　无人车主要参数

符号	描述
无人车车长 L/m	4.9m
无人车车宽 W/m	1.8m
无人车轴距 L_z/m	2.8m
无人车转动最大角度 δ_{max}/m	29.375°

通过题意我们可知无人车在泊车时先达到泊车地点附近再通过泊车的模型进行泊车，所以大致可以将整个运动过程分为两个阶段，下面先介绍到达指定泊车位置的模型

1. 从起始位置进行拐弯进入行驶区域;
2. 通过行驶区域到达停车点;
3. 进行停车。

首先我们先来建立无人车运动学模型

5.3.1 无人车运动学模型

分析无人车的运动状态取无人车的控制点为参考点。在考虑低速泊车的运动状态下，根据阿克曼转向原理，将无人车视为平面刚体，则可将无人车简化为只具有 x 方向、y 方向和横摆 3 个自由度。无人车在坐标系的坐标为 (x, y), 航向角为 ψ，无人车的运动学方程为

$$\begin{cases} \dot{x} = v \cdot cos\psi \\ \dot{y} = v \cdot sin\psi \\ \dot{\psi} = v \cdot tan\frac{\delta_f}{L_z} \end{cases} \tag{13}$$

其中：v 为车速;δ_f 为等效前轮转角。

5.3.2 从起始点出发的弯道模型

如图 1 所示，其中 A_1 和 A_2 分别表示初始点时无人车的内外两侧，B_1 和 B_2 表示转弯后无人车的内外两侧，其中大矩形和小矩形将图像分成了可行驶区域和非行驶区域(小矩形 ODC_2F 内),O 点为无人车进行最小转弯的原点，则 $A_1O = R_{min}$，其中的无人车在直角转弯过程的几何关系表示为

$$\begin{cases} OA_1 - A_1A_2 - A_2D = DO \\ R = \frac{L}{2Sin\theta}\theta_{max} = 29.375° \end{cases} \tag{14}$$

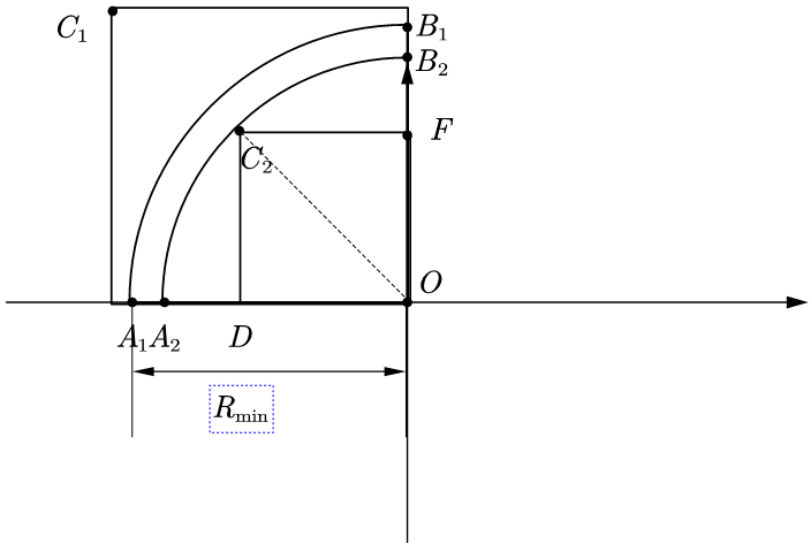


图 2　垂直转弯平面几何示意图

由图 1 可知无人车在直角转弯时存在以下关系

$$\begin{cases} DA_2 = R - W - OD \\ OC_2 = \sqrt{2}OD \\ OC_2 = R - W \end{cases} \tag{15}$$

接下来我们令 $\mu = DA_2$，联立上述方程组可以等到

$$R - (R - W - \mu)\sqrt{2} = W \tag{16}$$

经过简单的计算可以等到 $\mu = (1 - \frac{\sqrt{2}}{2})(R - W)$

5.3.3 无人车停车模型

由于第一问已经建立了相关模型，所以本模型是在问题一中建立的无人车直线的基础上的扩展，设 $H(x)$ 是小车到

下面开始介绍无人车的泊车阶段，根据停车位的不同，我们又将无人车的泊车问题又分为三种泊车策略：

1. 类似于图中 10 号泊车位的垂直泊车策略
2. 类似于图中 82 号泊车位的平行泊车策略
3. 类似于图中 31 号泊车位的倾斜停车位 (倾斜角为 45°) 泊车策略

假设汽车垂直泊车时初始位置航向角为 θ. 以汽车前进方向为 X 轴，自动泊车系统开始工作时刻汽车后轴中心为原点 O，原点左侧垂直方向为 Y 轴建立坐标系如图 3 所示，设垂直车位的两个顶点坐标为 (x_a, y_a) 和 (x_b, y_b) 可得到汽车相对于车位的初始位置航向角 θ 为:

$$\theta = \arctan \frac{y_b - y_a}{x_b - x_a} \tag{17}$$

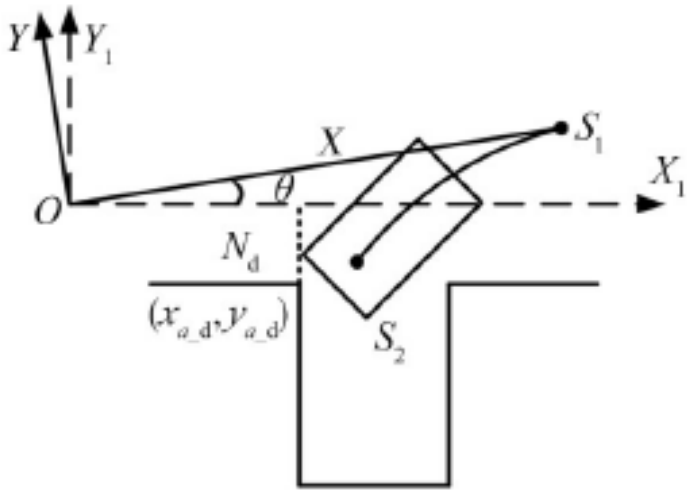


图 3　垂直泊车入库平面几何图

如图 3 定义初始位置航向角度向上位正，向下位负。

设泊车起始点为 S_1。无人车从 S_1 开始以最小转弯半径 R_{min} 进行倒车入库，考虑到垂直车位的右顶点 (x_b, y_b) 的约束，无人车在 S_2 点进行第一次停车，设行驶过的圆弧路径为 $\overset{\frown}{S_1S_2}$

根据无人车运动几何关系可知，车身外侧 M 点在倒车过程中与垂直车位中的右顶点 (x_b, y_b) 距离最近。我们假设两者之间最近的安全距离为 $L_{S1} = 0.1m$，可知 $O_1(x_{O1}, y_{o1})$ 的坐标为

$$\begin{cases} x_{o1} = x_b + \sqrt{(R - \frac{W}{2} - L_{s1})^2 - (R + y_b)^2} \\ y_{o1} = -R_{min} \end{cases} \tag{18}$$

从而求得起始点 S_1 的坐标为

$$\begin{cases} x_{s1} = x_{01} \\ y_{o1} = 0 \end{cases} \tag{19}$$

倘若无人车要只进行一次泊车便入库成功，则要求汽车沿规划路径 $\overset{\frown}{S_1S_2}$ 行驶过程中外侧顶点 N 不能与垂直车位左边线发生接触 (刮蹭)。为方便计算，我们将此时整个坐标系相对于车位顺时针转动 θ 角，旋转后坐标系为 OX_1Y_1，如图 4 所示。对垂直车位左顶点和无人车左后顶点 N 进行坐标变化后的坐标为 (x_{a_d}, y_{a_d}) 点和 N_d 点。同理，无人车进行泊车时候的起始点 S_1 和垂直车位右顶点也要做相应的坐标变换。

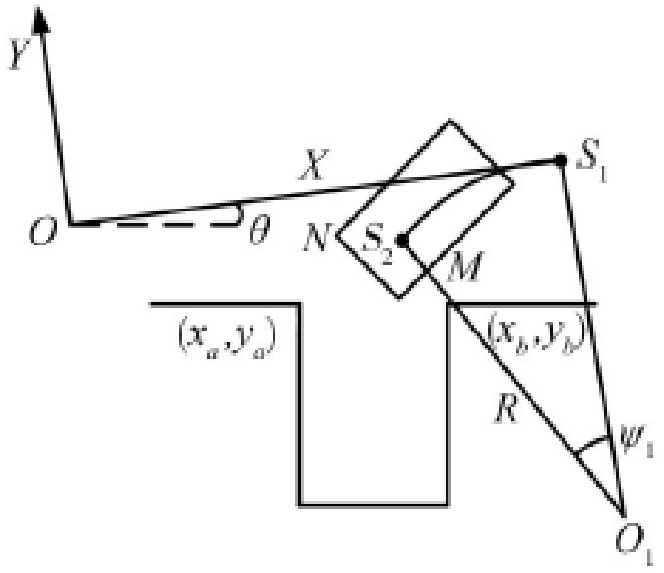


图 4　变化坐标系后泊车几何图

由图 4 可以看出，无人车在规划路径 $\overset{\frown}{S_1S_2}$ 行驶过程中，若无人车外侧顶点 N_d 满足下列约束条件，则无人车内侧后顶点不会与垂直车位的左侧车位线发生接触:

$$x_{N_d} \geq x_{a_d} + L_{s2} \tag{20}$$

中的 $L_{s2} = 0.1m$，为设置的 S_2 点的安全阀值。为了使得最终泊车较为规范，应使得无人车泊车最后的航向角在 OX_1Y_1 坐标系下接近 90°, 若能满足则可以实现一步到位泊车。

针对 81 号泊车位，这种平行泊车位的无人车路径规划，我们通过 [5] 等人的研究，可以设立出一下模型，以 81 号泊车位的四个顶点分别取 a、b、c、d，其中取 d 点为坐标原点。为降低碰撞风险，在车辆四周取长度 Ls 为安全距离。所以在终点 O_4 处 $S_4 = (X_4, Y_4, \theta_4) = (L_s + L_r, \frac{w}{2}, 0)$。

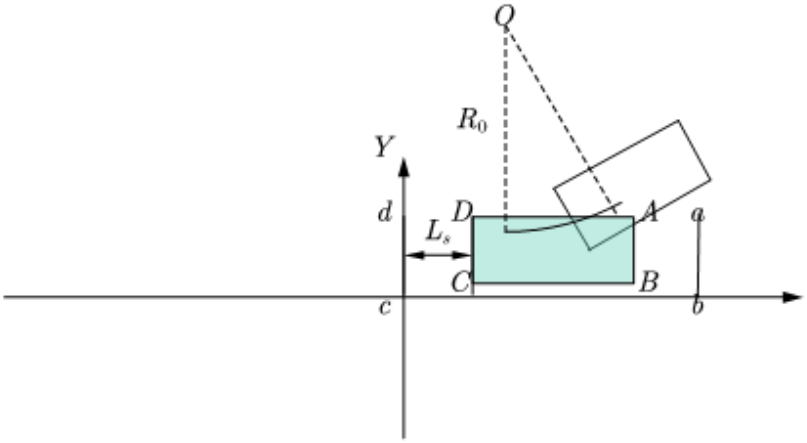


图 5　平行泊车平面几何图

按照“镜像”的方法，车辆左转向前进到平行泊车位，当车身的 B 点与车位 a d 点在同一条直线上时，无人车的转向盘恢复原位，车辆将沿着直线继续往前继续行驶到达与泊车位 a 点的最小距离为 L_s. 若求出 B 点的坐标为 (X_B, Y_B)，则平行泊车的最小长度为:

$$L_{P_{min}} = X_B + L_s \tag{21}$$

设虚圆路径转过的角度为 ϕ ，则 O_3 处车辆的横、纵坐标及车身航向角分别为:

$$\begin{cases} X_3 = X_{C10} + R_0 \sin(\phi - \psi) \\ Y_3 = Y_{C10} - R_0 \cos(\phi - \psi) \\ \theta_3 = \phi - 2\psi \end{cases} \tag{22}$$

根据车辆在 O_3 点处的坐标及无人车航向角可以得出 B 点的坐标为

$$\begin{cases} X_B = X_3 + R_{OB}\sin(\theta_3 - \delta) \\ Y_B = Y_3 - L_{OB}(\phi - \delta) = 0 \end{cases} \tag{23}$$

式子中：$L_{OB} = \sqrt{(L - L_r)^2 + (\frac{w}{2})^2}$ 为车身 B 点与后轴中心点的距离；$\delta = \arctan\frac{0.5W}{L - L_r}$ 为直线 O_3B 与车身中轴线之间的夹角。

联立上述式子可解出 $L_{P_{min}}$ 以及无人车在 O_3 点处的位置 $S_3 = (X_3, Y_3, \theta_3)$

5.4 问题二的模型求解

对上述模型进行求解当无人车倒车入库进行详细分析，在倒车的过程中，无人车在倒车入库之前的初速度可以看成 $v = 20km/h$，倒车入库的临界状态是无人车恰好倒车时速度为 0，所以无人车应该先做减速运动，再做加速圆周运动，再做减速直线运动使用上述模型可以得出各段路程所用的时间为

$$t = \sum t_n \tag{24}$$

我们通过程序将上述模型通过代码的形式表现出来了，首先取地图的左下角作为整个坐标系的原点，方向向右为 x 轴的正方向，方向向上为 y 轴的正方向，以车后轴的中心点即控制点作为 (x，y) 的坐标点，每时刻无人车的行驶路径长度、车辆朝向、速度、加速度、加加速度、角速度以及角加速度的相关数据使用 *python* 语言进行编写针对 82 号以及 10 号泊车位，我们使用 *matlab* 语言编写代码实现了，无人车的倒车入库的相关动画，其中附录 A 实现了无人车的垂直停车，附录 B 实现了无人车的水平停车.

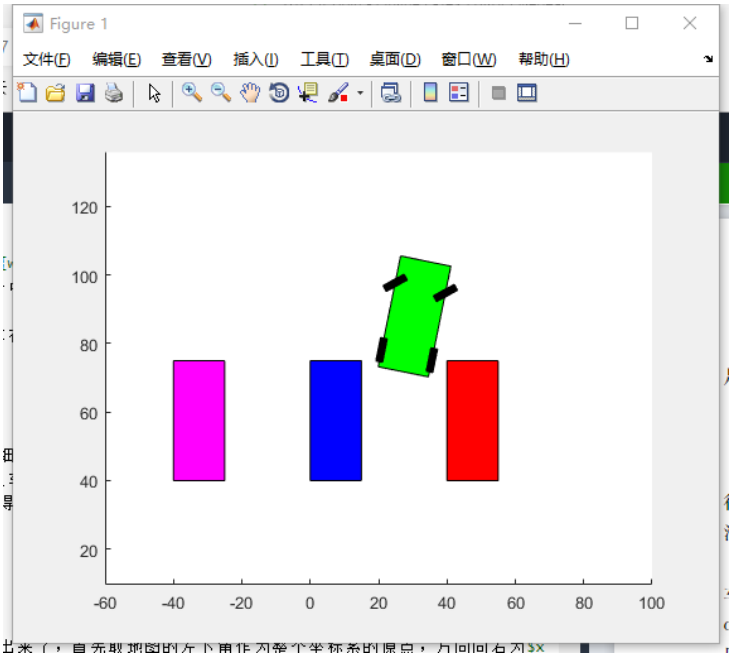


图 6　matlab 无人车垂直停车平面图

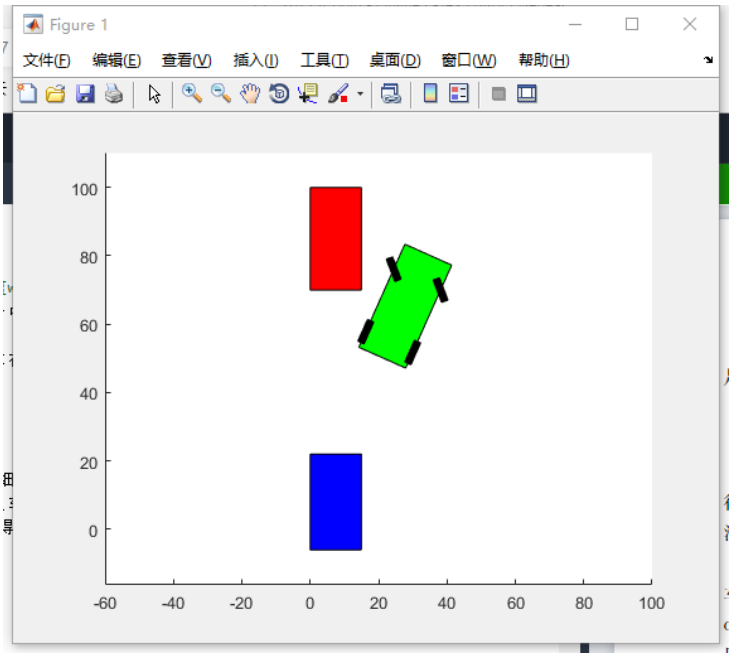
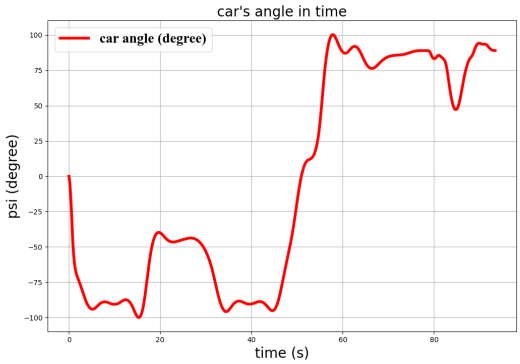
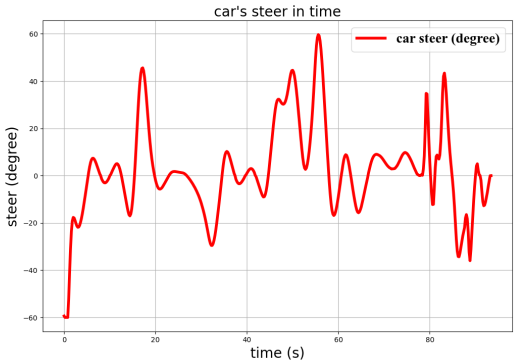


图 7　matlab 无人车水平停车平面图





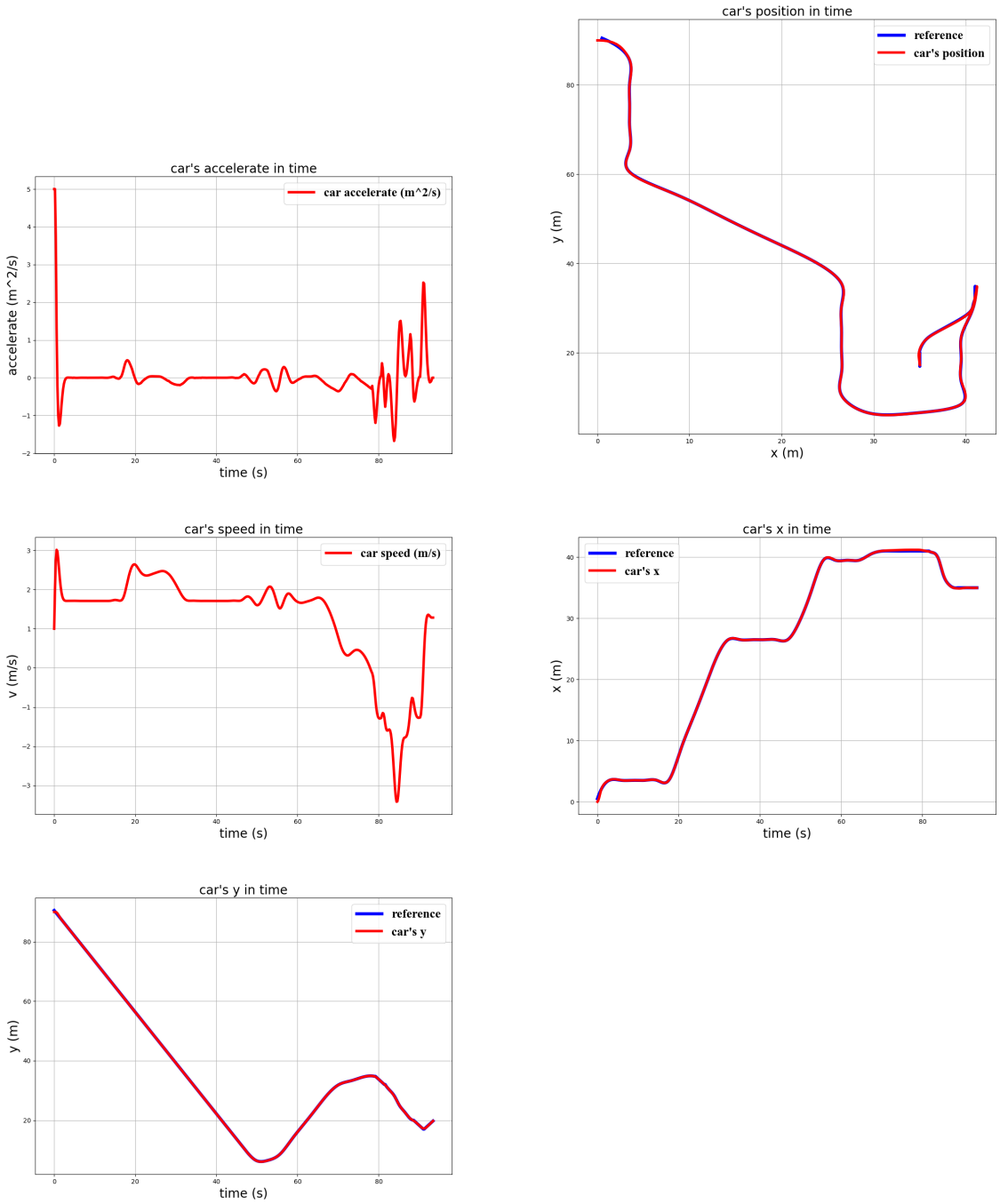


图 8　无人车各方面数据图

综上，我们对这些信息进行处理，针对上述求解模型进行处理，得到附件“支撑材料”中的三张动图。

5.5 问题三的模型建立

表 3　无人车主要参数

符号	描述
$V_{车}$	无人车车速
θ	无人车车头朝向角度
φ	无人车倒车入库时车头朝向角度
S	无人车行驶路程
b	为 x 轴方向的最大值

由问题分析并结合前两问的求解，为保证时间最短，我们令无人车在停泊过程中，无论是直线行驶、弯道行驶，还是倒车入库过程中车速 $V_{车}$ 为该路段限速的最高值。

我们对其可用的 n 个停车位的坐标进行假设, 分别为 $(x_i, y_i, \theta_i, \varphi_i), i = 1 \ldots n$, 其中 x_i 为 x 方向上的坐标，y_i 为 y 方向上的坐标，θ_i 为无人车在倒车入库之前一刻的车头指向方向，φ_i 为无人车在倒车入库过程中需要进行旋转角度。

我们先对弯道进行处理。

由题目所给的停车位平面图可知，对于弯道个数设为 W_i, 车头朝向角度的变化量为 $\triangle\theta$

$$W_i = \begin{cases} 0, & \triangle\theta = 0; \\ 1, & \triangle\theta = \frac{\pi}{2}; \\ 2, & \triangle\theta = \pi. \end{cases}$$

弯道路径总和为 $S_{1i} = 2R\pi \triangle \theta$, 设该路段的最高速度为 V_1, 所用时间 $t_{1i} = \frac{S_{1i}}{v_1}$

接下来我们进行直线行驶过程的模型建立，我们令直线行驶过程中起点到终点的坐标变化为 $\triangle x = x - x_i$,$\triangle y = y - y_i$, 则直线行驶的路程为

$$S_i = \begin{cases} |\triangle x|, & \triangle\theta = 0; \\ \triangle x_i + \triangle y_i, & \triangle\theta = \frac{\pi}{2}; \\ 2b - (x + x_i), & \triangle\theta = \pi. \end{cases}$$

通过上述 S_i 分别对其所需要的时间进行求解，令在减速带前 $5m$ 位置以内的速度为 V_2，其他直行道路的速度为 V_3.

当 $\triangle\theta = 0$, 带入上述公式求解的:

$$t_i = \begin{cases} \frac{b-x-5}{V_3} + \frac{b-x}{V_2} & b-x > 5, \triangle x > 0; \\ \frac{b-x}{V_2} & b-x \leq 5, \triangle x > 0; \\ \frac{-\triangle x}{V_2} & \triangle x < 0. \end{cases}$$

当 $\triangle\theta = \frac{\pi}{2}$ 时:

$$t_i = \begin{cases} \frac{b-x-5}{V_3} + \frac{b-x+\triangle y}{V_2} & b-x > 5, \triangle x > 0; \\ \frac{b-x+\triangle y}{V_2} & b-x \leq 5, \triangle x > 0; \end{cases}$$

当 $\triangle\theta = \pi$ 时:

$$t_i = \begin{cases} \frac{b-x-5}{V_3} + \frac{b-x+\triangle y_i}{V_2} + \frac{5}{V_2} + \frac{b-x_i-5}{V_3} & b-x_i > 5, b-x > 5; \\ \frac{b-x+\triangle y_i}{V_2} + \frac{b+x-5}{V_3} + \frac{b-x_i}{V_2} & b-x_i \leq 5, b-x > 5; \\ \frac{b+x+\triangle y_i}{V_2} + \frac{5}{V_2} + \frac{b+x-5}{V_3} & b-x_i \geq 5, b-x \leq 5; \\ \frac{b+x+\triangle y_i}{v_2} + \frac{5}{V_2} + \frac{b-x}{V_2} & b-x_i \leq 5, b-x \leq 5; \end{cases}$$

对于倒车路径 $S_{3i} = 2\pi R\varphi_i$

$$t_i = \begin{cases} \frac{S_{3i}}{V_2} & ; \\ \frac{S_{3i}}{V_2} & ; \end{cases}$$

综上所述令无人车对 n 个停车位的绝策系数记为 k_i 则总时间为 $t = \sum_{i=1}^{n} k_i(t_{1i} + t_{2i} + t_{3i})$ 则对于问题三的求解等价于对下列带约束的目标函数的优化问题:

5.6 问题三的模型求解

5.7 问题四的模型建立

问题四的模型是问题三模型的扩展, 在问题三建立的模型的基础再多考虑多辆车的同时停车问题, 所以可以得到以下流程图, 根据流程图就可以对 30 辆车同时停车问题进行求解, 则可以得到一个经过优化的解。

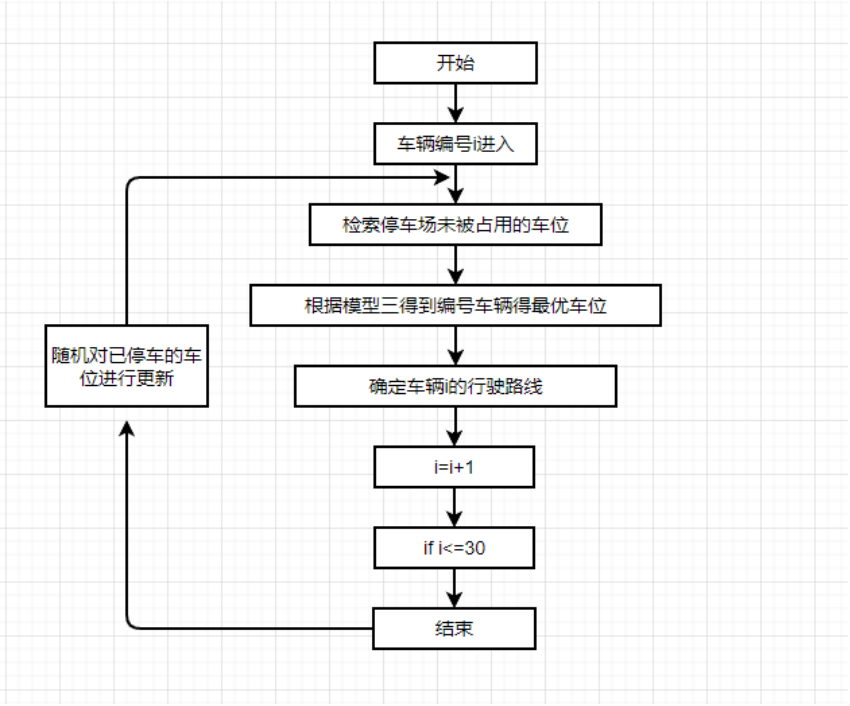


图 9　无人车转弯平面几何图

5.7.1 多无人车路径规划

多无人车问题定义为有 K 个无人车组成的群体，每辆无人车有各自的起点和终点。系统需为各无人车规划处一条从起点到终点的路径，且无人车之间的路径不能发生冲突。在无冲突基础上，令无人车群体路径的工时耗费总和最少是本问题的优化目标。

5.7.2 调度次序优化及带熔断机制的改进 HCA* 算法

HCA* 算法由协作 A*(Cooperative A*, CA*) 和反向回溯 A*(Reverse Resumable A*,RRA*) 算法组合而成。其中，CA* 在求解思路上将 MAPF 问题从一个整体问题分解为了有前后联系的单无人车路径规划问题，按顺序依次对无人车 $a_1, a_2, ..., a_k$ 做无人车的路径规划，并把前面规划的无人车路径信息作为障碍约束添加到后面的无人车规划当中。即在 A* 搜索过程中额外设立了一个预约表，当完成第 i 个无人车 a_i 的路径规划后，便将该无人车包含离散时间点的路径信息 p_i 放入该预约表并与前面已有的列表项共同组成下一个无人车 a_{i+1} 的带时限障碍物，依次对每个无人车执行上述操作，直至完成全部无人车的路径规划方案。在 CA* 基础上，Silver 对 A* 搜索过程中的启发函数做了进一步改进，在 CA* 算法下增设了启发函数计算层并使用 RRA* 算法对目标点到当前点进行反向 A* 搜索以获得当前点到目标点的实际距离，再将该距离作为启发函数，以进一步提高算法的搜索质量。

G-JHCA* 算法流程如下：

Algorithm 1 GJHCA* 算法

initialize:PopPrio(1)=Prio(1),Prio(2),...,Prio(n)
calculate fitness:F(1)=f(Prio(1)),f(Prio(2)),...,f(Prio(n))
while C(PopPrio(i))≠Ture do
for all prio(j) ∈ PopPrio(i) do
reservation table T= ⊘,paths= ⊘
for each agent α_k according to prio(j) do
path p_k=⊘
close$_r ra$⟵ rra*,N_r=length(close$_r ra$ *whileopen*≠ ⊘ and $N_c \leq \omega N_r$ do
P ⟵ pop(open) by f value
close ⟵ P
if P is goal node then
return P_k by trace close
end if
expend next node Q by P
open ⟵ Q if Q not conflict with T and not in close
update closerra
N_c = length(close), N_r=length(closerra)
end while
return P_k = ⊘
add p_k to paths
add p_k to T
end for
calculate fitness f(Prio(j) to F(i)
end for
select:$PopPrio'(i)$ = s(PopPrio(i) , F(i) , p_s, p_v)
crossover $PopPrio''(i)$ = c($PopPrio'(i)$, p_c)
mutate: $PopPrio'''(i)$ = m($PopPrio''(i)$, p_m)
generate new population : PopPRrio(i+1) = $PopPrio'''(i)$; i=i+1
end while

5.8 问题四的模型求解

六、 模型评价与推广

6.1 模型优点

1. 问题一的模型先确认了最小转弯半径由哪些因素影响，然后根据不同情况进行分类讨论并且分别建立模型，最后得出结果。在模型的分析与建立上条理清晰、思考严谨，排除掉影响不大的因素，找到可能影响结果的关键部分并合理建模，得到的结果可信度较高。
2. 问题二的模型在建立前考虑到两个方面的问题，首先是从初始位置到不同泊车点路径的不同，然后是进入泊车位的方式不同。通过对此问题的思考，在模型一的基础上，改进并建立无人车行驶到不同泊车位以及以不同方式及进入泊车位的模型。画出无人车进入停车位的几何示意图以及建立极坐标并运用极坐标方程，进行严格的分析与计算，讨论出无人车最合理进入车位的方案，使整个模型的框架清晰、可行性较高，减少了不必要的计算量和代码量。
3. 问题三的模型在模型一二的基础上分成了三部分建立，分别是无人车在直线、弯道以及倒库时所用的时间，然后将它们汇总。根据模型一和二的求解带入各路段的限速最高值，考虑了宏观方面的最好情况，使模型更加准确。
4. 问题四的模型是在问题三的模型的基础上建立的，整体不错，HCA* 算法可以快速处理多辆无人车在路径规划上的问题

6.2 模型改进

1. 我们思考可以在源模型的基础上更多的去考虑一些关于速度过快的无人车状态因素，在外界因素的考虑中再多下点功夫，就能保证模型在任意情况下都是适用
2. 在第二问的模型中，我们的模型在考虑倒车入库的时候忽略了多次入库这一重要路径规划，事实上，多次入库可以在路程上比一次入库节省不少路程。
3. 我们由于忽略了其他物理量，将速度定义成了路程规划的决策变量，事实上路程规划并不是速度就能决定的，还要考虑无人车在时间以及路程上的变化，才能最终使得无人车在路径规划上实现最优。
4. 我们的多无人车模型还是过于简单，如果数量级再上升几位的话，将会使得算法无法尽快做出抉择。

参考文献

[1] 吕鹏磊, 王子蕊, 刘利鹏. 大件运输车辆最小转弯半径影响因素及计算模型分析 [J]. 石家庄铁路职业技术学院学报,2021,20(02):79-84.

[2] 陈亚琳, 庄丽阳, 朱龙彪, 邵小江, 王恒. 基于改进 Dijkstra 算法的泊车系统路径规划研究 [J]. 现代制造工程,2017(08):63-67.DOI:10.16731/j.cnki.1671-3133.2017.08.012.

[3]Mitchell, W., Staniforth, A., and Scott, I., "Analysis of Ackermann Steering Geometry," SAE Technical Paper 2006-01-3638, 2006.

[4]Tseng DC, Chao TW, Chang JW. Image-Based Parking Guiding Using Ackermann Steering Geometry. AMM 2013;437:823–6.

[5] 胡勤明, 王金刚, 张小俊. 五次多项式优化的平行泊车路径规划 [J/OL]. 计算机工程与应用:1-9[2022-04-18].http://kns.cnki.net/kcms/detail/11.2127.TP.20210325.1007.008.html

[6] 高强, 陆洲, 段晨东, 徐婷. 汽车垂直泊车路径规划与路径跟踪研究 [J]. 汽车工程,2021,43(07):987-994+1012.DOI:10.19562/j.chinasae.qcgc.2021.07.005.

[7] 百度百科，最小转弯半径 [J]. 百度学术，https://baike.baidu.com/item/

[8] 百度百科，曲率 [J]. 百度学术，https://baike.baidu.com/item/

附录 A　第二问中实现垂直停车的 matlab 代码

```
clc
close all
clearvars
global wl;
global ww;
global D;
global L;
global WSeta;
wl=7;

ww=2;
    A=15;
    L=20;
    D=5;
    F=8;
    B=L+D+F;
    Parked_Car1x1=0;
    Parked_Car1y1=40;
    Parked_Car1x2=Parked_Car1x1;
    Parked_Car1y2=75;
    Parked_Car1x3=15;
    Parked_Car1y3=Parked_Car1y2;
    Parked_Car1x4=Parked_Car1x3;
    Parked_Car1y4=Parked_Car1y1;
    Parked_Car2x1=40;
    Parked_Car2y1=40;
    Parked_Car2x2=Parked_Car2x1;
    Parked_Car2y2=75;
    Parked_Car2x3=55;
    Parked_Car2y3=Parked_Car2y2;
    Parked_Car2x4=Parked_Car2x3;
    Parked_Car2y4=Parked_Car2y1;
    Parked_Car3x1=-40;
    Parked_Car3y1=40;
    Parked_Car3x2=Parked_Car3x1;
```

```
Parked_Car3y2=75;
Parked_Car3x3=-25;
Parked_Car3y3=Parked_Car3y2;
Parked_Car3x4=Parked_Car3x3;
Parked_Car3y4=Parked_Car3y1;
WSeta=50;
WSeta=(WSeta/180)*pi; % degree to radians
xi=Parked_Car2x2-3;
yi=Parked_Car2y2+25;
f1e=40;
axis([-60 100 -20 100],'equal');
t=0:0.01:pi;
seta=2*t;
Xc=xi-(L/tan(WSeta));
Yc=yi+D;
Rbl=sqrt(D^2+(L/tan(WSeta))^2);
phi=atan(D/(L/tan(WSeta)));
  i=1;
x=Xc+Rbl*cos(seta(i)+phi);
y=Yc-Rbl*sin(seta(i)+phi);
patch([Parked_Car2x1 Parked_Car2x2 Parked_Car2x3
    Parked_Car2x4],[Parked_Car2y1 Parked_Car2y2 Parked_Car2y3
    Parked_Car2y4],[1 0 0]);
patch([Parked_Car1x1 Parked_Car1x2 Parked_Car1x3
    Parked_Car1x4],[Parked_Car1y1 Parked_Car1y2 Parked_Car1y3
    Parked_Car1y4],[0 0 1]);
patch([Parked_Car3x1 Parked_Car3x2 Parked_Car3x3
    Parked_Car3x4],[Parked_Car3y1 Parked_Car3y2 Parked_Car3y3
    Parked_Car3y4],[1 0 1]);
Turn1(x,y,A,B,seta(i));
pause(4);
   for i=2:f1e
    x=Xc+Rbl*cos(seta(i)+phi);
    y=Yc-Rbl*sin(seta(i)+phi);
    cla;
    patch([Parked_Car2x1 Parked_Car2x2 Parked_Car2x3
        Parked_Car2x4],[Parked_Car2y1 Parked_Car2y2 Parked_Car2y3
```

```
        Parked_Car2y4],[1 0 0]);
    patch([Parked_Car1x1 Parked_Car1x2 Parked_Car1x3
        Parked_Car1x4],[Parked_Car1y1 Parked_Car1y2 Parked_Car1y3
        Parked_Car1y4],[0 0 1]);
    patch([Parked_Car3x1 Parked_Car3x2 Parked_Car3x3
        Parked_Car3x4],[Parked_Car3y1 Parked_Car3y2 Parked_Car3y3
        Parked_Car3y4],[1 0 1]);
    Turn1(x,y,A,B,seta(i));
    pause(.05);
end




xi=x+A*cos(seta(f1e));
yi=y-A*sin(seta(f1e));
Xc=xi+Rbl*cos(seta(f1e));
Yc=yi-Rbl*sin(seta(f1e));
pause(1);
for i=f1e:-1:1
        x=Xc-Rbl*cos(seta(i));
        y=Yc+Rbl*sin(seta(i));
        cla;
        patch([Parked_Car2x1 Parked_Car2x2 Parked_Car2x3
            Parked_Car2x4],[Parked_Car2y1 Parked_Car2y2 Parked_Car2y3
            Parked_Car2y4],[1 0 0]);
        patch([Parked_Car1x1 Parked_Car1x2 Parked_Car1x3
            Parked_Car1x4],[Parked_Car1y1 Parked_Car1y2 Parked_Car1y3
            Parked_Car1y4],[0 0 1]);
        patch([Parked_Car3x1 Parked_Car3x2 Parked_Car3x3
            Parked_Car3x4],[Parked_Car3y1 Parked_Car3y2 Parked_Car3y3
            Parked_Car3y4],[1 0 1]);
        [xa1, xb1, xb2, xa2, ya1, yb1, yb2, ya2]=Turn2(x,y,A,B,seta(i));
        pause(.05);
end
pause(1);
for i=1:50
```

```
ya1=ya1-.5;
yb1=yb1-.5;
ya2=ya2-.5;
yb2=yb2-.5;
cla;
patch([Parked_Car2x1 Parked_Car2x2 Parked_Car2x3
    Parked_Car2x4],[Parked_Car2y1 Parked_Car2y2 Parked_Car2y3
    Parked_Car2y4],[1 0 0]);
patch([Parked_Car1x1 Parked_Car1x2 Parked_Car1x3
    Parked_Car1x4],[Parked_Car1y1 Parked_Car1y2 Parked_Car1y3
    Parked_Car1y4],[0 0 1]);
patch([Parked_Car3x1 Parked_Car3x2 Parked_Car3x3
    Parked_Car3x4],[Parked_Car3y1 Parked_Car3y2 Parked_Car3y3
    Parked_Car3y4],[1 0 1]);
patch([xa1 xb1 xb2 xa2],[ya1 yb1 yb2 ya2],[0 1 0]);
ya11=ya1+(D-wl/2);
yb11=ya11+wl;
ya21=ya2+(D-wl/2);
yb21=ya21+wl;
xa11=xa2+(ww/2);
xb11=xa2+(ww/2);
xa21=xa2-(ww/2);
xb21=xa2-(ww/2);
patch([xa11 xb11 xb21 xa21],[ya11 yb11 yb21 ya21],[0 0 0]);
xa22=xa1-(ww/2);
xb22=xa1-(ww/2);
xa12=xa1+(ww/2);
xb12=xa1+(ww/2);
patch([xa12 xb12 xb22 xa22],[ya11 yb11 yb21 ya21],[0 0 0]);
ya13=ya11+L;
yb13=yb11+L;
patch([xa11 xb11 xb21 xa21],[ya13 yb13 yb13 ya13],[0 0 0]);
patch([xa12 xb12 xb22 xa22],[ya13 yb13 yb13 ya13],[0 0 0]);

pause(.02);
end
```

附录 B　第二问中实现水平停车的 matlab 代码

```
clc
close all
clearvars
global wl;
global ww;
global D;
global L;
global WSeta;
wl=7;
ww=2;
    A=15;
    L=20;
    D=5;
    F=8;
    B=L+D+F;

    %%停放汽车 1 的矩形坐标

    Parked_Car1x1=0;
    Parked_Car1y1=-6;
    Parked_Car1x2=0;
    Parked_Car1y2=22;
    Parked_Car1x3=15;
    Parked_Car1y3=22;
    Parked_Car1x4=15;
    Parked_Car1y4=-6;
    Parked_Car2x1=0;
    Parked_Car2y1=70;
    Parked_Car2x2=0;
    Parked_Car2y2=100;
    Parked_Car2x3=15;
    Parked_Car2y3= 100;
    Parked_Car2x4=15;
    Parked_Car2y4=70;
    WSeta=45;
```

```
WSeta=(WSeta/180)*pi; % degree to radians
S=3;
xi=Parked_Car2x4+S;
yi=Parked_Car2y4-9;
f1e=38;
    axis([-60 100 -20 100],'equal');
t=0:0.01:pi;
seta=2*t;
Xc=xi-(L/tan(WSeta));
Yc=yi+D;
Rbl=sqrt(D^2+(L/tan(WSeta))^2);
phi=atan(D/(L/tan(WSeta)));
          i=1;
       x=Xc+Rbl*cos(seta(i)+phi);
       y=Yc-Rbl*sin(seta(i)+phi);
       patch([Parked_Car2x1 Parked_Car2x2 Parked_Car2x3
           Parked_Car2x4],[Parked_Car2y1 Parked_Car2y2 Parked_Car2y3
           Parked_Car2y4],[1 0 0]);
       patch([Parked_Car1x1 Parked_Car1x2 Parked_Car1x3
           Parked_Car1x4],[Parked_Car1y1 Parked_Car1y2 Parked_Car1y3
           Parked_Car1y4],[0 0 1]);
       Turn1(x,y,A,B,seta(i));
       pause(4);
         for i=2:f1e
       x=Xc+Rbl*cos(seta(i)+phi);
       y=Yc-Rbl*sin(seta(i)+phi);
       cla;
       patch([Parked_Car2x1 Parked_Car2x2 Parked_Car2x3
           Parked_Car2x4],[Parked_Car2y1 Parked_Car2y2 Parked_Car2y3
           Parked_Car2y4],[1 0 0]);
       patch([Parked_Car1x1 Parked_Car1x2 Parked_Car1x3
           Parked_Car1x4],[Parked_Car1y1 Parked_Car1y2 Parked_Car1y3
           Parked_Car1y4],[0 0 1]);
       Turn1(x,y,A,B,seta(i));
       pause(.05);
end
```

```
    xi=x+A*cos(seta(f1e));
    yi=y-A*sin(seta(f1e));
   Xc=xi+Rbl*cos(seta(f1e));
   Yc=yi-Rbl*sin(seta(f1e));
   pause(1);



   for i=f1e:-1:1
          x=Xc-Rbl*cos(seta(i));
          y=Yc+Rbl*sin(seta(i));
          cla;
          patch([Parked_Car2x1 Parked_Car2x2 Parked_Car2x3
              Parked_Car2x4],[Parked_Car2y1 Parked_Car2y2 Parked_Car2y3
              Parked_Car2y4],[1 0 0]);
          patch([Parked_Car1x1 Parked_Car1x2 Parked_Car1x3
              Parked_Car1x4],[Parked_Car1y1 Parked_Car1y2 Parked_Car1y3
              Parked_Car1y4],[0 0 1]);
          [xa1, xb1, xb2, xa2, ya1, yb1, yb2, ya2]=Turn2(x,y,A,B,seta(i));
          pause(.05);
   end
pause(1);
for i=1:13
ya1=ya1+.5;
yb1=yb1+.5;
ya2=ya2+.5;
yb2=yb2+.5;
cla;
patch([Parked_Car2x1 Parked_Car2x2 Parked_Car2x3
    Parked_Car2x4],[Parked_Car2y1 Parked_Car2y2 Parked_Car2y3
    Parked_Car2y4],[1 0 0]);
patch([Parked_Car1x1 Parked_Car1x2 Parked_Car1x3
    Parked_Car1x4],[Parked_Car1y1 Parked_Car1y2 Parked_Car1y3
    Parked_Car1y4],[0 0 1]);
patch([xa1 xb1 xb2 xa2],[ya1 yb1 yb2 ya2],[0 1 0]);
ya11=ya1+(D-wl/2);
yb11=ya11+wl;
```

```
ya21=ya2+(D-wl/2);
yb21=ya21+wl;
xa11=xa2+(ww/2);
xb11=xa2+(ww/2);
xa21=xa2-(ww/2);
xb21=xa2-(ww/2);
patch([xa11 xb11 xb21 xa21],[ya11 yb11 yb21 ya21],[0 0 0]);
xa22=xa1-(ww/2);
xb22=xa1-(ww/2);
xa12=xa1+(ww/2);
xb12=xa1+(ww/2);
patch([xa12 xb12 xb22 xa22],[ya11 yb11 yb21 ya21],[0 0 0]);
ya13=ya11+L;
yb13=yb11+L;
patch([xa11 xb11 xb21 xa21],[ya13 yb13 yb13 ya13],[0 0 0]);
patch([xa12 xb12 xb22 xa22],[ya13 yb13 yb13 ya13],[0 0 0]);

pause(.02);
end
```

附录 C　第二问中实现水平停车的 matlab 代码

```
clc
close all
clearvars
global wl;
global ww;
global D;
global L;
global WSeta;
wl=7;
ww=2;
   A=15;
   L=20;
   D=5;
   F=8;
   B=L+D+F;
```

```
%%停放汽车 1 的矩形坐标

Parked_Car1x1=0;
Parked_Car1y1=-6;
Parked_Car1x2=0;
Parked_Car1y2=22;
Parked_Car1x3=15;
Parked_Car1y3=22;
Parked_Car1x4=15;
Parked_Car1y4=-6;
Parked_Car2x1=0;
Parked_Car2y1=70;
Parked_Car2x2=0;
Parked_Car2y2=100;
Parked_Car2x3=15;
Parked_Car2y3= 100;
Parked_Car2x4=15;
Parked_Car2y4=70;
WSeta=45;
WSeta=(WSeta/180)*pi; % degree to radians
S=3;
xi=Parked_Car2x4+S;
yi=Parked_Car2y4-9;
f1e=38;
    axis([-60 100 -20 100],'equal');
t=0:0.01:pi;
seta=2*t;
Xc=xi-(L/tan(WSeta));
Yc=yi+D;
Rbl=sqrt(D^2+(L/tan(WSeta))^2);
phi=atan(D/(L/tan(WSeta)));
          i=1;
       x=Xc+Rbl*cos(seta(i)+phi);
       y=Yc-Rbl*sin(seta(i)+phi);
       patch([Parked_Car2x1 Parked_Car2x2 Parked_Car2x3
          Parked_Car2x4],[Parked_Car2y1 Parked_Car2y2 Parked_Car2y3
```

```
            Parked_Car2y4],[1 0 0]);
        patch([Parked_Car1x1 Parked_Car1x2 Parked_Car1x3
            Parked_Car1x4],[Parked_Car1y1 Parked_Car1y2 Parked_Car1y3
            Parked_Car1y4],[0 0 1]);
        Turn1(x,y,A,B,seta(i));
        pause(4);
          for i=2:f1e
        x=Xc+Rbl*cos(seta(i)+phi);
        y=Yc-Rbl*sin(seta(i)+phi);
        cla;
        patch([Parked_Car2x1 Parked_Car2x2 Parked_Car2x3
            Parked_Car2x4],[Parked_Car2y1 Parked_Car2y2 Parked_Car2y3
            Parked_Car2y4],[1 0 0]);
        patch([Parked_Car1x1 Parked_Car1x2 Parked_Car1x3
            Parked_Car1x4],[Parked_Car1y1 Parked_Car1y2 Parked_Car1y3
            Parked_Car1y4],[0 0 1]);
        Turn1(x,y,A,B,seta(i));
        pause(.05);
end

 xi=x+A*cos(seta(f1e));
 yi=y-A*sin(seta(f1e));
Xc=xi+Rbl*cos(seta(f1e));
Yc=yi-Rbl*sin(seta(f1e));
pause(1);



for i=f1e:-1:1
        x=Xc-Rbl*cos(seta(i));
        y=Yc+Rbl*sin(seta(i));
        cla;
        patch([Parked_Car2x1 Parked_Car2x2 Parked_Car2x3
            Parked_Car2x4],[Parked_Car2y1 Parked_Car2y2 Parked_Car2y3
            Parked_Car2y4],[1 0 0]);
        patch([Parked_Car1x1 Parked_Car1x2 Parked_Car1x3
            Parked_Car1x4],[Parked_Car1y1 Parked_Car1y2 Parked_Car1y3
```

```
            Parked_Car1y4],[0 0 1]);
         [xa1, xb1, xb2, xa2, ya1, yb1, yb2, ya2]=Turn2(x,y,A,B,seta(i));
         pause(.05);
   end
pause(1);
for i=1:13
ya1=ya1+.5;
yb1=yb1+.5;
ya2=ya2+.5;
yb2=yb2+.5;
cla;
patch([Parked_Car2x1 Parked_Car2x2 Parked_Car2x3
    Parked_Car2x4],[Parked_Car2y1 Parked_Car2y2 Parked_Car2y3
    Parked_Car2y4],[1 0 0]);
patch([Parked_Car1x1 Parked_Car1x2 Parked_Car1x3
    Parked_Car1x4],[Parked_Car1y1 Parked_Car1y2 Parked_Car1y3
    Parked_Car1y4],[0 0 1]);
patch([xa1 xb1 xb2 xa2],[ya1 yb1 yb2 ya2],[0 1 0]);
ya11=ya1+(D-wl/2);
yb11=ya11+wl;
ya21=ya2+(D-wl/2);
yb21=ya21+wl;
xa11=xa2+(ww/2);
xb11=xa2+(ww/2);
xa21=xa2-(ww/2);
xb21=xa2-(ww/2);
patch([xa11 xb11 xb21 xa21],[ya11 yb11 yb21 ya21],[0 0 0]);
xa22=xa1-(ww/2);
xb22=xa1-(ww/2);
xa12=xa1+(ww/2);
xb12=xa1+(ww/2);
patch([xa12 xb12 xb22 xa22],[ya11 yb11 yb21 ya21],[0 0 0]);
ya13=ya11+L;
yb13=yb11+L;
patch([xa11 xb11 xb21 xa21],[ya13 yb13 yb13 ya13],[0 0 0]);
patch([xa12 xb12 xb22 xa22],[ya13 yb13 yb13 ya13],[0 0 0]);
```

```
pause(.02);
end
```

附录 D　第二问中模拟无人车运动轨迹代码

```
utils.py
\begin{appendices}
\section{第二问中实现水平停车的matlab代码}
\begin{lstlisting}[language=matlab]
clc
close all
clearvars
global wl;
global ww;
global D;
global L;
global WSeta;
wl=7;
ww=2;
    A=15;
    L=20;
    D=5;
    F=8;
    B=L+D+F;

    %%停放汽车 1 的矩形坐标

    Parked_Car1x1=0;
    Parked_Car1y1=-6;
    Parked_Car1x2=0;
    Parked_Car1y2=22;
    Parked_Car1x3=15;
    Parked_Car1y3=22;
    Parked_Car1x4=15;
    Parked_Car1y4=-6;
    Parked_Car2x1=0;
    Parked_Car2y1=70;
```

```
Parked_Car2x2=0;
Parked_Car2y2=100;
Parked_Car2x3=15;
Parked_Car2y3= 100;
Parked_Car2x4=15;
Parked_Car2y4=70;
WSeta=45;
WSeta=(WSeta/180)*pi; % degree to radians
S=3;
xi=Parked_Car2x4+S;
yi=Parked_Car2y4-9;
f1e=38;
    axis([-60 100 -20 100],'equal');
t=0:0.01:pi;
seta=2*t;
Xc=xi-(L/tan(WSeta));
Yc=yi+D;
Rbl=sqrt(D^2+(L/tan(WSeta))^2);
phi=atan(D/(L/tan(WSeta)));
          i=1;
       x=Xc+Rbl*cos(seta(i)+phi);
       y=Yc-Rbl*sin(seta(i)+phi);
       patch([Parked_Car2x1 Parked_Car2x2 Parked_Car2x3
           Parked_Car2x4],[Parked_Car2y1 Parked_Car2y2 Parked_Car2y3
           Parked_Car2y4],[1 0 0]);
       patch([Parked_Car1x1 Parked_Car1x2 Parked_Car1x3
           Parked_Car1x4],[Parked_Car1y1 Parked_Car1y2 Parked_Car1y3
           Parked_Car1y4],[0 0 1]);
       Turn1(x,y,A,B,seta(i));
       pause(4);
         for i=2:f1e
       x=Xc+Rbl*cos(seta(i)+phi);
       y=Yc-Rbl*sin(seta(i)+phi);
       cla;
       patch([Parked_Car2x1 Parked_Car2x2 Parked_Car2x3
           Parked_Car2x4],[Parked_Car2y1 Parked_Car2y2 Parked_Car2y3
           Parked_Car2y4],[1 0 0]);
```

```
        patch([Parked_Car1x1 Parked_Car1x2 Parked_Car1x3
            Parked_Car1x4],[Parked_Car1y1 Parked_Car1y2 Parked_Car1y3
            Parked_Car1y4],[0 0 1]);
        Turn1(x,y,A,B,seta(i));
        pause(.05);
   end

    xi=x+A*cos(seta(f1e));
    yi=y-A*sin(seta(f1e));
   Xc=xi+Rbl*cos(seta(f1e));
   Yc=yi-Rbl*sin(seta(f1e));
   pause(1);



   for i=f1e:-1:1
        x=Xc-Rbl*cos(seta(i));
        y=Yc+Rbl*sin(seta(i));
        cla;
        patch([Parked_Car2x1 Parked_Car2x2 Parked_Car2x3
            Parked_Car2x4],[Parked_Car2y1 Parked_Car2y2 Parked_Car2y3
            Parked_Car2y4],[1 0 0]);
        patch([Parked_Car1x1 Parked_Car1x2 Parked_Car1x3
            Parked_Car1x4],[Parked_Car1y1 Parked_Car1y2 Parked_Car1y3
            Parked_Car1y4],[0 0 1]);
        [xa1, xb1, xb2, xa2, ya1, yb1, yb2, ya2]=Turn2(x,y,A,B,seta(i));
        pause(.05);
   end
pause(1);
for i=1:13
ya1=ya1+.5;
yb1=yb1+.5;
ya2=ya2+.5;
yb2=yb2+.5;
cla;
patch([Parked_Car2x1 Parked_Car2x2 Parked_Car2x3
    Parked_Car2x4],[Parked_Car2y1 Parked_Car2y2 Parked_Car2y3
```

```
    Parked_Car2y4],[1 0 0]);
 patch([Parked_Car1x1 Parked_Car1x2 Parked_Car1x3
    Parked_Car1x4],[Parked_Car1y1 Parked_Car1y2 Parked_Car1y3
    Parked_Car1y4],[0 0 1]);
 patch([xa1 xb1 xb2 xa2],[ya1 yb1 yb2 ya2],[0 1 0]);
 ya11=ya1+(D-wl/2);
 yb11=ya11+wl;
 ya21=ya2+(D-wl/2);
 yb21=ya21+wl;
 xa11=xa2+(ww/2);
 xb11=xa2+(ww/2);
 xa21=xa2-(ww/2);
 xb21=xa2-(ww/2);
 patch([xa11 xb11 xb21 xa21],[ya11 yb11 yb21 ya21],[0 0 0]);
 xa22=xa1-(ww/2);
 xb22=xa1-(ww/2);
 xa12=xa1+(ww/2);
 xb12=xa1+(ww/2);
 patch([xa12 xb12 xb22 xa22],[ya11 yb11 yb21 ya21],[0 0 0]);
 ya13=ya11+L;
 yb13=yb11+L;
 patch([xa11 xb11 xb21 xa21],[ya13 yb13 yb13 ya13],[0 0 0]);
 patch([xa12 xb12 xb22 xa22],[ya13 yb13 yb13 ya13],[0 0 0]);

 pause(.02);
 end
```

pathplanning.py

```
import numpy as np
import math
import scipy.interpolate as scipy_interpolate
from utils import angle_of_line


####################### Functions #################

def interpolate_b_spline_path(x, y, n_path_points, degree=3):
```

```
    ipl_t = np.linspace(0.0, len(x) - 1, len(x))
    spl_i_x = scipy_interpolate.make_interp_spline(ipl_t, x, k=degree)
    spl_i_y = scipy_interpolate.make_interp_spline(ipl_t, y, k=degree)
    travel = np.linspace(0.0, len(x) - 1, n_path_points)
    return spl_i_x(travel), spl_i_y(travel)

def interpolate_path(path, sample_rate):
    choices = np.arange(0,len(path),sample_rate)
    if len(path)-1 not in choices:
           choices = np.append(choices , len(path)-1)
    way_point_x = path[choices,0]
    way_point_y = path[choices,1]
    n_course_point = len(path)*3
    rix, riy = interpolate_b_spline_path(way_point_x, way_point_y,
        n_course_point)
    new_path = np.vstack([rix,riy]).T
    # new_path[new_path<0] = 0
    return new_path

############## Path Planner #################
class AStarPlanner:

    def __init__(self, ox, oy, resolution, rr):
        """
        Initialize grid map for a star planning

        ox: x position list of Obstacles [m]
        oy: y position list of Obstacles [m]
        resolution: grid resolution [m]
        rr: robot radius[m]
        """

        self.resolution = resolution
        self.rr = rr
        self.min_x, self.min_y = 0, 0
        self.max_x, self.max_y = 0, 0
        self.obstacle_map = None
```

```
        self.x_width, self.y_width = 0, 0
        self.motion = self.get_motion_model()
        self.calc_obstacle_map(ox, oy)

    class Node:
        def __init__(self, x, y, cost, parent_index):
            self.x = x # index of grid
            self.y = y # index of grid
            self.cost = cost
            self.parent_index = parent_index

        def __str__(self):
            return str(self.x) + "," + str(self.y) + "," + str(
                self.cost) + "," + str(self.parent_index)

    def planning(self, sx, sy, gx, gy):
        """
        A star path search

        input:
            s_x: start x position [m]
            s_y: start y position [m]
            gx: goal x position [m]
            gy: goal y position [m]

        output:
            rx: x position list of the final path
            ry: y position list of the final path
        """

        start_node = self.Node(self.calc_xy_index(sx, self.min_x),
                               self.calc_xy_index(sy, self.min_y), 0.0, -1)
        goal_node = self.Node(self.calc_xy_index(gx, self.min_x),
                              self.calc_xy_index(gy, self.min_y), 0.0, -1)

        open_set, closed_set = dict(), dict()
        open_set[self.calc_grid_index(start_node)] = start_node
```

```
        while 1:
            if len(open_set) == 0:
                print("Open set is empty..")
                break

            c_id = min(
                open_set,
                key=lambda o: open_set[o].cost + self.calc_heuristic(goal_node,
                                                                     open_set[
                                                                         o]))
            current = open_set[c_id]

            if current.x == goal_node.x and current.y == goal_node.y:
                print("Find goal")
                goal_node.parent_index = current.parent_index
                goal_node.cost = current.cost
                break

            # Remove the item from the open set
            del open_set[c_id]

            # Add it to the closed set
            closed_set[c_id] = current

            # expand_grid search grid based on motion model
            for i, _ in enumerate(self.motion):
                node = self.Node(current.x + self.motion[i][0],
                                 current.y + self.motion[i][1],
                                 current.cost + self.motion[i][2], c_id)
                n_id = self.calc_grid_index(node)

                # If the node is not safe, do nothing
                if not self.verify_node(node):
                    continue

                if n_id in closed_set:
```

```
                continue

            if n_id not in open_set:
                open_set[n_id] = node # discovered a new node
            else:
                if open_set[n_id].cost > node.cost:
                    # This path is the best until now. record it
                    open_set[n_id] = node

    rx, ry = self.calc_final_path(goal_node, closed_set)

    return rx, ry

def calc_final_path(self, goal_node, closed_set):
    # generate final course
    rx, ry = [self.calc_grid_position(goal_node.x, self.min_x)], [
        self.calc_grid_position(goal_node.y, self.min_y)]
    parent_index = goal_node.parent_index
    while parent_index != -1:
        n = closed_set[parent_index]
        rx.append(self.calc_grid_position(n.x, self.min_x))
        ry.append(self.calc_grid_position(n.y, self.min_y))
        parent_index = n.parent_index

    return rx, ry

@staticmethod
def calc_heuristic(n1, n2):
    w = 1.0 # weight of heuristic
    d = w * math.hypot(n1.x - n2.x, n1.y - n2.y)
    return d

def calc_grid_position(self, index, min_position):
    """
    calc grid position

    :param index:
```

```
        :param min_position:
        :return:
        """
        pos = index * self.resolution + min_position
        return pos

    def calc_xy_index(self, position, min_pos):
        return round((position - min_pos) / self.resolution)

    def calc_grid_index(self, node):
        return (node.y - self.min_y) * self.x_width + (node.x - self.min_x)

    def verify_node(self, node):
        px = self.calc_grid_position(node.x, self.min_x)
        py = self.calc_grid_position(node.y, self.min_y)

        if px < self.min_x:
            return False
        elif py < self.min_y:
            return False
        elif px >= self.max_x:
            return False
        elif py >= self.max_y:
            return False

        # collision check
        if self.obstacle_map[node.x][node.y]:
            return False

        return True

    def calc_obstacle_map(self, ox, oy):

        self.min_x = round(min(ox))
        self.min_y = round(min(oy))
        self.max_x = round(max(ox))
        self.max_y = round(max(oy))
```

```
        self.x_width = round((self.max_x - self.min_x) / self.resolution)
        self.y_width = round((self.max_y - self.min_y) / self.resolution)

        # obstacle map generation
        self.obstacle_map = [[False for _ in range(self.y_width)]
                             for _ in range(self.x_width)]
        for ix in range(self.x_width):
            x = self.calc_grid_position(ix, self.min_x)
            for iy in range(self.y_width):
                y = self.calc_grid_position(iy, self.min_y)
                for iox, ioy in zip(ox, oy):
                    d = math.hypot(iox - x, ioy - y)
                    if d < self.rr:
                        self.obstacle_map[ix][iy] = True
                        break

    @staticmethod
    def get_motion_model():
        # dx, dy, cost
        motion = [[1, 0, 1],
                  [0, 1, 1],
                  [-1, 0, 1],
                  [0, -1, 1],
                  [-1, -1, math.sqrt(2)],
                  [-1, 1, math.sqrt(2)],
                  [1, -1, math.sqrt(2)],
                  [1, 1, math.sqrt(2)]]

        return motion


class PathPlanning:
    def __init__(self,obstacles):
        self.margin = 5
        #sacale obstacles from env margin to pathplanning margin
        obstacles = obstacles + np.array([self.margin,self.margin])
```

```
        obstacles = obstacles[(obstacles[:,0]>=0) & (obstacles[:,1]>=0)]

        self.obs = np.concatenate([np.array([[0,i] for i in
            range(100+self.margin)]),
                                   np.array([[100+2*self.margin,i] for i in
                                       range(100+2*self.margin)]),
                                   np.array([[i,0] for i in range(100+self.margin)]),
                                   np.array([[i,100+2*self.margin] for i in
                                       range(100+2*self.margin)]),
                                   obstacles])

        self.ox = [int(item) for item in self.obs[:,0]]
        self.oy = [int(item) for item in self.obs[:,1]]
        self.grid_size = 1
        self.robot_radius = 4
        self.a_star = AStarPlanner(self.ox, self.oy, self.grid_size,
            self.robot_radius)

    def plan_path(self,sx, sy, gx, gy):
        rx, ry = self.a_star.planning(sx+self.margin, sy+self.margin,
            gx+self.margin, gy+self.margin)
        rx = np.array(rx)-self.margin+0.5
        ry = np.array(ry)-self.margin+0.5
        path = np.vstack([rx,ry]).T
        return path[::-1]

############### Park Path Planner ##################

class ParkPathPlanning:
    def __init__(self,obstacles):
        self.margin = 5
        #sacale obstacles from env margin to pathplanning margin
        obstacles = obstacles + np.array([self.margin,self.margin])
        obstacles = obstacles[(obstacles[:,0]>=0) & (obstacles[:,1]>=0)]

        self.obs = np.concatenate([np.array([[0,i] for i in
            range(100+self.margin)]),
```

```
                            np.array([[100+2*self.margin,i] for i in
                                range(100+2*self.margin)]),
                            np.array([[i,0] for i in range(100+self.margin)]),
                            np.array([[i,100+2*self.margin] for i in
                                range(100+2*self.margin)]),
                            obstacles])

        self.ox = [int(item) for item in self.obs[:,0]]
        self.oy = [int(item) for item in self.obs[:,1]]
        self.grid_size = 1
        self.robot_radius = 4
        self.a_star = AStarPlanner(self.ox, self.oy, self.grid_size,
            self.robot_radius)

    def generate_park_scenario(self,sx, sy, gx, gy):
        rx, ry = self.a_star.planning(sx+self.margin, sy+self.margin,
            gx+self.margin, gy+self.margin)
        rx = np.array(rx)-self.margin+0.5
        ry = np.array(ry)-self.margin+0.5
        path = np.vstack([rx,ry]).T
        path = path[::-1]
        computed_angle =
            angle_of_line(path[-10][0],path[-10][1],path[-1][0],path[-1][1])

        s = 4
        l = 8
        d = 2
        w = 4

        if -math.atan2(0,-1) < computed_angle <= math.atan2(-1,0):
            x_ensure2 = gx
            y_ensure2 = gy
            x_ensure1 = x_ensure2 + d + w
            y_ensure1 = y_ensure2 - l - s
            ensure_path1 = np.vstack([np.repeat(x_ensure1,3/0.25),
                np.arange(y_ensure1-3,y_ensure1,0.25)[::-1]]).T
            ensure_path2 = np.vstack([np.repeat(x_ensure2,3/0.25),
```

```
            np.arange(y_ensure2,y_ensure2+3,0.25)[::-1]]).T
        park_path = self.plan_park_down_right(x_ensure2, y_ensure2)

    elif math.atan2(-1,0) <= computed_angle <= math.atan2(0,1):
        x_ensure2 = gx
        y_ensure2 = gy
        x_ensure1 = x_ensure2 - d - w
        y_ensure1 = y_ensure2 - l - s
        ensure_path1 = np.vstack([np.repeat(x_ensure1,3/0.25),
            np.arange(y_ensure1-3,y_ensure1,0.25)[::-1]]).T
        ensure_path2 = np.vstack([np.repeat(x_ensure2,3/0.25),
            np.arange(y_ensure2,y_ensure2+3,0.25)[::-1]]).T
        park_path = self.plan_park_down_left(x_ensure2, y_ensure2)

    elif math.atan2(0,1) < computed_angle <= math.atan2(1,0):
        x_ensure2 = gx
        y_ensure2 = gy
        x_ensure1 = x_ensure2 - d - w
        y_ensure1 = y_ensure2 + l + s
        ensure_path1 = np.vstack([np.repeat(x_ensure1,3/0.25),
            np.arange(y_ensure1,y_ensure1+3,0.25)]).T
        ensure_path2 = np.vstack([np.repeat(x_ensure2,3/0.25),
            np.arange(y_ensure2-3,y_ensure2,0.25)]).T
        park_path = self.plan_park_up_left(x_ensure2, y_ensure2)

    elif math.atan2(1,0) < computed_angle <= math.atan2(0,-1):
        x_ensure2 = gx
        y_ensure2 = gy
        x_ensure1 = x_ensure2 + d + w
        y_ensure1 = y_ensure2 + l + s
        ensure_path1 = np.vstack([np.repeat(x_ensure1,3/0.25),
            np.arange(y_ensure1,y_ensure1+3,0.25)]).T
        ensure_path2 = np.vstack([np.repeat(x_ensure2,3/0.25),
            np.arange(y_ensure2-3,y_ensure2,0.25)]).T
        park_path = self.plan_park_up_right(x_ensure2, y_ensure2)

    return np.array([x_ensure1, y_ensure1]), park_path, ensure_path1,
```

```
        ensure_path2


def plan_park_up_right(self, x1, y1):
        s = 4
        l = 8
        d = 2
        w = 4

        x0 = x1 + d + w
        y0 = y1 + l + s

        curve_x = np.array([])
        curve_y = np.array([])
        y = np.arange(y1,y0+1)
        circle_fun = (6.9**2 - (y-y0)**2)
        x = (np.sqrt(circle_fun[circle_fun>=0]) + x0-6.9)
        y = y[circle_fun>=0]
        choices = x>x0-6.9/2
        x=x[choices]
        y=y[choices]
        curve_x = np.append(curve_x, x[::-1])
        curve_y = np.append(curve_y, y[::-1])

        y = np.arange(y1,y0+1)
        circle_fun = (6.9**2 - (y-y1)**2)
        x = (np.sqrt(circle_fun[circle_fun>=0]) + x1+6.9)
        y = y[circle_fun>=0]
        x = (x - 2*(x-(x1+6.9)))
        choices = x<x1+6.9/2
        x=x[choices]
        y=y[choices]
        curve_x = np.append(curve_x, x[::-1])
        curve_y = np.append(curve_y, y[::-1])

        park_path = np.vstack([curve_x, curve_y]).T
        return park_path
```

```
def plan_park_up_left(self, x1, y1):
        s = 4
        l = 8
        d = 2
        w = 4

        x0 = x1 - d - w
        y0 = y1 + l + s

        curve_x = np.array([])
        curve_y = np.array([])
        y = np.arange(y1,y0+1)
        circle_fun = (6.9**2 - (y-y0)**2)
        x = (np.sqrt(circle_fun[circle_fun>=0]) + x0+6.9)
        y = y[circle_fun>=0]
        x = (x - 2*(x-(x0+6.9)))
        choices = x<x0+6.9/2
        x=x[choices]
        y=y[choices]
        curve_x = np.append(curve_x, x[::-1])
        curve_y = np.append(curve_y, y[::-1])

        y = np.arange(y1,y0+1)
        circle_fun = (6.9**2 - (y-y1)**2)
        x = (np.sqrt(circle_fun[circle_fun>=0]) + x1-6.9)
        y = y[circle_fun>=0]
        choices = x>x1-6.9/2
        x=x[choices]
        y=y[choices]
        curve_x = np.append(curve_x, x[::-1])
        curve_y = np.append(curve_y, y[::-1])

        park_path = np.vstack([curve_x, curve_y]).T
        return park_path
```

```
def plan_park_down_right(self, x1,y1):
        s = 4
        l = 8
        d = 2
        w = 4

        x0 = x1 + d + w
        y0 = y1 - l - s

        curve_x = np.array([])
        curve_y = np.array([])
        y = np.arange(y0,y1+1)
        circle_fun = (6.9**2 - (y-y0)**2)
        x = (np.sqrt(circle_fun[circle_fun>=0]) + x0-6.9)
        y = y[circle_fun>=0]
        choices = x>x0-6.9/2
        x=x[choices]
        y=y[choices]

        curve_x = np.append(curve_x, x)
        curve_y = np.append(curve_y, y)

        y = np.arange(y0,y1+1)
        circle_fun = (6.9**2 - (y-y1)**2)
        x = (np.sqrt(circle_fun[circle_fun>=0]) + x1+6.9)
        x = (x - 2*(x-(x1+6.9)))
        y = y[circle_fun>=0]
        choices = x<x1+6.9/2
        x=x[choices]
        y=y[choices]
        curve_x = np.append(curve_x, x)
        curve_y = np.append(curve_y, y)

        park_path = np.vstack([curve_x, curve_y]).T
        return park_path
```

```
    def plan_park_down_left(self, x1,y1):
            s = 4
            l = 8
            d = 2
            w = 4

            x0 = x1 - d - w
            y0 = y1 - l - s

            curve_x = np.array([])
            curve_y = np.array([])
            y = np.arange(y0,y1+1)
            circle_fun = (6.9**2 - (y-y0)**2)
            x = (np.sqrt(circle_fun[circle_fun>=0]) + x0+6.9)
            y = y[circle_fun>=0]
            x = (x - 2*(x-(x0+6.9)))
            choices = x<x0+6.9/2
            x=x[choices]
            y=y[choices]
            curve_x = np.append(curve_x, x)
            curve_y = np.append(curve_y, y)

            y = np.arange(y0,y1+1)
            circle_fun = (6.9**2 - (y-y1)**2)
            x = (np.sqrt(circle_fun[circle_fun>=0]) + x1-6.9)
            y = y[circle_fun>=0]
            choices = x>x1-6.9/2
            x=x[choices]
            y=y[choices]
            curve_x = np.append(curve_x, x)
            curve_y = np.append(curve_y, y)

            park_path = np.vstack([curve_x, curve_y]).T
            return park_path
```

main_autopark.py

```
import cv2
```

```
import numpy as np
from time import sleep
import argparse

from environment import Environment, Parking1
from pathplanning import PathPlanning, ParkPathPlanning, interpolate_path
from control import Car_Dynamics, MPC_Controller, Linear_MPC_Controller
from utils import angle_of_line, make_square, DataLogger

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--x_start', type=int, default=0, help='X of start')
    parser.add_argument('--y_start', type=int, default=90, help='Y of start')
    parser.add_argument('--psi_start', type=int, default=0, help='psi of
        start')
    parser.add_argument('--x_end', type=int, default=90, help='X of end')
    parser.add_argument('--y_end', type=int, default=80, help='Y of end')
    parser.add_argument('--parking', type=int, default=1, help='park position
        in parking1 out of 24')

    args = parser.parse_args()
    logger = DataLogger()

    ###### default variables ############################
    start = np.array([args.x_start, args.y_start])
    end   = np.array([args.x_end, args.y_end])
   ###############

    # environment margin : 5
    # pathplanning margin :

    ######################## defining obstacles
        ########################################
    parking1 = Parking1(args.parking)
    end, obs = parking1.generate_obstacles()

    # add squares
```

```
# square1 = make_square(10,65,20)
# square2 = make_square(15,30,20)
# square3 = make_square(50,50,10)
# obs = np.vstack([obs,square1,square2,square3])

# Rahneshan logo
# start = np.array([50,5])
# end = np.array([35,67])
# rah = np.flip(cv2.imread('READ_ME/rahneshan_obstacle.png',0), axis=0)
# obs = np.vstack([np.where(rah<100)[1],np.where(rah<100)[0]]).T

# new_obs = np.array([[78,78],[79,79],[78,79]])
# obs = np.vstack([obs,new_obs])
#### initialization ####################################################
env = Environment(obs)
my_car = Car_Dynamics(start[0], start[1], 0, np.deg2rad(args.psi_start),
    length=4, dt=0.2)
MPC_HORIZON = 5
controller = MPC_Controller()
# controller = Linear_MPC_Controller()

res = env.render(my_car.x, my_car.y, my_car.psi, 0)
cv2.imshow('environment', res)
key = cv2.waitKey(1)
############ path planning
    ################################################
park_path_planner = ParkPathPlanning(obs)
path_planner = PathPlanning(obs)

print('planning park scenario ...')
new_end, park_path, ensure_path1, ensure_path2 =
    park_path_planner.generate_park_scenario(int(start[0]),int(start[1]),int(end[0]),int(

print('routing to destination ...')
path =
    path_planner.plan_path(int(start[0]),int(start[1]),int(new_end[0]),int(new_end[1]))
path = np.vstack([path, ensure_path1])
```

```
    print('interpolating ...')
    interpolated_path = interpolate_path(path, sample_rate=5)
    interpolated_park_path = interpolate_path(park_path, sample_rate=2)
    interpolated_park_path = np.vstack([ensure_path1[::-1],
        interpolated_park_path, ensure_path2[::-1]])

    env.draw_path(interpolated_path)
    env.draw_path(interpolated_park_path)

    final_path = np.vstack([interpolated_path, interpolated_park_path,
        ensure_path2])

     ########## control ###################
    print('driving to destination ...')
    for i,point in enumerate(final_path):

            acc, delta = controller.optimize(my_car,
                final_path[i:i+MPC_HORIZON])
            my_car.update_state(my_car.move(acc, delta))
            res = env.render(my_car.x, my_car.y, my_car.psi, delta)
            logger.log(point, my_car, acc, delta)
            cv2.imshow('environment', res)
            key = cv2.waitKey(1)
            if key == ord('s'):
                cv2.imwrite('res.png', res*255)

    # zeroing car steer
    res = env.render(my_car.x, my_car.y, my_car.psi, 0)
    logger.save_data()
    cv2.imshow('environment', res)
    key = cv2.waitKey()
    ################################################################################

    cv2.destroyAllWindows()
```

environment.py

```
import cv2
import numpy as np

class Environment:
    def __init__(self,obstacles):
        self.margin = 5
        #coordinates are in [x,y] format
        self.car_length = 80
        self.car_width = 40
        self.wheel_length = 15
        self.wheel_width = 7
        self.wheel_positions = np.array([[25,15],[25,-15],[-25,15],[-25,-15]])

        self.color = np.array([0,0,255])/255
        self.wheel_color = np.array([20,20,20])/255

        self.car_struct = np.array([[+self.car_length/2, +self.car_width/2],
                                    [+self.car_length/2, -self.car_width/2],
                                    [-self.car_length/2, -self.car_width/2],
                                    [-self.car_length/2, +self.car_width/2]],
                                    np.int32)

        self.wheel_struct = np.array([[+self.wheel_length/2,
            +self.wheel_width/2],
                                      [+self.wheel_length/2, -self.wheel_width/2],
                                      [-self.wheel_length/2, -self.wheel_width/2],
                                      [-self.wheel_length/2, +self.wheel_width/2]],
                                      np.int32)

        #height and width
        self.background = np.ones((1000+20*self.margin,1000+20*self.margin,3))
        self.background[10:1000+20*self.margin:10,:] =
            np.array([200,200,200])/255
        self.background[:,10:1000+20*self.margin:10] =
            np.array([200,200,200])/255
        self.place_obstacles(obstacles)
```

```
def place_obstacles(self, obs):
    obstacles = np.concatenate([np.array([[0,i] for i in
        range(100+2*self.margin)]),
                            np.array([[100+2*self.margin-1,i] for i in
                                range(100+2*self.margin)]),
                            np.array([[i,0] for i in
                                range(100+2*self.margin)]),
                            np.array([[i,100+2*self.margin-1] for i in
                                range(100+2*self.margin)]),
                            obs + np.array([self.margin,self.margin])])*10
    for ob in obstacles:
        self.background[ob[1]:ob[1]+10,ob[0]:ob[0]+10]=0

def draw_path(self, path):
    path = np.array(path)*10
    color = np.random.randint(0,150,3)/255
    path = path.astype(int)
    for p in path:
        self.background[p[1]+10*self.margin:p[1]+10*self.margin+3,p[0]+10*self.margin:p[0]

def rotate_car(self, pts, angle=0):
    R = np.array([[np.cos(angle), -np.sin(angle)],
                  [np.sin(angle), np.cos(angle)]])
    return ((R @ pts.T).T).astype(int)

def render(self, x, y, psi, delta):
    # x,y in 100 coordinates
    x = int(10*x)
    y = int(10*y)
    # x,y in 1000 coordinates
    # adding car body
    rotated_struct = self.rotate_car(self.car_struct, angle=psi)
    rotated_struct += np.array([x,y]) +
        np.array([10*self.margin,10*self.margin])
    rendered = cv2.fillPoly(self.background.copy(), [rotated_struct],
        self.color)
```

```
        # adding wheel
        rotated_wheel_center = self.rotate_car(self.wheel_positions, angle=psi)
        for i,wheel in enumerate(rotated_wheel_center):

            if i <2:
                rotated_wheel = self.rotate_car(self.wheel_struct,
                    angle=delta+psi)
            else:
                rotated_wheel = self.rotate_car(self.wheel_struct, angle=psi)
            rotated_wheel += np.array([x,y]) + wheel +
                np.array([10*self.margin,10*self.margin])
            rendered = cv2.fillPoly(rendered, [rotated_wheel], self.wheel_color)

        # gel
        gel =
            np.vstack([np.random.randint(-50,-30,16),np.hstack([np.random.randint(-20,-10,8),
        gel = self.rotate_car(gel, angle=psi)
        gel += np.array([x,y]) + np.array([10*self.margin,10*self.margin])
        gel = np.vstack([gel,gel+[1,0],gel+[0,1],gel+[1,1]])
        rendered[gel[:,1],gel[:,0]] = np.array([60,60,135])/255

        new_center = np.array([x,y]) + np.array([10*self.margin,10*self.margin])
        self.background = cv2.circle(self.background,
            (new_center[0],new_center[1]), 2, [255/255, 150/255, 100/255], -1)

        rendered = cv2.resize(np.flip(rendered, axis=0), (700,700))
        return rendered


class Parking1:
    def __init__(self, car_pos):
        self.car_obstacle = self.make_car()
        self.walls = [[70,i] for i in range(-5,90) ]+\
                    [[30,i] for i in range(10,105)]+\
                    [[i,10] for i in range(30,36) ]+\
                    [[i,90] for i in range(70,76) ] #+ [[i,20] for i in
                        range(-5,50)]
```

```
        # self.walls = [0,100]
        self.obs = np.array(self.walls)
        self.cars = {1 : [[35,20]], 2 : [[65,20]], 3 : [[75,20]], 4 : [[95,20]],
                     5 : [[35,32]], 6 : [[65,32]], 7 : [[75,32]], 8 : [[95,32]],
                     9 : [[35,44]], 10: [[65,44]], 11: [[75,44]], 12: [[95,44]],
                     13: [[35,56]], 14: [[65,56]], 15: [[75,56]], 16: [[95,56]],
                     17: [[35,68]], 18: [[65,68]], 19: [[75,68]], 20: [[95,68]],
                     21: [[35,80]], 22: [[65,80]], 23: [[75,80]], 24: [[95,80]]}
        self.end = self.cars[car_pos][0]
        self.cars.pop(car_pos)

    def generate_obstacles(self):
        for i in self.cars.keys():
            for j in range(len(self.cars[i])):
                obstacle = self.car_obstacle + self.cars[i]
                self.obs = np.append(self.obs, obstacle)
        return self.end, np.array(self.obs).reshape(-1,2)

    def make_car(self):
        car_obstacle_x, car_obstacle_y = np.meshgrid(np.arange(-2,2),
            np.arange(-4,4))
        car_obstacle = np.dstack([car_obstacle_x, car_obstacle_y]).reshape(-1,2)
        return car_obstacle
```

control.py

```
import numpy as np
from scipy.optimize import minimize
import copy


class Car_Dynamics:
    def __init__(self, x_0, y_0, v_0, psi_0, length, dt):
        self.dt = dt          # sampling time
        self.L = length       # vehicle length
        self.x = x_0
        self.y = y_0
        self.v = v_0
```

```
        self.psi = psi_0
        self.state = np.array([[self.x, self.y, self.v, self.psi]]).T

    def move(self, accelerate, delta):
        x_dot = self.v*np.cos(self.psi)
        y_dot = self.v*np.sin(self.psi)
        v_dot = accelerate
        psi_dot = self.v*np.tan(delta)/self.L
        return np.array([[x_dot, y_dot, v_dot, psi_dot]]).T

    def update_state(self, state_dot):
        # self.u_k = command
        # self.z_k = state
        self.state = self.state + self.dt*state_dot
        self.x = self.state[0,0]
        self.y = self.state[1,0]
        self.v = self.state[2,0]
        self.psi = self.state[3,0]


class MPC_Controller:
    def __init__(self):
        self.horiz = None
        self.R = np.diag([0.01, 0.01])           # input cost matrix
        self.Rd = np.diag([0.01, 1.0])           # input difference cost matrix
        self.Q = np.diag([1.0, 1.0])             # state cost matrix
        self.Qf = self.Q                         # state final matrix

    def mpc_cost(self, u_k, my_car, points):
        mpc_car = copy.copy(my_car)
        u_k = u_k.reshape(self.horiz, 2).T
        z_k = np.zeros((2, self.horiz+1))

        desired_state = points.T
        cost = 0.0

        for i in range(self.horiz):
```

```
            state_dot = mpc_car.move(u_k[0,i], u_k[1,i])
            mpc_car.update_state(state_dot)

            z_k[:,i] = [mpc_car.x, mpc_car.y]
            cost += np.sum(self.R@(u_k[:,i]**2))
            cost += np.sum(self.Q@((desired_state[:,i]-z_k[:,i])**2))
            if i < (self.horiz-1):
                cost += np.sum(self.Rd@((u_k[:,i+1] - u_k[:,i])**2))
        return cost

    def optimize(self, my_car, points):
        self.horiz = points.shape[0]
        bnd = [(-5, 5),(np.deg2rad(-60), np.deg2rad(60))]*self.horiz
        result = minimize(self.mpc_cost, args=(my_car, points), x0 =
            np.zeros((2*self.horiz)), method='SLSQP', bounds = bnd)
        return result.x[0], result.x[1]



########################################################################

class Linear_MPC_Controller:
    def __init__(self):
        self.horiz = None
        self.R = np.diag([0.01, 0.01])           # input cost matrix
        self.Rd = np.diag([0.01, 1.0])           # input difference cost matrix
        self.Q = np.diag([1.0, 1.0])             # state cost matrix
        self.Qf = self.Q                         # state final matrix
        self.dt=0.2
        self.L=4

    def make_model(self, v, psi, delta):
        # matrices
        # 4*4
        A = np.array([[1, 0, self.dt*np.cos(psi)   , -self.dt*v*np.sin(psi)],
                      [0, 1, self.dt*np.sin(psi)    , self.dt*v*np.cos(psi) ],
                      [0, 0, 1                      , 0                     ],
```

```
                [0, 0, self.dt*np.tan(delta)/self.L, 1            ]])
    # 4*2
    B = np.array([[0   , 0                              ],
                [0     , 0                             ],
                [self.dt, 0                            ],
                [0     , self.dt*v/(self.L*np.cos(delta)**2)]])

    # 4*1
    C = np.array([[self.dt*v* np.sin(psi)*psi        ],
                [-self.dt*v*np.cos(psi)*psi          ],
                [0                                   ],
                [-self.dt*v*delta/(self.L*np.cos(delta)**2)]])

    return A, B, C

def mpc_cost(self, u_k, my_car, points):

    u_k = u_k.reshape(self.horiz, 2).T
    z_k = np.zeros((2, self.horiz+1))
    desired_state = points.T
    cost = 0.0
    old_state = np.array([my_car.x, my_car.y, my_car.v,
        my_car.psi]).reshape(4,1)

    for i in range(self.horiz):
        delta = u_k[1,i]
        A,B,C = self.make_model(my_car.v, my_car.psi, delta)
        new_state = A@old_state + B@u_k + C

        z_k[:,i] = [new_state[0,0], new_state[1,0]]
        cost += np.sum(self.R@(u_k[:,i]**2))
        cost += np.sum(self.Q@((desired_state[:,i]-z_k[:,i])**2))
        if i < (self.horiz-1):
            cost += np.sum(self.Rd@((u_k[:,i+1] - u_k[:,i])**2))

        old_state = new_state
    return cost
```

```
def optimize(self, my_car, points):
    self.horiz = points.shape[0]
    bnd = [(-5, 5),(np.deg2rad(-60), np.deg2rad(60))]*self.horiz
    result = minimize(self.mpc_cost, args=(my_car, points), x0 =
        np.zeros((2*self.horiz)), method='SLSQP', bounds = bnd)
    return result.x[0], result.x[1]
```