福特渲染接口文档输出

目录

序号	修改日期	作者	修改描述	项目版本
1		简惠灵/杨若飞/路海涛/王爽	1. 初稿	V1.0

第1章 福特渲染需求输入

功能		道路环境			障碍物			规划轨迹线
		立柱	减速带	车位	行人	锥桶	汽车	
HAVP	学习	x	x	v	x	x	x	N/A

	使用	v（地图文件）	v（地图文件）	v	v	v	v	V
APA&RPA	**搜库**	x	x	v	x	x	v	N/A
	泊车	x	x	v	x	x	v	V
Backtrack	**使用**	x	x	x	x	x	x	V

第2章 以太网信号数据

2.1 信号发送链路

实时数据传输：ACU通过以太网将功能使用过程中需要的渲染信息输出给IVI ，具体信号见2.2

地图文件静态数据传输：HAVP使用的地图文件和backtrack使用的路线轨迹为一次性发送

1. HAVP地图文件:格式为json文件格式，发送时机为当ACU端完成路线学习后发送，如下图所示。

2. backtrack路线轨迹文件：发送时机为当用户 点击倒车循迹入口时，当ACU判断此时历史轨迹可用时一次性发送当前需要使用的轨迹点集合，轨迹点的顺序为从使用功能的那一点开始，即如果如果是A到B的路线，那么轨迹点发送顺序是B到A。

2.2 福特信号列表-实时数据传输

信号名称				信号描述	字段类型	最小值	最大值	单位	发送周期（ms）	枚举值描述
类别	信号各字段名称									
	子字段1	子字段2								
障碍物	Obstacle（可移动障碍物）	id		障碍物id	int32	1	/	/	10Hz	
		timestamp		发送数据时的时间戳	uint64	/	/	us	10hz	

		type		障碍物显示的类型	enum	/	/	/	10Hz	OBSTACLE_TYPE_CAR= 0; // 汽车
										OBSTACLE_TYPE_Pedestrian_Stanstill= 1; // 静止大人
										OBSTACLE_TYPE_Pedestrian_Moving= 2; // 移动大人
										OBSTACLE_TYPE_Pedestrian_Kid_Stanstill= 3; // 静止儿童（预留）
										OBSTACLE_TYPE_Pedestrian_Kid_Moving= 4; // 移动儿童（预留）
										OBSTACLE_TYPE_CONE= 5; // 锥筒
										OBSTACLE_TYPE_UNKNOWN = 6；//其它

		speed_heading		障碍物运动方向与本车行驶方向夹角	double	-π	+π	rad	10Hz	
		bounding_box		障碍物包裹框，外接矩形(世界坐标系)	BoundingBox	/	/		10Hz	

	Bounding Box	left_front		框障碍物的左前坐标点，包括，x，y，z	Point	/	/	m	10Hz	
		right_front		框障碍物的右前坐标点，包括，x，y，z	Point	/	/	m	10Hz	
		left_back		框障碍物的左后坐标点，包括，x，y，z	Point	/	/	m	10Hz	
		right_back		框障碍物的右后坐标点，包括，x，y，z	Point	/	/	m	10Hz	
轨迹线	Planing	trajectory		泊车箭头轨迹点集合	Point	/	/	/	10Hz	
		trajectoryDir		泊车箭头轨迹点的方向集合	double	/	/	/	10Hz	
车位信息	ParkingSpaceInfo	bounding_point		车位轮廓描述	Point	/	/	/	10Hz	

		parkspace_type		车位类型	enum	/	/	/	10Hz	NO_TYPE= 0;
										CROSS= 1; //垂直车位
										PARALLEL = 2; //平行车位
										DIAGNOAL = 3; //斜车位

		parking_lot_status		车位状态空还是非空	enum	/	/	/	10Hz	NO_COMMENT= 0x0; PARKING_LOT_AVAIL= 0x1; 空车位 PARKING_LOT_UNAVAIL = 0x2非空车位

		ParkSpaceOdd		空车位是否可以规划泊入	enum	/	/	/	10Hz	ODD_NONE = 0;
										ODD_PASS = 1;//可以泊入
										ODD_FAIL = 2;//不可以泊入

		OccupyStatus		不可泊车位被占用的类型	enum	/	/	/	10Hz	OCCUPY_NONE= 0;//默认值

									OCCUPY_VEHICLE= 1// 车 other =2//其他

		endParkingSpaceInfo		是否为终点车位-HAVP	enum				10Hz	NO= 0;//其他功能为默认值
										YES= 1//仅HAVP下固定路线时当前车位为终点车位时发送此值

		id		车位ID	int32	1	999	/	10Hz	
自车当前的位置		current_pose		车辆当前的位置	Pose	/	/	/	10Hz	
		Pose	point	当前位置XYZ	Point	/	/	/	10Hz	
			euler_ypr	欧拉角	Euler	/	/	/	10Hz	
			timestamp	时间戳	uint64	/	/	/	10Hz	

2.3 福特信号列表-地图文件静态数据

2.3.1 HAVP路线文件

信号名称				信号描述	字段类型
类别	信号各字段名称				
	子字段1	子字段2			
HAVP路线地图	通用信息	id		路线id	string
	地图文件数据	roads-道路轨迹线信息	id	恒为lane	string
			type	恒为 “0200”,即车道	int

		geometry	路线轨迹的点集，每个点为有x,y,z三个值的空间坐标点	array<Point>
	roadmarks-减速带信息	id	减速带ID	string
		type	恒为 0500：减速带	int
		geometry	减速带四个角点	array<Point>
		indices	几何形状索引集	array<int>
	bndboxs-柱子信息	id	柱子ID	string
		type	恒为 “0800”,即立柱	string
		geometry	立柱矩形四个角点	array<Point>
		indices	几何形状索引集	array<int>
	regions-车位信息	id	车位ID	string
		type	恒为1停车位	int
		geometry	车位矩形坐标点集合	array<Point>
		indices	几何形状索引集	array<int>
	end_information-终点信息	end_point	终点坐标点	array<Point>
		end_park	终点车位ID	string

2.3.1 backtrack历史轨迹文件

信号名称				信号描述	字段类型
类别	信号各字段名称				
	子字段1	子字段2			

backtrack倒车循迹的路线信息	trajectory-backtrack		历史走过的轨迹点的集合	Point

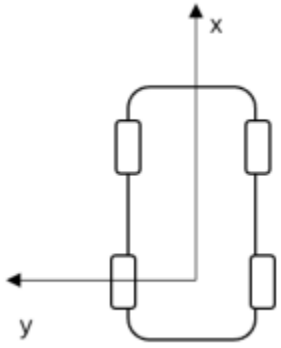
第3章 接口数据说明

3.1 坐标系定义

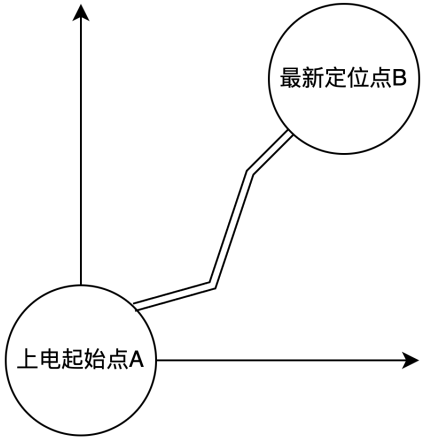
HAVP使用过程中涉及的坐标系定义如下

- 车体坐标系V：原点为车后轴中心在地面的投影点，方向前X；左Y；上Z

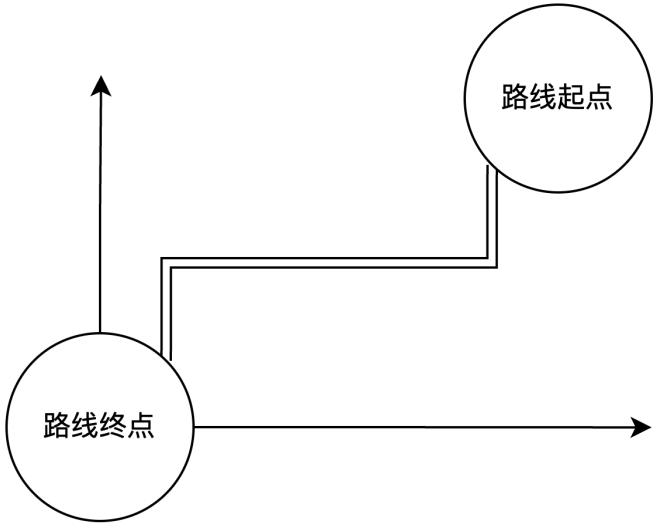
ACU需确保发出的动态信息即（车位，轨迹线，障碍物）的坐标原点为车体坐标系原点。



- 局部定位坐标系O：原点为上电时刻，方向为右X；前Y；上Z，如下图所示，即相对上电时刻的坐标点



- 全局定位坐标系W：原点为当前使用路线地图的终点作为起点，右X；前Y；上Z，如下图所示，适用功能：适用于HAVP地图。



3.2 历史轨迹信息渲染说明（适用HAVP/Backtrack）

3.2.1HAVP学习

HAVP学习过程中ACU实时发送车位信息给IVI进行渲染。

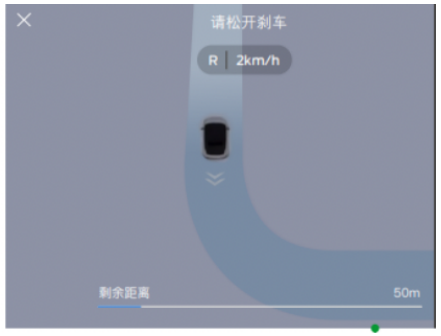
3.2.2HAVP使用

IVI使用HAVP地图模型文件（json）-全局坐标系，IVI从ACU接收当前的全局定位信息，提取地图文件对应位置周边的元素，随着车辆的前进，IVI将地图文件中存在的静态环境包括柱子、减速带进行绘图显示，同时IVI实时接收动态信息 （障碍物、规划轨迹线、车位信息）进行更新叠加显示。

3.2.3 backtrack使用路线

IVI使用地图轨迹点集合作为底图，从ACU接收当前的局部定位坐标，提取路线轨迹文件（局部定位坐标）对应位置区域显示，随着车辆的前进，IVI实时接收

动态信息 （规划轨迹线）进行更新显示。



1、历史轨迹信息

a.轨迹长度：长度最长为50m的轨迹点集合

b.轨迹定义和间隔：路线发送的形式为轨迹点的集合，其中每30cm间隔一个点；

c.坐标原点：坐标原点按照车辆里程计坐标系，即每次上电后时刻的点；

d.下电记忆：车辆正常下电后，backtrack可以记录下电前的路径，车辆上电后，轨迹的坐标原点需重置为最新上电时刻的点；

整车异常下电时（如直接切断蓄电池电源，电源系统故障导致下电等），记录轨迹丢失

历史走过的轨迹点举例，即点坐标的集合，单个点的坐标例子为，因福特目前是二维渲染，因此只用X和Y坐标即可，Z可以预留：

{ "x": 3.6541361237261776e+00,

"y": 7.5885431777488579e+00,

"z": -1.4825456462408355e+00}

2、箭头信息：

箭头信息和其他功能下ACU传输的规划轨迹信息trajectory一致，包括轨迹点和轨迹方向。

箭头轨迹点举例：

{ optional double x = 1;-X坐标

optional double y = 2;-Y坐标

optional double heading = 3;-当前点的角度}

3.3 渲染元素proto说明

3.3.1 障碍物信息

坐标系说明：障碍物信息均为车体坐标系，即相对车辆后轴中心

message Obstacle { // (可移动)障碍物

int32 id = 1; // **障碍物ID即ACU生成的用于标记障碍物身份识别的数字，此数字对IVI来说无特殊意义**

uint64 timestamp = 2; // **ACU生成的当前数据发送给IVI时的时间信息，用于IVI识别当前一组数据是否为同一时间发出**

ObstacleType type = 3; // **障碍物类型，包含和福特达成一致的障碍物类型枚举值，举例如IVI接收到1时，则代表此时障碍物为汽车**

enum ObstacleType { //障碍物类型

OBSTACLE_TYPE_CAR = 0; // 汽车

OBSTACLE_TYPE_Pedestrian_Stanstill = 1; // 静止大人

OBSTACLE_TYPE_Pedestrian_Moving= 2; // 移动大人

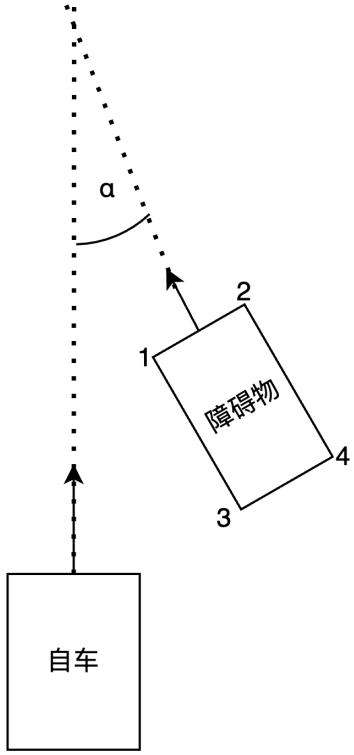
OBSTACLE_TYPE_Pedestrian_Kid_Stanstill= 3; // **静止儿童（预留）目前福特不支持大人和儿童的检测，仅预留接口**

OBSTACLE_TYPE_Pedestrian_Kid_Moving= 4; // **移动儿童（预留）目前福特不支持大人和儿童的检测，仅预留接口**

OBSTACLE_TYPE_CONE = 5; // 锥筒}

}

double speed_heading = 4; // **障碍物运动方向与本车行驶方向夹角，障碍物在自车的左边为正，右边为负值，如下图的角α。**



BoundingBox bounding_box = 5; // **障碍物包裹框，即障碍物接地四个角点的坐标，如上图所示的1,2,3,4角点**

// 障碍物包裹框

message BoundingBox { Point left_front = 1; // 左前角点

Point right_front = 2; // 右前角点

Point left_back = 3; // 左后角点

Point right_back = 4; // 右后角点 }

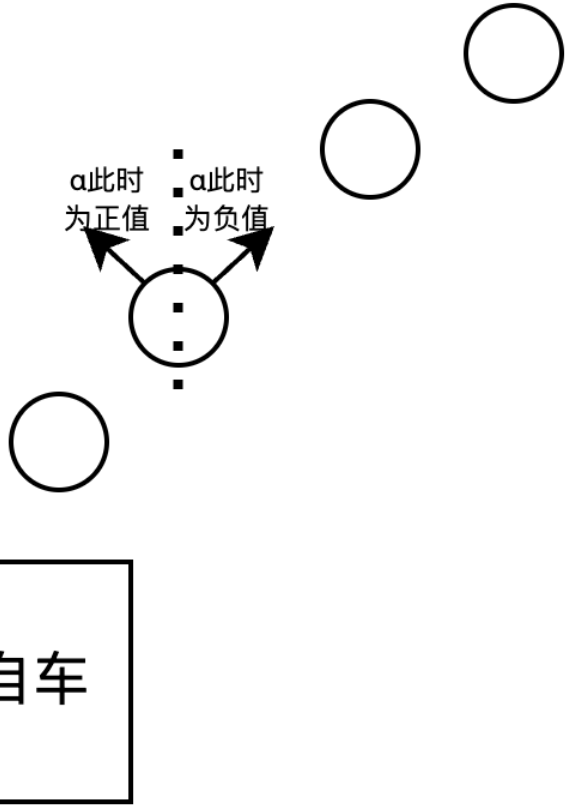
3.3.2 轨迹线信息

坐标系说明：障碍物信息均为车体坐标系，即相对车辆后轴中心

message Planing { //规划路线，为ACU当前规划路线，

repeated Point trajectory = 1; // **轨迹点，每0.3M一个点，共15m，IVI根据渲染需要进行选取和使用。**

repeated double trajectoryDir = 2; // **轨迹点的方向，即当前轨迹点的偏航角，相对正前方向，向右为负，向左为正**



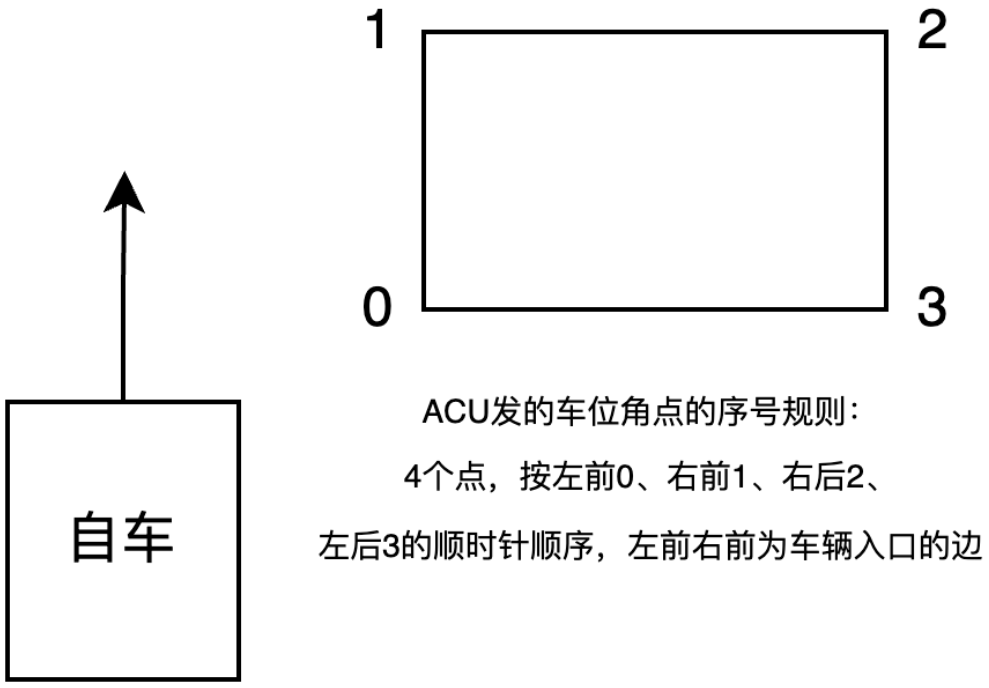
}

3.3.3 车位信息

坐标系说明：障碍物信息均为车体坐标系，即相对车辆后轴中心

message ParkingSpaceInfo {

repeated Point bounding_point = 1; // **(数组形式，4个点如下图，按顺时针左前0、右前1、右后2、左后3的顺时针顺序，左前右前为车辆入口的边)**



ParkSpaceType parkspace_type = 2; // 车位类型

```
                                    enum ParkSpaceType {
                                    NO_TYPE  = 0;
                                        CROSS    = 1; //垂直车位
                                    PARALLEL = 2; //平行车位
                                    DIAGNOAL = 3; //斜车位
                                     }
ParkingLotStatus parking_lot_status  = 3; // 车位状态空还是非空
                                       enum ParkingLotStatus {
                                            NO_COMMENT          = 0x0;
                                           PARKING_LOT_AVAIL   = 0x1;空车位
                                          PARKING_LOT_UNAVAIL = 0x2非空车位
                                        }
ParkSpaceOdd     parkspace_odd       = 4; // 如当前车位是空车位，是否可以规划泊入
                                      enum ParkSpaceOdd {
                                          ODD_NONE = 0;
                                          ODD_PASS = 1; // 可以泊入
                                          ODD_FAIL = 2; // 不可以泊入
                                        }
OccupyStatus     occupy_status       = 5;//当前车位被占的状态，被车占用或其他
                                       enum OccupyStatus {
                                          OCCUPY_NONE        = 0;
```

```
        OCCUPY_VEHICLE    = 1; // 车
        OCCUPY_OTHER      = 2; // 其他
    }
int32    id    = 6;//车位编号，即当前车位相对位置车辆位置，对应CAN信号的APA_DriverParkSlotReq=Slot_L1-L5,R1-R5。
endParkingSpaceInfo endParkingSpace_Info=7;  //当前是否为HAVP终点车位，HAVP下固定路线使用，表明当前车位是否为终点车位
    enum endParkingSpaceInfo {
        NO = 0; //其他功能为默认值
        YES  = 1 ;//仅HAVP下固定路线时当前车位为终点车位时发送此值
    }
}
```

3.3.4 HAVP地图文件数据说明

1. 数据主体结构


```
{"id":"9",
    "links": {},
    "bndboxs": {},
    "floors": {},
    "regions": {},
    "roads": {},
    "roadmarks": {},
    "end_information": {}}
```

字段	值类型	说明
id	string	地图的id.如9
bndboxs	object	立柱
regions	object	车位数据
roads	object	道路数据
roadmarks	object	减速带
end_information	object	泊车道路终点、目标车位等相关数据

2. bndboxs 建筑结构信息，即立柱


```
"pillar0": {
        "id": "pillar0",
        "type": "0800",
        "geometry": [
            { "x": -6.7624267898555708e+00, "y": -2.8634093462413039e+00, "z": 2.9984511136996694e-02 },
            { "x": -6.7624267898555708e+00, "y": -2.8634093462413039e+00, "z": 2.9984511136996694e-02 },
            { "x": -6.7624267898555708e+00, "y": -2.8634093462413039e+00, "z": 2.9984511136996694e-02 },
            { "x": -6.7624267898555708e+00, "y": -2.8634093462413039e+00, "z": 2.9984511136996694e-02 }
        ],
        "indices": [ 0, 1, 2, 0, 2, 3 ]
    },
```

字段	值类型	说明
id	string	地图的id，例如:"pillar0"
type	string	结构类型 立柱："0800"
indices	int array	索引数组。任何三维模型都是由三角面构成的，索引用于描述构成一个模型所用到的三角面需要由哪些坐标点构成。所以索引数组的长度必须是3的倍数。例如：模型包含4个顶点坐标，索引数据是[0,1,2,0,2,3]，说明该模型由两个三角形构成，这两个三角形分别由第1、2、3 和第1、3、4个坐标点所构成 渲染端拿到所有的索引集合，设置给自己创建的Mesh网格的索引数据。
geometry	object Array [{ x: double, y: double, z: double }, ...]	构成几何图形的坐标点集合。 用于绘制模型的空间信息。例如一个平面方形，就是由4个顶点构成的，一个立体方块则是由至少8个顶点构成。 每个顶点由三维空间中的x、y、z轴坐标数据构成，用于定位三维空间位置。例如: {"x": 8.1851043701171875e+00, "y": 1.1639657974243164e+01, "z": -6.7235745489597321e-02 } 渲染测拿到这四个坐标点，求出它的中心点（加起来除以四），作为柱子的渲染位置。 渲染端拿到所有的顶点集合，设置给自己创建的Mesh网格的顶点数据。

3. regions 车位信息

备注：这里的信息只是传递原始地图的车位四个角点，车位空还是非空按照ParkingSpaceInfo中的parking_lot_status

"park199": {
"id": "park199",

```
        "type": 1,
        "indices": [ 0, 1, 2, 0,  2,  3 ],
        "geometry": [
            { "x": 2.9420161541126468e+01, "y": -3.5859907331403221e+01, "z": -1.5139571630009108e+00  },
            { "x": 2.9420161541126468e+01, "y": -3.5859907331403221e+01, "z": -1.5139571630009108e+00  },
            { "x": 2.9420161541126468e+01, "y": -3.5859907331403221e+01, "z": -1.5139571630009108e+00  },
            { "x": 2.9420161541126468e+01, "y": -3.5859907331403221e+01, "z": -1.5139571630009108e+00  }
        ]
}
```


字段	值类型	说明
id	string	地图的id，例如:"park199"
type	int	车位类型 1：停车位
indices	int array	索引数组。任何三维模型都是由三角面构成的，索引用于描述构成一个模型所用到的三角面需要由哪些坐标点构成。所以索引数组的长度必须是3的倍数。例如：模型包含4个顶点坐标，索引数据是[0,1,2,0,2,3]，说明该模型由两个三角形构成，这两个三角形分别由第1、2、3 和第1、3、4个坐标点组成 渲染端拿到所有的索引集合，设置给自己创建的Mesh网格的索引数据。
geometry	object array [{ x: double, y: double,	构成几何图形的坐标点集合。 用于绘制模型的空间信息。例如一个平面方形，就是由4个顶点构成的，一个立体方块则是由至少8个顶点构成。 每个顶点由三维空间中的x、y、z轴坐标数据构成，用于定位三维空间位置。例如: {"x": 8.1851043701171875e+00, "y": 1.1639657974243164e+01,

	z: double }, ...]	"z": -6.7235745489597321e-02 } 渲染拿到这前四个坐标点，求出它的中心点（加起来除以四），作为车位的渲染位置。 渲染端拿到所有的顶点集合，设置给自己创建的Mesh网格的顶点数据。

4. road 道路信息

```
"roads": {
    "lane": {
      "id": "lane",
      "type": "0200",
      "indices": [ 0,1,2,2,3,1,...],
       "geometry": [
         { "x": -1.7441921119273172e+00,"y": 9.8558946535366836e+00, "z": -3.4792942124856265e-02},
         .....
       ]
    }
}
```

字段	类型	说明
id	string	地图的id，例如:"lane"
type	string	道路类型 普通道路："0200"

indices	int array	索引数组。任何三维模型都是由三角面构成的，索引用于描述构成一个模型所用到的三角面需要由哪些坐标点构成。所以索引数组的长度必须是3的倍数。例如：模型包含4个顶点坐标，索引数据是[0,1,2,0,2,3]，说明该模型由两个三角形构成，这两个三角形分别由第1、2、3 和第1、3、4个坐标点所构成 渲染端拿到路面的所有的索引集合，设置给自己创建的Mesh网格的索引数据。
geometry	object array [{ x: double, y: double, z: double }, ...]	构成几何图形的坐标点集合。 用于绘制模型的空间信息。例如一个平面方形，就是由4个顶点构成的，一个立体方块则是由至少8个顶点构成。 每个顶点由三维空间中的x、y、z轴坐标数据构成，用于定位三维空间位置。例如: {"x": 8.1851043701171875e+00, "y": 1.1639657974243164e+01, "z": -6.7235745489597321e-02 } 渲染端拿到路面的所有顶点集合，设置给自己创建的Mesh网格的顶点数据。

5. roadmarks 路面元素信息

减速带


```
"roadmarks": {
    "deceleration_strip0": {
      "id": "deceleration_strip0",
      "type": "0500",
      "indices": [ 0,1,2,0,2,3 ],
      "geometry": [
        {"x": 8.1851043701171875e+00, "y": 1.1639657974243164e+01,  "z": -6.7235745489597321e-02 },
```

{"x": 8.1851043701171875e+00, "y": 1.1639657974243164e+01, "z": -6.7235745489597321e-02 },
{"x": 8.1851043701171875e+00, "y": 1.1639657974243164e+01, "z": -6.7235745489597321e-02 },
{"x": 8.1851043701171875e+00, "y": 1.1639657974243164e+01, "z": -6.7235745489597321e-02 }
]
}, ...
}

字段	值类型	说明
id	string	路面元素id，例如：“deceleration_strip0”
type	string	0500：减速带
indices	int array	索引数组。任何三维模型都是由三角面构成的，索引用于描述构成一个模型所用到的三角面需要由哪些坐标点构成。所以索引数组的长度必须是3的倍数。例如：模型包含4个顶点坐标，索引数据是[0,1,2,0,2,3]，说明该模型由两个三角形构成，这两个三角形分别由第1、2、3 和第1、3、4个坐标点所构成。 渲染端拿到路面的所有的索引集合，设置给自己创建的Mesh网格的索引数据。
geometry	object array [{ x: double, y: double, z: double },	构成几何图形的坐标点集合。 用于绘制模型的空间信息。例如一个平面方形，就是由4个顶点构成的，一个立体方块则是由至少8个顶点构成。 每个顶点由三维空间中的x、y、z轴坐标数据构成，用于定位三维空间位置。例如: {"x": 8.1851043701171875e+00, "y": 1.1639657974243164e+01, "z": -6.7235745489597321e-02 }

	...]	渲染测拿到这前四个坐标点，求出它的中心点（加起来除以四），作为减速带的渲染位置。 渲染端拿到所有的顶点集合，设置给自己创建的Mesh网格的顶点数据。

6. end_information 终点信息

包含道路终点，和目标车位id，因目前HAVP不进行全局渲染，因此此信息正常发送预留

```
"end_information": {
    "park_type": 0,
    "end_point": {
        "x": 2.5637408991619641e+00,
        "y": -3.7775165607276705e+01,
        "z": -1.6349703630512709e+00
    },
    "end_park": {
        id: "park130"
    }
}
```

字段	值类型	说明
park_type	int	1：入库 ， 0 :出库
end_point	object	由三维空间中的x、y、z轴坐标数据构成，用于定位三维空间位置

	{
	x: double,
	y: double,
	z: double
	}	

end_park	object	目标车位信息
id	string	目标车位的id

3.3.5 数据量评估

1、实时发送的数据：渲染过程中实时发送

每一个空间中的点 由三个double类型的数（x y z）构成=8Byte*3 = 24Byte

每个3D障碍物由8个点包裹框构成 = 24 * 8 = 192 Byte(先按照8个点评估)

30个障碍物 = 192 * 30 = 5760

10个车位（每个车位四个点） = 960

16米的轨迹线（每米平均3个） = 48个点 = 1152

以上相加 = 7872 Byte

外加一些状态信号（每个障碍物的类型，障碍物的航向角，障碍物的时间戳等），估计整体在10 000Byte 大约10k每帧

2、HAVP路线文件：发送时间，学习完成后发送一次

50-100k-150m地图（和路线的弯曲度，车位类型有关）

3、backtrack路线：发送时间，功能条件检查通过后发送一次给IVI

倒车循迹50m，150个点大约4K

功能	传输数据	总计
APA	实时传输：障碍物（车辆，行人，其他）、车位、规划轨迹线	实时传输：10k每帧
HAVP	实时传输：障碍物（车辆，行人，锥桶、其他）、车位、规划轨迹线 发送一次地图文件：学习完成后	实时传输：10k每帧 地图：50-100k
backtrack	实时传输：箭头轨迹线 发送一次历史轨迹信息：功能条件检查通过后	实时传输：10k每帧 历史轨迹信息：4K

以上数据按照25hz的传输频率计算，按照最极端情况下总共114k/帧的传输数据量

不考虑backtrack历史轨迹信息和HAVP地图文件的大多数场景下持续发送数据，为250K/S的数据量，即0.24MB/s