差速小车综合模型设计及仿真验证

1. 实验目的

本实验以差速转向小车（CarR1Diff）为对象，演示如何在 RflySim 平台中构建并验证一个“控制器 + 动力学”一体化的**综合模型**。在原有差速小车动力学模型的基础上，使用 MATLAB/Simulink 复刻 PX4 Rover 的制导与混控逻辑，将控制器与 6DOF 动力学集成到同一个 `CarR1DiffNoPX4.slx` 文件中，经嵌入式代码生成（Embedded Coder）打包为动态链接库（`.dll` / `.so`），由 CopterSim 加载运行。该模型可在脱离 PX4 固件的情况下独立闭环运行（SIL=1，内部 Simulink 控制器生效），同时保留 HITL/SITL 兼容性（SIL=0 时外部 PWM 直通）。

通过本实验，期望达到以下目的：

- 掌握 RflySim 综合模型“**Simulink 建模 → 嵌入式代码生成 → 打包 DLL → CopterSim 加载仿真**”的完整开发流程；
- 理解 `inSILInts` / `inSILFloats` 控制协议（与四旋翼综合模型完全兼容）的位域含义及各控制模式的指令构造方法；
- 掌握利用 Python 脚本经 UDP 接口向综合模型发送解锁、位置控制、速度控制、速度+偏航角速率、停车等指令，实现小车闭环运动控制；
- 理解差速小车“双层级联 PID（速度环 + 偏航角速率环）+ 差速混控”的控制器架构及其与 PX4 Rover 混控映射的对应关系。

2. 实验要求

- 软件要求：Windows 10及以上版本；RflySim工具链[1]。
- 硬件要求：笔记本/台式电脑1台[2]。

3.实验地址

例程目录：
[安装目录]\RflySimAPIs\4.RflySimModel\3.CustExps\e5_CopterSimSILNoPX4\9.CarR1DiffModel

本实验文件夹下的文件目录结构如下：

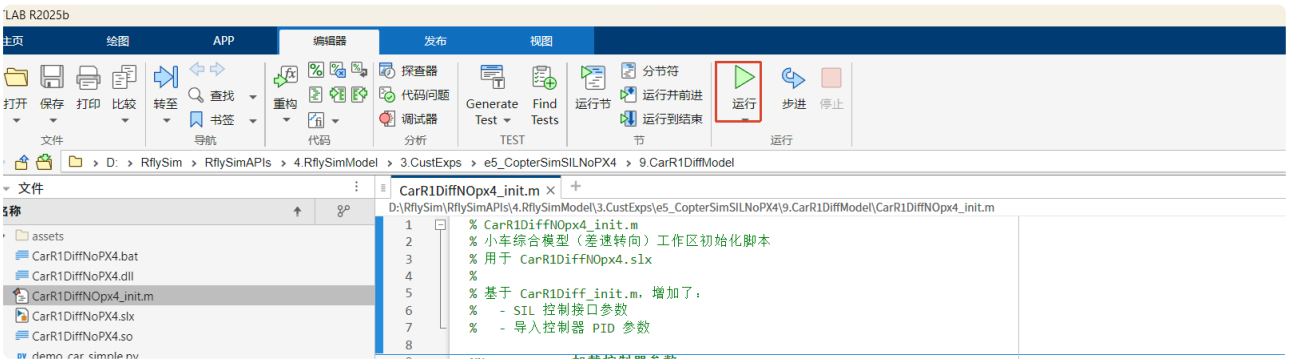
```
9.CarR1DiffModel
├── CarR1DiffNoPX4.slx              # 小车综合模型（控制器 + 6DOF 动力学，Simulink）
├── CarR1DiffNOpx4_init.m           # 车辆参数 + 控制器参数总初始化脚本（打开模型自动执行）
├── Init_control_car.m              # 控制器 PID 参数脚本（被 CarR1DiffNOpx4_init.m 调用）
├── GenerateModelDLLFile.p          # C++ 代码 → DLL/SO 打包脚本
├── CarR1DiffNoPX4.dll              # Windows 下的综合模型动态链接库（代码生成后打包）
├── CarR1DiffNoPX4.so               # Linux 下的综合模型动态链接库（代码生成后打包）
├── CarR1DiffNoPX4.bat              # 启动独立仿真脚本（SimMode=3，SIL=1，无需 PX4）
├── Python38Run.bat                 # RflySim Python 3.8 运行环境脚本
├── demo_car_simple.py              # 小车综合控制测试程序（解锁→位置→速度→偏航角速率→停车）
```

4. 实验内容或步骤

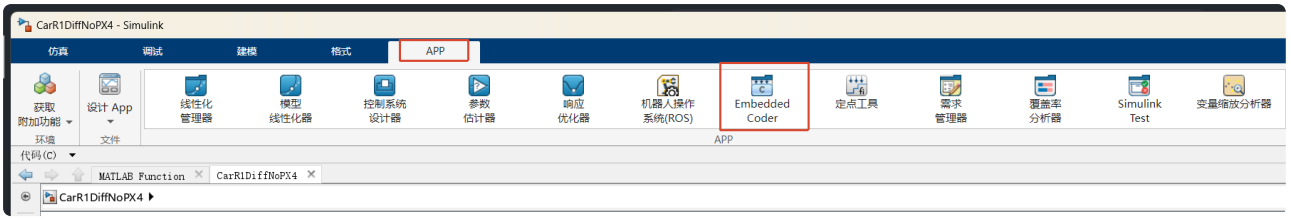
4.1 动态链接库编译

> 注：该步骤需要MATLAB中已经配置了编译生成C++代码得环境，可查阅例程：[RflySim安装目录]\RflySimAPIs\1.RflySimIntro\2.AdvExps\e6.VisualStudioInstall进行自行配置。

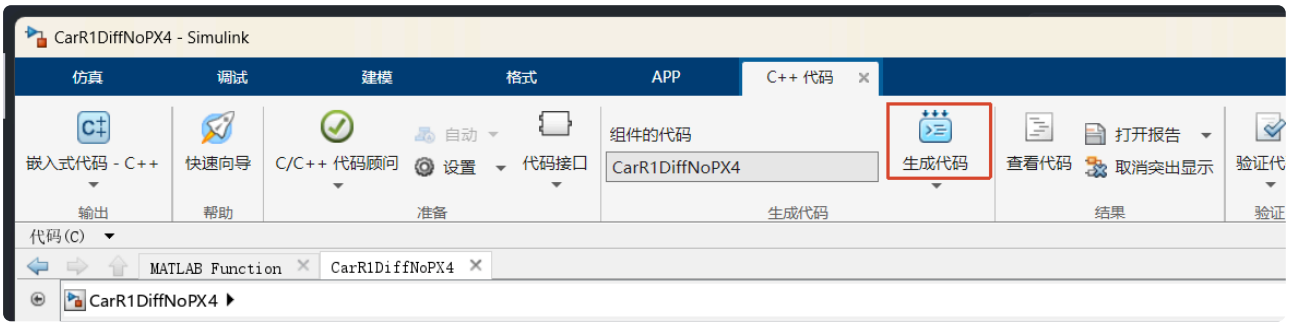
打开MATLAB软件导航到当前实验文件夹得目录下，运行`CarR1DiffNOpx4_init.m`脚本，



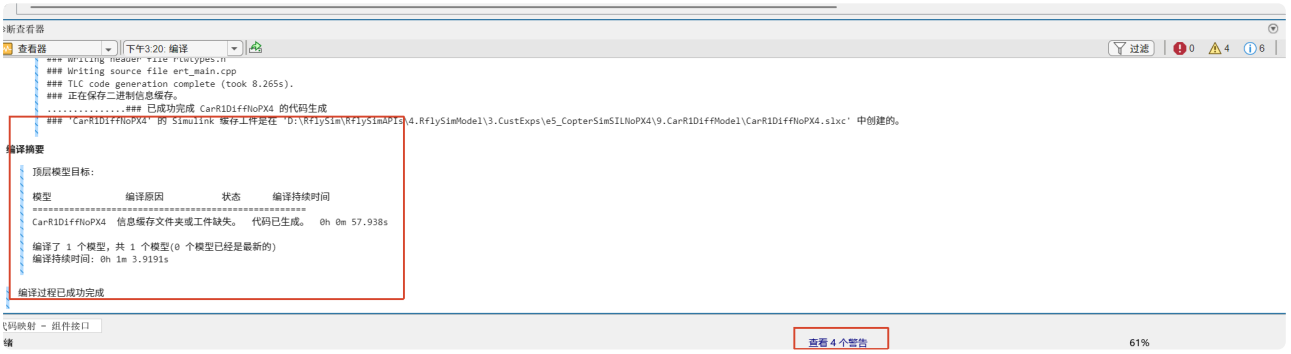
在MATLAB中双击打开`CarR1DiffNoPX4.slx`程序，打开“APP->Embedded Coder”。



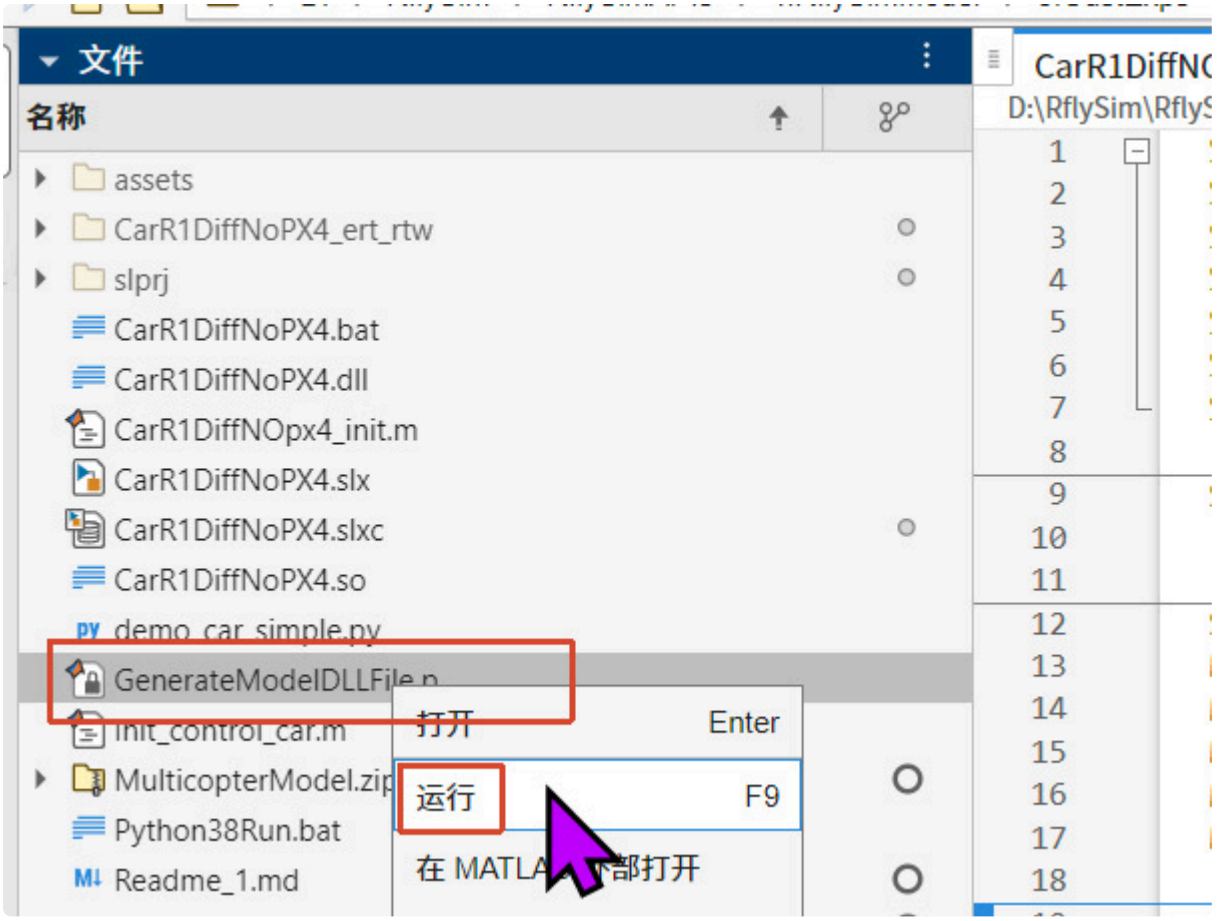
在弹出得标签页中点击“生成代码”。



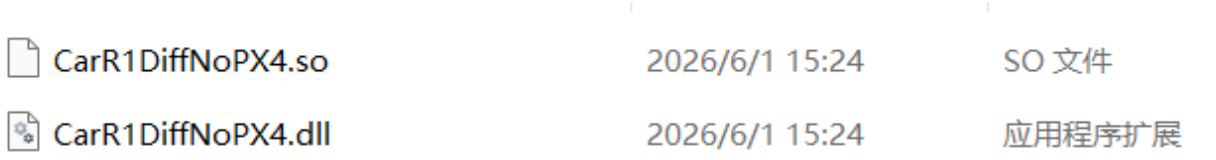
等待编译完成。



返回MATLAB软件，在MATLAB中运行`GenerateModelDLLFile.p`脚本，生成动态链接库。



该脚本将自动生成两个文件CarR1DiffNoPX4.dll、CarR1DiffNoPX4.so两个文件分别为在Windows和Linux系统下得动链接库。



4.2 仿真环境初始化

双击运行`CarR1DiffNoPX4.bat`文件，在弹出得对话框中输入`1`，等待仿真环境初始化完成。脚本将会启动 1 个 QGC 地面站， 1 个 CopterSim 、1 个 RflySim3D 软件。

4.3 运行控制程序

双击 `Python38Run.bat` 脚本，在弹出得对话框中输入：`python demo_car_simple.py` 运行小车得控制程序。

Python 通过 UDP 30100 端口向 CopterSim 发送 inSILInts/inSILFloats 指令。脚本首先发送 `inSILInts[0]=7`（hasCMD+SIL+Armed）解锁，然后发送位置控制指令。

指令速查表：

模式	inSILInts[0]	inSILInts[1]	说明
解锁	**7**	0	解锁后自动 PosHold
位置控制	**519**	**1**	1 = HasPos
速度控制	**65543**	**65538**	65538 = HasVel + NED
速度+偏航角速率	**65543**	**4114**	4114 = HasVel + HasYawRate + FRD
返航	**2055**	0	返回解锁时记录的 Home 点
停车	**1037**	0	减速停车

5. 关键知识点

5.1 实验整体思路与框架

本实验的核心是“**综合模型 + 外部 Python 控制**”的两段式闭环：

1. **建模与打包阶段**（MATLAB 侧）：在 `CarR1DiffNoPX4.slx` 中将“制导/控制器”与“6DOF 动力学”集成在一起；通过 Embedded Coder 生成 C++ 代码，再用 `GenerateModelDLLFile.p` 打包出 `CarR1DiffNoPX4.dll`（Windows）和 `CarR1DiffNoPX4.so`（Linux）。
2. **仿真运行阶段**（CopterSim + Python 侧）：`CarR1DiffNoPX4.bat` 以 `SimMode=3`、`DLLModel=CarR1DiffNoPX4` 启动 CopterSim，把上一步的 DLL 作为被控对象加载；Python 控制程序通过 UDP 接口把控制指令打包成 `inSILInts` / `inSILFloats` 两个数组发送给 CopterSim，由 DLL 内部的 Simulink 控制器解析并驱动小车运动，RflySim3D 负责三维可视化。

整体数据流如下：

```
Python (demo_car_simple.py)
   │  inSILInts[8] / inSILFloats[20]
   ▼  UDP 30100
CopterSim ── 加载 ──► CarR1DiffNoPX4.dll
                          │  GuidanceCore（指令解析 + 状态机 + 制导律）
                          │      ├─ 速度环 PID  → throttle
                          │      └─ 偏航角速率环 PID → differential
                          │  DiffMixer（差速混控）→ PWM_L / PWM_R
                          ▼  6DOF 动力学 + 地面接触
                       车辆状态 ──► RflySim3D 三维显示
```

其中 `SIL` 标志位决定控制来源：`SIL=1` 时内部 Simulink 控制器生效（本实验的独立仿真模式）；`SIL=0` 时外部 PWM 直通，用于 HITL/SITL。

5.2 inSILInts / inSILFloats 控制协议

综合模型的控制接口与四旋翼综合模型完全兼容，由两个数组承载：

`inSILInts[0]` 为 **Vehicle Command 位图**（按位置位后相加）：

bit0: hasCMD	bit1: SIL	bit2: Armed	bit8: Takeoff	bit9: Position	bit10: Land	bit11: Return	bit16: OffboardPos
有新命令	内部仿真控制	解锁	解锁起步	定点	停车	返航	外部位置指令

例如解锁指令 `7 = 1(hasCMD) + 2(SIL) + 4(Armed)`；位置控制 `519 = 7 + 512(bit9 Position)`；速度/外部控制 `65543 = 7 + 65536(bit16 OffboardPos)`；停车 `1037 = 1 + 8 + 1024(bit10 Land)`；返航 `2055 = 7 + 2048(bit11 Return)`。

`inSILInts[1]` 为 **Offboard 控制类型标志**（同样按位相加）：

值	含义	使用的 inSILFloats
1	HasPos — 位置控制	[0-1] = posX, posY（相对 Home 偏移，NED m）
2	HasVel — 速度控制	[3-4] = velX, velY（NED m/s）

值	含义	使用的 inSILFloats
8	HasYaw — 偏航角控制	[11] = yaw（rad）
16	HasYawRate — 偏航角速率控制	[14] = yawRate（rad/s）

> 联合控制时数值相加，如 HasVel + HasYawRate = 18；`65538 = 65536 + 2`、`4114 = 4096 + 16 + 2` 中的高位为坐标系标志（NED/FRD），对小车而言被忽略，真正生效的是 HasVel / HasYawRate 低位。所有角度、角速率单位均为**弧度**而非度。

5.3 控制程序 demo_car_simple.py 解析

该程序是本实验的控制入口，演示了“解锁 → 位置控制 → 速度控制 → 速度+偏航角速率 → 停车”的完整流程。下面按模块解析关键代码。

（1）导入 API 与初始化

程序通过把 RflySimSDK 的 `ctrl` 目录加入 `sys.path`，导入 `DllSimCtrlAPI` 控制接口类，并以 `CopterID=1` 创建实例（对应仿真中 1 号小车）。`CTRL_FREQ=30` 表示以 30 Hz 的固定频率持续发送指令——综合模型需要外部持续“喂”指令，而非发送一次即可。

```
sys.path.insert(0, r'C:\PX4PSP\RflySimAPIs\RflySimSDK\ctrl')
from DllSimCtrlAPI import DllSimCtrlAPI

CTRL_FREQ = 30
DT = 1.0 / CTRL_FREQ
...
dll = DllSimCtrlAPI(CopterID=1)
```

（2）底层指令封装 sendCarCmd

这是理解整个协议的关键函数。它把高层意图翻译成底层的 `inSILInts[8]` 与 `inSILFloats[20]` 数组：`ints0` 写入 `inSILInts[0]`（Command 位图），`ints1` 写入 `inSILInts[1]`（控制类型）；期望的位置/速度/偏航量则按协议约定的索引填入 `inSILFloats`（注意 MATLAB 1-index 与 Python 0-index 的换算），最后调用 `dll.sendSILIntFloat()` 发送。

```python
def sendCarCmd(ints0, ints1, posXY=None, velXY=None, yaw=None, yawRate=None):
    inSILInts = [0] * 8
    inSILFloats = [0.0] * 20
    inSILInts[0] = int(ints0)
    inSILInts[1] = int(ints1)

    if posXY is not None:
        inSILFloats[0] = float(posXY[0])
        inSILFloats[1] = float(posXY[1])
    if velXY is not None:
        inSILFloats[3] = float(velXY[0])
        inSILFloats[4] = float(velXY[1])
    if yaw is not None:
        inSILFloats[11] = float(yaw)
    if yawRate is not None:
        inSILFloats[14] = float(yawRate)

    dll.sendSILIntFloat(inSILInts, inSILFloats)
```

（3）阶段 0：解锁

发送 `ints0=7`（hasCMD+SIL+Armed）、`ints1=0`（无控制类型）持续 2 秒。小车没有“起飞”概念，解锁后自动进入位置保持（PosHold）原地待命。循环中用 `next_tick += DT` 配合 `sleep_until()` 保证严格的 30 Hz 节拍。

```python
while time.time() - phase_start < 2.0:
    next_tick += DT
    sleep_until(next_tick)
    sendCarCmd(ints0=7, ints1=0)
```

（4）阶段 1：位置控制 → (10, 5)

`ints0=519`（在解锁基础上叠加 Position 位）、`ints1=1`（HasPos），目标位置写入 `inSILFloats[0-1]`。目标 = **解锁时记录的 Home 点 + 偏移量**，因此 `(10, 5)` 表示相对起点向北 10 m、向东 5 m。

```python
while time.time() - phase_start < 10.0:
    next_tick += DT
    sleep_until(next_tick)
    sendCarCmd(ints0=519, ints1=1, posXY=[POS_X, POS_Y])
```

（5）阶段 2：速度控制 vx=2, vy=0

这里改用 API 封装好的 `sendDllVelNEDNoYaw()`，内部等效于 `ints0=65543`、`ints1=65538`（HasVel+NED），向 NED 北向给定 2 m/s 速度。小车控制器把 NED 速度矢量解算为前向速度 + 航向对准。

```
while time.time() - phase_start < 5.0:
    next_tick += DT
    sleep_until(next_tick)
    dll.sendDllVelNEDNoYaw(vx=VEL_X, vy=VEL_Y, vz=0)
```

（6）阶段 3：速度 + 偏航角速率（画圆）

调用 `sendDllVelFRD()`，等效于 `ints0=65543`、`ints1=4114`（HasVel+HasYawRate+FRD）。此时控制器进入“直接控制”：前向速度 `V_des=vx`、偏航角速率 `r_des=yawRate` 直接下发。以 1 m/s 前进 + 0.5 rad/s 持续右转，小车画出弧线/圆周轨迹。

```
while time.time() - phase_start < 10.0:
    next_tick += DT
    sleep_until(next_tick)
    dll.sendDllVelFRD(vx=SPEED_FWD, vy=0, vz=0, yawRate=YAWRATE)
```

（7）停车

发送 `ints0=1037`（hasCMD+SIL+Armed+Land）持续 3 秒，小车以恒减速度 ramp down 至 0 后自动 Disarm。

```
while time.time() - phase_start < 3.0:
    next_tick += DT
    sleep_until(next_tick)
    sendCarCmd(ints0=1037, ints1=0)
```

5.4 控制器架构与差速混控

与四旋翼的四层级联 PID 不同，小车采用**双层级联**结构：

```
位置控制(hasPos):            位置误差 → L1 制导/P 控制 → V_des, r_des
速度控制(hasVel):            NED 速度矢量 → 前向速度 V_des + 航向对准 → r_des
直接控制(hasVel+hasYawRate): V_des = vx,  r_des = yawRate

V_des → 速度环 PID    → throttle（油门）
r_des → 偏航角速率环 PID → differential（差速量）
throttle, differential → DiffMixer → PWM_L / PWM_R → 电机
```

差速混控 `DiffMixer` 匹配 PX4 `rover_diff_sitl.main.mix`：

```
Motor1(左前) = thrust + steering
Motor2(左后) = thrust + steering
Motor3(右前) = thrust - steering
Motor4(右后) = thrust - steering
```

输出范围为 `[-1, 1]`，与 PX4 `HIL_ACTUATOR_CONTROLS` 一致：1.0 全速前进、0.0 停止、-1.0 全速倒车。

5.5 控制器参数（Init_control_car.m）

控制器的 PID 与限幅参数在 `Init_control_car.m` 中集中定义，修改后需重新编译 DLL 才能生效：

参数	默认值	说明
`CAR_SPEED_P` / `CAR_SPEED_I`	1.0 / 0.2	速度环 PID 比例 / 积分增益
`CAR_YAWRATE_P` / `CAR_YAWRATE_I`	1.0 / 0.05	偏航角速率环 PID 比例 / 积分增益
`CAR_K_HEADING`	2.0	速度模式下航向对准 P 增益
`CAR_L1_DISTANCE`	3.0 m	L1 制导前视距离
`CAR_CRUISE_SPEED`	2.0 m/s	默认巡航速度
`CAR_MAX_SPEED`	5.0 m/s	最大前向速度
`CAR_MAX_YAWRATE`	π/2 rad/s	最大偏航角速率（90°/s）
`CAR_LAND_DECEL`	1.0 m/s^2	停车减速度

6.参考资料

1. RflySim官方文档： https://rflysim.com/doc/zh/
2. RflySim 安装与配置指南： https://rflysim.com/doc/zh/HowToInstall.pdf
3. RflySim 综合模型与通信接口说明（DLL/SO 模型）：见安装目录下 `RflySimAPIs\4.RflySimModel\API.pdf`

4. PX4 Rover 差速混控器映射：见 PX4 固件 `Firmware\ROMFS\px4fmu_common\mixers-sitl\rover_diff_sitl.main.mix`
5. MathWorks Embedded Coder（C/C++ 代码生成）官方文档：https://www.mathworks.com/help/ecoder/

7.常见问题

Q1：运行 Python 控制程序后小车没有任何运动。

A1：通常是指令位图或解锁未生效所致，可按以下顺序排查：① 确认 `inSILInts[0]` 同时包含 SIL 位（含 2）和 Armed 位（含 4），解锁指令应为 `7`；② 确认 CopterSim 已用本实验的 `CarR1DiffNoPX4.dll` 启动（`CarR1DiffNoPX4.bat` 中 `DLLModel=CarR1DiffNoPX4`、`SimMode=3`）；③ 确认模型修改后已重新生成 DLL（Embedded Coder 编译 + `GenerateModelDLLFile.p` 打包）；④ 确认 Python 端持续以约 30 Hz 发送指令，而非只发一次。

Q2：位置控制时小车不沿直线、歪着走或到不了目标点。

A2：多与航向/速度反馈或 Home 点有关：① 检查车辆 yaw（欧拉角第 3 个分量）是否正确反馈到航向控制；② 速度反馈应使用 NED 地速模 `sqrt(ve_x² + ve_y²)`，而非体轴速度，确保转弯时速度测量准确；③ 位置目标 = 解锁时记录的 Home 点 + `inSILFloats` 偏移，若 Home 点记录异常会导致整体偏移，可重新解锁以刷新 Home 点。

Q3：小车启动/加速过猛，或停车不平稳。

A3：综合模型内置了 `V_des` 速率限制（约 2 m/s^2）。若仍觉过猛，可在 `Init_control_car.m` 中调低 `CAR_SPEED_P`（速度环比例增益）或 `CAR_MAX_SPEED`；停车减速度由 `CAR_LAND_DECEL` 控制。修改参数后需重新编译并重新生成 DLL 才能生效。

Q4：MATLAB 中编译 / 生成 DLL 报错。

A4：① 确认已按要求配置好 C++ 代码生成环境（Embedded Coder + 受支持的编译器），可参考 VisualStudio 安装配置例程；② 确认模型 InitFcn 回调已设置为 `run('CarR1DiffNOpx4_init.m')`，否则工作区参数缺失会导致编译失败；③ 将 RflySim 平台更新至最新版本后重试。

---

1. https://rflysim.com/ ↩
2. 推荐配置请见： https://rflysim.com/doc/zh/HowToInstall.pdf ↩