

2025无人系统具身智能算法挑战赛---四足机器狗场景应用挑战赛使用手册



介绍

本手册专为"2025无人系统具身智能算法挑战赛"中的四足机器狗场景应用挑战赛参赛队伍设计，提供完整的大模型-四足机器狗协同开发指导手册。手册围绕"视觉感知-决策控制-机械狗执行"的技术闭环，帮助参赛者快速构建基于九格大模型的四足机器狗控制系统。

本手册采用"理论→工具→实践"的递进式设计，助力参赛团队快速实现"语言指令→场景理解→动作执行"的智能四足机器狗控制闭环，为大赛竞技提供坚实的技术支撑。

1	# 2025无人系统具身智能算法挑战赛 使用手册限制条款
2	
3	© 2025 无人系统具身智能算法挑战赛组委会 版权所有
4	
5	**使用授权范围：**
6	本手册仅授权以下主体在赛事期间使用：
7	
8	1. 经组委会认证的参赛团队队员
9	2. 赛事官方裁判及技术监督人员
10	3. 组委会授权的培训导师
11	
12	**严格禁止事项：**
13	
14	- 任何形式的商业性使用或二次销售
15	- 向非参赛组织或个人进行传播
16	- 改编后用于其他赛事或商业项目
17	- 在线平台/文库的公开传播
18	
19	**使用约束：**
20	手册所含技术方案、赛事规则及数据参数等知识产权归组委会所有，参赛者仅限：
21	
22	- 赛事筹备期用于技术方案设计参考
23	- 正式竞赛期间作为操作规范依据
24	- 赛后总结阶段用于技术复盘分析
25	
26	**免责声明：**
27	本手册内容按"现有状态"提供：
28	组委会不承担因手册信息导致的技术方案偏差责任
29	不保证所含方案满足特定技术场景的实施需求
30	对使用后果不承担直接或间接法律责任
31	
32	*违反本条款者组委会有权取消参赛资格并追究法律责任*

目录

2025无人系统具身智能算法挑战赛---四足机器狗场景应用挑战赛使用手册

介绍

目录

（一）环境配置

1.ROS安装

1.1 安装

1.2 测试ROS

1.3 安装依赖

2.Anaconda安装

2.1 官网下载

2.2 安装

2.3 环境配置

2.4 验证环境

3.九格大模型环境配置

3.1 创建虚拟环境inference

3.2 环境配置

3.3 测试模型

4.yolov8安装

4.1 安装yolov8

5.Isaac Sim安装

5.1 官网下载

5.2 环境配置

（二）快速启动

1.下载仿真代码

2.运行四足机器狗仿真

（三）大模型接口

1.模型加载接口

2.推理调用接口

（四）四足机器狗接口

1.运动控制接口

2.传感器接口

（五）机械臂接口

1.运动控制接口

2.相机接口

3.夹爪接口

4.规划控制接口

(一) 环境配置

本次大赛推荐使用的操作系统为ubuntu20.04、显卡显存22G及以上

1.ROS安装

1.1 安装

官网安装: <https://www.ros.org/blog/getting-started/>

同时支持一键安装, 极大提升安全效率与便捷性

一键安装: 打开终端, 输入下面命令, 进行一键安装, 推荐使用ROS Noetic版本

```
1 | wget http://fishros.com/install -O fishros && . Fishros
```

1.2 测试ROS

需求: 判断是否能使用键盘控制小乌龟移动

操作:

开启终端一:

```
1 | #启动roscore核心
2 | roscore
```

开启终端二:

```
1 | #出现图形化界面
2 | rosrn turtlesim turtlesim_node
```

开启终端三:

```
1 | #启动键盘控制节点
2 | rosrn turtlesim turtle_teleop_key
```

注意: 光标必须聚焦在键盘控制窗口, 否则不能控制乌龟运动

1.3 安装依赖

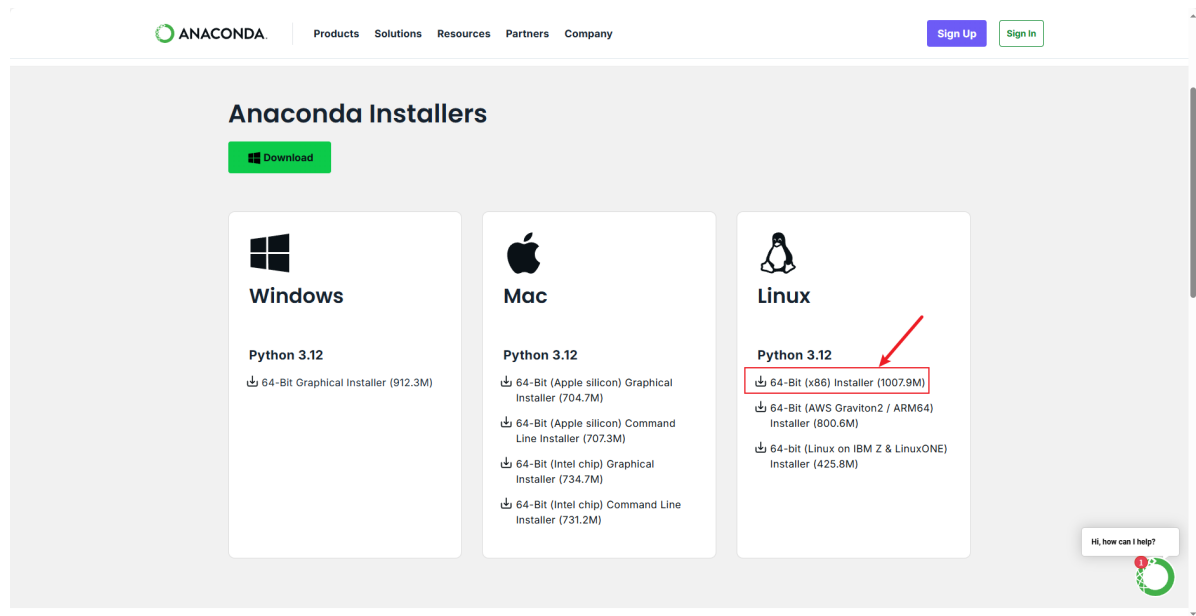
```
1 | sudo apt install nlohmann-json3-dev
2 | sudo apt install ros-noetic-move-base*
3 | sudo apt install ros-noetic-amcl*
4 | sudo apt install ros-noetic-map-server*
5 | sudo apt install ros-noetic-teb-local-planner*
```

2.Anaconda安装

2.1 官网下载

通过官网下载安装包，官网地址：[Download Now | Anaconda](https://www.anaconda.com/download/)

具体版本根据自己的电脑环境进行选择



2.2 安装

找到安装包的位置，点击右键选择在终端中打开，输入

```
1 | sh 安装包名
```

回车出现 `Please answer yes or no` 选项后，输入 `yes`

回车出现 `You can undo this by running 'conda init --reverse $SHELL' ? [yes|no]` 选项后，输入 `no`

2.3 环境配置

进入主目录，通过 `Ctrl+H` 命令显示隐藏文件，找到 `.bashrc` 文件，打开后在最后一行添加

```
1 | source ~/anaconda3/bin/activate
```

2.4 验证环境

关掉前面的终端，开一个新的终端，当 `(bash)` 出现在命令提示符前面说明成功安装了环境。

3. 九格大模型环境配置

3.1 创建虚拟环境inference

先启动anaconda环境，输入命令，创建虚拟环境 `inference`（环境名）

```
1 | conda create -n inference python=3.10
```

```
q@q:~$ source .bashrc
(base) q@q:~$ conda create -n inference python=3.10
2 channel Terms of Service accepted
Channels:
 - defaults
Platform: linux-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

environment location: /home/q/anaconda3/envs/inference

added / updated specs:
 - python=3.10

The following packages will be downloaded:
```

package	build	
openssl-3.0.17	h5eee18b_0	5.2 MB
python-3.10.18	h1a3bd86_0	26.5 MB
setuptools-78.1.1	py310h06a4308_0	1.7 MB

回车出现 `Proceed ([y] / n)` 选项后，输入 y

完成进程后，输入下面命令进入之前创建的虚拟环境

```
1 | conda activate inference
```

当 (inference) 出现在命令提示符前面说明成功创建了虚拟环境。

3.2 官网下载大模型

官网下载：可在 <https://thunlp-model.oss-cn-wulanchabu.aliyuncs.com/9G4B.tar> 下载4B模型。

百度网盘：https://pan.baidu.com/s/1o8GMcEl_SyC2euaiFVKstQ?pwd=8888 提取码: 8888

阿里云网盘：<https://www.alipan.com/s/be42mHqfrWX> 提取码: 1zp2

3.2 环境配置

找到模型文件所在目录 `/model/Embodied/` 点击右键选择在终端打开，输入下面命令

```
1 | #先进入上面创建的虚拟环境
2 | conda activate inference
3 | #下载
4 | pip install -r requirements.txt
5 | #如果下载比较慢，添加清华源
6 | pip install -r requirements.txt -i https://pypi.tuna.tsinghua.edu.cn/simple
```

3.3 测试模型

进入 `/model/Embodied/inference/test.py`，修改 `model_file` 路径为选手自己电脑路径（图中24行）

```
test.py
~/model/Embodied/inference
保存(S)

1 """
2 Name test
3 Date 2025/3/5 13:39
4 Version 1.0
5 TODO:推理
6 """
7
8 import torch
9 from PIL import Image
10 from transformers import AutoModel, AutoTokenizer
11
12 if __name__ == '__main__':
13     prompt = f"""### 背景 ###
14     您需要对图片中的内容进行识别。
15     ### 输出格式 ###
16     您的输出由以下两部分组成，确保您的输出包含这两部分：
17     ### 思考 ###
18     考虑饮料外的标识，辨别饮料的种类，饮料容器。并且识别饮料为'有糖'或者'无糖'，给出你的思考过程。
19     ### 识别结果 ###
20     若图中出现了饮料，请以json形式从左到右对他们进行描述，包括饮料：种类，是否有糖，饮料容器。
21     """
22
23
24     model_file = '/home/q/model/FM9G4B-V'
25     model = AutoModel.from_pretrained(model_file, trust_remote_code=True,
26                                       attn_implementation='sdpa', torch_dtype=torch.bfloat16)
27     model = model.eval().cuda()
28     tokenizer = AutoTokenizer.from_pretrained(model_file, trust_remote_code=True)
29
30     image = Image.open('step.jpg').convert('RGB')
31
32     msgs = [{'role': 'user', 'content': [image, prompt]]]
```

进入 /model/Embodied/inference 运行下面代码

```
1 python3 test.py
```

```
(inference) qm: /model/Embodied/inference$ python3 test.py
self.grid_size: 8
self.num_queries: 64
self.embed_dim: 2560
self.num_heads: 16
self.adaptive: True
self.max_size: (70, 70)
self.kv_dim: 112
/home/q/anaconda3/envs/inference/lib/python3.10/site-packages/transformers/models/auto/image_processing_auto.py:513: FutureWarning: The ImageProcessorClass argument is deprecated and will be removed in v4.42. Please use 'slow_image_processor_class', or 'fast_image_processor_class' instead
warnings.warn(
The 'seen_tokens' attribute is deprecated and will be removed in v4.41. Use the 'cache_position' model input instead.
=====
### 思考 ###
1. **饮料种类识别**：
   - 从图片中可以看到三种不同的饮料罐，分别有红色、绿色和黑色的罐子。
   - 红色罐子上印有“可口可乐”的标志，绿色罐子上印有“雪碧”的标志，黑色罐子上印有“百事可乐”的标志。
2. **是否有糖**：
   - 可口可乐通常是含糖版本，雪碧和百事可乐通常是含糖版本。
3. **饮料容器**：
   - 所有三种饮料都是铝制罐装。

### 识别结果 ###
```json
[
 {
 "种类": "可口可乐",
 "是否有糖": "有糖",
 "饮料容器": "铝制罐"
 },
 {
 "种类": "雪碧",
 "是否有糖": "有糖",
 "饮料容器": "铝制罐"
 },
 {
 "种类": "百事可乐",
 "是否有糖": "有糖",
 "饮料容器": "铝制罐"
 }
]
```

## 4.yolov8安装

### 4.1 安装yolov8

打开终端

```
1 #创建 yolov8 虚拟环境
2 conda create -n yolov8 python=3.8
3 #下载nvidia-cuda-toolkit
4 sudo apt install nvidia-cuda-toolkit
```

```

5 #查看conda版本
6 nvidia-smi
7 #安装pytorch, 到官网寻找相关的版本
8 https://pytorch.org/ 选择对应版本
9 #进入yolov8虚拟环境
10 conda activate yolov8
11 #选择对应pytorch版本的对应命令进行下载
12 #下载YOLOv8官方代码库
13 git clone https://github.com/ultralytics/ultralytics.git
14 #安装YOLOv8所有依赖
15 pip install -r requirements.txt

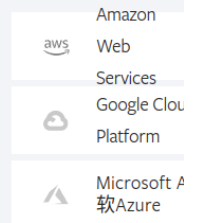
```

**NOTE:** Latest PyTorch requires Python 3.8 or later. For more details, see Python section below.

注意:最新的PyTorch需要Python 3.8或更高版本。有关详细信息, 请参阅下面的Python部分。

PyTorch Build PyTorch构建	Stable (2.2.1) 稳定的(2.2.1)		Preview (Nightly) 预览(夜间)	
Your OS 你的	Linux	Mac	Windows 窗户。	
Package 包	Conda 康达	Pip 皮普	LibTorch Lib火炬	Source 源
Language 语言	Python		C++ / Java C / Java	
Compute Platform 计算平台	CUDA 11.8 CUDA 11.8 (英语)	CUDA 12.1	ROCm 5.7	CPU
Run this Command: 运行命令:	<p><code>pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu118</code></p> <p>Pip3安装torch torchvision torchaudio——index-url https://download.pytorch.org/whl/cu118</p>			

Previous versions of PyTorch >  
PyTorch的旧版本



## 5.Isaac Sim安装

### 5.1 官网下载

Isaac Sim官方网站: <https://docs.isaacsim.omniverse.nvidia.com/4.5.0/installation/download.html>



4.5.0

By downloading or using the NVIDIA Isaac Sim WebRTC Streaming Client, you agree to the [NVIDIA Isaac Sim WebRTC Streaming Client License Agreement](#).

Table of Contents

Isaac Sim

What Is Isaac Sim?

Release Notes

Installation

Isaac Sim Requirements

Download Isaac Sim

Workstation Installation

Container Installation

Cloud Deployment

Livestream Clients

Python Environment Installation

ROS and ROS 2 Installation

Setup Tips

Linux Troubleshooting

Help & FAQ

Getting Started

Getting Started Tutorials

Workflows

Examples

Assets and Robots

Reference Architecture

Isaac Sim Applications

Latest Release

Latest Release

Name	Version	Release Date	Links
Isaac Sim	4.5.0	January 2025	<a href="#">Linux (6.7 GB)</a> <a href="#">Windows (6.5 GB)</a>
Isaac Sim Compatibility Checker	4.5.0	January 2025	<a href="#">Linux (249.8 MB)</a> <a href="#">Windows (246.6 MB)</a>
Isaac Sim WebRTC Streaming Client	1.0.6	January 2025	<a href="#">Linux (214.8 MB)</a> <a href="#">Windows (168.1 MB)</a> <a href="#">macOS x86_64 (208.2 MB)</a> <a href="#">macOS arm64 (197.0 MB)</a>
Isaac Sim Assets	4.5.0	January 2025	<a href="#">Pack 1 of 3 (33.5 GB)</a> <a href="#">Pack 2 of 3 (28.6 GB)</a> <a href="#">Pack 3 of 3 (24.1 GB)</a>
Omniverse Streaming Client	103.1.1	April 2022	<a href="#">Linux (385.5 MB)</a> <a href="#">Windows (102.0 MB)</a>
[DEPRECATED]			<a href="#">Windows (102.0 MB)</a>

On this page

Latest Release

Download Archive

选择Isaac Sim 4.5.0版本，点击Linux下载后，将下载好的压缩包进行解压，解压后的文件放在/home的主目录下。

## 5.2 环境配置

```
1 # 编辑 ~/.bashrc 文件的命令
2 sudo gedit ~/.bashrc
3 # 在文件末尾添加以下内容（替换 /home/your_username/isaacsim/python.sh 为您的实际路径）
4 export ISAACSIM_PATH="${HOME}/isaacsim"
5 export ISAACSIM_PYTHON_EXE="/home/your_username/isaacsim/python.sh"
```

按 Ctrl+S 保存

```
1 # 刷新环境
2 source .bashrc
```

## （二）快速启动

### 1. 下载仿真代码

```
1 仿真代码与本手册位于同一目录下
```

将EAICON源代码拷贝复制到/home/your\_username下

### 2. 运行四足机器狗仿真

询问方式：开始导航/开始巡线

操作：

开启终端一：

```
1 #启动roscore核心
2 roscore
```

开启终端二：

```
1 #进入工作空间
2 cd Go2
3 sudo cp EAICON/Content/Go2/lidar/L1.json /home/q/isaac-
sim/exts/isaacsim.sensors.rtx/data/lidar_configs/NVIDIA/
4 #启动仿真环境
5 sh run_go2_sim.sh
```

开启终端三：

```
1 #进入工作空间
2 cd Go2
3 #退出当前环境，返回默认环境
4 conda deactivate
5 #运行 launch文件
6 roslaunch go2_slam go2_bring.launch
```

开启终端四：

```
1 #进入工作空间
2 cd Go2
3 #激活 inference 环境
4 conda activate inference
5 #运行 Python文件
6 rosrn go2_scale go2_scale.py
```

**注意：**每次操作前都需要在终端输入 `source ./devel/setup.bash` 刷新环境，否则有概率出现莫名其妙的报错。

## （三）大模型接口

该版本通用大模型参数量为80亿，具有高效训练与推理和高效适配与部署的技术特点，具备文本问答、文本分类、机器翻译、文本摘要等自然语言处理能力。九格百亿级通用基础大模型的参数量为8B（80亿）。具体的模型训练、推理等内容见：[quick start](#)

本表聚焦“九格”接口设计中与大模型相关的部分，将其抽象为模型加载、推理调用两大核心单元，具体接口列表如下：

接口名称	描述	调用方式	输入参数	输出	异常处理
模型加载接口	从本地或远程路径加载大模型及其Tokenizer	<code>AutoModel.from_pretrained</code> <code>AutoTokenizer.from_pretrained</code>	- <code>model_file</code> (字符串): 权重与配置存放路径 - <code>trust_remote_code</code> (布尔): 是否信任远程自定义代码	- <code>self.model</code> (模型对象) - <code>self.tokenizer</code> (分词器对象)	捕获并 <code>rospy.logerr</code> , 加载失败时置空并退出订阅流程
推理调用接口	根据输入图像与文本 Prompt, 调用模型生成推理结果	<code>model.chat(image=None, msgs, tokenizer=self.tokenizer)</code>	- <code>msgs</code> (列表): 每项为字典		

## 1.模型加载接口

```
1 self.model = AutoModel.from_pretrained(
2 model_file: str,
3 trust_remote_code: bool = True,
4 attn_implementation: str = 'sdpa',
5 torch_dtype: torch.dtype = torch.bfloat16
6)
7 self.tokenizer = AutoTokenizer.from_pretrained(
8 model_file: str,
9 trust_remote_code: bool = True
10)
```

### 参数说明

**model\_file**: 本地或远程路径，预训练模型权重与配置所在目录。

**trust\_remote\_code**: 是否信任并执行仓库中的自定义代码。

**attn\_implementation** 与 **torch\_dtype**: 可选优化参数。

### 输出说明

**self.model**: 已加载并 `eval()` 的模型实例，已切换到 CUDA（若可用）。

**self.tokenizer**: 对应的分词器，用于构造输入tokens。

### 异常处理

捕获任何加载错误，调用 `rospy.logerr("模型加载失败: %s", e)` 并将`self.model/self.tokenizer`置为 `None`，后续流程根据空值判断跳过订阅与推理。

## 2.推理调用接口

```
1 model_res = self.model.chat(
2 image=None,
3 msgs: List[Dict[str, Any]],
4 tokenizer=self.tokenizer
5)
```

### 输入说明

**msgs**: 长度可变的消息列表，每条消息格式为：

```
1 {
2 'role': 'user',
3 'content': [pil_image: PIL.Image.Image, prompt: str]
4 }
```

**pil\_image**: 从最新 ROS 彩色帧转换而来。

**prompt**: 用户或上层脚本动态输入的文本提示。

输出说明

**model\_res**: 大模型返回的推理结果，可为文本、结构化数据或二次封装，随后转换为字符串发布。

调用时机

在 self.new\_bbox\_request == True 且最新图像帧已获取时触发。

异常处理

推理过程中捕获任何异常并调用 rospy.logerr("调用大模型进行处理时出错: %s", e)，当前帧推理终止，不影响后续请求。

## （四）四足机器狗接口

### 1.运动控制接口

话题名称	消息类型	发布/订阅	功能说明
/cmd_vel	geometry_msgs/Twist	订阅	接收机器人运动控制指令（线速度和角速度）
/cmd_vel_x	std_msgs/Float32	发布	发布处理后的x方向线速度分量
/cmd_vel_y	std_msgs/Float32	发布	发布处理后的y方向线速度分量
/cmd_vel_yaw	std_msgs/Float32	发布	发布处理后的绕z轴的角速度分量
/odom	nav_msgs/Odometry	发布	发布机器人里程计信息（位置、姿态和速度）
/tf	tf2_msgs/TFMessage	发布	广播各坐标系之间的变换（坐标变换树）

### 2.传感器接口

话题名称	消息类型	发布/订阅	功能说明
/camera/front	sensor_msgs/Image	订阅	前视摄像头原始图像 用于机械狗的巡线
/camera/global	sensor_msgs/Image	订阅	全局摄像头
/lidar/points	sensor_msgs/PointCloud2	发布	发布三维点云数据
/scan	ensor_msgs/LaserScan	发布	发布二维激光扫描数据
/imu/data	sensor_msgs/Imu	发布	发布惯性测量单元数据，包括线加速度、角速度和姿态

## （五）机械臂接口

### 1.运动控制接口

话题名称	消息类型	发布/订阅	功能说明
<code>end_effector/pose</code>	<code>geometry_msgs/PoseStamped</code>	订阅	末端执行器当前位姿
<code>/end_effector/target_pose</code>	<code>geometry_msgs/PoseStamped</code>	发布	末端执行器目标位姿

### 2.相机接口

话题名称	消息类型	发布/订阅	功能说明
/camera/wrist/info	sensor_msgs/CameraInfo	订阅	获取腕部相机内参 (焦距fx/fy, 光心cx/cy)
/camera/wrist/depth	sensor_msgs/Image	订阅	深度相机图像，像素值为深度（单位：米），用于场景深度感知
/camera/wrist/rgb	sensor_msgs/Image	订阅	RGB 彩色图像（编码： <code>rgb8</code> ），用于视觉检测、语义分割或显示画面

### 3.夹爪接口

话题名称	消息类型	发布/订阅	功能说明
/gripper/target	std_msgs/Float32	发布	控制夹爪开合宽度 (单位：米)

## 4.规划控制接口

接口名称	函数签名	功能描述	替换示例
<b>PathPlannerInterface</b>	<pre>plan(start: np.ndarray, goal: np.ndarray, env_cfg: Any) → np.ndarray</pre>	从起点 <code>start</code> 到终点 <code>goal</code> ，在环境 <code>env_cfg</code> （碰撞体、关节限位等信息）下生成一条关节空间或笛卡尔空间轨迹，返回形状为 <code>(T, D)</code> 的轨迹点序列。	<code>RRTPlanner</code> ， <code>PRMPlanner</code> ， <code>LVSPPlanner</code>
<b>TrajectoryTrackerInterface</b>	<pre>track(trajecotory: np.ndarray, state: np.ndarray) → np.ndarray</pre>	接收规划器输出的轨迹 <code>trajectory</code> 和当前系统状态 <code>state</code> ，计算下一个执行指令（如关节位置/速度/力矩），返回控制命令 <code>u</code> 。	<code>PIDTracker</code> ， <code>MPCTracker</code> ， <code>ImpedanceTracker</code>
<b>PathPlannerCfg</b>	<pre>python&lt;br&gt;@configclass&lt;br&gt;class PathPlannerCfg:&lt;br&gt;    class_name: str&lt;br&gt;    params: dict = {}&lt;br&gt;</pre>	在配置文件里指定要用的规划器类名和初始化参数，框架会根据 <code>class_name</code> 动态 import 并实例化。	<pre>yaml&lt;br&gt;planner:&lt;br&gt;    class_name:&lt;br&gt;    "RRTPlanner"&lt;br&gt;    params:&lt;br&gt;    step_size: 0.1</pre>
<b>TrajectoryTrackerCfg</b>	<pre>python&lt;br&gt;@configclass&lt;br&gt;class TrajectoryTrackerCfg:&lt;br&gt;    class_name: str&lt;br&gt;    params: dict = {}&lt;br&gt;</pre>	在配置文件里指定要用的跟踪器类名和初始化参数，框架会根据 <code>class_name</code> 动态 import 并实例化。	<pre>yaml&lt;br&gt;tracker:&lt;br&gt;    class_name:&lt;br&gt;    "MPCTracker"&lt;br&gt;    params:&lt;br&gt;    horizon: 20</pre>
<b>PlannerFactory</b>	<pre>get_planner(cfg: PathPlannerCfg) → PathPlannerInterface</pre>	根据 <code>PathPlannerCfg</code> ，动态加载并返回对应的规划器实例。	—
<b>TrackerFactory</b>	<pre>get_tracker(cfg: TrajectoryTrackerCfg) → TrajectoryTrackerInterface</pre>	根据 <code>TrajectoryTrackerCfg</code> ，动态加载并返回对应的跟踪器实例。	—
<b>execute_motion(planner, tracker)</b>	<pre>execute_motion(start, goal, env_cfg)</pre>	<b>封装调用流程：</b> 1. 调用 <code>planner.plan(start, goal, env_cfg)</code> 2. 在仿真或真实机械臂循环中，调用 <code>tracker.track(...)</code> 并下发控制命令	—