智能无人系统控制实践

第二章　实验平台配置

作者：戴训华　中南大学

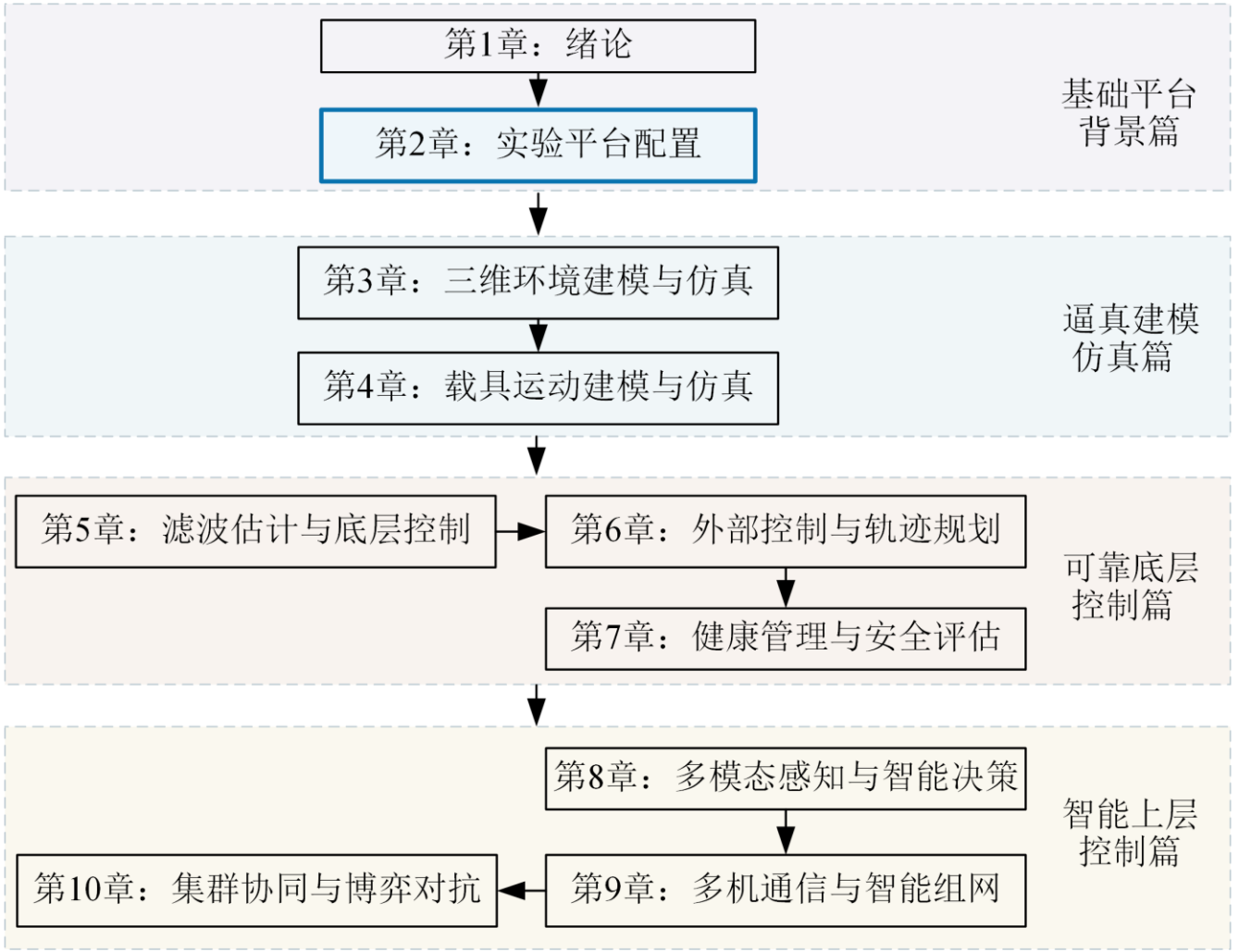
主页：faculty.csu.edu.cn/daixunhua

邮箱：dai.xh@csu.edu.cn

平台官网：rflysim.com

配套教材





教材：戴训华，全权，任雪峰. 智能无人系统控制实践[M]. 北京：电子工业出版社，2026.

本章学习目标

学完第二章能够完成什么

1. 说明主要工具的职责与连接关系
2. 完成平台安装并运行首次仿真
3. 比较仿真模式并识别接口输入输出
4. 复现配套例程并解释一次改动的结果

看懂工具间的数据流，才能判断改动是否真正改善了系统行为。

本章目录

关键工具：从运行环境到仿真闭环

第二章　实验平台配置

2.1　关键工具介绍

2.2　平台安装与首次仿真

2.3　平台架构与开发流程

2.4　API与SDK开发接口

2.5　例程体系与实验路线

2.6　习题与课堂讨论

本节 4 个教学单元

2.1.1 开发基础与运行环境

教材对应：第 2.1.2.2、2.1.2.3、2.1.2.7 节

2.1.2 模型与算法开发

教材对应：第 2.1.2.1、2.1.2.3、2.1.2.7 节

2.1.3 系统组成与通信

教材对应：第 2.1.2.4–2.1.2.6 节

2.1.4 仿真工具与开发路线

教材对应：第 2.1.2.1、2.1.2.8 节

2.1.1 开发基础与运行环境：工具分类

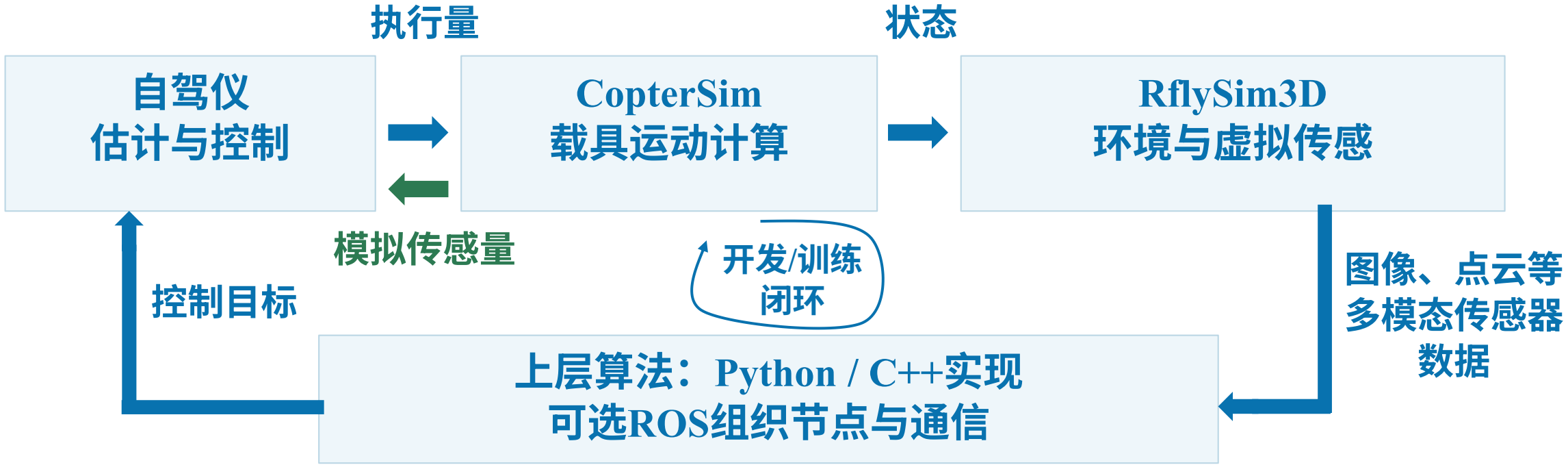
工具链中的五类对象

- **运行环境：** Windows、Ubuntu承载程序；适用于Linux的Windows子系统（WSL）承载Linux机载环境模拟（PX4自驾仪固件编译、ROS开发等）。
- **语言与库：** Python、C/C++编写程序，OpenCV等库提供图像处理等现成功能。
- **开发工具：** MATLAB分析数值与数据，Simulink用模块和信号连接建立系统模型。
- **接口与协作：** 应用程序接口（API）规定功能的调用方式。
机器人操作系统（ROS）是组织模块协作的软件框架。
- **仿真与自驾仪：** RflySim把载具模型、PX4等自驾仪与地面站、上层算法和三维场景连成闭环，用虚拟实验检验系统行为。

运行环境承载程序，算法处理数据，通信把各模块连接成仿真闭环。

2.1.1 开发基础与运行环境：先看一次仿真闭环

RflySim把模型、控制与场景接起来



运动、控制、环境和上层算法通过接口共同形成仿真闭环。

无论是仿真训练环境搭建，还是算法开发，都需要大量工具 → RflySim工具链

2.1.1 开发基础与运行环境：操作系统

Windows与Linux分别承载哪些程序

- **Windows**：运行运动仿真、三维渲染、控制算法与地面站程序，承担仿真组织、显示和交互。
- **Ubuntu**：Linux发行版，可开发、编译并运行配套自驾仪与ROS程序，也用于机载算法部署。
- **跨系统协作**：两种系统中的程序经接口交换数据，各用对应的库与工具链。



Ubuntu：Linux发行版

部署到机载板时：同为Linux，也要匹配处理器架构与驱动。

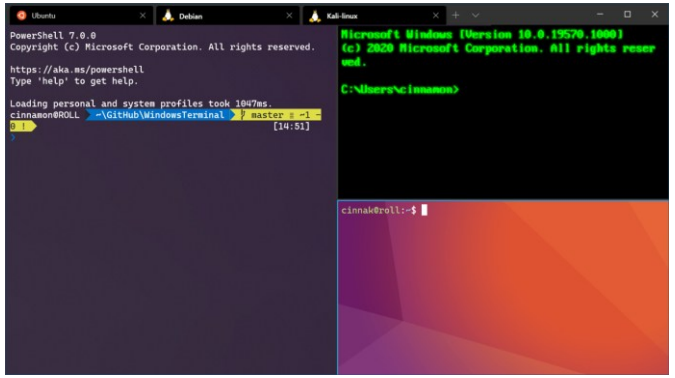
Windows系统与开发文档
https://learn.microsoft.com/zh-cn/windows/

Ubuntu安装与命令行教程
https://ubuntu.com/tutorials

同一台计算机可承载两种运行环境，但程序、依赖和设备架构必须匹配。

2.1.1 开发基础与运行环境：跨系统与环境封装

WSL提供Linux环境，Docker封装应用



同一终端可运行Windows与Linux命令

- **WSL：**在Windows中运行Linux程序。WSL1使用兼容层，WSL2运行Linux内核。
- **Docker：**镜像保存应用及依赖，容器是运行实例；同一环境的容器共享内核。
- **组合关系：**Linux容器可借助WSL2运行；WSL容器（WSLC）2026-06起公开预览。

镜像 → 容器实例

WSL安装与配置
https://learn.microsoft.com/windows/wsl/

Docker镜像与容器入门
https://docs.docker.com/get-started/

WSL承载Linux程序，Docker用镜像复用应用和依赖，两者可组合使用。

2.1.1 开发基础与运行环境：系统与环境补充

处理器架构与WinWSL环境

注：日常使用的开发计算机通常采用 x86-64（也称 AMD64）处理器架构；而手机与机载计算机对功耗要求极为苛刻，因此多采用 ARM64 架构。两者名称中的“64”均指 64 位系统字长。

注：RflySim 工具链内置 WinWSL 编译环境，该环境基于 WSL 1/2 与 Ubuntu 22.04 构建，并支持Docker容器化技术，便于算法向真机迁移。

处理器架构决定可执行文件适配；WSL与Docker承担不同的环境封装。

2.1.1 开发基础与运行环境：选学·跨架构构建

从x86开发机（Windows仿真）到ARM64目标（机载板卡）

开发计算机便于调试，机载板卡受体积、功耗与算力约束。常见x86-64开发机与ARM64板卡的指令集不同，即使同用Ubuntu，程序与依赖库也须面向目标架构构建。

- **Buildx构建：**Docker Buildx可指定目标平台，使用支持ARM64的基础镜像和依赖生成目标镜像。
  - **两条常见路径：**QEMU模拟目标指令，便于执行构建步骤但可能较慢；交叉编译器在开发机生成目标机器码。
  - **板上运行：**程序启动依赖目标系统的库，取图依赖相机驱动；图像输入、算法输出与处理时延共同反映部署效果。

Docker多架构镜像构建：https://docs.docker.com/build/building/multi-platform/

跨架构构建适配机器码，板上运行还依赖系统库与设备驱动。

2.1.1 开发基础与运行环境：命令行基础

终端、CMD与BAT分别负责什么

- **终端窗口：**提供输入和显示区域，可承载CMD、PowerShell或Linux Shell。
- **命令解释器（Shell）：**CMD解释Windows命令，PowerShell与Linux Shell各有语法。
- **批处理脚本：**BAT保存一组CMD命令，可设置目录和参数、启动程序，便于复现实验的启动条件。
- **运行Python：**先用cd进入脚本所在目录，再执行python ShootBall3.py，Shell启动解释器运行脚本。

Windows Terminal使用
https://learn.microsoft.com/windows/terminal/

CMD命令速查（SS64）
https://ss64.com/nt/

终端承载交互，Shell解释启动命令，Python解释器执行算法脚本。

2.1.1 开发基础与运行环境：语言与程序库

用Python组织实验，用库实现具体算法

红球例程中的语言与库

层次	在例程中做什么	可定位的对象
Python主程序	组织采样、判断与循环	ShootBall3.py
数值／视觉库	保存图像、分割目标、计算质心	NumPy、OpenCV
RflySim API	取图像、读状态、发送运动目标	图像采集与控制接口
C/C++及编译库	实现底层控制或高性能运算	固件、驱动、库函数

Python手册：https://docs.python.org/3/　　C++开发文档：https://learn.microsoft.com/cpp/

Python用于算法快速开发与测试，C/C++常用于机载端的性能优化与部署。

2.1.1 开发基础与运行环境：Python入门

Python程序、解释器与程序包

- **语言与解释器：**Python代码写入.py文件，由解释器执行，函数和循环组织实验。
- **程序包：**import引入功能。NumPy处理数组，OpenCV处理图像，平台API连接仿真。
- **开发优势：**交互试验和现成库便于验证算法；耗时运算可调用已编译的实现。
- **环境隔离：**venv隔离包版本；平台例程使用配套解释器。用sys.executable核对路径，python -m pip为当前解释器装包。

Python语法与程序组织
https://docs.python.org/zh-cn/3/tutorial/

NumPy数组入门
https://numpy.org/learn/

解释器执行主程序，程序包提供功能，环境决定实际使用哪一套依赖。

2.1.1 开发基础与运行环境：C/C++与动态库

C与C++：把算法写成程序

思考： 假设小型无人机上机载算力实在有限，你该如何优化取舍？

- **C的表达：** C 是通用编程语言，常用函数、数组和结构体组织数据与计算；指针表示内存地址，访问边界与对象寿命影响程序正确性。
- **C++的扩展：** C++ 提供类、模板和标准库等组织工具，可封装模型状态并复用算法；它与 C 紧密相关，但两者并非完全兼容。
- **选择依据：** 本机代码与明确的数据布局便于优化重复计算，也便于接入既有模型库；实际速度仍取决于算法、实现和目标硬件。
- **典型用途：** C/C++常用于自驾仪固件、驱动、运动模型和ROS节点，能直接复用已有原生代码与库。

C++语法与练习（LearnCpp）：https://www.learncpp.com/

C/C++便于复用原生代码和优化计算，适合固件、驱动与模型开发。

2.1.1 开发基础与运行环境：C/C++与动态库

编辑器、编译器与构建工具如何分工

- **编辑与调试：**编辑器负责写代码；集成开发环境把编辑、构建和调试接在一起。VS Code 的 C/C++ 扩展仍需配套编译器，安装扩展不等于装齐工具链。
- **编译与链接：**编译器把源代码转换为目标代码，链接器组合目标文件并处理外部引用；MSVC、GCC、Clang 是常见编译工具链选择。
- **工程配置：**CMake 读取 CMakeLists.txt，确定源文件、依赖和输出目标，再生成对应构建系统；它本身不替代 C++ 编译器。
- **实际构建：**Ninja、Make 或 MSBuild 按依赖调用工具；修改源文件后重新构建，通常只处理受影响的部分，再运行和调试新产物。

CMake构建入门教程：https://cmake.org/cmake/help/latest/guide/tutorial/index.html

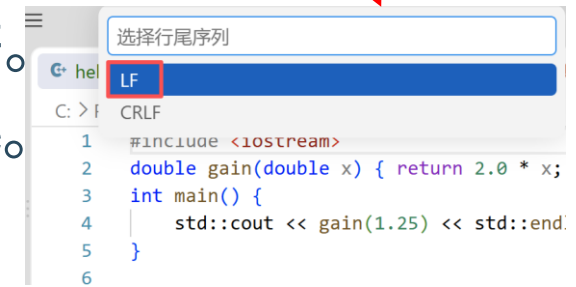
工程配置说明构建什么，构建工具安排步骤，编译器与链接器生成产物。

2.1.1 开发基础与运行环境：C++最小练习

一个增益函数怎样变成运行结果

```
#include <iostream>
double gain(double x) { return 2.0 * x; }
int main() {
    std::cout << gain(1.25) << std::endl;
}
```

```
C:\PX4PSP\Firmware>wsl
root@DaiPC:/mnt/c/PX4PSP/Firmware# g++ hello.cpp -o hello
root@DaiPC:/mnt/c/PX4PSP/Firmware# ./hello
2.5
```



- **工具环境：**安装MSVC C++工具后，打开x64 Native Tools命令提示符。
- **编译与运行：**保存hello.cpp；执行cl /EHsc hello.cpp，再执行hello.exe。
- **结果核对：**原输入得到2.5；将1.25改为-2.0，重新构建后得到-4。

MSVC命令行构建与环境：https://learn.microsoft.com/cpp/build/building-on-the-command-line

源文件描述计算，构建产物执行计算，修改后要重新构建并检查输出。

2.1.1 开发基础与运行环境：C/C++与动态库

从源文件到可执行文件与动态库

- **源文件与头文件：**源文件 .c/.cpp 保存实现，头文件 .h 常提供共享声明；声明告诉编译器怎样调用，实际定义仍要随工程参与构建。
- **预处理与编译：**预处理展开头文件和宏，编译检查语法与类型并转换代码；经汇编得到 .obj/.o 目标文件，此时还不是完整程序。
- **链接生成产物：**链接器组合目标文件和库，解析外部引用，按目标生成可执行文件或动态库；缺少函数定义常在链接阶段报错。
- **运行加载依赖：**启动程序或加载库时，系统还要找到所需动态库及其依赖；编译、链接成功后仍可能因缺少依赖而无法运行。

Clang编译与链接：https://clang.llvm.org/docs/UsersManual.html
GCC编译阶段与选项：https://gcc.gnu.org/onlinedocs/gcc/Overall-Options.html

编译生成目标代码，链接解析函数引用，加载器在运行时装入依赖库。

2.1.1 开发基础与运行环境：语言与构建补充

语言选择、Linux编译与交叉编译

注：Python的优势是算法的快速实现与验证，但在可靠性、执行速度与系统资源管理方面不及C/C++，后者适合最终的产品落地。

注：读者也可在 Linux 系统中通过 gcc 或 g++ 进行编译和运行。具体方法请自行查阅相关资料，并在 RflySim 提供的 WSL 环境中进行实践。
核心思路：保存hello.cpp（要使用LF结尾）；在终端输入命令 g++ hello.cpp -o hello 进行编译生成可执行文件；随后输入 ./hello 执行该程序。

注：在个人x86架构计算机上通过gcc编译出来的代码，不能在ARM架构上运行，需要使用交叉编译器（如arm-none-eabi-gcc）来编译。之所以WSL可以编译出PX4能跑的固件，就是基于此原理。

Python用于快验算法；C/C++跨架构部署须选对编译器。

2.1.1 开发基础与运行环境：C/C++与动态库

可执行文件与动态链接库：EXE、DLL、.so

- **可执行文件（EXE）：** Windows 的常见程序产物，启动后由操作系统创建进程并进入程序入口；它可以包含算法，也可以依赖动态库完成工作。
- **动态链接库（DLL）：** Windows 的共享代码模块，由宿主程序加载并调用导出接口；原生模型 DLL 通常不作为独立应用启动，也不自动建立独立进程。
- **Linux共享库（.so）：** Linux 常用的共享库产物，作用类似 DLL；二进制格式和平台接口不同，不能通过改后缀变成可加载的 Windows DLL。
- **两种链接方式：** 静态链接把所需库代码纳入产物；动态链接在运行时使用共享模块，便于复用和分发更新，但更新仍须保持接口兼容。

Windows DLL原理：https://learn.microsoft.com/windows/win32/dlls/dynamic-link-libraries

EXE 可作为程序启动；DLL 与 .so 供匹配的宿主加载使用。

2.1.1 开发基础与运行环境：库文件辨析

静态库、导入库与运行时动态库

- **静态库：**目标文件的归档，常见后缀.a或.lib；链接时将需要的代码纳入产物。
- **导入库：**Windows的.lib也可能是DLL导入库，供链接器解析导出符号，不包含可替代整个DLL的实现。
- **运行时依赖：**程序通过导入库链接DLL后，运行时仍需找到DLL及其依赖。只交付.lib通常不够。
- **显式加载：**ctypes等接口可直接加载DLL并查找函数；此时仍需声明正确的参数、返回类型与调用约定。

CMake静态库、共享库与导入库：https://cmake.org/cmake/help/latest/command/add_library.html

静态库提供目标代码，导入库提供链接信息，DLL提供运行时实现。

2.1.1 开发基础与运行环境：动态库匹配条件

DLL 能否调用：接口、架构与依赖匹配

- **接口契约：**导出函数名、参数和返回类型决定怎样调用；结构体布局、数组长度及内存由谁释放也要一致，否则可能读错数据或崩溃。
- **处理器架构：**宿主进程与库须面向兼容架构；常见 x64 程序不能直接加载 x86 DLL，64 位操作系统不代表每个进程都是 64 位。
- **运行依赖：**DLL 可能依赖其他动态库和 C/C++ 运行库；主文件存在却提示找不到模块，可能是依赖缺失或搜索路径不对。
- **跨语言接口：**通常用明确的C风格函数封装C++实现，参数类型、数组长度与内存释放规则仍需一致。

DLL导出符号与名称修饰：https://learn.microsoft.com/cpp/build/reference/decorated-names

可调用的 DLL 需要接口说得清、架构配得上、运行依赖找得到。

2.1.1 开发基础与运行环境：动态库最小练习

把增益函数导出为Windows DLL

```
// gain.cpp
extern "C" __declspec(dllexport)
double gain(double x) { return 2.0 * x; }
```

- **构建条件：** 使用MSVC的x64 Native Tools命令提示符，在文件目录中构建。
- **构建命令：** cl /LD /MD /EHsc gain.cpp /link /OUT:gain.dll
- **接口含义：** extern "C"使用C链接，dllexport导出函数；/LD指定DLL目标。
- **产物边界：** 得到gain.dll；它演示函数调用，不是CopterSim专用运动模型。

MSVC创建与调用DLL
https://learn.microsoft.com/cpp/build/walkthrough-creating-and-using-a-dynamic-link-library-cpp

先明确导出函数与构建目标，再由匹配架构的宿主加载动态库。

2.1.1 开发基础与运行环境：动态库补充

静态库取舍与Linux共享库

注：静态链接可减少对应库的运行时依赖，但代码体积较大且更新须重新链接；动态库便于共享和局部升级，但存在依赖缺失风险。

注：在WSL中先删去__declspec(dllexport)，保留extern "C"，再执行

```
g++ -fPIC -shared gain.cpp -o gain.so
```

最终得到gain.so文件，可以被其他程序调用。

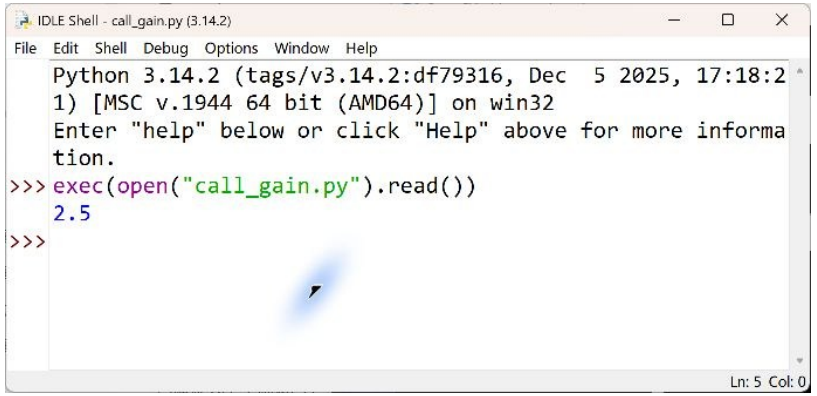
库类型影响部署依赖；Windows DLL与Linux .so分别构建和加载。

2.1.1 开发基础与运行环境：C/C++与动态库

Python调用C/C++计算模块

```
import ctypes
from pathlib import Path
lib = ctypes.CDLL(str(Path("gain.dll").resolve()))
lib.gain.argtypes = [ctypes.c_double]
lib.gain.restype = ctypes.c_double
print(lib.gain(1.25))
```

图：IDLE运行call_gain.py的输出



- **运行条件：**代码存为call_gain.py，在gain.dll所在目录运行，Python须为x64。
- **结果与区别：**打印2.5；这是Python进程内的函数调用，不经网络，也不启动仿真。

ctypes类型与调用约定：https://docs.python.org/3/library/ctypes.html

ctypes在Python进程中调用原生函数，参数与返回类型决定数据解释。

2.1.1 开发基础与运行环境：命令行与启动脚本

启动脚本、解释器和算法各做一件事

环节	红球例程中的文件／命令	检查什么
启动仿真	ShootBall3SITL.bat	场景、运动模型和控制软件就绪
准备解释器	Python38Run.bat	窗口提示Python3.8环境已设置
执行算法	python ShootBall3.py	25 s后弹出掩膜窗口并开始跟踪

- **看懂脚本：** 批处理脚本（BAT）调用程序并传参；终端显示命令与报错。
- **保存条件：** 脚本、模型、依赖和算法版本共同决定能否复现实验。

BAT准备启动条件，解释器执行脚本，算法把观测转为运动目标。

2.1.1 开发基础与运行环境：版本管理

Git把实验修改保存为可追溯版本



分支独立尝试，合并整合修改

- **版本控制：** Git保存文件修改历史，便于比较差异和恢复已记录的版本。
- **提交与分支：** 提交记录一次项目快照；分支隔离方案，合并整合确认后的修改。
- **实验记录：** 代码版本与参数、场景、初始状态和结果一起保存，支持重复比较。

Git命令手册
https://git-scm.com/docs

GitHub仓库托管
https://github.com/

Gitee仓库托管
https://gitee.com/

提交保存一个版本，分支隔离不同尝试，实验条件把版本与结果对应起来。

2.1.1 开发基础与运行环境：Git入门

Git记录修改，托管平台共享项目

- **Git与托管平台：**Git在本地保存版本历史。GitHub、Gitee托管远程仓库，便于协作和获取源码。
- **先读再改：**git status查看改动范围，git diff查看内容差异，先确认本次究竟改了什么。
- **保存一个版本：**git add选择文件，git commit保存本地提交，不会自动上传。
- **最小练习：**在例程副本中git init并提交基线；只改一个阈值再提交，记录场景参数，用git log查看历史。

Pro Git中文书：提交与分支
https://git-scm.com/book/zh/v2

Git分支交互练习
https://learngitbranching.js.org/

代码提交与实验条件一起保存，才能解释结果为什么发生变化。

本章目录

关键工具：从运行环境到仿真闭环

第二章　实验平台配置

2.1　关键工具介绍

2.2　平台安装与首次仿真

2.3　平台架构与开发流程

2.4　API与SDK开发接口

2.5　例程体系与实验路线

2.6　习题与课堂讨论

本节 4 个教学单元

2.1.1 开发基础与运行环境

教材对应：第 2.1.2.2、2.1.2.3、2.1.2.7 节

2.1.2 模型与算法开发

教材对应：第 2.1.2.1、2.1.2.3、2.1.2.7 节

2.1.3 系统组成与通信

教材对应：第 2.1.2.4–2.1.2.6 节

2.1.4 仿真工具与开发路线

教材对应：第 2.1.2.1、2.1.2.8 节

2.1.2 模型与算法开发：任务关系总览

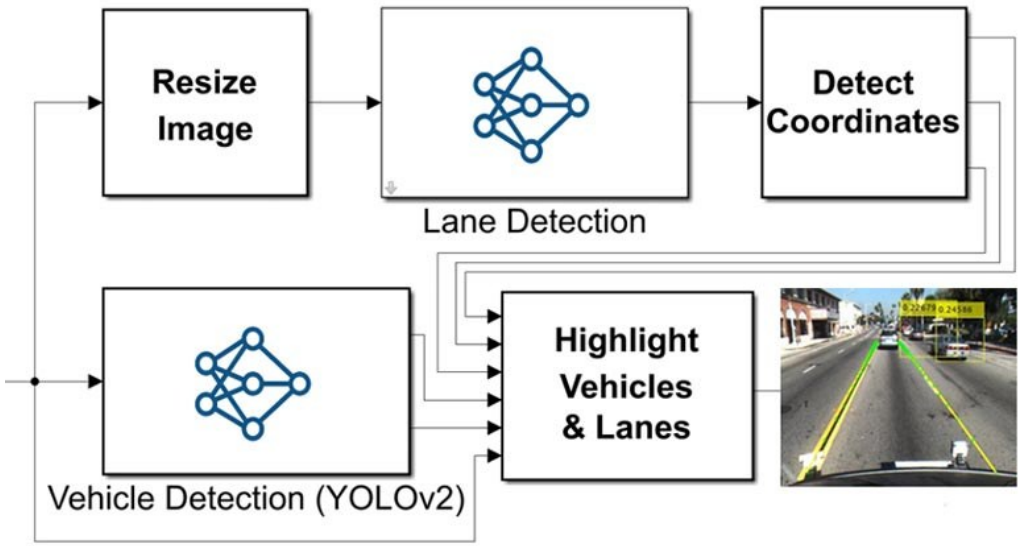
模型、控制器与视觉算法接入同一闭环

开发对象	输入与开发工具	产物与闭环位置
运动模型	参数、执行量；Simulink/C++	模型DLL → CopterSim状态
控制器	状态、目标；Simulink/C++	控制程序 → 自驾仪执行量
视觉算法	图像；OpenCV、Python/C++	目标信息 → 上层控制目标
选学：训练	样本；PyTorch/TensorFlow	已训练模型 → 视觉推理
按需：加速	模型；GPU/CUDA/TensorRT	推理提速 → 上层算法

模型管运动，控制器管执行，视觉管观测；训练和加速按任务选用。

2.1.2 模型与算法开发：建模与仿真

MATLAB分析数据，Simulink组织模型



图：车道与车辆检测的数据流

- **MATLAB**：用数值计算和脚本准备输入、调整参数，分析误差与结果。
- **Simulink**：用模块表示运算，信号线传递数据，按时间推进仿真。
- **图中流程**：上支路检测车道，下支路检测车辆，结果叠加回原图。

MATLAB处理数值与脚本，Simulink用模块和信号描述可仿真的系统。

2.1.2 模型与算法开发：建模工具入门

MATLAB与Simulink的入门任务

- **两类工具：**MATLAB用矩阵与脚本分析数据，Simulink用模块、信号和求解器推进动态系统。
- **基于模型的设计（MBD）：**同一模型贯穿设计、仿真与测试；代码生成把受支持的模块转为C/C++，编译后成为自驾仪固件或模型DLL。
- **最小练习：**MATLAB用plot画正弦曲线；Simulink用Sine Wave→Gain→Scope，增益由1改为2后幅值加倍。

MATLAB交互入门课程
https://matlabacademy.mathworks.com/

Simulink建模与仿真教程
https://www.mathworks.com/help/simulink/

数值脚本便于分析数据，模块模型便于观察系统随时间变化的行为。

2.1.2 模型与算法开发：模型与生成产物

运动模型与控制器分别去哪里

被控对象

运动模型
力、力矩与运动状态

➡

动态链接库（DLL）
符合约定的动态库

➡

CopterSim
加载模型并计算运动

控制算法

控制模型
状态输入与控制输出

➡

生成与编译
工具链及目标配置

➡

自驾仪程序
执行控制算法

模型接口：CopterSim传入执行量，动态库计算并返回运动状态。

运动模型描述被控对象，控制代码实现控制规律，产物去向不同。

2.1.2 模型与算法开发：C/C++与动态库

CopterSim怎样加载模型DLL

- **实验入口：**第2章基础实验e2_DLL-Load，用文本编辑器打开Exp1_MinModelTemp_SITL.bat。
- **模型选择：**DLLModel设置为Exp1_MinModelTemp，对应Exp1_MinModelTemp.dll。
- **导入链条：**启动脚本将模型DLL复制到CopterSim/external/model目录，再把模型名传给CopterSim.exe。
- **运行分工：**模型DLL按平台接口更新运动状态，CopterSim调度调用，RflySim3D根据状态更新场景。

CopterSim模型DLL配置与导入：https://rflysim.com/doc/zh/3.Soft/coptersim.html

模型DLL计算运动，CopterSim负责调用，三维程序呈现状态。

2.1.2 模型与算法开发：OpenCV入门

OpenCV的能力、优势与语言接口

- **能力范围：**开源计算机视觉库（OpenCV）提供图像读写、滤波、颜色处理、几何变换、相机标定和特征检测等现成算法。
- **语言接口：**提供C++与Python接口。Python便于交互试验，C++便于接入原生工程。
- **经典方法：**阈值与几何测量无需训练样本，中间结果可见，适合建立检测基线。
- **使用条件：**规则对光照和遮挡敏感。接口的颜色、尺寸与数据类型必须匹配，计算速度应实际测量。

OpenCV模块与示例源码：https://github.com/opencv/opencv

现成算法与双语言接口降低视觉开发成本，可靠检测仍取决于数据和场景。

2.1.2 模型与算法开发：OpenCV入门

OpenCV中的图像数组与颜色通道

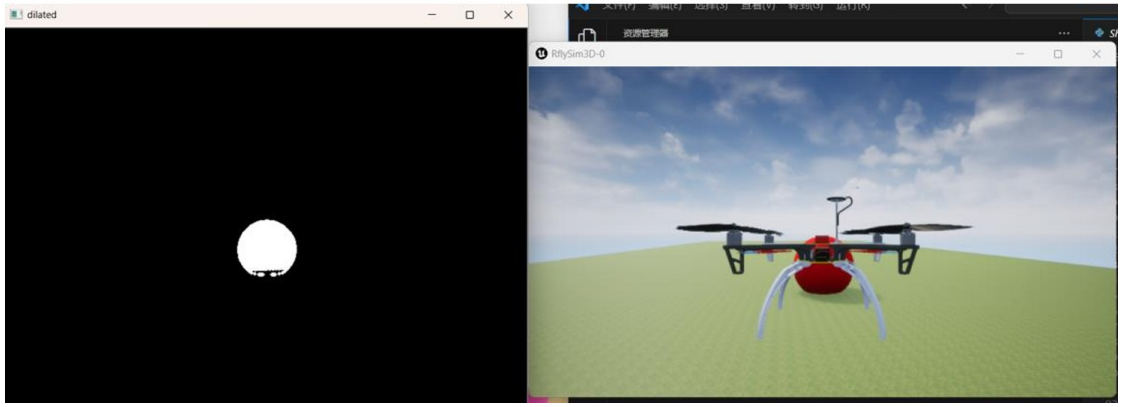
- **读图结果：** cv2.imread返回NumPy数组；路径错误时返回None，处理前先检查。
- **数组模型：** Python 中图像通常是 NumPy 数组，彩色按高、宽、通道排列，灰度通常为二维；访问像素先行后列。
- **颜色通道：** 八位彩色图的单通道值为0至255。默认彩色读图按蓝绿红（BGR），其他接口可能用红绿蓝（RGB）。
- **使用边界：** 颜色顺序由接口约定，不能看文件名猜测；交换通道须显式转换，像素坐标也不能直接解释为世界位置。

像素、通道与数组：https://docs.opencv.org/4.x/d3/df2/tutorial_py_basic_ops.html

图像数组按行、列和通道索引，颜色通道顺序由接口约定。

2.1.2 模型与算法开发：视觉与学习框架

同一视觉任务，可以采用不同检测方法



颜色分割结果 → 面积、质心

OpenCV　　TensorFlow

- **OpenCV方法：**按颜色筛选并清理像素，计算目标面积与中心。
- **学习方法：**PyTorch／TensorFlow用样本训练模型，再识别新图像。
- **共同输出：**目标位置、范围或类别，继续供感知决策与运动控制使用。
- **方法取舍：**阈值法省去训练，学习方法依赖样本覆盖实际场景。

颜色检测用人工规则筛选像素，学习模型用数据形成检测能力。

2.1.2 模型与算法开发：OpenCV入门

颜色分割怎样输出目标中心

- **颜色筛选：**读入图像后转换为色相、饱和度与明度（HSV），用阈值选出目标颜色，得到前景白、背景黑的二值掩膜。
- **噪声清理：**形态学开运算可去除小块噪声，闭运算可填补小孔；核的大小影响保留哪些细节，需要结合目标尺寸选择。
- **中心输出：**提取轮廓并按面积筛选，用图像矩计算目标质心，同时输出面积与有效标志，供后续跟踪或误差计算使用。
- **失败判据：**没有合格轮廓或面积为零时返回无效；光照变化、遮挡和同色背景会干扰检测，二维中心尚不包含深度。

HSV颜色分割：https://docs.opencv.org/4.x/df/d9d/tutorial_py_colorspaces.html

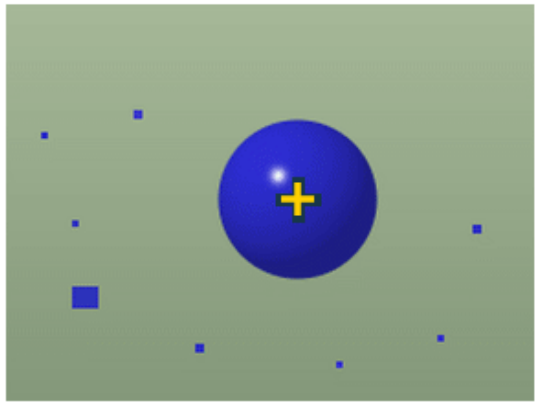
颜色筛选、形态清理和轮廓测量可把图像变成目标中心。

2.1.2 模型与算法开发：OpenCV入门

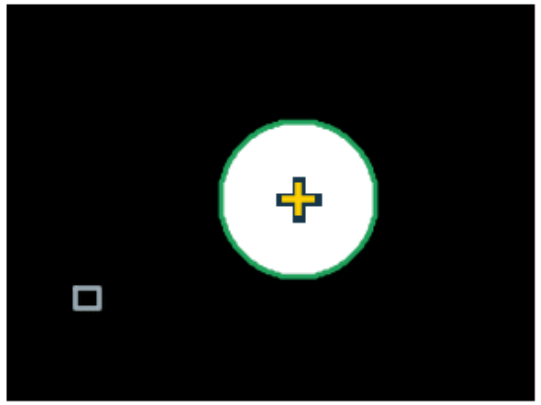
颜色分割的中间结果与无效判定

1 HSV 阈值　2 开运算　3 闭运算　4 轮廓筛选　**5 质心**　6 目标移出

合成图像



OpenCV 掩膜



质心 (px)
(132.0, 88.0)

有效面积 (px)
4049

有效目标

面积 ≥ 300 px → 质心；否则 None

图：合成图像上的OpenCV逐步运算（教学示意）

无目标也要显式报告，视觉控制才不会把失效质心当作新观测。

2.1.2 模型与算法开发：OpenCV最小练习

一幅图像的输入、测量与失败检查

- **输入确认：**用代码生成含单一蓝色矩形的图片；
若改为读入文件，先检查结果不是None，再打印shape和dtype。
- **逐步观察：**8位OpenCV图像转HSV后，H为0至179，S、V为0至255。筛选蓝色区间，核对原图、掩膜与中心标记。
- **结果核对：**矩形角点为(80,60)和(140,100)时，预期中心约为(110,80)，
坐标单位为像素。
- **失败处理：**换成全黑图后输出无效标志。没有轮廓或面积为零时，不继续计算质心，也不沿用旧观测。

掩膜形态学处理：https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html

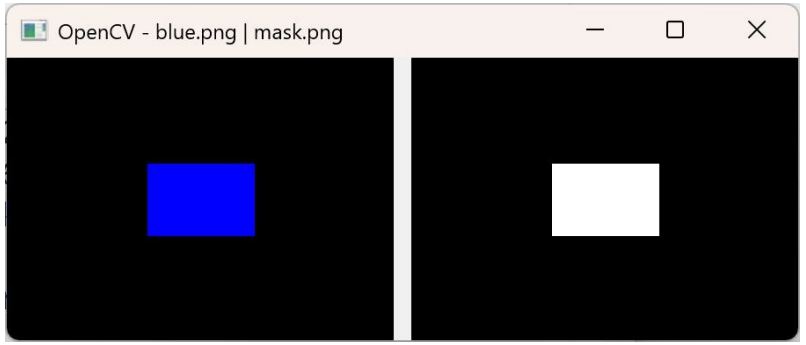
先在静态图上核对中间结果和失败状态，再处理仿真的连续图像。

2.1.2 模型与算法开发：OpenCV最小练习

生成图像并提取蓝色掩膜

```
import cv2
import numpy as np
img = np.zeros((160, 220, 3), dtype=np.uint8)
cv2.rectangle(img, (80, 60), (140, 100), (255, 0, 0), -1)
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
mask = cv2.inRange(hsv, (110, 100, 100), (130, 255, 255))
cv2.imwrite("blue.png", img)
cv2.imwrite("mask.png", mask)
```

图：OpenCV 输出窗口



- **观察结果：**同一目录得到蓝色矩形原图与白色矩形掩膜，下一步测量中心。

绘图函数：https://docs.opencv.org/4.x/dc/da5/tutorial_py_drawing_functions.html

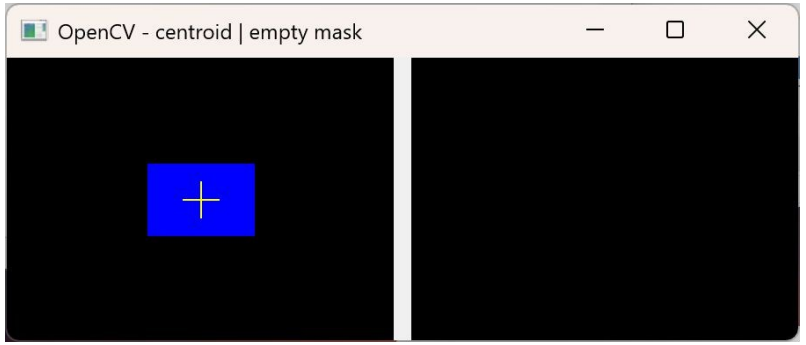
颜色命中只是目标候选，还要检验面积与位置是否可信。

2.1.2 模型与算法开发：OpenCV最小练习

计算质心，并验证无目标分支

```
def center(mask):
    m = cv2.moments(mask, binaryImage=True)
    if m["m00"] == 0:
        return None
    return (m["m10"] / m["m00"], m["m01"] / m["m00"])
print(center(mask))
print(center(np.zeros_like(mask)))
```

图：质心与空掩膜窗口



- **接续运行：**紧接前页代码执行，核对下面两种输入的输出。
- **蓝色矩形：**一阶矩除以面积得到质心，预期输出(110.0, 80.0)。
- **全黑掩膜：**面积为零时返回None，明确标记观测无效。

图像矩与质心：https://docs.opencv.org/4.x/dd/d49/tutorial_py_contour_features.html

2.1.2 模型与算法开发：选学·学习型视觉

训练框架、推理程序与视觉库

- **训练任务：**PyTorch、TensorFlow提供张量运算、自动求导和模型组件，用样本和损失函数更新参数。
- **推理任务：**部署时用已经训练的模型处理新图像，输出类别、目标框或掩膜，通常不再更新参数。
- **OpenCV配合：**读取图像、调整尺寸、转换颜色和显示结果，也可用深度神经网络（DNN）模块运行受支持的模型。
- **学习顺序：**先完成单图推理，核对输入与输出，再连接实时图像和运动控制。

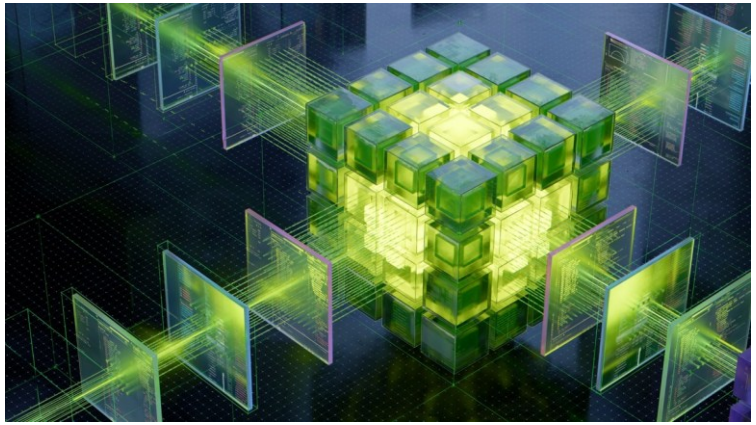
PyTorch张量与模型训练
https://pytorch.org/tutorials/

TensorFlow训练与预测
https://www.tensorflow.org/tutorials

训练得到模型参数，推理生成目标信息，视觉库负责前后处理和显示。

2.1.2 模型与算法开发：选学·推理加速

并行计算与模型推理加速



图：图形处理器的并行计算示意

- **并行硬件：**图形处理器（GPU）适合大量并行运算；CUDA提供NVIDIA GPU计算平台。
- **推理优化：**TensorRT优化并运行已训练模型，把新图像转换成检测等结果。
- **完整时延：**输入预处理、数据搬运、模型推理和后处理共同决定输出速度。

预处理 → 数据传输 → 模型推理 → 后处理

CUDA计算文档
https://docs.nvidia.com/cuda/

TensorRT推理文档
https://docs.nvidia.com/tensorrt/

并行硬件和推理优化降低计算耗时，端到端时延还包括数据传输与处理。

2.1.2 模型与算法开发：选学·并行与推理加速

GPU计算、推理与训练的源码示例

CUDA并行计算示例： https://github.com/NVIDIA/cuda-samples
先看设备查询、向量运算与数据传输示例，理解主机和GPU之间的协作。

TensorRT推理示例： https://github.com/NVIDIA/TensorRT
从samples目录理解模型输入、执行引擎与输出，核对版本和运行依赖。

PyTorch训练示例： https://github.com/pytorch/examples
从图像分类等小例子观察数据、模型、损失和参数更新，先读README。

TensorFlow模型与示例： https://github.com/tensorflow/models
按任务选择模型示例，先核对依赖与数据，再区分训练和推理入口。

加速模型计算只缩短推理环节，整条视觉链路还受数据搬运等开销影响。

本章目录

关键工具：从运行环境到仿真闭环

第二章　实验平台配置

2.1　关键工具介绍

2.2　平台安装与首次仿真

2.3　平台架构与开发流程

2.4　API与SDK开发接口

2.5　例程体系与实验路线

2.6　习题与课堂讨论

本节 4 个教学单元

2.1.1 开发基础与运行环境

教材对应：第 2.1.2.2、2.1.2.3、2.1.2.7 节

2.1.2 模型与算法开发

教材对应：第 2.1.2.1、2.1.2.3、2.1.2.7 节

2.1.3 系统组成与通信

教材对应：第 2.1.2.4–2.1.2.6 节

2.1.4 仿真工具与开发路线

教材对应：第 2.1.2.1、2.1.2.8 节

2.1.3 系统组成与通信：生态分工

Pixhawk、PX4与地面站是什么

pixhawk

PX4 autopilot

本书统称“自驾仪”，无人机中通常俗称“飞控”。

对象	所属层次	在系统中的作用
Pixhawk	硬件标准与生态	兼容板卡承载计算、传感与输出
PX4 / ArduPilot	自驾仪软件	状态估计、控制、任务与保护
QGroundControl	地面站软件	配置、任务规划与运行监控
MAVLink	消息协议	约定指令、状态等消息格式

硬件承载自驾仪软件，地面站通过消息接口配置和监控系统。

2.1.3 系统组成与通信：自驾仪生态入门

硬件识别、固件配置与地面站学习资料

Pixhawk硬件标准与生态：https://pixhawk.org/
辨认控制板、传感器输入与执行输出，先理解硬件职责。

PX4载具配置与控制文档：https://docs.px4.io/main/zh/
从Getting Started认识载具类型、固件、状态估计与控制。

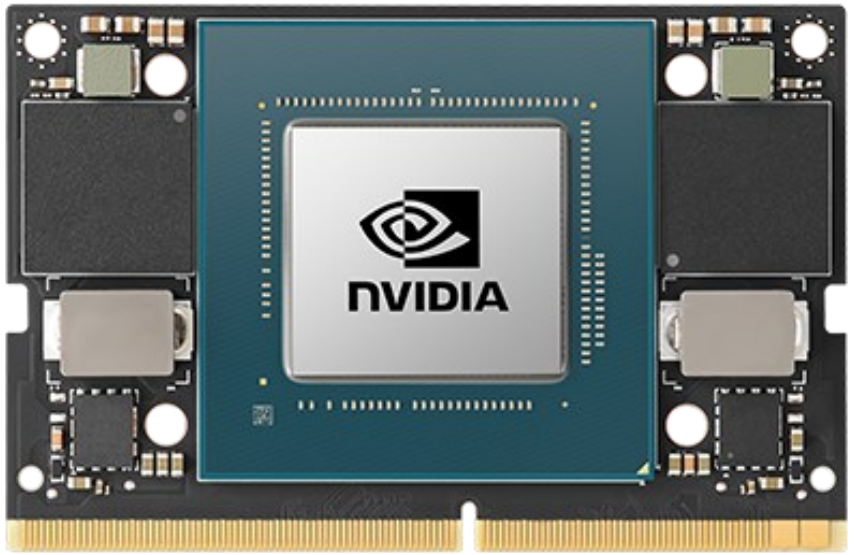
ArduPilot各类载具手册：https://ardupilot.org/
按载具类型进入对应手册，理解共同能力与配置差异。

QGroundControl（QGC）配置、规划与监控手册：https://docs.qgroundcontrol.com/
从界面概览认识配置、任务规划与运行观察的分工。

硬件提供计算与输入输出，固件实现自驾仪，地面站提供配置和观察入口。

2.1.3 系统组成与通信：机载计算

自驾仪与机载计算机怎样分工



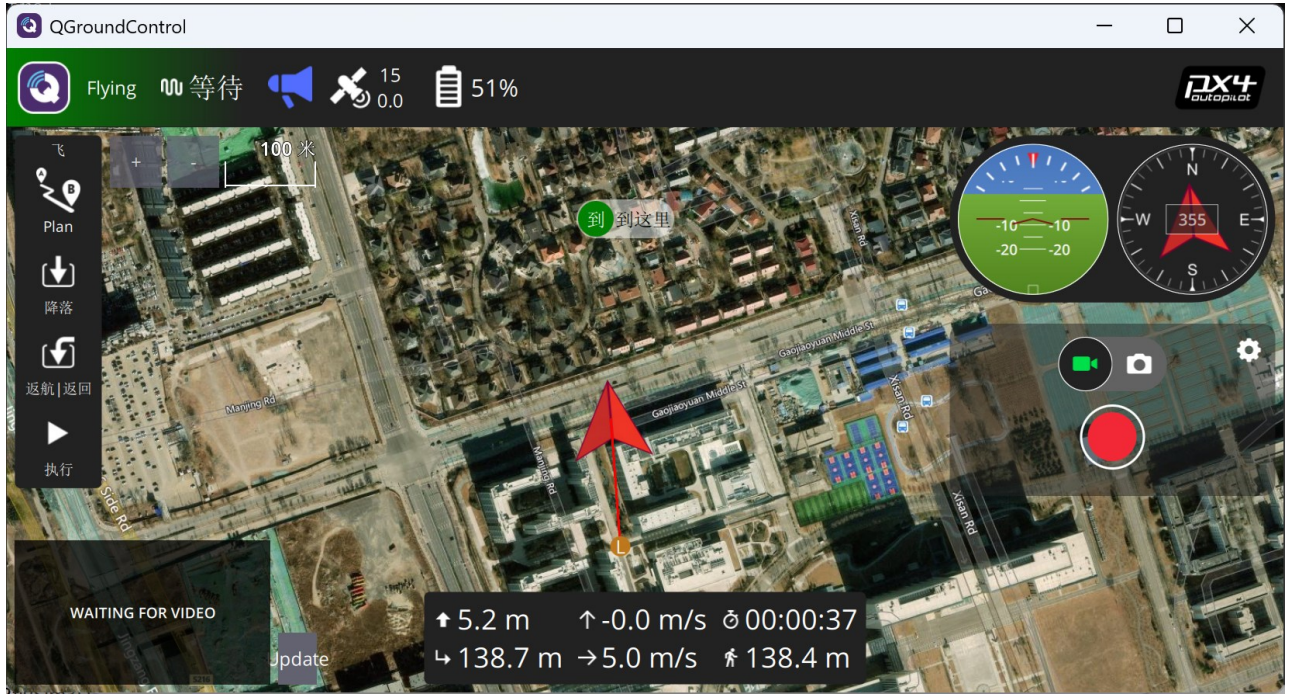
图：NVIDIA Jetson机载计算模块

- **自驾仪：**接收传感量、估计状态，运行底层控制并输出执行指令。
- **机载计算机：**处理图像和点云，完成识别、路径规划与任务逻辑。
- **红球跟踪示例：**上层由目标质心与面积计算速度目标，自驾仪控制载具运动。
- **反馈与约束：**位置、速度和模式回传上层；算力、功耗、散热与接口制约选型。

感知规划给出运动目标，自驾仪结合状态估计把目标落实为执行量。

2.1.3 系统组成与通信：地面站

从地面站读状态、发动作、看响应



图：QGroundControl（QGC）飞行视图

- **状态栏：**上方显示连接、模式、电池等运行状态。
- **动作栏：**起飞与返航入口发送请求，是否执行由自驾仪状态决定。
- **地图与仪表：**地图呈现位置，仪表呈现姿态。

地面站把运行状态、操作者指令和载具响应放到同一观察界面。

2.1.3 系统组成与通信：地面站工作流

QGC贯穿任务准备、运行与复盘

- **参数配置：**按自驾仪支持项配置机型、传感器和控制参数，配置值应与实际载具及仿真任务一致。
- **任务规划：**编辑地图中的航点和任务项，检查高度、顺序与结束动作，再上传任务。
- **运行监控：**观察位置、模式和电池遥测，比较目标与实际响应，异常时查看消息。
- **日志复盘：**日志按时间保存状态与控制信息；将目标、实际响应和模式变化对齐，可定位响应滞后发生在哪个环节。

QGC任务规划与界面源码
https://github.com/mavlink/qgroundcontrol

PX4飞行日志在线分析
https://logs.px4.io/

地面站组织配置、任务和观察，自驾仪实际执行状态估计与运动控制。

2.1.3 系统组成与通信：消息协议

MAVLink约定消息含义，链路负责传输



上层程序／地面站

编码控制目标

➡

串口或网络

传递MAVLink消息

➡

自驾仪

解码并执行控制

回传：姿态、位置、速度、模式等状态

- **消息内容：**如速度目标消息含坐标系、vx/vy/vz（m/s）与目标载具编号。
- **传输通道：**串口匹配端口与波特率，网络匹配地址与端口。
- **状态回传：**自驾仪反馈实际运动和模式，上层据此更新目标。

消息格式回答数值表示什么，传输通道回答消息怎样到达目标程序。

2.1.3 系统组成与通信：进程间通信

库调用与进程间通信

- **程序与进程：**程序是可执行代码，运行后形成进程；各进程通常有独立地址空间。
- **库调用：**宿主进程加载DLL后调用函数，参数与返回值约定双方怎样交换数据。
- **跨进程交换：**进程间通信（IPC）让独立程序交换数据或同步，常见方式有管道、套接字和共享内存。
- **平台中的区别：**CopterSim调用模型DLL属于宿主内的模型计算，多程序间交换图像和状态还需要通信接口。

Python套接字通信：https://docs.python.org/3/library/socket.html

函数调用连接进程内部的实现，进程间通信连接独立运行的程序。

2.1.3 系统组成与通信：传输通道

串口、网络与共享内存怎样选择

通道	传输机制	无人系统中的用途与取舍
串口	按字节传输的设备链路	计算机连接自驾仪，双方匹配波特率与格式。
用户数据报协议UDP	保留数据报边界 不保证送达及顺序	RflySim组件间互传状态与控制指令，丢包与时效由应用处理。
传输控制协议TCP	可靠、有序的字节流 丢失数据可重传	传送需要完整到达的数据，重传可能增加等待。
共享内存	本机进程共享缓冲区	红球例程中RflySim3D向Python传图；需同步读写与管理缓冲。

TCP提供可靠有序传输，UDP由应用处理丢包，同机大数据可用共享内存。

2.1.3 系统组成与通信：传输通道

同次丢包下的UDP与TCP交付过程



图：相同网络条件：3号包丢失，4号包延迟（教学示意）

对照同次丢包：UDP应用取最新，TCP补齐后按序交付。

2.1.3 系统组成与通信：ROS入门

ROS节点与发布订阅

- **组织优势：**ROS提供通信库、开发工具和软件包生态，可以独立开发并复用感知、规划与控制模块。
- **节点分工：**节点是承担一项逻辑职责的计算单元，可在同机或跨机协作；相机采集、目标检测和显示可以分别组织。
- **发布订阅：**发布者向命名话题发送消息，订阅者按话题与类型接收；一个图像源可同时供检测、显示和记录使用。
- **消息模型：**消息是有字段类型的结构；图像消息包含宽高、编码、字节数据以及采集时间和坐标系，帮助程序一致地解释图像。

ROS2 Jazzy基础教程
https://docs.ros.org/en/jazzy/

ROS1节点与话题教程
https://wiki.ros.org/ROS/Tutorials

ROS 用有类型的消息连接节点，让多个程序共享观测与结果。

2.1.3 系统组成与通信：ROS基本概念

ROS怎样组织机器人程序

ROS：运行于操作系统之上的机器人软件框架

节点是功能单元，不等于进程；也可在同一进程内组合。

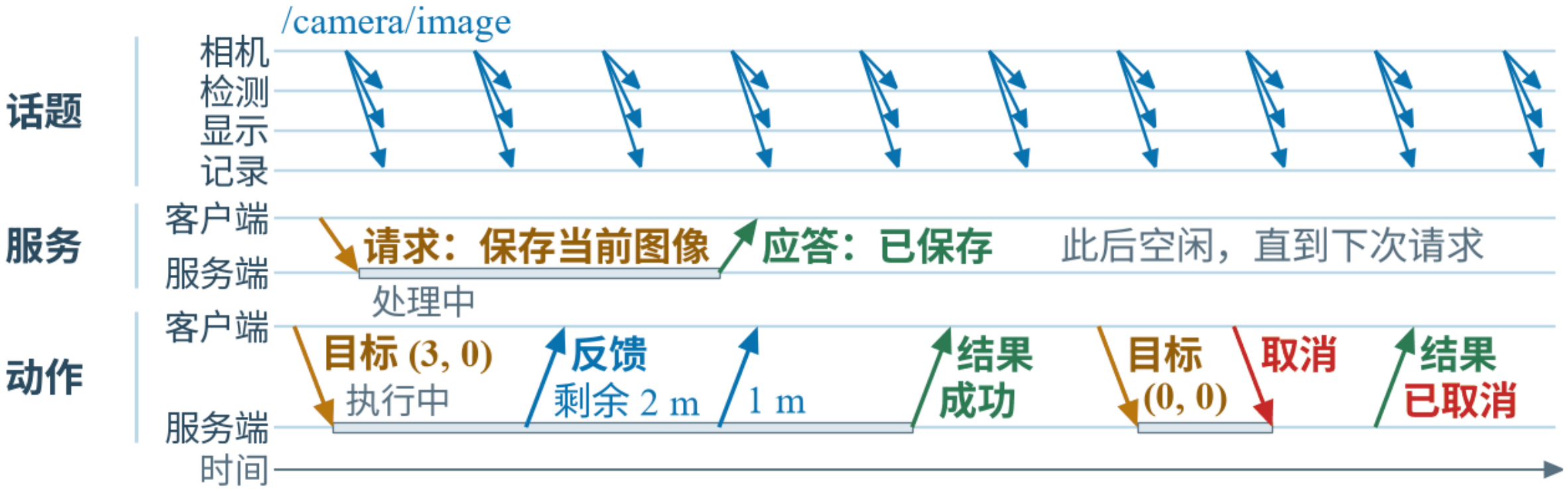


概念	作用	示例
话题	按消息类型发布与订阅	相机图像、里程计状态
服务	一次请求与应答	查询配置、触发操作
动作	有反馈且可取消的长任务	导航到目标位置
参数	保存节点的可配置值	帧率、阈值、坐标设置

ROS用节点组织功能，用话题、服务和动作表达不同交互。

2.1.3 系统组成与通信：ROS基本概念

话题、服务与动作的消息时序



图：三类交互共用时间轴，消息传输时间已放慢（教学示意）

2.1.3 系统组成与通信：ROS版本

ROS 1与ROS 2保留概念，改变实现

比较项	ROS 1	ROS 2
组织概念	节点、话题、服务等	延续概念并扩展能力
通信与发现	通常由Master协调发现 节点之间传输消息	中间件接口（RMW） 常用数据分发服务（DDS）
构建生态	常见catkin及配套工具	常见ament与colcon
迁移条件	旧包依赖与消息接口	客户端API、包构建与通信配置

RMW隔离通信实现；DDS提供数据发布、订阅与发现机制。

catkin与ament参与包构建；colcon组织多包构建。

ROS两代延续节点与消息概念，通信实现和构建工具发生变化。

2.1.3 系统组成与通信：ROS入门

ROS版本与平台环境匹配

- **已有课程：**按已验证的环境复现实验。ROS1 Noetic于2025年5月结束官方维护，Ubuntu 22.04起没有官方ROS1安装包。
- **新项目：**按依赖选择ROS2稳定版，例如Ubuntu 24.04与Jazzy、22.04与Humble，并核对支持期限。
- **平台接入：**WinWSL承载Linux与ROS环境，图像接口发布观测，节点处理结果后再接课程控制接口。
- **通信兼容：**话题、类型与网络决定数据通路；服务质量（QoS）还约定交付策略，尽力发送不能满足可靠接收要求。

ROS2版本与目标平台
https://www.ros.org/reps/rep-2000.html

RflySim配套WinWSL环境
https://rflysim.com/doc/zh/3.Soft/winwsl.html

课程复现保留已验证依赖，新学习按系统与软件包选择稳定版。

2.1.3 系统组成与通信：ROS入门

ROS可视化与话题录包

- **通信检查：**命令行列出节点与话题，查看消息类型、数值和频率；节点关系图工具 rqt_graph 展示谁发布、谁订阅。
- **数据可视：**三维可视化工具 RViz 可显示图像、点云和坐标变换，显示的是输入数据；固定参考系与时间戳错误会造成错位。
- **录包回放：**ROS1 用 rosbag、ROS2 用 ros2 bag 保存话题数据，再按时间重放，便于重复调参、共享观测和比较算法。
- **复现边界：**录包保留选定消息，不会自动保存整个仿真世界和全部参数；图像、相机参数、坐标变换及算法配置应按实验需要一起保留。

RViz三维显示源码（ROS2）
https://github.com/ros2/rviz

ROS1录包与通信源码
https://github.com/ros/ros_comm

RViz显示消息中的空间数据，录包保存观测供算法重复处理和比较。

2.1.3 系统组成与通信：ROS最小练习

消息的发现、观察与回放

- **准备环境：**在已装turtlesim的独立Jazzy环境中，每个新Bash终端先执行source /opt/ros/jazzy/setup.bash，再启动仿真与键盘节点。
- **列出与查看：**ROS2用ros2 topic list、info和echo查看话题与数据，ROS1对应使用rostopic。
- **录制与回放：**记录/turtle1/pose；重复录制换新目录名。停止采集源后回放，同一段位姿消息再次出现。
- **消息含义：**/turtle1/pose发布位姿观测，/turtle1/cmd_vel接收速度指令；本练习回放的是观测数据。

ROS2录包与回放示例
https://github.com/ros2/rosbag2

ROS1 turtlesim源码（Noetic）
https://github.com/ros/ros_tutorials/tree/noetic-devel

位姿话题描述运动结果，速度话题传递运动指令，二者的数据方向不同。

2.1.3 系统组成与通信：ROS2命令练习

用Jazzy练习话题观察与录包回放

01 两个终端 分别启动	ros2 run turtlesim turtlesim_node ros2 run turtlesim turtle_teleop_key
02 新终端 查看与订阅	ros2 topic info /turtle1/pose ros2 topic echo /turtle1/pose
03 新终端录制 移动后停止	ros2 bag record -o pose_demo /turtle1/pose 键盘产生运动；录制终端按Ctrl+C结束。
04 停止源节点 回放并观察	ros2 bag play pose_demo 先停仿真与键盘节点，保留echo观察。

ROS2通信示例（Jazzy分支）：https://github.com/ros2/demos/tree/jazzy

位姿录包可重复提供同一段观测，便于调试订阅端与比较算法。

2.1.3 系统组成与通信：适配库

怎样在程序中使用MAVLink

适配库封装消息收发，把应用功能接到MAVLink链路。

工具	主要用法	适合的任务
Pymavlink	Python直接处理MAVLink消息	检查报文、实现特定消息交互
MAVSDK	以较高层API调用遥测、动作和任务	快速实现受支持的应用功能
MAVROS	在ROS与MAVLink系统之间适配	把自驾仪接入ROS节点网络

检查底层报文还是调用高层功能，决定选用哪一层MAVLink接口。

2.1.3 系统组成与通信：通信工具入门资料

MAVLink消息收发与适配库示例

MAVLink消息字段与协议： https://mavlink.io/en/
阅读消息定义中的字段、单位和坐标系，辨认目标值与实际状态。

Pymavlink消息收发教程： https://mavlink.io/en/mavgen_python/
先读建立连接与接收消息的示例，理解消息类型和超时。

MAVSDK连接与遥测教程： https://mavsdk.mavlink.io/
先读QuickStart和遥测示例，理解高层API与底层消息的关系。

MAVROS节点与话题源码： https://github.com/mavlink/mavros
先读README与话题接口，用ROS节点连接自驾仪消息。

同一消息协议可以有不同语言和框架入口，选择取决于应用所需的接口。

2.1.3 系统组成与通信：外部控制·MAVLink示例

上层程序给目标，自驾仪完成运动控制

视觉算法		PX4自驾仪		载具／运动模型
质心与面积 计算运动目标	➡	状态估计与控制 产生执行量	➡	运动响应 形成新的图像

观测反馈：图像中心偏差、目标面积与实际运动状态

- **速度接口：** SendVelFRD(vx, vy, vz, yawrate)：速度单位m/s，偏航角速度单位rad/s。
- **方向约定：** 前、右、下（FRD）规定速度方向；vz小于0表示上升目标。
- **外部控制：** 外部控制模式（Offboard）持续接收目标，自驾仪完成底层控制。

视觉误差先变成运动目标，自驾仪再把运动目标变成执行量。

2.1.3 系统组成与通信：外部控制模式

Offboard目标流与自驾仪响应

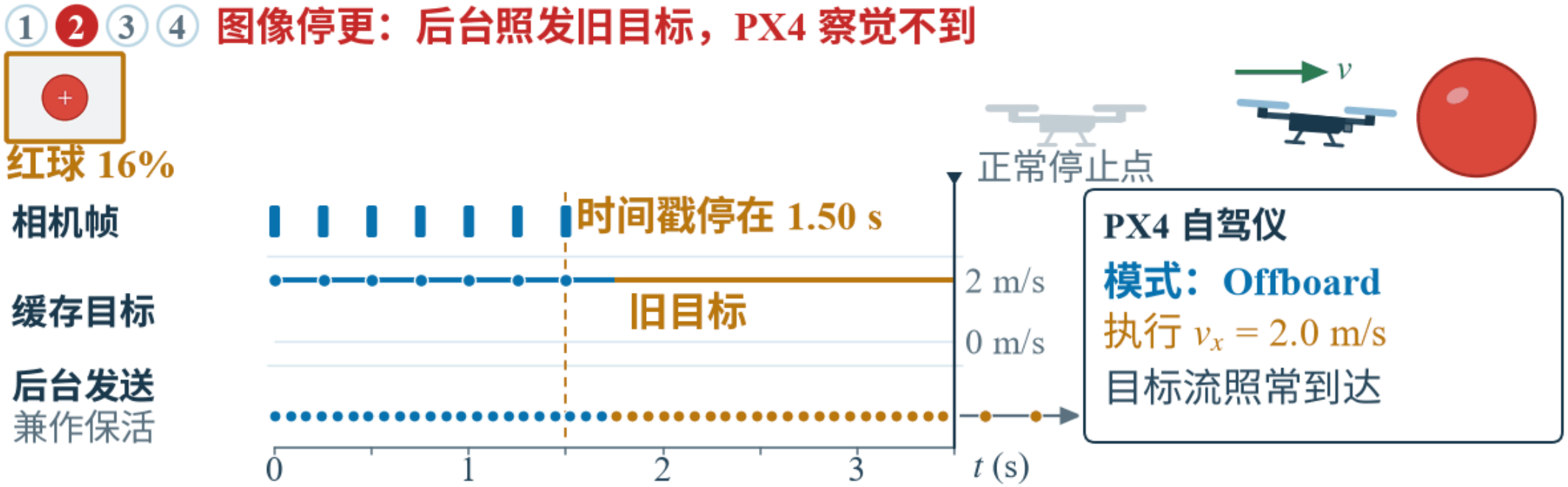
- **控制分工：**外部程序给出位置或速度目标，PX4依据状态估计计算控制输出；实际速度随控制器和载具响应变化。
- **模式条件：**位置、速度控制分别依赖有效的位置、速度估计；切入前建立外部控制保活流，切入后持续发送。
- **失联机制：**MAVLink目标流同时承担保活；ROS2单独发送OffboardControlMode。保活超时后退出模式并执行配置动作。
- **旧目标风险：**后台可持续发送缓存目标，即使视觉已停更。程序还需检查图像时间戳；观测超时应覆盖旧目标，转入预设保持或降落流程。

PX4 Offboard模式要求：https://docs.px4.io/main/en/flight_modes/offboard

后台持续发包时，视觉停更仍可能导致旧目标继续执行。

2.1.3 系统组成与通信：外部控制模式

视觉停更为何仍可能继续执行旧目标



图：同一起点的四种情形（简化仿真，非实测）

对照帧时间戳与发包点：持续发包不代表视觉仍在更新。

本章目录

关键工具：从运行环境到仿真闭环

第二章　实验平台配置

2.1　关键工具介绍

2.2　平台安装与首次仿真

2.3　平台架构与开发流程

2.4　API与SDK开发接口

2.5　例程体系与实验路线

2.6　习题与课堂讨论

本节 4 个教学单元

2.1.1 开发基础与运行环境

教材对应：第 2.1.2.2、2.1.2.3、2.1.2.7 节

2.1.2 模型与算法开发

教材对应：第 2.1.2.1、2.1.2.3、2.1.2.7 节

2.1.3 系统组成与通信

教材对应：第 2.1.2.4–2.1.2.6 节

2.1.4 仿真工具与开发路线

教材对应：第 2.1.2.1、2.1.2.8 节

2.1.4 仿真工具与开发路线：在环仿真

软件在环与硬件在环的关键差别

先比在环模式，再看场景与物理工具，最后按任务选路。

比较项	软件在环（SITL）	硬件在环（HITL）
控制程序	运行在计算机环境	运行在真实目标硬件
被控对象	计算机仿真模型	计算机仿真模型
主要用途	算法与功能快速迭代	板上运行、时序与接口联调
未覆盖范围	真实硬件时序与物理环境	真实动力、环境与整机行为

SITL在计算机上运行控制软件，HITL把真实控制硬件放入闭环。

2.1.4 仿真工具与开发路线：场景与系统仿真

三维引擎与机器人仿真平台

- **Unreal Engine**：通用三维引擎，利用几何、材质、光照和相机生成视觉观测。
- **Unity**：通用三维引擎；选学ML-Agents可将场景接入策略训练流程。
- **Gazebo与Webots**：面向机器人系统仿真，连接载具模型、虚拟传感器与控制程序。
- **RflySim中的分工**：CopterSim计算运动状态，RflySim3D据此更新场景与相机图像。
- **红球闭环**：运动改变目标在图像中的位置和大小，视觉程序据此更新控制目标。

三维引擎侧重场景与交互，机器人仿真还要连接物理、传感器和控制。

2.1.4 仿真工具与开发路线：三维场景

虚幻引擎与RflySim3D的分工

- **场景引擎：**虚幻引擎（UE）组织几何、材质、光照和相机，把场景渲染为图像。
- **平台集成：**RflySim3D基于UE实现可视化与虚拟传感器，接收载具位姿并更新场景。
- **运动计算：**CopterSim计算载具运动，状态驱动画面，视觉程序读取相机图像。
- **感知影响：**光照变化可影响颜色分割；模型参数先改变运动，再改变图像。

RflySim3D场景与接口：https://rflysim.com/doc/zh/3.Soft/3d-eng.html
UE场景、材质与相机文档：https://dev.epicgames.com/documentation/unreal-engine/

运动模型给出载具状态，三维引擎把场景和相机条件转成可用观测。

2.1.4 仿真工具与开发路线：选学·场景与机器人仿真

场景制作与机器人仿真入门教程

UE场景制作课程： https://dev.epicgames.com/community/learning
在课程中选择Unreal Engine，学习对象、关卡、材质和相机的设置。

Unity编辑器与交互课程： https://learn.unity.com/
从编辑器和基础项目入门，学习场景、脚本与交互。

Gazebo模型与传感器教程： https://gazebosim.org/docs/
从Getting Started学习世界、模型、传感器及控制接口。

Webots机器人与控制器教程： https://cyberbotics.com/doc/guide/index
从Tutorials学习机器人、环境和控制器的连接关系。

先根据场景制作或机器人仿真任务选一个工具，再完成它的最小示例。

2.1.4 仿真工具与开发路线：选学·机器人学习

机器人学习工具怎样分工

以强化学习为例：策略选择动作，训练算法根据奖励更新策略。

工具	主要作用	典型任务
MuJoCo / MJX	多体动力学与接触 MJX支持并行仿真	关节控制、接触运动与策略训练
Isaac Sim / Lab	Sim构建仿真环境 Lab组织学习任务	机器人感知与运动控制
Genesis	刚体与软体等物理仿真 构建机器人交互环境	机械臂操控、刚柔体交互

仿真提供动作后的状态与观测，任务奖励引导策略学习。

2.1.4 仿真工具与开发路线：选学·机器人学习资料

物理仿真与策略训练的入门资料

MuJoCo模型与MJX文档： https://mujoco.readthedocs.io/
先读Overview与模型示例，MJX用于相应的并行仿真工作流。

Isaac Sim场景与传感器教程： https://docs.isaacsim.omniverse.nvidia.com/
从Getting Started理解场景、机器人模型和传感器。

Isaac Lab任务与策略训练教程： https://isaac-sim.github.io/IsaacLab/
从Tutorials理解任务环境、观测、动作、奖励和训练接口。

Genesis实体与仿真教程： https://genesis-world.readthedocs.io/
从User Guide的基础示例理解实体、物理属性和仿真步进。

仿真器提供状态变化与观测，训练框架组织任务和策略更新。

2.1.4 仿真工具与开发路线：开发路线

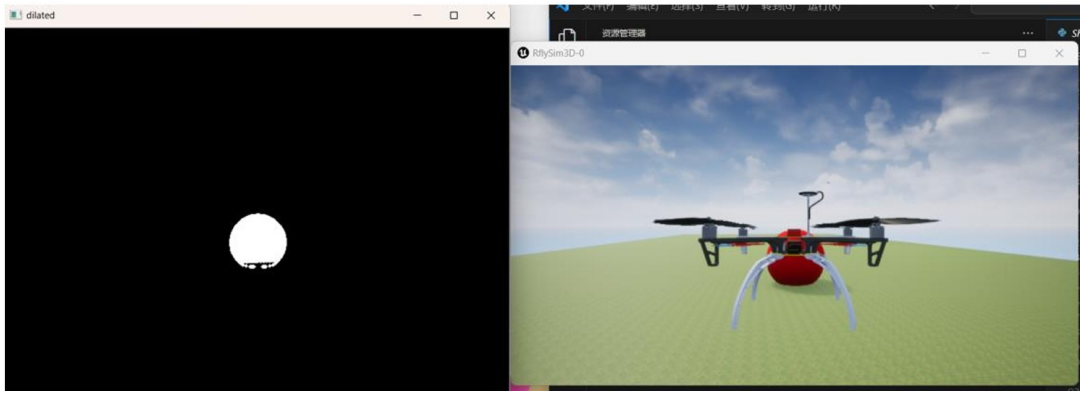
从开发任务选择工具与交付物

要完成的任务	主要工具	形成的产物
复现基础仿真	统一入口、CopterSim、地面站	运行记录与状态判读
修改视觉跟踪	Python、OpenCV、平台API	检测／控制脚本与对照结果
建模与控制设计	MATLAB、Simulink、编译工具	模型DLL或控制程序
集成机器人模块	Linux环境、ROS及适配接口	节点、消息与集成配置
学习型感知决策	训练框架、仿真与部署工具	数据、模型／策略与评测

先确定改动处在观测、运动还是控制环节，再选对应工具。

2.1.4 仿真工具与开发路线：例程贯通

贯穿例程：红球信息怎样变成跟踪动作



白色掩膜对应红球区域

- **获取观测：** VisionCaptureApi读取相机图像。
- **定位目标：** calc_centroid计算目标面积与质心。
- **生成动作：** controller把误差变成速度目标。
- **形成闭环：** SendVelFRD发目标，仿真返回新图像。

例程索引：第1章 → 基础实验 → e7_RunPythonProject

视觉提取目标误差，控制器据此改变运动，新图像再反馈跟踪效果。

2.1.4 仿真工具与开发路线：单参数对照

只改一个阈值，预测并验证运动差异

比较对象：ShootBall3.py 中的 max_prop（目标面积占比阈值）

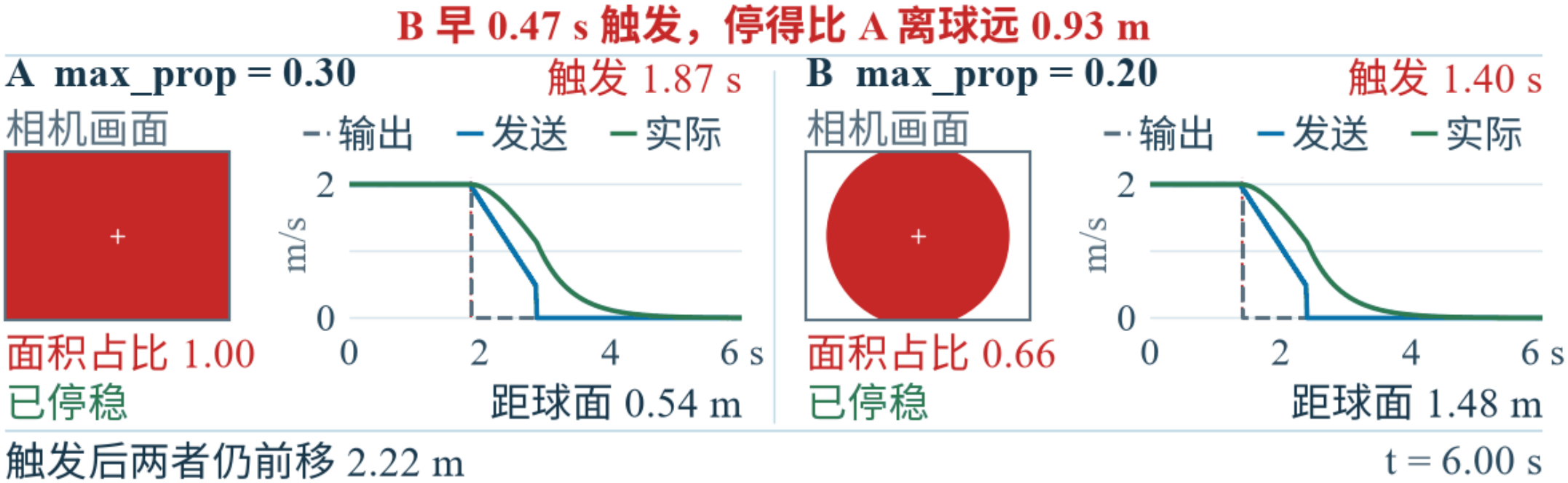
比较内容	方案A：0.30	方案B：0.20
初始条件	同一红球、机位与相机配置	沿用方案A条件
指令判断	占比达到30%时停止前进指令	占比达到20%时停止前进指令
待验证预测	作为基准，记录停止位置	更早开始减速，停得离球更远

记录：对齐目标占比、平滑前后速度目标和实际运动，分辨指令与响应。

面积阈值决定前进目标何时归零，实际减速还受指令平滑与载具响应影响。

2.1.4 仿真工具与开发路线：单参数对照

面积阈值怎样改变触发时刻与停止位置



图：仅改变面积阈值：A为0.30，B为0.20（简化仿真，非实测）

对照三条曲线：阈值更小会先撤销前进指令，载具随后减速。

2.1.4 仿真工具与开发路线：官方学习资料

围绕一个问题使用官方资料

当前问题	优先查什么	查阅入口
例程如何运行	Readme、启动文件、预期现象	RflySim文档
函数怎样使用	参数类型、返回值与最小示例	**Python** / **OpenCV**
目标如何被解释	消息字段、坐标系与模式支持	**MAVLink** / **PX4**
研究怎样扩展	物理、传感、接口与训练任务	仿真与机器人学习文档

RflySim课程与实验总目录：https://rflysim.com/doc/zh/

先读课程例程，再查当前用到的函数、消息与运行条件。

2.1.4 仿真工具与开发路线：分层学习

基础复现、算法修改与进阶开发

- **先复现实验：**使用统一入口运行例程，查看状态，说明脚本作用与实验结果。
- **再修改算法：**学习Python变量、函数与数组，处理一幅图像，再解释OpenCV输出怎样影响控制目标。
- **按任务补基础：**开发模型时学C++与DLL，集成机器人模块时学ROS及消息接口。
- **最后扩展研究：**场景制作、GPU推理和策略训练按需选学，记录输入、输出、运行条件及比较结果。

RflySim功能演示与案例视频
https://rflysim.com/doc/zh/0.Show/videos.html

ML-Agents策略训练（选学）
https://unity-technologies.github.io/ml-agents/

围绕当前任务逐步增加工具，每一步都用可观察的小结果检验理解。

本节思考与衔接

能否解释红球例程中的工具与数据

- **工具分工：**红球图像由谁产生、谁处理？PX4与CopterSim各负责哪段响应？
- **接口理解：**把颜色分割换成学习模型，哪些输入输出约定需要保持一致？
- **结果解释：**max_prop触发后速度目标已归零，载具仍移动，应该观察什么？
- **学习顺序：**复现这个Python例程，哪些工具现在需要，哪些可留待拓展？

下一步：2.2 平台安装与首次仿真

能追踪图像、目标与运动反馈，才真正理解例程的工具分工。