DAY1 真机定点飞行

1、要点理解

代码解析/wiki --> 仿真复现 --> 真机部署

1.1 ROS

ROS 是一个用于编写机器人软件的灵活框架。它是一个开源的元操作系统（meta-operating system），提供了类似于操作系统的服务，包括硬件抽象、底层设备控制、常用功能实现、进程间消息传递以及包管理等。它也提供了一系列工具和库，帮助软件开发者创建复杂且健壮的机器人行为。

ROS-本质是通讯方式

- **节点（Nodes）**：ROS中的每个组件都是一个节点，可以发布消息、订阅话题或提供服务。
- **话题（Topics）**：节点之间通过话题进行通信，类似于消息传递。
- **服务（Services）**：是节点之间的一种请求-响应通信方式。
- **动作（Actions）**：是服务的扩展，允许处理可能需要较长时间才能完成的请求。

1.2 MAVROS

MAVROS 是一个ROS包，它作为ROS和MAVLink协议之间的桥梁。MAVLink 是一种轻量级的通信协议，广泛用于无人机和地面站之间的通信。MAVROS 允许ROS节点与PX4飞控系统或其他支持MAVLink的飞控系统进行通信。

- **功能**：通过MAVROS，ROS节点可以发送控制命令到无人机，接收无人机的状态和传感器数据，实现高级控制和数据处理。
- **安装**：MAVROS可以通过二进制文件或源文件安装。推荐使用源文件安装，以获取最新的功能和改进。

1.3 飞控固件与飞控硬件

飞控固件（Flight Control Firmware）是运行在无人机或其他飞行控制系统上的**软件**，负责处理无人机的飞行控制逻辑和行为。它通过接收传感器数据，执行算法来控制无人机的飞行动作，确保无人机能够按照预定的路径或指令飞行。飞控固件通常包括以下功能：

- **传感器数据处理**：读取陀螺仪、加速度计、磁力计、气压计等传感器的数据。
- **姿态估计**：根据传感器数据计算无人机的当前姿态（例如俯仰角、横滚角、偏航角）。
- **控制算法**：实现控制算法（如PID控制器）以调整无人机的飞行姿态和位置。
- **导航**：根据GPS和其他传感器数据，实现路径规划和导航功能。
- **通信**：与其他设备（如地面控制站、其他无人机）进行通信。

PX4是一个开源的无人机飞控固件项目，广泛用于多种类型的无人机，包括多旋翼、固定翼和VTOL（垂直起降）飞机。

另外还有ArduPilot，ArduPilot是另一个流行的开源飞控固件

飞控硬件：飞控硬件是实际的物理设备，它包括微处理器、传感器（陀螺仪、加速度计、磁力计、气压计等）以及与其他组件通信的接口。常见的飞控硬件除了Pixhawk外，还包括：

- **APM（ArduPilot Mega）**：由3DRobotics开发，是ArduPilot飞控固件的硬件平台之一。
- **Omnibus**：另一款流行的飞控硬件，具有多种配置选项。

- **Nazea**：一款轻量级的飞控硬件，适用于小型多旋翼无人机。
- **KK2**：一款集成了FMU（飞行控制单元）和PMU（电源管理单元）的飞控硬件。
- **Betaflight**：主要用于穿越机（FPV无人机），也是一款流行的飞控硬件。

1.4 ROS与PX4例子

场景描述

假设我们想要通过ROS节点发送命令给PX4飞控，让无人机起飞并悬停在特定高度。

步骤概述

1. **设置ROS环境**：确保ROS环境已安装并配置好。
2. **安装MAVROS**：在ROS环境中安装MAVROS包，它作为ROS与PX4之间的通信桥梁。
3. **编写ROS节点**：创建一个ROS节点，用于发送控制命令。
4. **启动仿真或实体无人机**：使用Gazebo或连接实体无人机进行测试。

1.5 机载计算机与飞控

飞控（Flight Controller）：飞控是无人机的"大脑"，负责实时处理来自传感器的数据，并执行飞行控制算法来控制无人机的飞行姿态和轨迹。

机载计算机（Onboard Computer）：机载计算机是无人机上的一个附加计算平台，用于处理更复杂的任务，如图像处理、数据分析、机器学习等。

内部飞控和机载计算机的连接方式：物理层串口连接；软件协议使用MAVROS连接（MAVLINK<-->ROS）

MAVROS有传有收

1.6 数传

无人机中的数传，全称为"数据传输"（Data Transmission），指的是无人机与地面控制站（Ground Control Station, GCS）或遥控器（Remote Controller）之间传输数据的过程。这种数据包括但不限于：

1. **遥测数据**：无人机实时发送其状态信息给地面控制站，包括位置、速度、高度、姿态、电池电量、飞行模式等。
2. **传感器数据**：无人机上的各种传感器数据，如GPS信号、气压计数据、陀螺仪和加速度计数据等，也可以通过数传系统发送到地面。
3. **图像和视频**：无人机搭载的摄像头捕获的图像和视频流可以通过数传系统实时传输到地面。
4. **控制指令**：从地面控制站或遥控器发出的控制指令，如起飞、降落、改变飞行路径等，通过数传系统发送给无人机。
5. **软件更新和设置调整**：地面控制站可以无线更新无人机的飞行控制软件或调整设置参数。

数传系统通常使用以下技术实现：

- **无线电频率**：使用特定的无线电频率进行数据传输，这是最常见的数传方式。
- **Wi-Fi**：一些小型无人机使用Wi-Fi进行数据传输，但由于距离和干扰限制，通常用于短距离通信。
- **蜂窝网络**：利用移动通信网络进行数据传输，可以实现更远距离的通信。
- **卫星通信**：对于长航时或执行特殊任务的无人机，可能使用卫星通信系统进行数据传输。

数传系统的设计需要考虑以下因素：

- **传输速率**：数据传输的快慢，影响实时性能。
- **传输距离**：通信的范围，决定无人机的控制半径。
- **抗干扰能力**：抵抗其他无线电信号干扰的能力。
- **安全性**：数据传输过程中的加密和安全措施，防止被未授权访问。

- **可靠性**：系统在各种环境条件下的稳定性和可靠性。

数传是无人机系统中不可或缺的一部分，它确保了无人机与操作员之间的有效通信，是实现远程控制和监控的关键技术。

1.7 GPS与RTK

GPS

- GPS是一个全球卫星导航系统，由美国建立和维护，能够提供全球覆盖的定位服务。
- GPS系统通过至少4颗卫星发出的信号，接收器可以计算出地球上的精确位置（经纬度）和高度。
- 标准GPS服务的定位精度通常在几米范围内。

RTK

- RTK是一种利用GPS信号进行高精度定位的技术，它能够将定位精度提高到厘米级别。
- RTK技术通过实时处理两个GPS接收器（一个固定位置的基站和一个移动站）的数据，消除卫星信号中的大气延迟和误差，从而实现高精度定位。
- RTK需要一个已知位置的固定基站，它将校正数据发送给移动站，移动站利用这些校正信息来提高自身的定位精度。

RTK和GPS的关系

1. **技术基础**：RTK基于GPS技术，使用GPS信号作为其数据来源。
2. **精度提升**：RTK通过差分技术显著提高了GPS的定位精度。
3. **应用场景**：RTK通常用于需要高精度定位的应用，如土地测绘、精准农业、建筑施工、自动驾驶等。
4. **实时性**：RTK提供的是实时定位信息，适合动态定位需求。
5. **成本和复杂性**：RTK系统相对于标准GPS系统来说成本更高，技术也更复杂。
6. **基站依赖**：RTK操作需要一个基站作为参考点，而标准GPS不需要。
7. **信号遮挡问题**：RTK由于其高精度要求，对信号遮挡和多径效应更敏感，需要更精确的信号处理。

总结

RTK是GPS的一个高级应用，它通过差分定位技术显著提高了GPS的定位精度。虽然RTK系统更为复杂和昂贵，但它在需要高精度定位的领域中是不可或缺的。标准GPS服务适用于一般定位需求，而RTK服务则适用于对精度要求极高的专业应用。

1.8 串口与网口

问题：无人机通讯时的网口和串口，现在的无人机有飞控、机载计算机、两个minihomer天空端和遥控器和minihomer地面端

网口和串口是两种常见的通信接口，它们各自有不同的特点和用途。

网口（Ethernet Port）

1. **定义**：网口通常指的是以太网接口，它是一种有线通信方式，可以提供高速的数据传输。
2. **速度**：网口的传输速度通常很快，可以达到100Mbps或1000Mbps（1Gbps）。
3. **用途**：在无人机系统中，网口常用于连接高性能的机载计算机和飞控系统，以实现高速数据传输和实时控制。
4. **优点**：稳定性高，传输速度快，适合大量数据传输。
5. **缺点**：需要物理连接，不适合长距离通信。

串口（Serial Port）

1. **定义**：串口是一种串行通信接口，数据以串行方式传输，即一次传输一个比特。
2. **速度**：串口的传输速度相对较慢，常见的速率有9600bps、19200bps、115200bps等。
3. **用途**：串口在无人机系统中常用于连接飞控系统与其他传感器或外设，如GPS、IMU等。

4. **优点**：灵活性高，可以通过简单的线缆实现连接，适合短距离通信。
5. **缺点**：传输速度慢，不适合大量数据的快速传输。

无人机通信系统示例

以阿木实验室的P450机型为例，无人机的通信系统可能包含以下几个部分：

- **飞控系统**：无人机的大脑，负责处理飞行数据和控制无人机的飞行。
- **机载计算机**：可能用于处理更复杂的任务，如图像处理、数据分析等。
- **Mini homer天空端**：用于空中定位和通信，可能与飞控系统或机载计算机通过网口或串口连接。
- **遥控器**：操作员通过遥控器发送指令给飞控系统，通常使用无线通信。
- **Mini homer地面端**：用于地面定位和通信，可能与遥控器或其他地面设备连接。

在实际应用中，无人机的各个组件之间的通信可能会根据具体需求和设计选择使用网口或串口。例如，如果需要高速传输大量数据，可能会选择网口；如果只需要传输少量控制信号，可能会选择串口。

通信协议（遥控器与地面端）

无人机的遥控器和地面端设备之间的通信协议主要有以下几种：

1. **MAVLink协议**：
   - **定义**：MAVLink（Micro Air Vehicle Link）是一种轻量级的通信协议，专门为无人机系统设计，用于在无人机和地面站之间传输数据和命令。
   - **特点**：MAVLink协议具有良好的扩展性和兼容性，能够适应不同的硬件平台和应用需求。它可以传输位置数据、姿态数据、传感器数据等，并支持发送和接收命令。MAVLink协议在数据传输中采用了包的方式，每个包包含有关数据类型和数据内容的信息。数据内容可以根据具体的应用需求进行定义和扩展。
   - **传输介质**：支持不同的传输介质，如串口、无线电链路等。MAVLink设定了心跳包机制，可用于检测无人设备与地面站之间连通性的检测。123
2. **PWM（Pulse Width Modulation）协议**：
   - **定义**：PWM是一种脉宽调制信号，主要用于控制单个电调和舵机。
   - **特点**：PWM信号是一个周期性的方波信号，周期为20ms，高电平的持续时间代表控制量。例如，1100us对应0油门，1900us对应满油门。PWM信号具有较高的抗干扰能力，传输过程全部使用满电压传输，非0即1，类似于数字信号。
   - **应用**：广泛用于接收机与舵机、电调的控制。456
3. **PPM（Pulse Position Modulation）协议**：
   - **定义**：PPM是一种脉冲位置调制信号，用于传输多个通道的数据。
   - **特点**：PPM将多个通道的数值一个接一个合并进一个通道，用2个高电平之间的宽度来表示一个通道的值。PPM协议最多传输10个通道，使用一个定时器就可以轻松解决。
   - **应用**：适用于需要集中获取接收机的多个通道的值再做其他用途的场合，如教练模式、模拟器、多轴无人机等。46
4. **S.BUS（Serial Bus）协议**：
   - **定义**：S.BUS是一种串行通信协议，最早由日本厂商FUTABA引入。
   - **特点**：S.BUS使用RS232C串口的硬件协议，采用TTL电平，负逻辑，100K的波特率。每11个位表示一个通道数值，串口通信。S.BUS一帧数据的长度为25个字节，包含16个伺服通道的数据和其他控制信息。
   - **应用**：适用于与飞控连接，获取遥控器上所有通道的数据。

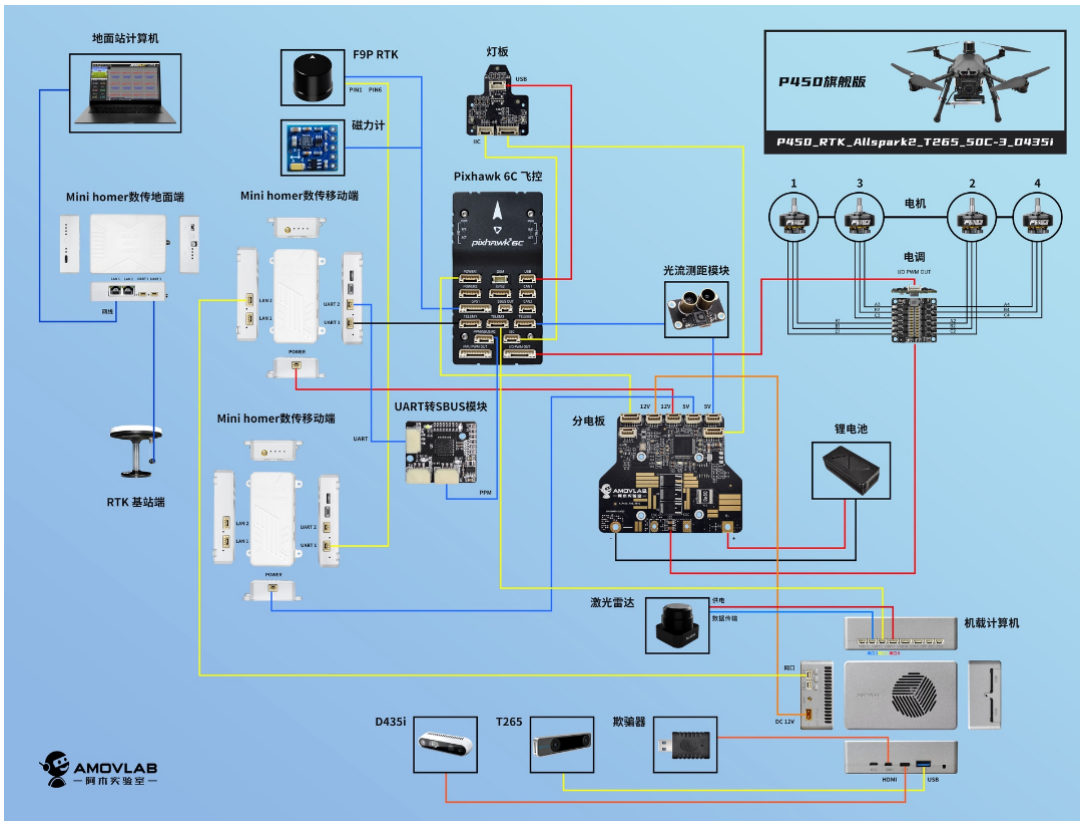
2、amov P450相关

EKF 一堆传感器数据做位姿估计

视觉追踪使用相机 D435i 最远距离10m--**做识别**

T265--**做定位**

还有一个光流，在T265失效时可定位，但无里程内容



机载计算机 NVIDIA Jetson Orin NX,CPU是arm核不是x86，显存和内存共用

amov自己做的底板和外壳，核心板是NVIDIA，将核心板插到底板上

minihomer 双工通信 每一个（地面站、机载计算机）都有自己的ip地址，不是传统的点对点通信

遥控器美国手 左摇杆是上下，表示提高/降低无人机高度；右摇杆上下左右表示前后左右

QGC-飞控地面站

prometheus-机载计算机地面站

no machine-进入机载电脑

DAY2 仿真

仿真系统

模拟环境-Gazebo；对无人机物理模型及运动控制模型搭建-PX4-SITL

1、 .sh --> .launch --> .cpp/.py

1. `.sh` 脚本

`.sh` 是Shell脚本文件的扩展名，通常用于Linux操作系统中的脚本编程。在ROS中，`.sh` 脚本可能用于初始化环境变量、启动ROS节点或执行一些自动化任务。

2. `.launch` 文件

`.launch` 文件是ROS中用于启动多个节点的配置文件。它允许你定义一个或多个节点的启动参数，包括节点的名称、类型、参数设置等。`.launch` 文件通常由ROS的 `roslaunch` 工具调用，它能够解析这些配置并启动相应的节点。

3. `.cpp` 或 `.py` 文件

这些文件是实际编写机器人行为逻辑的源代码文件。`.cpp` 是C++语言的源文件，而 `.py` 是Python语言的源文件。在ROS中，这些文件定义了节点的逻辑，包括订阅话题、发布话题、服务调用等。

关系描述

- `.sh` 脚本可以设置环境并启动 `.launch` 文件。
- `.launch` 文件定义了要启动的节点和它们的参数。
- `.cpp` 或 `.py` 文件包含了节点的实现代码。

例子

假设我们正在开发一个简单的ROS机器人项目，它包括一个用于检测障碍物的节点和一个用于控制机器人移动的节点。

1. `.sh` **脚本** (`start_robot.sh`):

```
sh#!/bin/bash
source /opt/ros/noetic/setup.bash  # 设置ROS环境
roslaunch my_robot robot.launch  # 启动机器人的.launch文件
```

2. `.launch` **文件** (`robot.launch`):

```
xml<launch>
    <node name="obstacle_detector" pkg="my_robot"
type="obstacle_detector.cpp" output="screen" />
    <node name="motion_controller" pkg="my_robot"
type="motion_controller.py" output="screen" />
</launch>
```

3. `.cpp` **文件** (`obstacle_detector.cpp`):

```
cpp// 包含ROS头文件和障碍物检测逻辑
#include "ros/ros.h"
// ... 节点实现代码 ...
```

4. `.py` **文件** (`motion_controller.py`):

```
python#!/usr/bin/env python
import rospy
# ... 节点实现代码 ...
```

在这个例子中，当你运行 `start_robot.sh` 脚本时，它会首先设置ROS环境，然后调用 `robot.launch` 文件启动两个节点：`obstacle_detector` 和 `motion_controller`。这两个节点分别由 `obstacle_detector.cpp` 和 `motion_controller.py` 文件实现。

2、如何编译

要进入到该目录下，拖动至terminal或命令行执行 ./compile_xx.sh

.yaml .CMakelist XML文件

DAY3 目标检测

1、问题相关

输入图像，输出检测到的图像的位置

打印list：

```
rostopic list
```

摄像头image_raw信息--> d435i有信息 -->yolov5有信息

识别到的是小车特征，跟踪的是二维码

2、执行、启动

```
chomd +x *.launch
```

设置执行权限，使得所有 `.launch` 文件都可执行

具有执行权限后可以使用roslaunch来启动

```
roslaunch package_name launch_file.launch
```

在ROS中启动一个包

3、cmake理解

3.1 make cmake Makefile catkin_make

1. **Make**:
   - Make 是一个构建自动化工具，它使用 Makefile（一个包含指令的脚本文件）来管理和自动化软件的编译过程。
   - Makefile 定义了一组规则和依赖关系，指导 make 如何编译项目。
   - Make 是一个比较传统的工具，适用于较小的项目或者那些不需要复杂构建系统的场景。
2. **CMake**:
   - CMake（Cross-platform Make）是一个跨平台的自动化构建系统，它使用 CMakeLists.txt 文件来描述项目的构建过程。
   - CMake 能够生成标准的 Makefile 或其他类型的构建文件（比如 Ninja），这些文件随后可以被 make 或其他构建工具使用。
   - CMake 特别适合于大型项目，支持复杂的依赖关系和多种编译器和平台。
3. **cmake：**

- 在ROS中，通常首先使用`cmake`命令来配置构建系统，这会在工作空间的`build`目录中生成Makefile。
- 然后，使用`make`命令来编译项目，这会根据Makefile中的指令来编译源代码。
- 在catkin工作空间中，`catkin_make`是一个封装了上述两个步骤的命令，它会自动调用cmake和make来构建整个工作空间。

3.2 流程

1. 在工作空间的根目录下，运行`catkin_make`。
2. `catkin_make`会查找所有的包和依赖，然后运行`cmake`来生成Makefile。
3. 接着，`catkin_make`会调用`make`来编译所有的包。
4. 最后，`catkin_make`还会设置环境，使得编译出的节点可以被ROS找到和运行。

3.3 CMakeLists.txt

功能：

`CMakeLists.txt`定义了项目的构建逻辑，而 CMake 是执行这些逻辑的工具，它读取`CMakeLists.txt`并生成必要的构建文件，以便于编译和构建项目

举例：

```
# 指定CMake的最低版本要求
cmake_minimum_required(VERSION 3.10)

# ##定义项目名称和语言
project(MyProject VERSION 1.0 LANGUAGES CXX)
# project(prometheus_demo)

# 设置C++标准
set(CMAKE_CXX_STANDARD 14)
set(CMAKE_CXX_STANDARD_REQUIRED True)

# ##Find catkin macros and libraries
# find_package(catkin REQUIRED COMPONENTS
  # message_generation
  # roscpp
  # rospy
  # geometry_msgs
  # sensor_msgs
  # mavros
  # nav_msgs
  # std_msgs
  # std_srvs
  # tf2_ros
  # tf2_eigen
  # mavros_msgs
  # prometheus_msgs
# )

# 包含头文件目录
include_directories(include)

# ##添加一个可执行文件
add_executable(MyExecutable src/main.cpp src/helper.cpp)
# add_executable(takeoff_land basic/takeoff_land/src/takeoff_land.cpp)

# 链接一个库
target_link_libraries(MyExecutable PRIVATE someLibrary)
# target_link_libraries(takeoff_land ${catkin_LIBRARIES})

# 启用测试
enable_testing()
```

```
# 添加测试
add_test(NAME MyTest COMMAND MyExecutable)

# 安装规则
install(TARGETS MyExecutable RUNTIME DESTINATION bin)
```

3.4 举例

```
prometheus_demo/
|
├── src/                    # 源代码目录
|   └── prometheus_demo/    # 包目录
|       ├── include/        # 头文件目录
|       |   └── prometheus_demo/   # 包的头文件子目录
|       |       └── takeoff_land.h  # 示例头文件
|       ├── src/            # 源文件目录
|       |   └── takeoff_land.cpp   # 示例源文件
|       ├── launch/         # launch文件目录
|       |   └── takeoff_land.launch  # 示例launch文件
|       ├── scripts/        # 脚本文件目录
|       |   └── takeoff_land.sh  # 示例shell脚本
|       └── CMakeLists.txt  # 构建配置文件
|
├── build/                  # 构建目录，由catkin_make生成
|   ├── CMakeFiles/         # CMake生成的文件
|   ├── prometheus_demo/    # 与源代码目录结构对应的构建目录
|   |   └── CMakeFiles/   # 特定于包的CMake文件
|   └── Makefile            # 顶层Makefile
|
├── devel/                  # devel空间，由catkin_make生成
|   ├── lib/                # 存放编译后的库文件
|   ├── etc/                # 存放环境设置脚本
|   └── share/              # 存放其他共享资源
|
├── CMakeLists.txt          # 顶层CMakeLists.txt，配置整个工作空间
└── package.xml             # 包的元数据文件
```

步骤 1: 创建ROS工作空间

首先，你需要创建一个ROS工作空间。这通常包含三个目录：`src`、`build` 和 `devel`。

```
mkdir -p prometheus_demo/src
cd prometheus_demo
catkin_make
```

步骤 2: 创建包和源文件

在 `src` 目录中创建一个新的ROS包 `prometheus_demo` 以及相应的 `CMakeLists.txt` 和 `package.xml` 文件。然后，添加 `takeoff_land.cpp` 源文件。

在 `prometheus_demo/src/prometheus_demo` 目录中：

- `CMakeLists.txt` 示例：

```
cmake_minimum_required(VERSION 3.0)
```

```
project(prometheus_demo)

# 找到catkin宏和库
find_package(catkin REQUIRED COMPONENTS
  roscpp
  std_msgs
)

# 定义一个可执行文件
add_executable(takeoff_land src/takeoff_land.cpp)

# 链接catkin库
target_link_libraries(takeoff_land ${catkin_LIBRARIES})

# 包含头文件
include_directories(${catkin_INCLUDE_DIRS})
```

- `takeoff_land.cpp` 示例：

```
#include "ros/ros.h"
#include "std_msgs/String.h"

void messageCallback(const std_msgs::String::ConstPtr& msg)
{
  ROS_INFO("I heard: [%s]", msg->data.c_str());
}

int main(int argc, char **argv)
{
  ros::init(argc, argv, "takeoff_land");
  ros::NodeHandle n;
  ros::Subscriber sub = n.subscribe("chatter", 1000, messageCallback);
  ros::spin();
  return 0;
}
```

- `package.xml` 示例：

```
<package>
  <name>prometheus_demo</name>
  <version>0.0.0</version>
  <description>A demo package for prometheus</description>

  <maintainer email="your.email@example.com">Maintainer Name</maintainer>

  <license>BSD</license>

  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>std_msgs</build_depend>
  <run_depend>roscpp</run_depend>
  <run_depend>std_msgs</run_depend>
</package>
```

步骤 3: 添加脚本和launch文件

在 `prometheus_demo/src/prometheus_demo` 目录中添加 `takeoff_land.sh` 和 `takeoff_land.launch` 文件。

- `takeoff_land.sh` 示例：

```
bash#!/bin/bash
source /opt/ros/<ros_version>/setup.bash
rosrun prometheus_demo takeoff_land
```

- `takeoff_land.launch` 示例：

```
xml<launch>
  <!-- 启动takeoff_land节点 -->
  <node name="takeoff_land" pkg="prometheus_demo" type="takeoff_land"
output="screen" />
</launch>
```

确保 `takeoff_land.sh` 脚本具有执行权限：

```
chmod +x prometheus_demo/src/prometheus_demo/takeoff_land.sh
```

步骤 4: 构建工作空间

回到工作空间的根目录，运行 `catkin_make` 来构建整个工作空间。

```
cd ..
catkin_make
```

步骤 5: 源代码环境

构建完成后，你需要源代码环境，以便能够运行你的ROS节点。

```
source devel/setup.bash
# source 用于读取并执行指定文件
```

Q：为什么还需要源代码环境?

即使构建完成后，你仍然需要设置**源代码环境**，原因如下：

1. **环境变量**：构建过程可能会修改环境变量，如 `PATH`、`LD_LIBRARY_PATH` 等，以确保系统能够找到新编译的可执行文件和库。
2. **ROS环境**：ROS需要特定的环境变量来运行节点，如 `ROS_PACKAGE_PATH` 来查找包，`ROS_MASTER_URI` 指向ROS Master节点。
3. **依赖关系**：新编译的包可能依赖于其他包，需要确保所有依赖包的环境都正确设置。Q

步骤 6: 运行节点

```
rosrun prometheus_demo takeoff_land
```

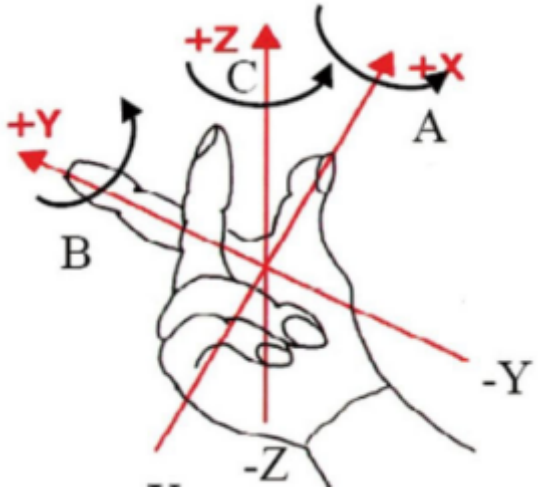
步骤 7: 运行launch文件

使用 `roslaunch` 命令启动 `takeoff_land.launch` 文件。

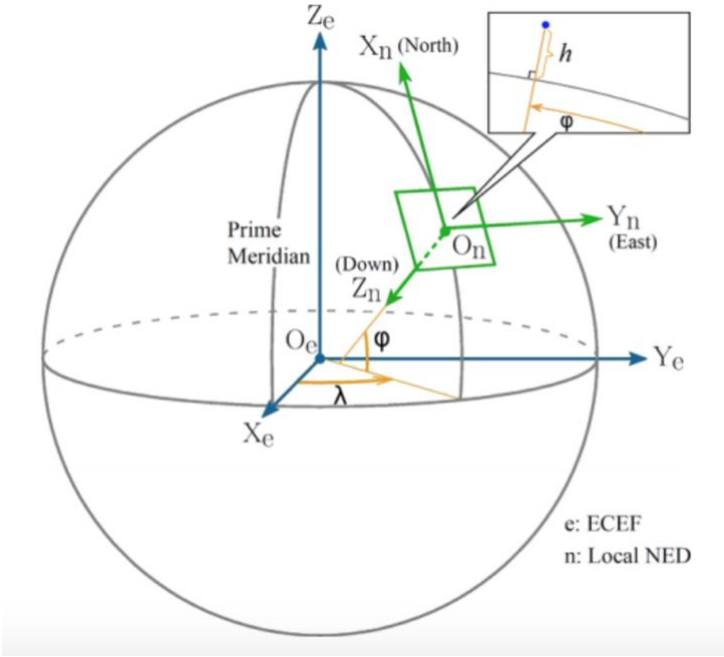
```
roslaunch prometheus_demo takeoff_land.launch
```

4、相机坐标系、机体坐标系

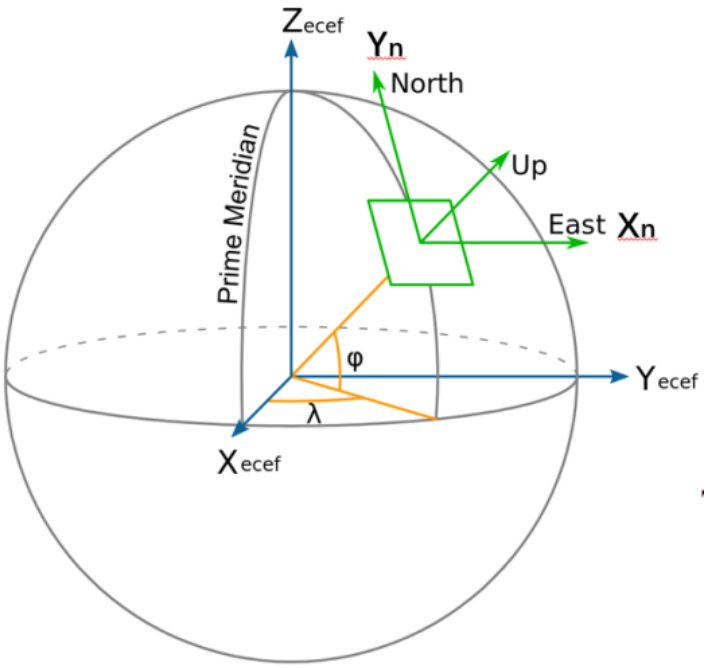
4.1 右手定则



4.2NED惯性系（北东地）-PX4原生坐标系



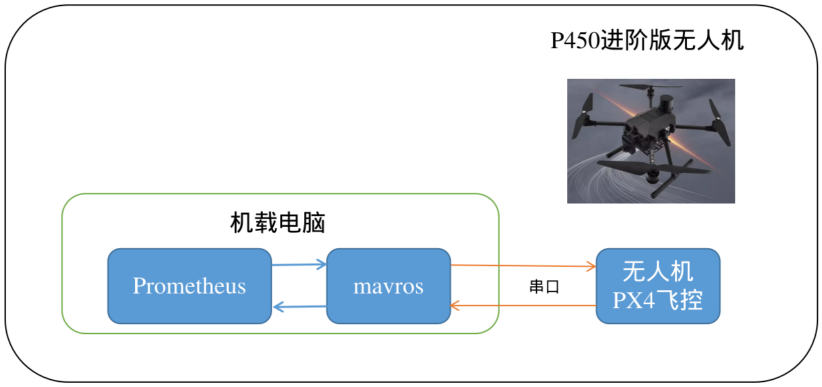
4.3 ENU惯性系（东北天）-mavros常使用

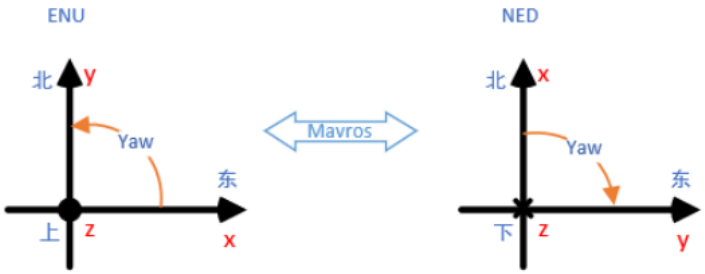


4.4 坐标系转换：机载电脑prometheus <-->飞控PX4

从mavros通信中会涉及到ENU惯性系（东北天）与NED惯性系（北东地）之间坐标的互相转换

- 当机载电脑中Prometheus获取飞控中定位数据，则是NED惯性系（北东地）转换为ENU惯性系（东北天）
- 当机载电脑中Prometheus向飞控发送位置控制指令，则是ENU惯性系（东北天）转换为NED惯性系（北东地）





从数值上看NED惯性系（北东地）转换ENU惯性系（东北天），就是xy轴数据互换，z轴数据取反。Prometheus地面站显示xyz坐标数据便是ENU惯性系（东北天）下，对应PX4原生坐标系数据xy轴数据互换，z轴数据取反。

$$Yaw_{NED} = -Yaw_{ENU} + 90$$

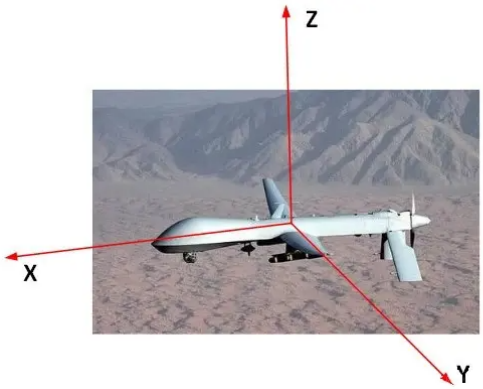
$$X_{NED} = Y_{ENU}$$

$$Y_{NED} = X_{ENU}$$

$$Z_{NED} = -Z_{ENU}$$

4.5 机体系 -- mavros常使用

mavros下常使用坐标系-机体系。body系（前左上），以无人机当前位置为原点，机头向前方向为x轴，垂直机头90度向左为y轴，垂直xy平面向上为z轴



4.6 转换代码

```
// 对无人机的控制代码部分
// 相机系下：前方Z为正，右方x为正，下方y为正
// g_command_now.velocity_ref[0] 为无人机的机体坐标系的x，指向前， 与相机坐标系的z轴相关，所以与相机坐标系的z轴为线性关系。
// g_command_now.velocity_ref[1] 为无人机的机体坐标系的y，指向左，与相机坐标系的x轴相关，为相机坐标系相反的方向。所以与相机坐标系的px为线性关系。并且指向相反，所以前面加负号。
// g_command_now.velocity_ref[2] 为无人机的机体坐标系的z，指向上。与相机坐标系的py相关。为相机坐标系py相反的方向。所以与相机坐标系py呈线性关系。
g_command_now.velocity_ref[0] = 0.5 * (g_Detection_raw.pz - 2.0);
//无人机x轴的控制参考量，0.5为比例系数，如果调大，无人机的前行跟踪速度会变快。调小，降低跟踪速度。
//pz代表相机坐标系下目标离相机的直线距离。2.0设置为保持2米的距离。
//为什么是正的0.5，相机离目标的距离变小，代表无人机离目标的距离也要变小，所以是正数。
g_command_now.velocity_ref[1] = -0.8 * g_Detection_raw.px;
//无人机y轴的控制参考量
//px代表相机系下，向右为正。
//无人机的坐标系，向N/左方向飞行。与Px的向右为正相反。所以要加负号。0.8调整跟踪速度的快慢，不需要保持比例。
//无人机z轴的控制参考量
//相机系下，向下py为正。
//目标中心点的视线角度 [相机系下：右方x角度为正，下方y角度为正] 单位：度
//无人机往上飞，视线角ay会向下，为正值。
//如果需要保持某一个视线范围内，需要调整无人机。
//当无人机的视线角小于-20，代表无人机看到的目标是在图像中心的上方，所以需要调整无人机往上飞。
if(g_Detection_raw.los_ay <= -20)
{
    g_command_now.velocity_ref[2] = 0.8 * g_Detection_raw.py;
}
//当无人机的实现角大于-15，代表无人机过高了，所以需要调整无人机往下飞。
else if(g_Detection_raw.los_ay >= -15)
```

```
{
    g_command_now.velocity_ref[2] = -0.8 * g_Detection_raw.py;
}
//其他情况下，不控制无人机的z轴上下的速度。
else{

    g_command_now.velocity_ref[2] = 0;
}
```

5、rqt-graph 节点关系图

6、RViz image-raw

RViz 是 ROS（Robot Operating System）中的一个可视化工具，它允许用户查看和交互机器人的传感器数据和状态信息。在ROS中，使用RViz查看相机画面或视频推流通常涉及以下步骤：

1. **安装依赖项**：首先需要安装一些ROS的包，比如`usb_cam`用于USB摄像头，以及`image_view`用于显示图像。
2. **运行摄像头节点**：启动摄像头节点，它会从USB摄像头读取图像数据并发布到ROS话题上，例如`/usb_cam/image_raw`。
3. **使用RViz显示图像**：打开RViz界面，添加一个"Image"显示类型，并设置其"Image Topic"为摄像头发布图像的话题，如`/usb_cam/image_raw`，这样就能在RViz中看到摄像头的实时图像了。

DAY4 实验

1、OSI七层网络复习

应用层 -- prometheus地面站（为应用软件提供网络服务）

MAVROS MAVLINK

HTTP（网络服务的最终用户接口）

传输层 -- TCP/UDP （端到端的数据传输，保证可靠）

网络层 -- Socket IP（不可靠）（负责数据包从源到目的地的传输和路由选择；使用ip地址进行设备间的通信）

物理层 -- minihomer 900MHz（频段带宽窄，传输距离远，能够穿墙）

串口/WIFI

2、室内无人机光流定位飞行

概述&原理

光流算法：

1. 通过下视摄像头获得图像数据，分析图像的不同时刻的帧数据，得到像素的移动速度；
2. 将像素的移动速度转换成飞行器的移动速度；
3. 光流传感器输出的是xy两个轴向的**速度数据**，位置可以通过速度积分获得（不可避免会产生漂移）

如何判定图像的运动? 选择对应的特征点，通过特征点的运动来判定图像的运动。

如何根据前后两个像素点估计出运动?

- 光流算法的两个假设，即认为两个时刻的**运动很小**和**亮度恒定不变**
- 基于梯度的方法：利用时变图像灰度（或其滤波形式）的时空微分（即时空梯度函数）来计算像素的速度矢量

得到光流输出的数据，飞控又如何使用?

step1 预处理（注意与高度的线性关系；姿态补偿；先积分再微分，得到速度数据。好处是滤波+数据标定）

step3 位置估计算法

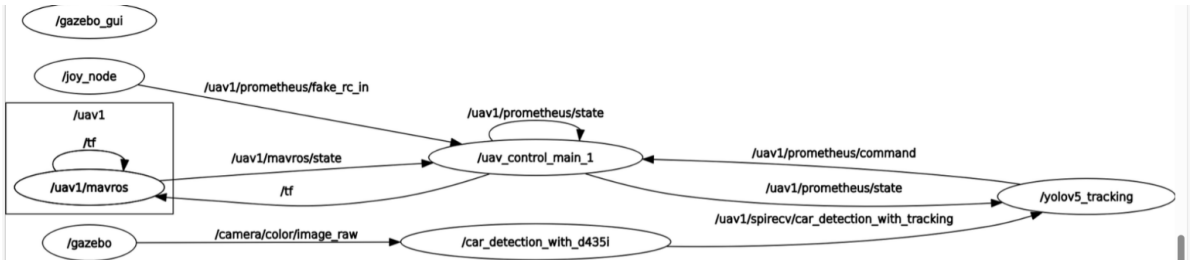
详细：https://mp.weixin.qq.com/s/JGPSqJ_oimw0de1FMN189A

3、代码解析-car_detection_with_tracking_d435i

4、代码解析-yolov5_tracking

DAY5 比赛日

1、节点-以yolo追踪人为例



主要包含`/joy_node`、`/uav1/mavros`、`/uav_control_main_1`、`/yolov5_tracking`、`/car_detection_with_d435i`、`/gazebo`等ROS节点。

- `/joy_node`节点为**遥控器ROS驱动节点**，用以获取遥控器数据,通过话题`uav1/prometheus/fake_rc_in`将遥控器数据传输给`/uav_control_main_1`节点。
- `/uav1/mavros`节点为**飞控ROS驱动节点**，与飞控进行数据交互。在仿真中，该驱动节点与模拟飞控进行数据交互。
- `/uav_control_main_1`节点为Prometheus项目中**最基础**的ROS节点，所有Prometheus项目的功能模块都通过该节点与无人机进行数据交互。
- `/gazebo`节点为**gazebo ROS驱动节点**，为`/car_detection_with_d435i`节点**提供实时图像**`/camera/color/image_raw`，后期检测
- `/car_detection_with_d435i`节点为**SpireCV 检测节点**，识别图像`/camera/color/image_raw`中是否有通用目标-person，并输出目标位置等信息`/uav1/spirecv/car_detection_with_d435i`

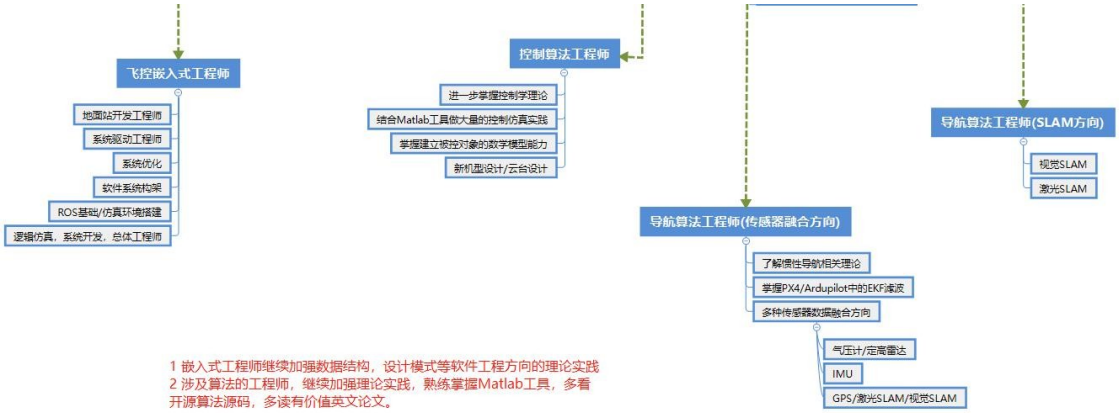
- `/yolov5_tracking`节点为**yolov5跟踪节点**，通过接收`/car_detection_with_d435i`检测数据，来进行目标跟踪发送`/uav1/prometheus/command`控制命令，让无人机跟踪目标。

从节点运行图中，我们可以看到有如下的数据话题

- /uav1/prometheus/command：无人机控制接口，对应的消息为`prometheus_msgs/UAVCommand`
- /uav1/prometheus/state：无人机状态，对应的消息为`prometheus_msgs/UAVState`
- /uav1/prometheus/fake_rc_in：仿真模拟PX4遥控器数据
- /uav1/spirecv/car_detection_with_d435i：SpireCV检测输出目标位置等信息
- /camera/color/image_raw:仿真D435i摄像头采集到的实时图像画面

见：https://docs.amovlab.com/Prometheus_Simulation_Development_Kit_wiki/#/src/%E4%BD%BF%E7%94%A8%E8%AF%B4%E6%98%8E/%E4%BB%BF%E7%9C%9FDemo/%E4%BB%BF%E7%9C%9FDemo-%E4%BA%8C%E7%BB%B4%E7%A0%81%E7%82%B9%E5%87%BB%E8%B7%9F%E8%B8%AA

2、bbs社区 知识体系



1 嵌入式工程师继续加强数据结构，设计模式等软件工程方向的理论实践
2 涉及算法的工程师，继续加强理论实践，熟练掌握Matlab工具，多看开源算法源码，多读有价值英文论文。