1.实验名称及目的

1.1 实验名称

Apm飞控环境部署与固件下载编译实验

1.2 实验目的

主要讲解如何部署Apm飞控环境以及固件下载编译的方法。

1.3 关键知识点

无

2.实验效果

能够正确部署Apm飞控环境以及完成固件下载编译

3.文件目录

例程目录：[安装目录]\RflySimAPIs\2.RflySimUsage\0.ApiExps\e7_ApmBuild

文件夹/文件名称	说明
environment_install	环境配置文件

4.运行环境

4.1 软件要求

Windows 10及以上版本；RflySim工具链。

> ①：若使用Pixhawk 6X飞控，平台安装时的编译命令为：px4_fmu-v6x_default，推荐PX4固件版本为：1.12.3。其他配套飞控及编译命令请见：https://rflysim.com/doc/zh/1/Hardware.html

4.2 硬件要求

笔记本/台式电脑① 1台。

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

5.实验步骤

Apm开发环境的部署方法

0.ApiExps\e7_ApmBuild\environment_install中的文件，是从Apm源码中提取出来的环境配置脚本，由于PX4和APM的gcc_arm编译器版本有区别，本平台对脚本进行了拆分。

1）将environment_install文件夹拷贝到Ubuntu系统中.

2）在拷贝到Ubuntu系统的environment_install文件夹中打开终端，运行install-prereqs-ubuntu.sh就能完成依赖环境的配置，运行命令为./install-prereqs-ubuntu.sh。

3）等待上一条命令运行完毕后，同样在文件夹中打开终端，运行gcc_arm.sh，就能安装gcc_arm-10版编译器，并注册环境，运行命令为./gcc_arm.sh。

如果上述步骤中出现下面的提示，按照提示运行命令就行了：E: dpkg被中断，您必须手工运行‘sudo dpkg --configure -a’解决此问题。

注：如果虚拟机配置困难，可以直接使用 **WinWSL 编译器** 进行实验。

如果使用 **RflySim 自带的 WSL（Ubuntu）**测试，可以省略以上步骤，直接进行以下实验，实验完成后需要 **重置 WSL 环境**，避免影响后续实验。

我们同样配置的有已经部署Apm开发环境的虚拟机，打包放在百度网盘，如果有需要可从百度网盘中进行下载使用。链接:https://pan.baidu.com/s/1gqc7i8vzL_i_DXRRL3Ix-Q?pwd=y9rt 提取码: y9rt。我们所配置的虚拟机账号以及密码均为：nvidia。与NX保持一致，方便后续向真机转

移。克隆官方源码PX4
源代码存储在 Github 的ArduPilot/Autopilot 存储库中。

要将最新版本（“main”）获取到您的计算机上，请在终端中输入以下命令**（注：运行之前，需要保证您的电脑可以正常访问Github网站）**：

git clone --recursive https://github.com/ArduPilot/ardupilot.git

之后，导航到ardupilot目录

cd ardupilot

之后，需要构建ArduCopter

./waf configure --board CubeBlack

./waf copter

第一个命令应该只调用一次，或者在您想要更改配置选项时调用。一种经常使用的配置是从一个板切换到另一个板的选项。例如，我们可以切换到SkyViper
GPS无人机并重新构建：--board

等待进行编译，如果编译成功，说明环境配置正确。

注：后续会在百度网盘中上传，如果有网络问题，可以直接进行下载，之后在进行编译。

Apm源码学习

APM （ArduPilot
Mega）是一款开源的飞控系统，其源码可以在GitHub上找到。本文将对APM飞控系统的源码进行讲解。

在飞行控制模块中，最重要的是飞行姿态控制算法。源码中实现了多种不同的姿态控制算法，包括经典的PID控制器和先进的模型预测控制器。这些算法通过读取飞行器的传感器数据，如加速度计和陀螺仪，来计算飞行器当前的姿态，并根据目标姿态进行调整。源码中还实现了飞行器的导航控制算法，通过GPs数据和地面站指令，实现导航目标的控制。

另一个重要的模块是传感器数据采集模块，用于获取飞行器的传感器数据。源码中实现了对加速度计、陀螺仪、磁力计等传感器的读取和数据处理。通过对传感器数据的处理，可以获取飞行器的姿态、加速度、角速度等信息，用于姿态控制和导航控制。

通信模块负责与地面站进行通信，传输飞行器的状态和控制指令。源码中实现了多种通信方式，包括串口通信和无线通信，以适应不同的应用场景。通过与地面站的通信，可以实时监

控飞行器的状态，并发送控制指令进行调整。

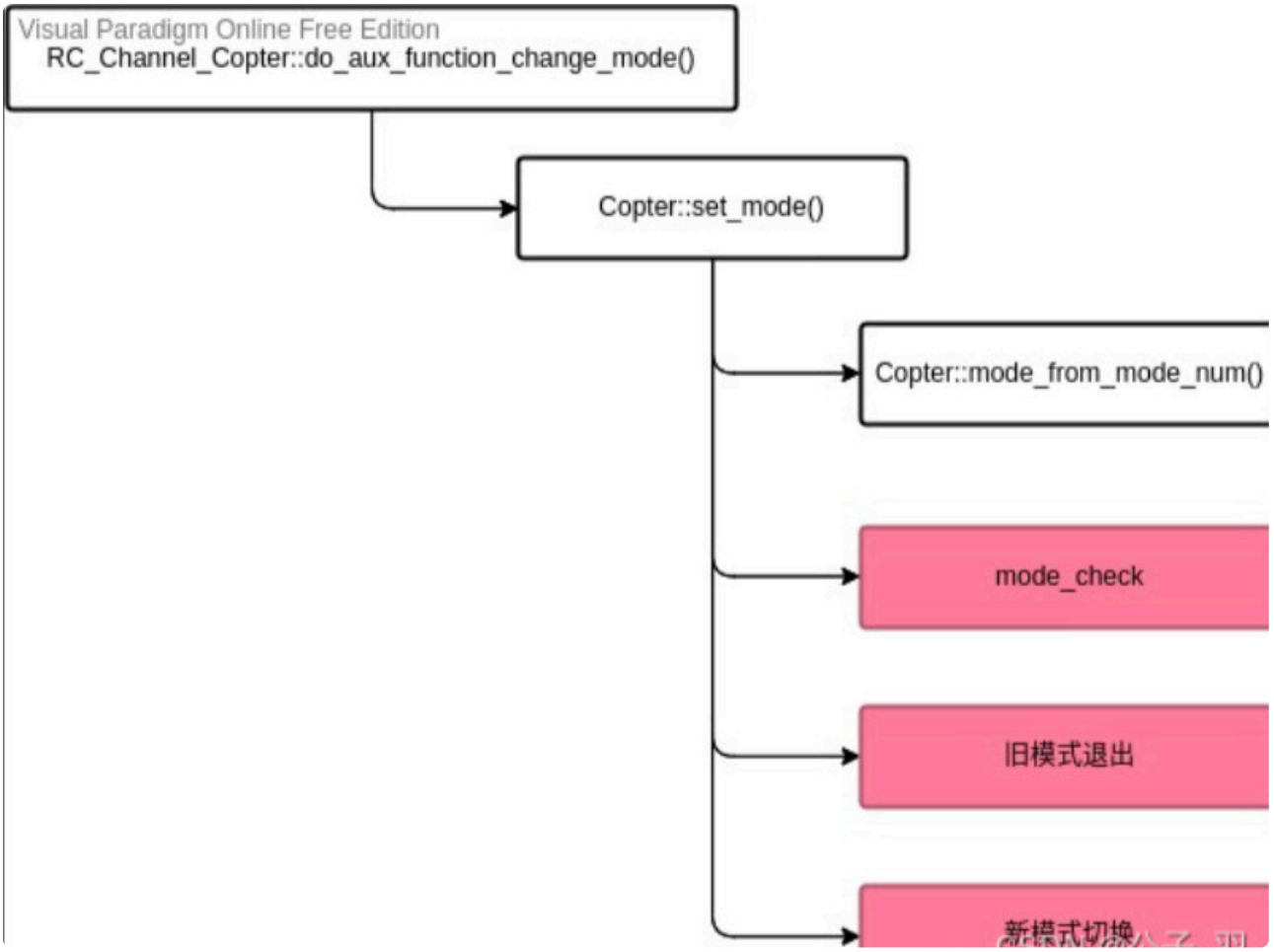
此外，源码中还包括一些辅助功能模块，如传感器校准、飞行任务管理、参数配置等。这些模块可以帮助用户对飞行器进行配置和管理，提供更加精确和可靠的飞行控制。

总之，APM飞控系统的源码实现了一套完整的飞行控制系统，包括姿态控制、传感器数据采集、通信等关键功能。通过对源码的深入理解和学习，可以帮助开发者更好地应用和开发飞行控制系统，实现更高级的飞行功能和应用。

APM飞控中有很多种方式可以改变飞行模式。其中最常见的为遥控器切换和地面站切换两种方式。

```cpp
// interface to set the vehicles mode
/* 设置飞行模式的缘由 */
enum class ModeReason : uint8_t {
    UNKNOWN,
    RC_COMMAND,
    GCS_COMMAND,
    RADIO_FAILSAFE,
    BATTERY_FAILSAFE,
    GCS_FAILSAFE,
    EKF_FAILSAFE,
    GPS_GLITCH,
    MISSION_END,
    THROTTLE_LAND_ESCAPE,
    FENCE_BREACHED,
    TERRAIN_FAILSAFE,
    BRAKE_TIMEOUT,
    FLIP_COMPLETE,
    AVOIDANCE,
    AVOIDANCE_RECOVERY,
    THROW_COMPLETE,
    TERMINATE,
    TOY_MODE,
    CRASH_FAILSAFE,
    SOARING_FBW_B_WITH_MOTOR_RUNNING,
    SOARING_THERMAL_DETECTED,
    SOARING_THERMAL_ESTIMATE_DETERIORATED,
    VTOL_FAILED_TRANSITION,
    VTOL_FAILED_TAKEOFF,
    FAILSAFE, // general failsafes, prefer specific fail
    INITIALISED,
    SURFACE_COMPLETE,
    BAD_DEPTH,
    LEAK_FAILSAFE,
    SERVOTEST,
```

下面以遥控器切换为例详解飞行模式切换逻辑。



APM遥控器切换飞行模式的源代码在RC_Channel.cpp文件中的RC_Channel_Copter::do_aux_function_change_mode()函数中。

```
switch(ch_flag) {
case AuxSwitchPos::HIGH: {
    // engage mode (if not possible we remain in current flight mode)
    const bool success = copter.set_mode(mode, ModeReason::RC_COMMAND);
    if (copter.ap.initialised) {
        if (success) {
            AP_Notify::events.user_mode_change = 1;
        } else {
            AP_Notify::events.user_mode_change_failed = 1;
        }
    }
    break;
}
default:
    // return to flight mode switch's flight mode if we are currently
    // in this mode
    if (copter.flightmode->mode_number() == mode) {
        rc().reset_mode_switch();
```

const bool success = copter.set_mode()调用Copter::set_mode()函数，在set_mode()函数中调用

mode_from_mode_num()函数进行预切换。new_flightmode不会立即切换到新模式，而是会根据新模式所需的设备状态判断是否要切换过去。

```
#if FRAME_CONFIG != HELI_FRAME
    // ensure vehicle doesn't leap off the ground if a user switches
    // into a manual throttle mode from a non-manual-throttle mode
    // (e.g. user arms in guided, raises throttle to 1300 (not enough to
    // trigger auto takeoff), then switches into manual):
    // 如果驾驶员从非手动油门模式切换到手动油门模式，确保飞行器不会离开地面。
    // 例如：驾驶员在引导模式下解锁，将油门提高到1300(不足以触发起飞)，然后切换到手动模式。
    bool user_throttle = new_flightmode->has_manual_throttle();
#if MODE_DRIFT_ENABLED == ENABLED
    if (new_flightmode == &mode_drift) {
        user_throttle = true;
    }
#endif
    if (!ignore_checks &&
        ap.land_complete &&
        user_throttle &&
        !copter.flightmode->has_manual_throttle() &&
        new_flightmode->get_pilot_desired_throttle() > copter.get_non_takeoff_throttle()) {
        mode_change_failed(new_flightmode, "throttle too high");
        return false;
    }
#endif

    if (!ignore_checks &&
        new_flightmode->requires_GPS() &&
        !copter.position_ok()) {
        mode_change_failed(new_flightmode, "requires position");
        return false;
```

第一个检查：飞机在地面上时，遥控器的油门值不能太高，太高则不能切换。

第二个检查：有的飞行模式(定点、任务模式)需要GPS正常才能运行，检查GPS状态。

第三个检查：EKF高度估计是否正常

如果以上内容检查通过了，则退出旧模式，设置新模式。

```
if (!new_flightmode->init(ignore_checks)) {
    mode_change_failed(new_flightmode, "initialisation failed");
    return false;
}

// perform any cleanup required by previous flight mode
// 清理上一个模式所需要的东西
exit_mode(flightmode, new_flightmode);

// store previous flight mode (only used by tradeheli's autorotation
// 存储上一模式(仅用于传统直升机自动旋转)
prev_control_mode = flightmode->mode_number();

// update flight mode
/* 这里是真正的模式切换的地方 */
flightmode = new_flightmode;
control_mode_reason = reason;
logger.Write_Mode((uint8_t)flightmode->mode_number(), reason);
gcs().send_message(MSG_HEARTBEAT);
```

在模式切换中有个问题，姿态自稳模式和定点模式切换的时候，如何避免大角度的突变？

答：在APM飞控中只做了Z轴的输出值平滑，XY水平方向上没做。因此在实际飞行过程中，从姿态自稳模式切换到定点模式的时候，会看到飞机抖一下。

在exit_mode()函数中set_accel_throttle_I_from_pilot_throttle()函数用于输出平滑。简而言之，通过设置PID控制器的积分项，达到输出值的平滑。

```
327  // exit_mode - high level call to organise cleanup as a flight mode is exited
328  void Copter::exit_mode(Mode *&old_flightmode,
329                         Mode *&new_flightmode)
330  {
331      // smooth throttle transition when switching from manual to automatic flight modes
332      if (old_flightmode->has_manual_throttle() && !new_flightmode->has_manual_throttle() && motors->armed() && !ap.land_complete) {
333          // this assumes all manual flight modes use get_pilot_desired_throttle to translate pilot input to output throttle
334          set_accel_throttle_I_from_pilot_throttle();
335      }
336
337      // cancel any takeoffs in progress
338      old_flightmode->takeoff_stop();
339
340      // perform cleanup required for each flight mode
341      old_flightmode->exit();
342
343  #if FRAME_CONFIG == HELI_FRAME
344      // firmly reset the flybar passthrough to false when exiting acro mode.
345      if (old_flightmode == &mode_acro) {
346          attitude_control->use_flybar_passthrough(false, false);
347          motors->set_acro_tail(false);
348      }
349
350      // if we are changing from a mode that did not use manual throttle,
351      // stab col ramp value should be pre-loaded to the correct value to avoid a twitch
352      // heli_stab_col_ramp should really only be active switching between Stabilize and Acro modes
353      if (!old_flightmode->has_manual_throttle()){
354          if (new_flightmode == &mode_stabilize){
355              input_manager.set_stab_col_ramp(1.0);
356          } else if (new_flightmode == &mode_acro){
357              input_manager.set_stab_col_ramp(0.0);
358          }
359      }
360  #endif //HELI_FRAME
```

```
// set_accel_throttle_I_from_pilot_throttle - smoothes transition from pilot controlled throttle to autopilot throttle
void Copter::set_accel_throttle_I_from_pilot_throttle()
{
    // get last throttle input sent to attitude controller
    float pilot_throttle = constrain_float(attitude_control->get_throttle_in(), 0.0f, 1.0f);
    // shift difference between pilot's throttle and hover throttle into accelerometer I
    pos_control->get_accel_z_pid().set_integrator((pilot_throttle-motors->get_throttle_hover()) * 1000.0f);
```

将flightmode父类指针指向新的子类，在Copter::fast_loop()函数中以400Hz的频率调用flightmode->run()。

因此，每种飞行模式的运行频率都是400Hz。

1. Apm的核心架构

APM（Autopilot Management System，自动驾驶飞行管理系统）飞控系统是指用于无人机等飞行器上的一套自动控制和管理系统。其核心架构通常包括以下几个关键组成部分：

1. 飞行控制器（Flight Controller）：作为飞控系统的核心，飞行控制器负责处理传感器数据，执行飞行算法，控制无人机的姿态和导航。它通常包括一个或多个微处理器，运行专门的飞行软件。

2. 传感器模块：包括陀螺仪、加速度计、磁力计、高度计、barometer气压计、GPS等，用于获取无人机的位置、速度、姿态和高度等信息。

3.

执行器模块：执行器包括电机、伺服、液压或气动装置，根据飞行控制器的指令来控制无人机的运动，如调整螺旋桨转速、改变推力等。

4.

电源管理：无人机电源管理系统负责监控和分配电源，确保飞控系统和其他电子设备在飞行过程中有稳定的电力供应。

5.

通信模块：用于处理与地面控制站、其他无人机或系统的通信，包括数据传输、指令接收和响应发送。

6.

导航与制导：利用GPS、惯性导航系统（INS）、视觉辅助或其他辅助系统来确定无人机的当前位置，并规划航线。

7.

任务管理：管理无人机的任务执行，如拍照、数据采集、目标跟踪等，并确保任务按照预定计划执行。

8.

安全与冗余系统：包括故障检测、故障切换和冗余设计，确保在关键系统失效时无人机能够安全飞行或安全着陆。

9.

软件系统：飞控系统的软件包括飞行控制算法、数据处理程序、任务调度程序等，它们运行在飞行控制器上，控制着无人机的所有飞行活动。

这些组成部分共同工作，确保无人机能够安全、稳定和高效地执行任务。APM飞控系统的设计需要考虑无人机的类型、任务需求、环境条件等多种因素，以实现最优的飞行性能和任务完成能力。

固件装机使用的简单步骤

Build命令有一个--upload选项，用于将构建的二进制文件上传到连接的板上。Pixhawk和基于Linux的板支持此选项。下面的命令使用--targets选项，这将在下一项中进行解释。

./waf --targets bin/arducopter –upload

对于Linux板，您需要首先配置要上传到的板的IP。这在配置阶段完成，包括：

./waf configure --board <board> --rsync-dest <destination>

下面的命令给出了一个具体的示例（板和目标IP将根据使用的板而变化）：

./waf configure --board navio2 --rsync-dest root@192.168.1.2😉

./waf --target bin/arducopter –upload

这允许设置一个目标，--upload选项将二进制文件上载到该目标。在引擎盖下，它安装到一个临时位置，并调用rsync＜temp_install_location＞/＜destination＞。

在Linux板上还有一个install命令，它将安装到某个目录，就像上面的临时安装一样。分销商可以使用它来创建.deb、.rpm或其他包类型：

./waf copter

DESTDIR=/my/temporary/location ./waf install

6.参考资料

1. 无。

7.常见问题

Q:在进行APM环境部署以及获取源码失败?

N:检查网络是否正常，在可能的情况下，尝试使用代理再次运行。