PX4软件系统简介

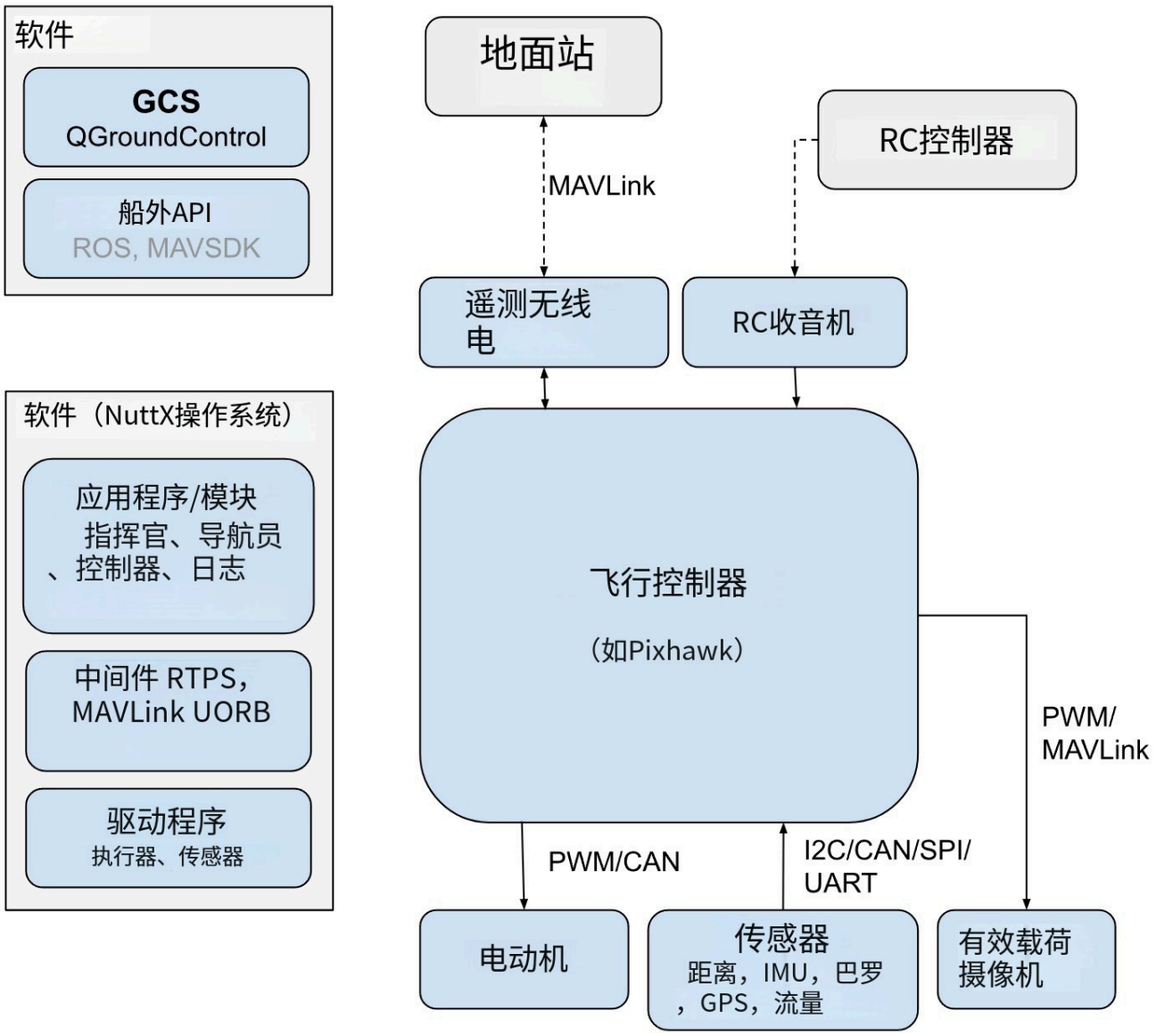
PX4由瑞士苏黎世联邦理工学院(ETH)计算机视觉与几何实验室的一个软硬件项目PIXHAWK演变而来，该飞控系统完全开源，为全球各地的飞控爱好者和研究团队提供一款低成本高性能的高端自驾仪。经过来自工业界和学术界的世界级开发人员多年的开发与完善，目前PX4飞控系统已经形成完善合理的软件架构，可支持单旋翼、多旋翼、飞艇等多种载具。官网链接为： https://docs.px4.io/main/zh/

PX4架构

以下各节简要概述了两个“典型”PX4 系统的 PX4
硬件和软件堆栈;一个只有一个飞行控制器，另一个有一个飞行控制器和一个配套计算机（也称为“任务计算机”）。

飞行控制器

下图提供了基于飞行控制器的典型“简单”PX4 系统的高级概述。



硬件包括：

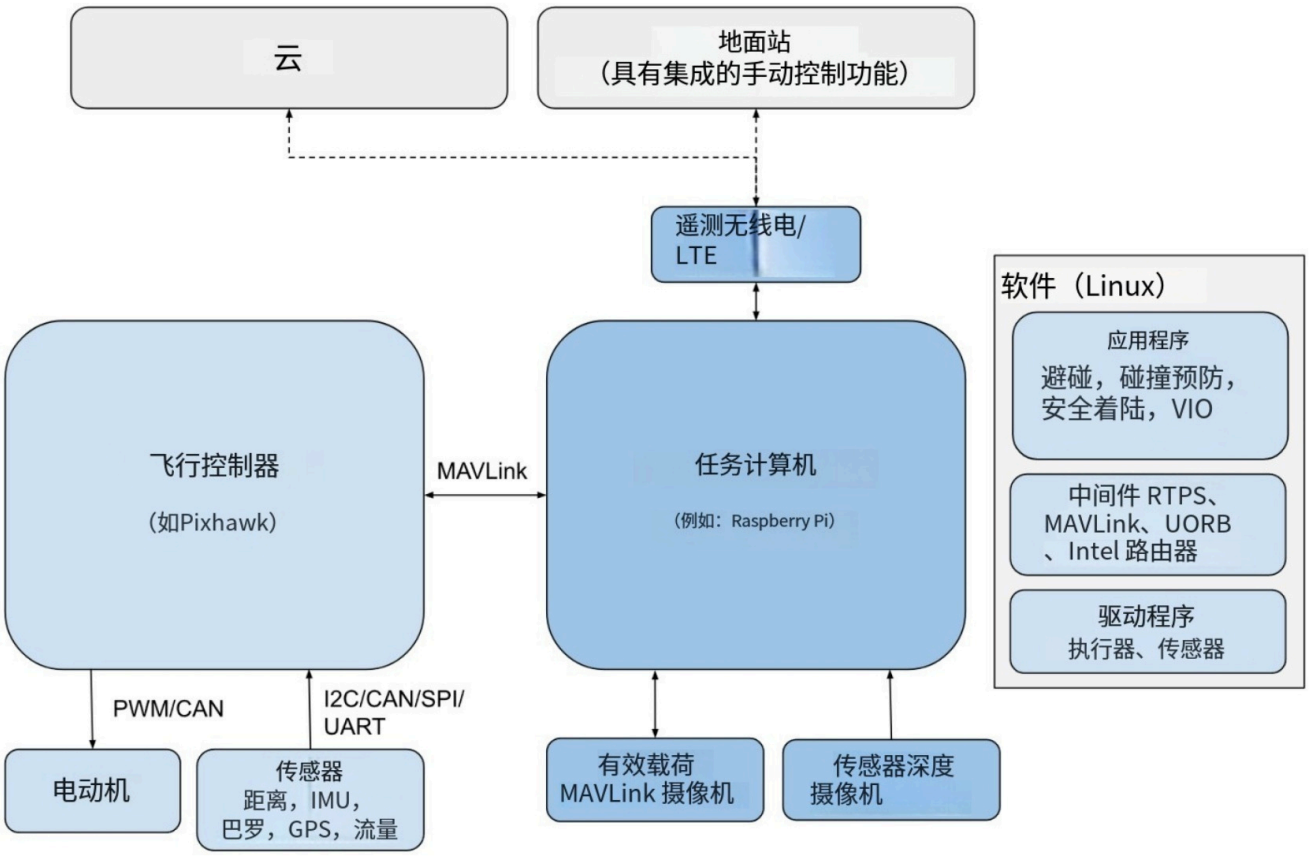
飞行控制器（运行 PX4 飞行堆栈）。这通常包括内部 IMU、指南针和气压计。连接到 PWM
输出、DroneCAN（DroneCAN
允许双向通信，而不是如图所示的单向通信）或其他总线的电机

ESC。通过I2C、SPI、CAN、UART等连接的传感器（GPS、指南针、距离传感器、气压计、光流计、气压计、ADSB应答器等）。相机或其他有效载荷。相机可以连接到
PWM 输出或通过
MAVLink。用于连接到地面站计算机/软件的遥测无线电。用于手动控制的RC控制系统。

图的左侧显示了软件堆栈，它与图的硬件部分水平对齐（大约）。地面站计算机通常运行QGroundControl（或其他一些地面站软件）。它还可以运行
MAVSDK 或 ROS
等机器人软件。运行在飞控上的PX4飞行堆栈包括驱动器、通信模块、控制器、估计器等中间件和系统模块。

飞行控制器和配套计算机

下图显示了一个 PX4
系统，该系统包括一个飞行控制器和一个配套计算机（此处称为“任务计算机”）。



飞行控制器运行正常的 PX4
飞行堆栈，而伴侣计算机则提供物体回避和碰撞预防等高级功能。这两个系统通过快速串行或
IP 链路连接，通常使用 MAVLink
协议进行通信。与地面站和云的通信通常通过配套计算机进行路由（例如，使用MAVLink路由器（来自Intel））。

PX4 系统通常在配套计算机上运行 Linux 操作系统（因为 PX4/PX4-Avoidance
项目提供了为 Linux 设计的基于 ROS 的回避库）。Linux 是比 NuttX
更好的“通用”软件开发平台;有更多的Linux开发人员，并且已经编写了很多有用的软件（例如，用于计算机视觉，通信，云集成，硬件驱动程序）。出于同样的原因，伴侣计算机有时运行
Android。

注：该图显示了通过 LTE 的云或地面站连接，这种方法已被许多基于 PX4
的系统使用。PX4 不提供专门用于 LTE 和/或云集成的软件（这需要定制开发）。

PX4软件框架

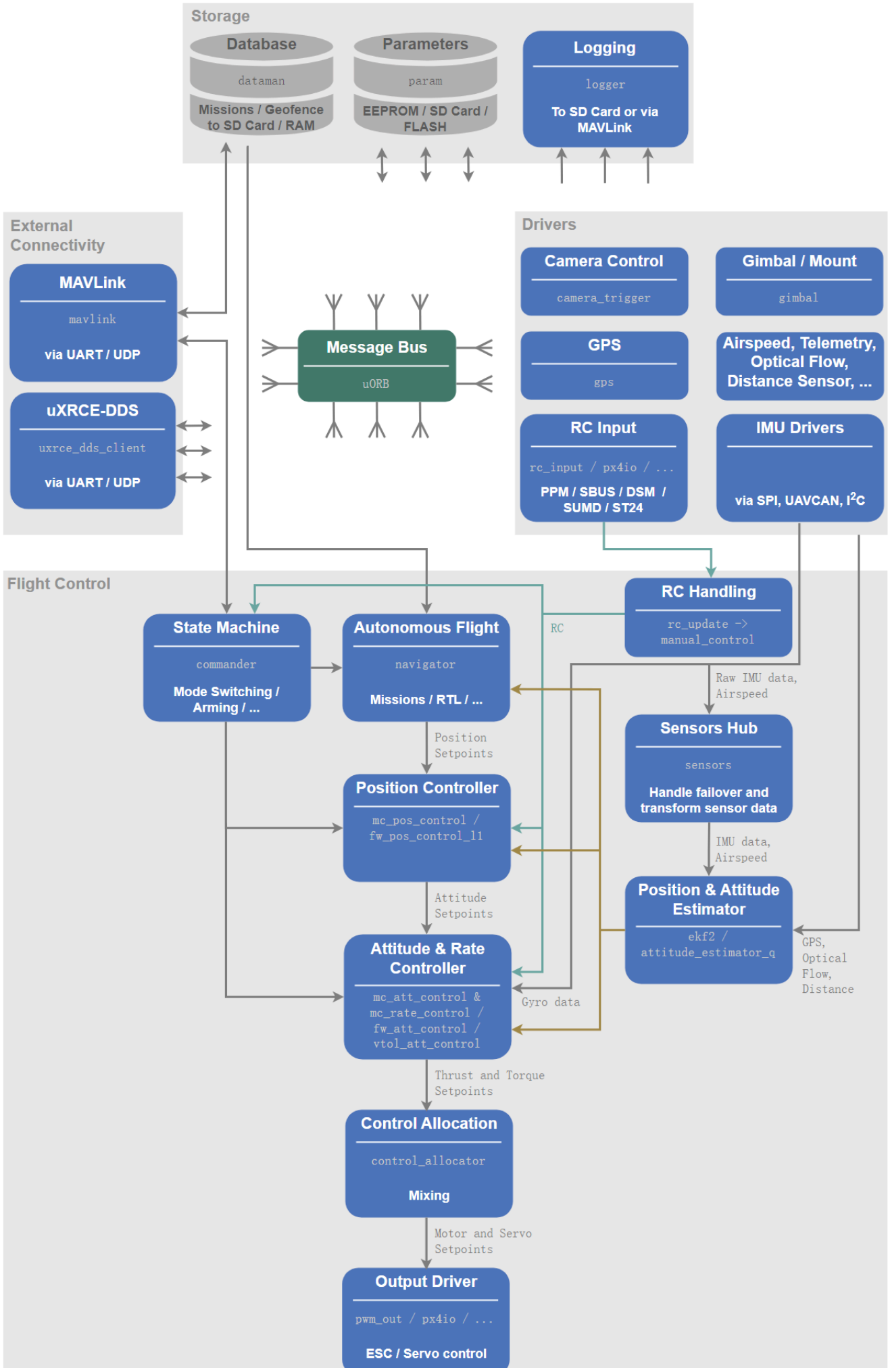
PX4 由两个主要部分组成：一是**飞行控制栈**（flight stack）
，该部分主要包括状态估计和飞行控制系统；另一个是 **中间件**

，该部分是一个通用的机器人应用层，可支持任意类型的自主机器人，主要负责机器人的内部/外部通讯和硬件整合。所有的 PX4 支持的 无人机机型
（包括其他诸如无人船、无人车、无人水下航行器等平台）均共用同一个代码库。
整个系统采用了 响应式（reactive）(opens new window)设计，这意味着：

- 所有的功能都可以被分割成若干可替换、可重复使用的部件。
- 通过异步消息传递进行通信。
- 系统可以应对不同的工作负载。

顶层软件架构

下面的架构图对 PX4 的各个积木模块以及各模块之间的联系进行了一个详细的概述。
图的上半部分包括了中间件模块，而下半部分展示的则是飞行控制栈的组件。


Storage
Database
dataman
Missions / Geofence to SD Card / RAM
Parameters
param
EEPROM / SD Card / FLASH
Logging
logger
To SD Card or via MAVLink
External Connectivity
MAVLink
mavlink
via UART / UDP
uXRCE-DDS
uxrce_dds_client
via UART / UDP
Message Bus
uORB
Drivers
Camera Control
camera_trigger
Gimbal / Mount
gimbal
GPS
gps
Airspeed, Telemetry, Optical Flow, Distance Sensor, ...
RC Input
rc_input / px4io / ...
PPM / SBUS / DSM / SUMD / ST24
IMU Drivers
via SPI, UAVCAN, I²C
Flight Control
RC Handling
rc_update -> manual_control
RC
State Machine
commander
Mode Switching / Arming / ...
Autonomous Flight
navigator
Missions / RTL / ...
Raw IMU data, Airspeed
Sensors Hub
sensors
Handle failover and transform sensor data
Position Setpoints
Position Controller
mc_pos_control / fw_pos_control_l1
IMU data, Airspeed
Position & Attitude Estimator
ekf2 / attitude_estimator_q
GPS, Optical Flow, Distance
Attitude Setpoints
Attitude & Rate Controller
mc_att_control & mc_rate_control / fw_att_control / vtol_att_control
Gyro data
Thrust and Torque Setpoints
Control Allocation
control_allocator
Mixing
Motor and Servo Setpoints
Output Driver
pwm_out / px4io / ...
ESC / Servo control

源代码被拆分为许多相互独立的模块/程序（如图所示）。通常来说一个图中的积木块对应一个功能模块。

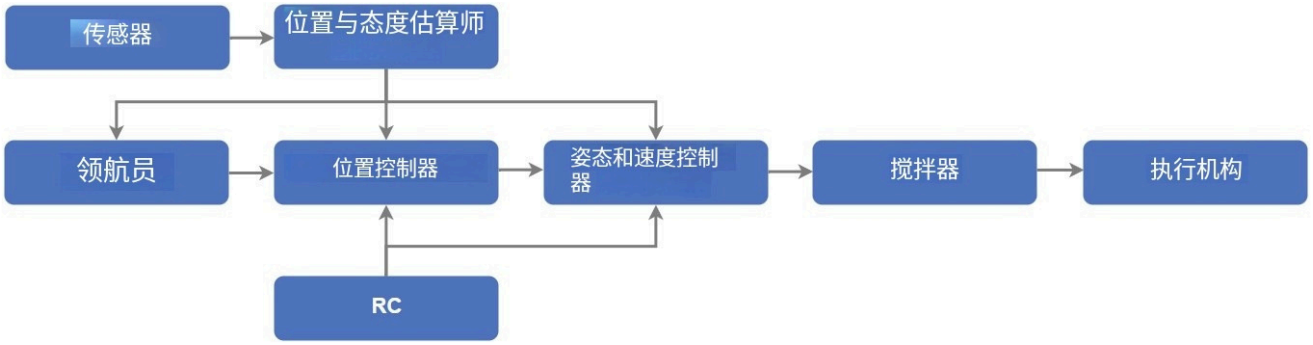
上图中的箭头表示的是各个模块之间 最重要的
信息流连接。实际运行时各模块之间信息流的连接数目比图中展示出来的要多很多，且部分数据（比如：配置参数）会被大部分模块访问。有关每个模块的更多信息，请参阅《模块和命令参考》。

箭头表示模块之间最重要连接的信息流。使用 发布-订阅 消息总线这个方案意味着：

- 系统是 “响应式” 的 — 系统异步运行，新数据抵达时系统立即进行更新。
- 系统所有的活动和通信都是完全并行的。
- 系统组件在任何地方都可以在保证线程安全的情况下使用数据。

飞行控制栈是针对自主无人机设计的导航、制导和控制算法的集合。它包括了为固定翼、旋翼和
VTOL 无人机设计的控制器，以及相应的姿态、位置估计器。

下图展示了飞行控制栈的整体架构， 下图展示了飞行控制栈的整体架构，
它包含了从传感器数据、 RC 控制量输入 到自主飞行控制（制导控制器，Navigator
），再到电机、舵机控制（执行器，Actuators）的全套通路。



估计器采用一个或多个传感器输入，将它们组合起来，并计算车辆状态（例如，来自 IMU
传感器数据的姿态）。

控制器是一个组件，它把设定值和测量或估计状态（过程变量）作为输入。其目标是调整过程变量的值，使其与设定值匹配。输出是最终达到该设定点的校正。例如，位置控制器将位置设定值作为输入，过程变量是当前估计的位置，输出是将车辆推向所需位置的姿态和推力设定值。

混频器接收力命令（例如“右转”）并将其转换为单独的电机命令，同时确保不超过某些限制。这种转换是特定于车辆类型的，并且取决于各种因素，例如电机相对于重心的布置或车辆的旋转惯性。

中间件主要由嵌入式传感器的设备驱动程序、与外部世界（伴侣计算机、GCS
等）的通信以及 uORB 发布-订阅消息总线组成。

此外，中间件还包括一个模拟层，允许PX4飞行代码在桌面操作系统上运行，并在模拟的“世界”中控制计算机建模的车辆。

更新速率

由于模块等待消息更新，因此驱动程序通常会定义模块更新的速度。大多数 IMU 驱动器以
1kHz 的频率对数据进行采样，对其进行积分并以 250Hz 的频率发布。系统的其他部分（如
）不需要如此高的更新率，因此运行速度要慢得多。

通过运行，可以在系统上实时检查消息更新率。

4.运行环境

PX4 运行在各种提供 POSIX-API 的操作系统（如 Linux、macOS、NuttX 或
QuRT）上。它还应该具有某种形式的实时调度（例如FIFO）。

模块间通信（使用 uORB）基于共享内存。整个 PX4
中间件运行在单个地址空间中，即内存在所有模块之间共享。

有 2 种不同的方法可以执行模块：

- 任务：模块在自己的任务中运行，具有自己的堆栈和进程优先级。
- 工作队列任务：模块在共享工作队列上运行，与队列中的其他模块共享相同的堆栈和工作队列线程优先级。
  - 所有任务都必须协同工作，因为它们不能相互干扰。
  - 多个工作队列任务可以在一个队列上运行，并且可以有多个队列。
  - 通过指定将来的固定时间，或者通过 uORB 主题更新回调来调度工作队列任务。

在工作队列上运行模块的优点是它使用较少的
RAM，并可能导致较少的任务切换。缺点是不允许工作队列任务休眠或轮询消息，也不允许阻塞
IO（例如从文件中读取）。长时间运行的任务（执行繁重的计算）也可能在单独的任务中运行，或者至少在单独的工作队列中运行。

后台任务。px4_task_spawn_cmd()用于启动独立于调用（父）任务运行的新任务 （NuttX）
或线程 （POSIX - Linux/macOS）：

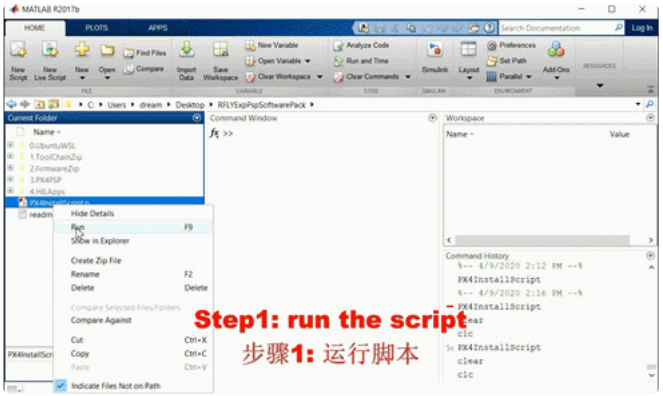
```
independent_task = px4_task_spawn_cmd( "commander", // Process name SCHED_DEFAULT, // Scheduling type (RR or FIFO) SCHED_PRIORITY_DEFAULT + 40, // Scheduling priority 3600, // Stack size of the new task or thread commander_thread_main, // Task (or thread) main function (char * const *)&argv[0] // Void pointer to pass to the new task // (here the commandline arguments). );
```

NuttX 是在飞行控制板上运行 PX4 的主要
RTOS。它是开源的（BSD许可证），轻量级，高效且非常稳定。模块作为任务执行，它们有自己的文件描述符列表，但它们共享一个地址空间。任务仍然可以启动一个或多个共享文件描述符列表的线程。每个任务/线程都有一个固定大小的堆栈，并且有一个定期任务，用于检查所有堆栈是否有足够的剩余可用空间（基于堆栈颜色）。

在 Linux 或 macOS 上，PX4
在单个进程中运行，模块在自己的线程中运行（任务和线程之间没有像 NuttX
那样的区别）。

RflySim平台的飞控固件编译环境部署

RflySim平台支持一键部署PX4软件环境，并支持卓翼和Pixhawk部分系列飞控硬件，编译界面如下图所示，可自定义选择不同的飞控编译命令和PX4固件版本，RflySim会将选定的PX4
Firmware源代码部署在设定的安装路径上，安装过程中，平台会自动识别所部署的固件类型，并进行全新部署。

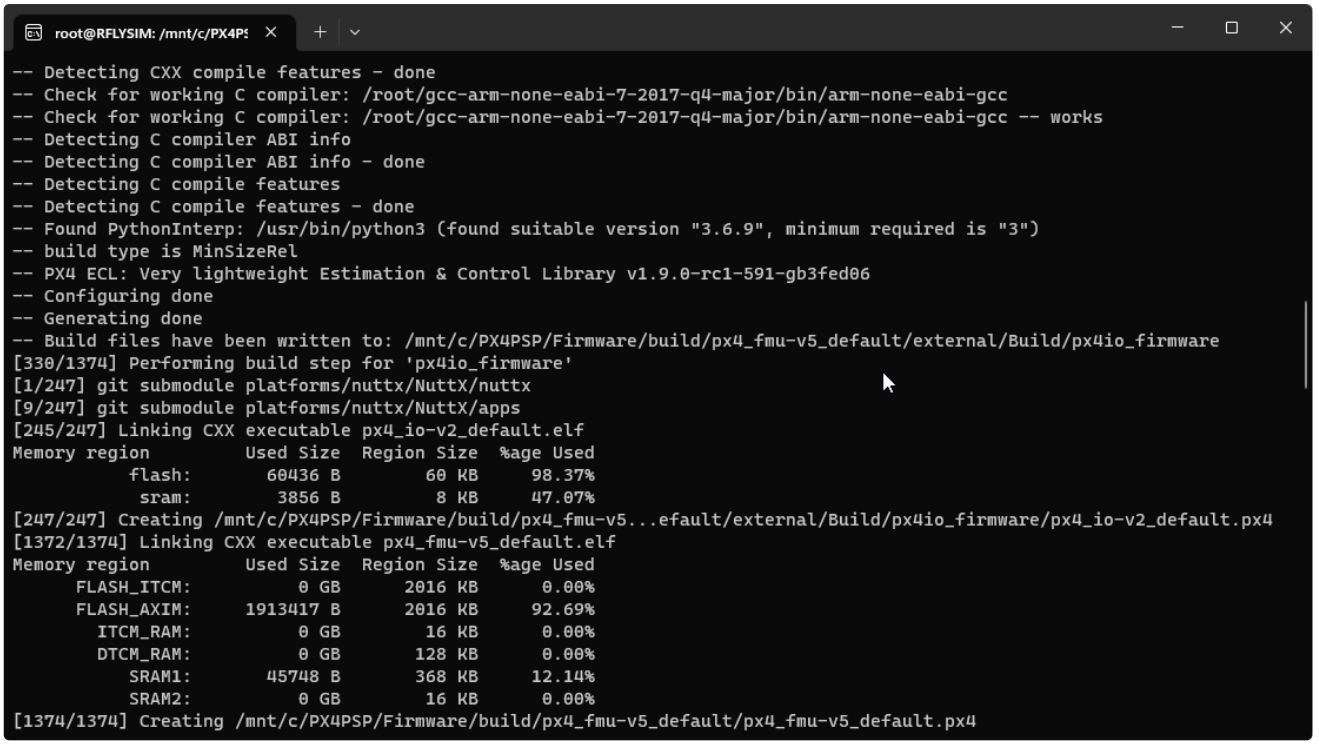


RflySim平台安装成功后，可在Windows下是无法直接进行飞控固件编译，一般采取的方式是在Windows安装PX4固件编译所需的工具链，在PX4官网提供三种工具链分别是：Windows
Cygwin 工具链、Virtual Machine工具链以及Bash on
Windows工具链。RflySim平台所采用的方式为Bash on
Windows工具链的方式，在平台安装时，默认WinWSL子系统，无需用户进行自行配置。安装完成之后，在桌面生成的"*\桌面\RflyTools\WinWSL.lnk"双击即可打开WinWSL子系统，输入对应的飞控编译命令即可完成飞控固件的编译。如：

##若子系统的根目录未在平台固件安装目录下，需输入：

cd /mnt/c/PX4PSP/Firmware

make droneyee_zyfc-h7_default



PX4 Firmware文件夹解析

PX4源码Firmware是一个内容庞大的文件夹，里面有许多的子文件夹，代表着不同的功能模块。文件夹结构如下图所示：

*/PX4PSP/Firmware	build	编译生成的文件		
	boards	存储各种飞控板卡的配置文件		
	Documentation	开发者文档目录，包含代码说明		
	launch	仿真环境所用到的文件		
	mavlink	mavlink消息的定义，打包， 发送，接收，解包的函数		
	msg	uORB通信用到的数据结构		
	platforms	系统平台实现的文件		
	ROMFS	ROM file system的缩写， 是文件系统文件夹， 里面存放的飞控系统的启动脚本， 负责系统初始化。		
	src	PX4源代码目录	drivers	飞控硬件系统中使用的所有传感器的驱 包括陀螺仪、加速度计、磁力计、气压 GPS、光流等，也包括pwm输出、px4i 不同遥控器通信协议等功能的驱动。
			examples	PX4官方给出的例程， 帮助新手开发者入门进行二次开发。
			include	其他代码需要用到的头文件和库。
			lib	标准库，有矩阵运算、PID控制、 传感器校准等。
			modules	功能模块文件夹，也是二次开发主要的 包括姿态解算，姿态控制，位置估计， 位置控制，指令控制，落地检测， 传感器初始化等。

*/PX4PSP/Firmware	build	编译生成的文件		
			systemcmds	系统指令文件来， 都是飞控终端支持的命令的源码，如to reboot命令等。
			templates	存放功能模块的代码模板，可以用于参
	Tools	一些工具，比如下载工具、 仿真环境等。		

src文件夹中modules:包含所有上层应用的模块实现，每个模块占用一个线程独立运行，PX4的上层程序都是通过模块的形式来运行（类似于ROS里面的节点），每一个模块订阅和发布uORB信息。模块包括姿态解算、姿态控制、位置控制、命令处理（commander）等。例如：

attitude_estimator_ekf：采用EKF算法实现的姿态估计

attitude_estimator_q: 使用mahony的互补滤波算法实现姿态解算。

mc_att_control：即multi-copter attitude
control，多轴飞行器的姿态控制算法实现，主要就是姿态的内外环PID控制，外环角度控制，内环角速度控制。

mc_pos_control: multi-copter position
control，多轴飞行器的位置控制算法实现，主要是位置的内外环PID控制，外环速度控制，内环加速度控制。

commander: 整个系统的任务调度，包括命令处理、事件处理、飞行模式切换等。

land_detector:
飞行过程中使用land模式降落或者落地时的落地监测部分，内部会监测Z轴速度和加速度等。

local_position_estimator: 常说的LPE算法实现位置解算。

mavlink:
和地面站通信的通信协议，结合地面站QGC源码配合修改，或者仅仅调用mavlink内部的API接口，即可通过无线信号将所需的数据显示在地面站QGC上，此方法是一种实时监测目标数据的方法。

logger: 关于log日志的读写函数。

PX4多旋翼源码解析

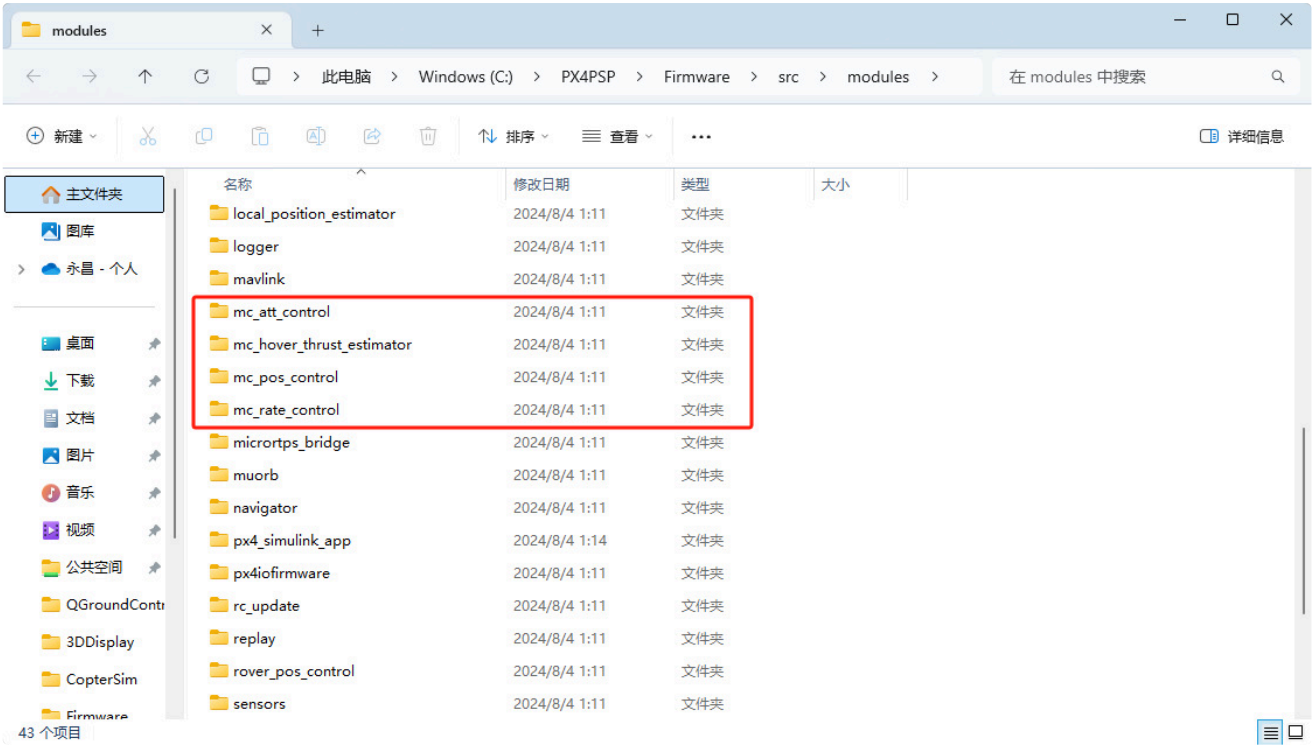
多旋翼相关源码放置在“*:\PX4PSP\Firmware\src\modules”目录下，并以“mc”开头命名。

“mc_att_control”文件夹中，存放的是姿态环控制相关的代码。

“mc_hover_thrust_estimator”文件夹中，存放的是悬停推力估计器相关的代码。

“mc_pos_control” 文件夹中，存放的是位置环控制相关的代码。

“mc_rate_control” 文件夹中，存放的是速度环控制相关的代码。



姿态环控制源码

【mc_att_control_params.c】

PX4 使用 param
子系统（和值的平面表）和文本文件（用于启动脚本）来存储其配置。参数定义包含两个部分：参数元数据指定固件中每个参数的默认值，以及用于在地面控制站和文档中显示（和编辑）参数的其他元数据。C/C++
代码，提供从 PX4 模块和驱动程序中获取和/或订阅参数值的访问权限。

参数名称不得超过 16 个 ASCII
字符。按照惯例，组中的每个参数都应共享相同（有意义的）字符串前缀，后跟下划线，并且用于专门与多旋翼或固定翼系统相关的参数。此约定不强制执行。名称在代码和参数元数据中必须匹配，才能正确地将参数与其元数据（包括固件中的默认值）关联。

PX4
使用广泛的参数元数据系统来驱动面向用户的参数呈现，并为固件中的每个参数设置默认值。参数元数据可以作为
.c 或 .yaml 参数定义存储在源树中的任何位置（YAML
定义更新且更灵活）。通常，它与关联的模块一起存储。

该文件中，就是用“.c”文件存储的多旋翼姿态控制器参数。该文件中的代码都已以下的形式编写，首先是一些注释信息，其中定义了一些最小值和最大值等。注释下面按照规则编写：参数的类型、参数的名称
（必须与代码中使用的名称匹配）
以及固件中的默认值。下面第一行代码的意思是定义了一个FLOAT类型的参数，其名称为MC_ROLL_P，默认值为6.5。

```
/** * Roll P gain * * Roll proportional gain, i.e. desired angular speed in rad/s for error 1 rad. * * @min 0.0 * @max 12 * @decimal 2 * @increment 0.1 * @group Multicopter Attitude Control */ PARAM_DEFINE_FLOAT(MC_ROLL_P, 6.5f); PARAM_DEFINE_FLOAT(MC_PITCH_P, 6.5f); PARAM_DEFINE_FLOAT(MC_YAW_P, 2.8f); PARAM_DEFINE_FLOAT(MC_YAW_WEIGHT, 0.4f); PARAM_DEFINE_FLOAT(MC_ROLLRATE_MAX, 220.0f); PARAM_DEFINE_FLOAT(MC_PITCHRATE_MAX, 220.0f); PARAM_DEFINE_FLOAT(MC_YAWRATE_MAX, 200.0f); PARAM_DEFINE_FLOAT(MC_MAN_TILT_TAU, 0.0f);
```

【mc_att_control.hpp】

该文件是C++头文件，主要声明了多旋翼飞行器姿态控制应用程序启动/停止处理功能以及MulticopterAttitudeControl类，该类继承了ModuleBase<MulticopterAttitudeControl>、ModuleParams和px4::WorkItem类。

```
extern "C" __EXPORT int mc_att_control_main(int argc, char *argv[]); class MulticopterAttitudeControl : public ModuleBase<MulticopterAttitudeControl>, public ModuleParams, public px4::WorkItem { ..... };
```

【mc_att_control_main.cpp】

该文件是多旋翼姿态控制器的具体实现，以下是具体实现的函数。

```
MulticopterAttitudeControl::MulticopterAttitudeControl(bool vtol) MulticopterAttitudeControl::~MulticopterAttitudeControl() bool MulticopterAttitudeControl::init() void MulticopterAttitudeControl::parameters_updated() float MulticopterAttitudeControl::throttle_curve(float throttle_stick_input) void MulticopterAttitudeControl::generate_attitude_setpoint(const Quatf &q, float dt, bool reset_yaw_sp) void MulticopterAttitudeControl::Run() int MulticopterAttitudeControl::task_spawn(int argc, char *argv[]) int MulticopterAttitudeControl::custom_command(int argc, char *argv[]) int MulticopterAttitudeControl::print_usage(const char *reason) int mc_att_control_main(int argc, char *argv[])
```

【AttitudeControl/AttitudeControl.hpp】

该文件是该C++头文件，声明了基于四元数的姿态控制器类，及其方法。

```
class AttitudeControl { ..... };
```

【AttitudeControl/AttitudeControl.cpp】

该文件是基于四元数的姿态控制器类的具体实现。

```
void AttitudeControl::setProportionalGain(const matrix::Vector3f &proportional_gain, const float yaw_weight) matrix::Vector3f AttitudeControl::update(const Quatf &q) const
```

【AttitudeControl/ AttitudeControlTest.cpp】

该文件是一个测试代码，为编写的姿态控制器做了测试。

```
TEST(AttitudeControlTest, AllZeroCase) { AttitudeControl attitude_control; Vector3f rate_setpoint = attitude_control.update(Quatf()); EXPECT_EQ(rate_setpoint, Vector3f()); } class AttitudeControlConvergenceTest : public ::testing::Test { public: AttitudeControlConvergenceTest() { _attitude_control.setProportionalGain(Vector3f(.5f, .6f, .3f), .4f); _attitude_control.setRateLimit(Vector3f(100.f, 100.f, 100.f)); } void checkConvergence() { int i; // need function scope to check how many steps Vector3f rate_setpoint(1000.f, 1000.f, 1000.f); _attitude_control.setAttitudeSetpoint(_quat_goal, 0.f); for (i = 100; i > 0; i--) { const Vector3f rate_setpoint_new = _attitude_control.update(_quat_state); _quat_state = _quat_state * Quatf(AxisAnglef(rate_setpoint_new)); _quat_state = -_quat_state; if (rate_setpoint_new.norm() >= rate_setpoint.norm()) { break; } rate_setpoint = rate_setpoint_new; } EXPECT_EQ(_quat_state.canonical(), _quat_goal.canonical()); EXPECT_GT(i, 0); } AttitudeControl _attitude_control; Quatf _quat_state; Quatf _quat_goal; }; TEST_F(AttitudeControlConvergenceTest, AttitudeControlConvergence) { const int inputs = 8; const Quatf QArray[inputs] = { Quatf(), Quatf(0, 1, 0, 0), Quatf(0, 0, 1, 0), Quatf(0, 0, 0, 1), Quatf(0.698f, 0.024f, -0.681f, -0.220f), Quatf(-0.820f, -0.313f, 0.225f, -0.423f), Quatf(0.599f, -0.172f, 0.755f, -0.204f), Quatf(0.216f, -0.662f, 0.290f, -0.656f) }; for (int i = 0; i < inputs; i++) { for (int j = 0; j < inputs; j++) { printf("--- Input combination: %d %d\n", i, j); _quat_state = QArray[i]; _quat_goal = QArray[j]; _quat_state.normalize(); _quat_goal.normalize(); checkConvergence(); } } } TEST(AttitudeControlTest, YawWeightScaling) { AttitudeControl attitude_control; const float yaw_gain = 2.8f; const float yaw_sp = .1f; Quatf pure_yaw_attitude(cosf(yaw_sp / 2.f), 0, 0, sinf(yaw_sp / 2.f)); attitude_control.setProportionalGain(Vector3f(6.5f, 6.5f, yaw_gain), .4f); attitude_control.setRateLimit(Vector3f(1000.f, 1000.f, 1000.f)); attitude_control.setAttitudeSetpoint(pure_yaw_attitude, 0.f); Vector3f rate_setpoint = attitude_control.update(Quatf()); EXPECT_EQ(Vector2f(rate_setpoint), Vector2f()); EXPECT_NEAR(rate_setpoint(2), yaw_sp * yaw_gain, 1e-4f); attitude_control.setProportionalGain(Vector3f(6.5f, 6.5f, yaw_gain), 0.f); rate_setpoint = attitude_control.update(Quatf()); EXPECT_EQ(rate_setpoint, Vector3f()); }
```

悬停推力估计器源码

【hover_thrust_estimator_params.c】

悬停推力估计器使用的参数。

```
PARAM_DEFINE_FLOAT(HTE_HT_NOISE, 0.0036); PARAM_DEFINE_FLOAT(HTE_ACC_GATE, 3.0); PARAM_DEFINE_FLOAT(HTE_HT_ERR_INIT, 0.1);
```

【MulticopterHoverThrustEstimator.hpp】

悬停推力估计器的类。

```
class MulticopterHoverThrustEstimator : public ModuleBase<MulticopterHoverThrustEstimator>, public ModuleParams, public px4::WorkItem { ..... };
```

【MulticopterHoverThrustEstimator.cpp】

悬停推力估计器接口类的实现。

```
MulticopterHoverThrustEstimator::MulticopterHoverThrustEstimator() : MulticopterHoverThrustEstimator::~MulticopterHoverThrustEstimator() bool MulticopterHoverThrustEstimator::init() void MulticopterHoverThrustEstimator::reset() void MulticopterHoverThrustEstimator::updateParams() void MulticopterHoverThrustEstimator::Run() void MulticopterHoverThrustEstimator::publishStatus(const hrt_abstime &timestamp_sample) void MulticopterHoverThrustEstimator::publishInvalidStatus() int MulticopterHoverThrustEstimator::task_spawn(int argc, char *argv[]) int MulticopterHoverThrustEstimator::custom_command(int argc, char *argv[]) int MulticopterHoverThrustEstimator::print_status() int MulticopterHoverThrustEstimator::print_usage(const char *reason) extern "C" __EXPORT int mc_hover_thrust_estimator_main(int argc, char *argv[])
```

【zero_order_hover_thrust_ekf.hpp】

单状态悬停推力估计器的接口类。

```
class ZeroOrderHoverThrustEkf { ..... };
```

【zero_order_hover_thrust_ekf.cpp】

单状态悬停推力估计器接口类的实现。

```
void ZeroOrderHoverThrustEkf::predict(const float dt) void ZeroOrderHoverThrustEkf::fuseAccZ(const float acc_z, const float thrust) inline float ZeroOrderHoverThrustEkf::computeH(const float thrust) const inline float ZeroOrderHoverThrustEkf::computeInnovVar(const float H) const float ZeroOrderHoverThrustEkf::computeInnov(const float acc_z, const float thrust) const float ZeroOrderHoverThrustEkf::computePredictedAccZ(const float thrust) const inline float ZeroOrderHoverThrustEkf::computeKalmanGain(const float H, const float innov_var) const inline float ZeroOrderHoverThrustEkf::computeInnovTestRatio(const float innov, const float innov_var) const inline bool ZeroOrderHoverThrustEkf::isTestRatioPassing(const float innov_test_ratio) const inline void ZeroOrderHoverThrustEkf::updateState(const float K, const float innov) inline void ZeroOrderHoverThrustEkf::updateStateCovariance(const float K, const float H) inline bool ZeroOrderHoverThrustEkf::isLargeOffsetDetected() const inline void ZeroOrderHoverThrustEkf::bumpStateVariance() inline void ZeroOrderHoverThrustEkf::updateLpf(const float residual, const float signed_innov_test_ratio) inline void ZeroOrderHoverThrustEkf::updateMeasurementNoise(const float residual, const float H)
```

【zero_order_hover_thrust_ekf_test.cpp】

零级悬停推力估计的测试代码。

```
class ZeroOrderHoverThrustEkfTest : public ::testing::Test { public: struct Status { float hover_thrust; float hover_thrust_var; float innov; float innov_var; float innov_test_ratio; float accel_noise_var; }; ZeroOrderHoverThrustEkfTest() { _random_generator.seed(42); } float computeAccelFromThrustAndHoverThrust(float thrust, float hover_thrust); Status runEkf(float hover_thrust_true, float thrust, float time, float accel_noise = 0.f, float thr_noise = 0.f); private: ZeroOrderHoverThrustEkf _ekf{}; static constexpr float _dt = 0.02f; std::normal_distribution<float> _standard_normal_distribution; std::default_random_engine _random_generator; // Pseudo-random generator with constant seed protected: static constexpr float _accel_noise_var_min = 1.f; // Constrained in the implementation }; float ZeroOrderHoverThrustEkfTest::computeAccelFromThrustAndHoverThrust(float thrust, float hover_thrust) { return CONSTANTS_ONE_G * thrust / hover_thrust - CONSTANTS_ONE_G; } ZeroOrderHoverThrustEkfTest::Status ZeroOrderHoverThrustEkfTest::runEkf(float hover_thrust_true, float thrust, float time, float accel_noise, float thr_noise) { Status status{}; for (float t = 0.f; t <= time; t += _dt) { _ekf.predict(_dt); float noisy_thrust = thrust + thr_noise * _standard_normal_distribution(_random_generator); float accel_theory = computeAccelFromThrustAndHoverThrust(thrust, hover_thrust_true); float noisy_accel = accel_theory + accel_noise * _standard_normal_distribution(_random_generator); _ekf.fuseAccZ(noisy_accel, noisy_thrust); } status.hover_thrust = _ekf.getHoverThrustEstimate(); status.hover_thrust_var = _ekf.getHoverThrustEstimateVar(); status.innov = _ekf.getInnovation(); status.innov_var = _ekf.getInnovationVar(); status.innov_test_ratio = _ekf.getInnovationTestRatio(); status.accel_noise_var = _ekf.getAccelNoiseVar(); return status; } TEST_F(ZeroOrderHoverThrustEkfTest, testStaticCase) { const float thrust = 0.5f; const float hover_thrust_true = 0.5f; ZeroOrderHoverThrustEkfTest::Status status = runEkf(hover_thrust_true, thrust, 2.f); EXPECT_NEAR(status.hover_thrust, hover_thrust_true, 1e-4f); EXPECT_NEAR(status.hover_thrust_var, 0.f, 1e-3f); EXPECT_NEAR(status.accel_noise_var, _accel_noise_var_min, 1.f); } TEST_F(ZeroOrderHoverThrustEkfTest, testStaticConvergence) { const float thrust = 0.72f; const float hover_thrust_true = 0.72f; ZeroOrderHoverThrustEkfTest::Status status = runEkf(hover_thrust_true, thrust, 2.f); EXPECT_NEAR(status.hover_thrust, hover_thrust_true, 1e-2f); EXPECT_NEAR(status.hover_thrust_var, 0.f, 1e-3f); EXPECT_NEAR(status.accel_noise_var, _accel_noise_var_min, 1.f); // The noise learning is slow and takes more time to go to zero } TEST_F(ZeroOrderHoverThrustEkfTest, testStaticConvergenceWithNoise) { const float sigma_noise = 3.f; const float noise_var = sigma_noise * sigma_noise; const float thrust = 0.72f; const float hover_thrust_true = 0.72f; const float t_sim = 10.f; ZeroOrderHoverThrustEkfTest::Status status = runEkf(hover_thrust_true, thrust, t_sim, sigma_noise); EXPECT_NEAR(status.hover_thrust, hover_thrust_true, 5e-2f); EXPECT_NEAR(status.hover_thrust_var, 0.f, 1e-3f); EXPECT_NEAR(status.accel_noise_var, noise_var, 0.2f * noise_var); } TEST_F(ZeroOrderHoverThrustEkfTest, testLargeAccelNoiseAndBias) { const float sigma_noise = 7.f; const float noise_var = sigma_noise * sigma_noise; const float thrust = 0.4f; // Below hover thrust const float hover_thrust_true = 0.72f; const float t_sim = 15.f; ZeroOrderHoverThrustEkfTest::Status status = runEkf(hover_thrust_true, thrust, t_sim, sigma_noise); EXPECT_NEAR(status.hover_thrust, hover_thrust_true, 7e-2); EXPECT_NEAR(status.hover_thrust_var, 0.f, 1e-3f); EXPECT_NEAR(status.accel_noise_var, noise_var, 0.2f * noise_var); } TEST_F(ZeroOrderHoverThrustEkfTest, testThrustAndAccelNoise) { const float accel_noise = 2.f; const float accel_var = accel_noise * accel_noise; const float thr_noise = 0.01f; const float thrust = 0.72f; // Above hover thrust const float hover_thrust_true = 0.6f; const float t_sim = 15.f; ZeroOrderHoverThrustEkfTest::Status status = runEkf(hover_thrust_true, thrust, t_sim, accel_noise, thr_noise); EXPECT_NEAR(status.hover_thrust, hover_thrust_true, 5e-2f); EXPECT_NEAR(status.hover_thrust_var, 0.f, 1e-3f); EXPECT_NEAR(status.accel_noise_var, accel_var, 0.4f * accel_var); } TEST_F(ZeroOrderHoverThrustEkfTest, testHoverThrustJump) { const float accel_noise = 2.f; const float accel_var = accel_noise * accel_noise; const float thr_noise = 0.01f; float thrust = 0.8; // At hover float hover_thrust_true = 0.8f; float t_sim = 10.f; ZeroOrderHoverThrustEkfTest::Status status = runEkf(hover_thrust_true, thrust, t_sim, accel_noise, thr_noise); thrust = 0.3; hover_thrust_true = 0.3f; t_sim = 10.f; status = runEkf(hover_thrust_true, thrust, t_sim, accel_noise, thr_noise); EXPECT_NEAR(status.hover_thrust, hover_thrust_true, 5e-2f); EXPECT_NEAR(status.hover_thrust_var, 0.f, 1e-3f); EXPECT_NEAR(status.accel_noise_var, accel_var, 2.f * accel_var); }
```

位置环控制源码

【mc_pos_control_params.c】

多旋翼机位置控制器参数。

PARAM_DEFINE_FLOAT(MPC_THR_MIN, 0.12f); PARAM_DEFINE_FLOAT(MPC_THR_HOVER, 0.5f);
PARAM_DEFINE_INT32(MPC_USE_HTE, 1); PARAM_DEFINE_INT32(MPC_THR_CURVE, 0);
PARAM_DEFINE_FLOAT(MPC_THR_MAX, 1.0f); PARAM_DEFINE_FLOAT(MPC_MANTHR_MIN, 0.08f);
PARAM_DEFINE_FLOAT(MPC_Z_P, 1.0f); PARAM_DEFINE_FLOAT(MPC_Z_VEL_P_ACC, 4.0f);
PARAM_DEFINE_FLOAT(MPC_Z_VEL_I_ACC, 2.0f); PARAM_DEFINE_FLOAT(MPC_Z_VEL_D_ACC, 0.0f);
PARAM_DEFINE_FLOAT(MPC_Z_VEL_MAX_UP, 3.0f); PARAM_DEFINE_FLOAT(MPC_Z_VEL_MAX_DN, 1.0f);
PARAM_DEFINE_FLOAT(MPC_XY_P, 0.95f); PARAM_DEFINE_FLOAT(MPC_XY_VEL_P_ACC, 1.8f);
PARAM_DEFINE_FLOAT(MPC_XY_VEL_I_ACC, 0.4f); PARAM_DEFINE_FLOAT(MPC_XY_VEL_D_ACC, 0.2f);
PARAM_DEFINE_FLOAT(MPC_XY_CRUISE, 5.0f); PARAM_DEFINE_FLOAT(MPC_XY_TRAJ_P, 0.5f);
PARAM_DEFINE_FLOAT(MPC_XY_ERR_MAX, 2.0f); PARAM_DEFINE_FLOAT(MPC_VEL_MANUAL, 10.0f);
PARAM_DEFINE_FLOAT(MPC_XY_VEL_MAX, 12.0f); PARAM_DEFINE_FLOAT(MPC_TILTMAX_AIR, 45.0f);
PARAM_DEFINE_FLOAT(MPC_TILTMAX_LND, 12.0f); PARAM_DEFINE_FLOAT(MPC_LAND_SPEED, 0.7f);
PARAM_DEFINE_INT32(MPC_LAND_RC_HELP, 0); PARAM_DEFINE_FLOAT(MPC_TKO_SPEED, 1.5f);
PARAM_DEFINE_FLOAT(MPC_MAN_TILT_MAX, 35.0f); PARAM_DEFINE_FLOAT(MPC_MAN_Y_MAX, 150.0f);
PARAM_DEFINE_FLOAT(MPC_MAN_Y_TAU, 0.08f); PARAM_DEFINE_FLOAT(MPC_HOLD_DZ, 0.1f);
PARAM_DEFINE_FLOAT(MPC_HOLD_MAX_XY, 0.8f); PARAM_DEFINE_FLOAT(MPC_HOLD_MAX_Z, 0.6f);
PARAM_DEFINE_FLOAT(MPC_VELD_LP, 5.0f); PARAM_DEFINE_FLOAT(MPC_ACC_HOR_MAX, 5.0f);
PARAM_DEFINE_FLOAT(MPC_ACC_HOR, 3.0f); PARAM_DEFINE_FLOAT(MPC_ACC_UP_MAX, 4.0f);
PARAM_DEFINE_FLOAT(MPC_ACC_DOWN_MAX, 3.0f); PARAM_DEFINE_FLOAT(MPC_JERK_MAX, 8.0f);
PARAM_DEFINE_FLOAT(MPC_JERK_AUTO, 4.0f); PARAM_DEFINE_INT32(MPC_ALT_MODE, 0);
PARAM_DEFINE_FLOAT(MPC_XY_MAN_EXPO, 0.6f); PARAM_DEFINE_FLOAT(MPC_Z_MAN_EXPO, 0.6f);
PARAM_DEFINE_FLOAT(MPC_YAW_EXPO, 0.6f); PARAM_DEFINE_FLOAT(MPC_YAWRAUTO_MAX, 45.0f);
PARAM_DEFINE_FLOAT(MPC_LAND_ALT1, 10.0f); PARAM_DEFINE_FLOAT(MPC_LAND_ALT2, 5.0f);
PARAM_DEFINE_FLOAT(MPC_TKO_RAMP_T, 3.0f); PARAM_DEFINE_INT32(MPC_POS_MODE, 4);
PARAM_DEFINE_FLOAT(MPC_SPOOLUP_TIME, 1.0f); PARAM_DEFINE_INT32(MPC_YAW_MODE, 0);
PARAM_DEFINE_FLOAT(SYS_VEHICLE_RESP, -0.4f); PARAM_DEFINE_FLOAT(MPC_XY_VEL_ALL, -10.0f);
PARAM_DEFINE_FLOAT(MPC_Z_VEL_ALL, -3.0f);

【MulticopterPositionControl.hpp】

多旋翼机位置控制器的接口类。

class MulticopterPositionControl : public ModuleBase<MulticopterPositionControl>, public control::SuperBlock, public ModuleParams, public px4::ScheduledWorkItem { ..... };

【MulticopterPositionControl.cpp】

多旋翼机位置控制器接口类的实现。

MulticopterPositionControl::MulticopterPositionControl(bool vtol) : MulticopterPositionControl::~MulticopterPositionControl() bool MulticopterPositionControl::init() int MulticopterPositionControl::parameters_update(bool force) PositionControlStates MulticopterPositionControl::set_vehicle_states(const vehicle_local_position_s &local_pos) void MulticopterPositionControl::Run() void MulticopterPositionControl::failsafe(const hrt_abstime &now, vehicle_local_position_setpoint_s &setpoint, const PositionControlStates &states, bool warn) void MulticopterPositionControl::reset_setpoint_to_nan(vehicle_local_position_setpoint_s &setpoint) int MulticopterPositionControl::task_spawn(int argc, char *argv[]) int MulticopterPositionControl::custom_command(int argc, char *argv[]) int MulticopterPositionControl::print_usage(const char *reason) extern "C" __EXPORT int mc_pos_control_main(int argc, char *argv[])

【Takeoff/Takeoff.hpp】

处理所有起飞状态和平稳斜坡发动机的接口类。

enum class TakeoffState { disarmed = takeoff_status_s::TAKEOFF_STATE_DISARMED, spoolup = takeoff_status_s::TAKEOFF_STATE_SPOOLUP, ready_for_takeoff = takeoff_status_s::TAKEOFF_STATE_READY_FOR_TAKEOFF, rampup = takeoff_status_s::TAKEOFF_STATE_RAMPUP, flight = takeoff_status_s::TAKEOFF_STATE_FLIGHT }; class Takeoff { ..... };

【Takeoff/Takeoff.cpp】

处理所有起飞状态和平稳斜坡发动机接口类的实现。

void Takeoff::generateInitialRampValue(float velocity_p_gain) void Takeoff::updateTakeoffState(const bool armed, const bool landed, const bool want_takeoff, const float takeoff_desired_vz, const bool skip_takeoff, const hrt_abstime &now_us) float Takeoff::updateRamp(const float dt, const float takeoff_desired_vz)

【Takeoff/TakeoffTest.cpp】

起飞的测试代码。

TEST(TakeoffTest, Initialization) { Takeoff takeoff; EXPECT_EQ(takeoff.getTakeoffState(), TakeoffState::disarmed); } TEST(TakeoffTest, RegularTakeoffRamp) { Takeoff takeoff; takeoff.setSpoolupTime(1.f); takeoff.setTakeoffRampTime(2.0); takeoff.generateInitialRampValue(CONSTANTS_ONE_G / 0.5f); takeoff.updateTakeoffState(false, true, false, 1.f, false, 0); EXPECT_EQ(takeoff.getTakeoffState(), TakeoffState::disarmed); takeoff.updateTakeoffState(true, false, false, 1.f, false, 500_ms); EXPECT_EQ(takeoff.getTakeoffState(), TakeoffState::spoolup); takeoff.updateTakeoffState(true, false, false, 1.f, false, 2_s); EXPECT_EQ(takeoff.getTakeoffState(), TakeoffState::ready_for_takeoff); takeoff.updateTakeoffState(true, false, true, 1.f, false, 3_s); EXPECT_EQ(takeoff.getTakeoffState(), TakeoffState::rampup); takeoff.updateTakeoffState(true, false, true, 1.f, false, 4_s); EXPECT_FLOAT_EQ(takeoff.updateRamp(.5f, 1.5f), 0.f); EXPECT_FLOAT_EQ(takeoff.updateRamp(.5f, 1.5f), .5f); EXPECT_FLOAT_EQ(takeoff.updateRamp(.5f, 1.5f), 1.f); EXPECT_FLOAT_EQ(takeoff.updateRamp(.5f, 1.5f), 1.5f); EXPECT_FLOAT_EQ(takeoff.updateRamp(.5f, 1.5f), 1.5f); takeoff.updateTakeoffState(true, false, true, 1.f, false, 6500_ms); EXPECT_EQ(takeoff.getTakeoffState(), TakeoffState::flight); }

【PositionControl/ControlMath.hpp】

不适合矩阵库的矢量操作的简单函数的接口。

namespace ControlMath { …… }

【PositionControl/ControlMath.cpp】

不适合矩阵库的矢量操作的简单函数的实现。

namespace ControlMath { void thrustToAttitude(const Vector3f &thr_sp, const float yaw_sp, vehicle_attitude_setpoint_s &att_sp) void limitTilt(Vector3f &body_unit, const Vector3f &world_unit, const float max_angle) void bodyzToAttitude(Vector3f body_z, const float yaw_sp, vehicle_attitude_setpoint_s &att_sp) Vector2f constrainXY(const Vector2f &v0, const Vector2f &v1, const float &max) bool cross_sphere_line(const Vector3f &sphere_c, const float sphere_r, const Vector3f &line_a, const Vector3f &line_b, Vector3f &res) void addIfNotNan(float &setpoint, const float addition) void addIfNotNanVector3f(Vector3f &setpoint, const Vector3f &addition) void setZeroIfNanVector3f(Vector3f &vector) }

【PositionControl/ControlMathTest.cpp】

不适合矩阵库的矢量操作的简单函数的测试代码。

```
TEST(ControlMathTest, LimitTiltUnchanged) { Vector3f body = Vector3f(0.f, 0.f, 1.f).normalized(); Vector3f body_before = body; limitTilt(body, Vector3f(0.f, 0.f, 1.f), M_DEG_TO_RAD_F * 45.f); EXPECT_EQ(body, body_before); body = Vector3f(0.f, .1f, 1.f).normalized(); body_before = body; limitTilt(body, Vector3f(0.f, 0.f, 1.f), M_DEG_TO_RAD_F * 45.f); EXPECT_EQ(body, body_before); } TEST(ControlMathTest, LimitTiltOpposite) { Vector3f body = Vector3f(0.f, 0.f, -1.f).normalized(); limitTilt(body, Vector3f(0.f, 0.f, 1.f), M_DEG_TO_RAD_F * 45.f); float angle = acosf(body.dot(Vector3f(0.f, 0.f, 1.f))); EXPECT_NEAR(angle * M_RAD_TO_DEG_F, 45.f, 1e-4f); EXPECT_FLOAT_EQ(body.length(), 1.f); } TEST(ControlMathTest, LimitTiltAlmostOpposite) { // This case doesn't trigger corner case handling but is very close to it Vector3f body = Vector3f(0.001f, 0.f, -1.f).normalized(); limitTilt(body, Vector3f(0.f, 0.f, 1.f), M_DEG_TO_RAD_F * 45.f); float angle = acosf(body.dot(Vector3f(0.f, 0.f, 1.f))); EXPECT_NEAR(angle * M_RAD_TO_DEG_F, 45.f, 1e-4f); EXPECT_FLOAT_EQ(body.length(), 1.f); } TEST(ControlMathTest, LimitTilt45degree) { Vector3f body = Vector3f(1.f, 0.f, 0.f); limitTilt(body, Vector3f(0.f, 0.f, 1.f), M_DEG_TO_RAD_F * 45.f); EXPECT_EQ(body, Vector3f(M_SQRT1_2_F, 0, M_SQRT1_2_F)); body = Vector3f(0.f, 1.f, 0.f); limitTilt(body, Vector3f(0.f, 0.f, 1.f), M_DEG_TO_RAD_F * 45.f); EXPECT_EQ(body, Vector3f(0.f, M_SQRT1_2_F, M_SQRT1_2_F)); } TEST(ControlMathTest, LimitTilt10degree) { Vector3f body = Vector3f(1.f, 1.f, .1f).normalized(); limitTilt(body, Vector3f(0.f, 0.f, 1.f), M_DEG_TO_RAD_F * 10.f); float angle = acosf(body.dot(Vector3f(0.f, 0.f, 1.f))); EXPECT_NEAR(angle * M_RAD_TO_DEG_F, 10.f, 1e-4f); EXPECT_FLOAT_EQ(body.length(), 1.f); EXPECT_FLOAT_EQ(body(0), body(1)); body = Vector3f(1, 2, .2f); limitTilt(body, Vector3f(0.f, 0.f, 1.f), M_DEG_TO_RAD_F * 10.f); angle = acosf(body.dot(Vector3f(0.f, 0.f, 1.f))); EXPECT_NEAR(angle * M_RAD_TO_DEG_F, 10.f, 1e-4f); EXPECT_FLOAT_EQ(body.length(), 1.f); EXPECT_FLOAT_EQ(2.f * body(0), body(1)); } TEST(ControlMathTest, ThrottleAttitudeMapping) { /* expected: zero roll, zero pitch, zero yaw, full thr mag * reason: thrust pointing full upward */ Vector3f thr{0.f, 0.f, -1.f}; float yaw = 0.f; vehicle_attitude_setpoint_s att{}; thrustToAttitude(thr, yaw, att); EXPECT_FLOAT_EQ(att.roll_body, 0.f); EXPECT_FLOAT_EQ(att.pitch_body, 0.f); EXPECT_FLOAT_EQ(att.yaw_body, 0.f); EXPECT_FLOAT_EQ(att.thrust_body[2], -1.f); /* expected: same as before but with 90 yaw * reason: only yaw changed */ yaw = M_PI_2_F; thrustToAttitude(thr, yaw, att); EXPECT_FLOAT_EQ(att.roll_body, 0.f); EXPECT_FLOAT_EQ(att.pitch_body, 0.f); EXPECT_FLOAT_EQ(att.yaw_body, M_PI_2_F); EXPECT_FLOAT_EQ(att.thrust_body[2], -1.f); /* expected: same as before but roll 180 * reason: thrust points straight down and order Euler * order is: 1. roll, 2. pitch, 3. yaw */ thr = Vector3f(0.f, 0.f, 1.f); thrustToAttitude(thr, yaw, att); EXPECT_FLOAT_EQ(att.roll_body, -M_PI_F); EXPECT_FLOAT_EQ(att.pitch_body, 0.f); EXPECT_FLOAT_EQ(att.yaw_body, M_PI_2_F); EXPECT_FLOAT_EQ(att.thrust_body[2], -1.f); } TEST(ControlMathTest, ConstrainXYPriorities) { const float max = 5.f; // v0 already at max Vector2f v0(max, 0.f); Vector2f v1(v0(1), -v0(0)); Vector2f v_r = constrainXY(v0, v1, max); EXPECT_FLOAT_EQ(v_r(0), max); EXPECT_FLOAT_EQ(v_r(1), 0.f); // norm of v1 exceeds max but v0 is zero v0.zero(); v_r = constrainXY(v0, v1, max); EXPECT_FLOAT_EQ(v_r(1), -max); EXPECT_FLOAT_EQ(v_r(0), 0.f); v0 = Vector2f(.5f, .5f); v1 = Vector2f(.5f, -.5f); v_r = constrainXY(v0, v1, max); const float diff = Vector2f(v_r - (v0 + v1)).length(); EXPECT_FLOAT_EQ(diff, 0.f); // v0 and v1 exceed max and are perpendicular v0 = Vector2f(4.f, 0.f); v1 = Vector2f(0.f, -4.f); v_r = constrainXY(v0, v1, max); EXPECT_FLOAT_EQ(v_r(0), v0(0)); EXPECT_GT(v_r(0), 0.f); const float remaining = sqrtf(max * max - (v0(0) * v0(0))); EXPECT_FLOAT_EQ(v_r(1), -remaining); } TEST(ControlMathTest, CrossSphereLine) { const Vector3f prev = Vector3f(0.f, 0.f, 0.f); const Vector3f curr = Vector3f(0.f, 0.f, 2.f); Vector3f res; bool retval = false; // on line, near, before previous waypoint retval = cross_sphere_line(Vector3f(0.f, 0.f, -.5f), 1.f, prev, curr, res); EXPECT_TRUE(retval); EXPECT_EQ(res, Vector3f(0.f, 0.f, 0.5f)); // on line, near, before target waypoint retval = cross_sphere_line(Vector3f(0.f, 0.f, 1.f), 1.f, prev, curr, res); EXPECT_TRUE(retval); EXPECT_EQ(res, Vector3f(0.f, 0.f, 2.f)); // on line, near, after target waypoint retval = cross_sphere_line(Vector3f(0.f, 0.f, 2.5f), 1.f, prev, curr, res); EXPECT_TRUE(retval); EXPECT_EQ(res, Vector3f(0.f, 0.f, 2.f)); // near, before previous waypoint retval = cross_sphere_line(Vector3f(0.f, .5f, -.5f), 1.f, prev, curr, res); EXPECT_TRUE(retval); EXPECT_EQ(res, Vector3f(0.f, 0.f, .366025388f)); // near, before target waypoint retval = cross_sphere_line(Vector3f(0.f, .5f, 1.f), 1.f, prev, curr, res); EXPECT_TRUE(retval); EXPECT_EQ(res, Vector3f(0.f, 0.f, 1.866025448f)); // near, after target waypoint retval = ControlMath::cross_sphere_line(matrix::Vector3f(0.f, .5f, 2.5f), 1.f, prev, curr, res); EXPECT_TRUE(retval); EXPECT_EQ(res, Vector3f(0.f, 0.f, 2.f)); // far, before previous waypoint retval = ControlMath::cross_sphere_line(matrix::Vector3f(0.f, 2.f, -.5f), 1.f, prev, curr, res); EXPECT_FALSE(retval); EXPECT_EQ(res, Vector3f()); // far, before target waypoint retval = ControlMath::cross_sphere_line(matrix::Vector3f(0.f, 2.f, 1.f), 1.f, prev, curr, res); EXPECT_FALSE(retval); EXPECT_EQ(res, Vector3f(0.f, 0.f, 1.f)); // far, after target waypoint retval = ControlMath::cross_sphere_line(matrix::Vector3f(0.f, 2.f, 2.5f), 1.f, prev, curr, res); EXPECT_FALSE(retval); EXPECT_EQ(res, Vector3f(0.f, 0.f, 2.f)); } TEST(ControlMathTest, addIfNotNan) { float v = 1.f; // regular addition ControlMath::addIfNotNan(v, 2.f); EXPECT_EQ(v, 3.f); // addition is NAN and has no influence ControlMath::addIfNotNan(v, NAN); EXPECT_EQ(v, 3.f); v = NAN; // both summands are NAN ControlMath::addIfNotNan(v, NAN); EXPECT_TRUE(isnan(v)); // regular value gets added to NAN and overwrites it ControlMath::addIfNotNan(v, 3.f); EXPECT_EQ(v, 3.f); }
```

【PositionControl/PositionControl.hpp】

级联位置控制器仅用于位置/速度控制的接口类。

class PositionControl { ..... };

【PositionControl/PositionControl.cpp】

级联位置控制器仅用于位置/速度控制接口类的实现。

void PositionControl::setVelocityGains(const Vector3f &P, const Vector3f &I, const Vector3f &D) void PositionControl::setVelocityLimits(const float vel_horizontal, const float vel_up, const float vel_down) void PositionControl::setThrustLimits(const float min, const float max) void PositionControl::updateHoverThrust(const float hover_thrust_new) void PositionControl::setState(const PositionControlStates &states) void PositionControl::setInputSetpoint(const vehicle_local_position_setpoint_s &setpoint) bool PositionControl::update(const float dt) void PositionControl::_positionControl() void PositionControl::_velocityControl(const float dt) void PositionControl::_accelerationControl() bool PositionControl::_updateSuccessful() void PositionControl::getLocalPositionSetpoint(vehicle_local_position_setpoint_s &local_position_setpoint) const void PositionControl::getAttitudeSetpoint(vehicle_attitude_setpoint_s &attitude_setpoint) const

【PositionControl/PositionControlTest.cpp】

级联位置控制器仅用于位置/速度控制的测试代码。

```
TEST(PositionControlTest, EmptySetpoint) { PositionControl position_control; vehicle_local_position_setpoint_s output_setpoint{}; position_control.getLocalPositionSetpoint(output_setpoint); EXPECT_FLOAT_EQ(output_setpoint.x, 0.f); EXPECT_FLOAT_EQ(output_setpoint.y, 0.f); EXPECT_FLOAT_EQ(output_setpoint.z, 0.f); EXPECT_FLOAT_EQ(output_setpoint.yaw, 0.f); EXPECT_FLOAT_EQ(output_setpoint.yawspeed, 0.f); EXPECT_FLOAT_EQ(output_setpoint.vx, 0.f); EXPECT_FLOAT_EQ(output_setpoint.vy, 0.f); EXPECT_FLOAT_EQ(output_setpoint.vz, 0.f); EXPECT_EQ(Vector3f(output_setpoint.acceleration), Vector3f(0.f, 0.f, 0.f)); EXPECT_EQ(Vector3f(output_setpoint.jerk), Vector3f(0.f, 0.f, 0.f)); EXPECT_EQ(Vector3f(output_setpoint.thrust), Vector3f(0, 0, 0)); vehicle_attitude_setpoint_s attitude{}; position_control.getAttitudeSetpoint(attitude); EXPECT_FLOAT_EQ(attitude.roll_body, 0.f); EXPECT_FLOAT_EQ(attitude.pitch_body, 0.f); EXPECT_FLOAT_EQ(attitude.yaw_body, 0.f); EXPECT_FLOAT_EQ(attitude.yaw_sp_move_rate, 0.f); EXPECT_EQ(Quatf(attitude.q_d), Quatf(1.f, 0.f, 0.f, 0.f)); EXPECT_EQ(Vector3f(attitude.thrust_body), Vector3f(0.f, 0.f, 0.f)); EXPECT_EQ(attitude.roll_reset_integral, false); EXPECT_EQ(attitude.pitch_reset_integral, false); EXPECT_EQ(attitude.yaw_reset_integral, false); EXPECT_EQ(attitude.fw_control_yaw, false); EXPECT_FLOAT_EQ(attitude.apply_flaps, 0.f);//vehicle_attitude_setpoint_s::FLAPS_OFF); // TODO why no reference? } class PositionControlBasicTest : public ::testing::Test { ..... }; class PositionControlBasicDirectionTest : public PositionControlBasicTest { ..... }; TEST_F(PositionControlBasicDirectionTest, PositionDirection) { _input_setpoint.x = .1f; _input_setpoint.y = .1f; _input_setpoint.z = -.1f; EXPECT_TRUE(runController()); checkDirection(); } TEST_F(PositionControlBasicDirectionTest, VelocityDirection) { _input_setpoint.vx = .1f; _input_setpoint.vy = .1f; _input_setpoint.vz = -.1f; EXPECT_TRUE(runController()); checkDirection(); } TEST_F(PositionControlBasicTest, TiltLimit) { _input_setpoint.x = 10.f; _input_setpoint.y = 10.f; _input_setpoint.z = -0.f; EXPECT_TRUE(runController()); Vector3f body_z = Quatf(_attitude.q_d).dcm_z(); float angle = acosf(body_z.dot(Vector3f(0.f, 0.f, 1.f))); EXPECT_GT(angle, 0.f); EXPECT_LE(angle, 1.f); _position_control.setTiltLimit(0.5f); EXPECT_TRUE(runController()); body_z = Quatf(_attitude.q_d).dcm_z(); angle = acosf(body_z.dot(Vector3f(0.f, 0.f, 1.f))); EXPECT_GT(angle, 0.f); EXPECT_LE(angle, .50001f); _position_control.setTiltLimit(1.f); // restore original } TEST_F(PositionControlBasicTest, VelocityLimit) { _input_setpoint.x = 10.f; _input_setpoint.y = 10.f; _input_setpoint.z = -10.f; EXPECT_TRUE(runController()); Vector2f velocity_xy(_output_setpoint.vx, _output_setpoint.vy); EXPECT_LE(velocity_xy.norm(), 1.f); EXPECT_LE(abs(_output_setpoint.vz), 1.f); } TEST_F(PositionControlBasicTest, PositionControlMaxThrustLimit) { _input_setpoint.x = 10.f; _input_setpoint.y = 10.f; _input_setpoint.z = -10.f; runController(); Vector3f thrust(_output_setpoint.thrust); EXPECT_FLOAT_EQ(thrust(0), 0.f); EXPECT_FLOAT_EQ(thrust(1), 0.f); EXPECT_FLOAT_EQ(thrust(2), -0.9f); EXPECT_EQ(_attitude.thrust_body[0], 0.f); EXPECT_EQ(_attitude.thrust_body[1], 0.f); EXPECT_FLOAT_EQ(_attitude.thrust_body[2], -0.9f); EXPECT_FLOAT_EQ(_attitude.roll_body, 0.f); EXPECT_FLOAT_EQ(_attitude.pitch_body, 0.f); } TEST_F(PositionControlBasicTest, PositionControlMinThrustLimit) { _input_setpoint.x = 10.f; _input_setpoint.y = 0.f; _input_setpoint.z = 10.f; runController(); Vector3f thrust(_output_setpoint.thrust); EXPECT_FLOAT_EQ(thrust.length(), 0.1f); EXPECT_FLOAT_EQ(_attitude.thrust_body[2], -0.1f); EXPECT_FLOAT_EQ(_attitude.roll_body, 0.f); EXPECT_FLOAT_EQ(_attitude.pitch_body, -1.f); } TEST_F(PositionControlBasicTest, FailsafeInput) { _input_setpoint.vz = .1f; _input_setpoint.thrust[0] = _input_setpoint.thrust[1] = 0.f; _input_setpoint.acceleration[0] = _input_setpoint.acceleration[1] = 0.f; EXPECT_TRUE(runController()); EXPECT_FLOAT_EQ(_attitude.thrust_body[0], 0.f); EXPECT_FLOAT_EQ(_attitude.thrust_body[1], 0.f); EXPECT_LT(_output_setpoint.thrust[2], -.1f); EXPECT_GT(_output_setpoint.thrust[2], -.5f); EXPECT_GT(_attitude.thrust_body[2], -.5f); EXPECT_LE(_attitude.thrust_body[2], -.1f); } TEST_F(PositionControlBasicTest, IdleThrustInput) { // High downwards acceleration to make sure there's no thrust Vector3f(0.f, 0.f, 100.f).copyTo(_input_setpoint.acceleration); EXPECT_TRUE(runController()); EXPECT_FLOAT_EQ(_output_setpoint.thrust[0], 0.f); EXPECT_FLOAT_EQ(_output_setpoint.thrust[1], 0.f); EXPECT_FLOAT_EQ(_output_setpoint.thrust[2], -.1f); } TEST_F(PositionControlBasicTest, InputCombinationsPosition) { _input_setpoint.x = .1f; _input_setpoint.y = .2f; _input_setpoint.z = .3f; EXPECT_TRUE(runController()); EXPECT_FLOAT_EQ(_output_setpoint.x, .1f); EXPECT_FLOAT_EQ(_output_setpoint.y, .2f); EXPECT_FLOAT_EQ(_output_setpoint.z, .3f); EXPECT_FALSE(isnan(_output_setpoint.vx)); EXPECT_FALSE(isnan(_output_setpoint.vy)); EXPECT_FALSE(isnan(_output_setpoint.vz)); EXPECT_FALSE(isnan(_output_setpoint.thrust[0])); EXPECT_FALSE(isnan(_output_setpoint.thrust[1])); EXPECT_FALSE(isnan(_output_setpoint.thrust[2])); } TEST_F(PositionControlBasicTest, InputCombinationsPositionVelocity) { _input_setpoint.vx = .1f; _input_setpoint.vy = .2f; _input_setpoint.z = .3f; // altitude EXPECT_TRUE(runController()); // EXPECT_TRUE(isnan(_output_setpoint.x)); // EXPECT_TRUE(isnan(_output_setpoint.y)); EXPECT_FLOAT_EQ(_output_setpoint.z, .3f); EXPECT_FLOAT_EQ(_output_setpoint.vx, .1f); EXPECT_FLOAT_EQ(_output_setpoint.vy, .2f); EXPECT_FALSE(isnan(_output_setpoint.vz)); EXPECT_FALSE(isnan(_output_setpoint.thrust[0])); EXPECT_FALSE(isnan(_output_setpoint.thrust[1])); EXPECT_FALSE(isnan(_output_setpoint.thrust[2])); } TEST_F(PositionControlBasicTest, SetpointValiditySimple) { EXPECT_FALSE(runController()); _input_setpoint.x = .1f; EXPECT_FALSE(runController()); _input_setpoint.y = .2f; EXPECT_FALSE(runController()); _input_setpoint.acceleration[2]
```

```
= .3f; EXPECT_TRUE(runController()); } TEST_F(PositionControlBasicTest, SetpointValidityAllCombinations) { float *const setpoint_loop_access_map[] = {&_input_setpoint.x, &_input_setpoint.vx, &_input_setpoint.acceleration[0], &_input_setpoint.y, &_input_setpoint.vy, &_input_setpoint.acceleration[1], &_input_setpoint.z, &_input_setpoint.vz, &_input_setpoint.acceleration[2] }; for (int combination = 0; combination < 512; combination++) { resetInputSetpoint(); for (int j = 0; j < 9; j++) { if (combination & (1 << j)) { // Set arbitrary finite value, some values clearly hit the limits to check these corner case combinations *(setpoint_loop_access_map[j]) = static_cast<float>(combination) / static_cast<float>(j + 1); } } const bool has_x_setpoint = ((combination & 7) != 0); const bool has_y_setpoint = (((combination >> 3) & 7) != 0); const bool has_z_setpoint = (((combination >> 6) & 7) != 0); const bool has_xy_pairs = (combination & 7) == ((combination >> 3) & 7); const bool expected_result = has_x_setpoint && has_y_setpoint && has_z_setpoint && has_xy_pairs; EXPECT_EQ(runController(), expected_result) << "combination " << combination << std::endl << "input" << std::endl << "position " << _input_setpoint.x << ", " << _input_setpoint.y << ", " << _input_setpoint.z << std::endl << "velocity " << _input_setpoint.vx << ", " << _input_setpoint.vy << ", " << _input_setpoint.vz << std::endl << "acceleration " << _input_setpoint.acceleration[0] << ", " << _input_setpoint.acceleration[1] << ", " << _input_setpoint.acceleration[2] << std::endl << "output" << std::endl << "position " << _output_setpoint.x << ", " << _output_setpoint.y << ", " << _output_setpoint.z << std::endl << "velocity " << _output_setpoint.vx << ", " << _output_setpoint.vy << ", " << _output_setpoint.vz << std::endl << "acceleration " << _output_setpoint.acceleration[0] << ", " << _output_setpoint.acceleration[1] << ", " << _output_setpoint.acceleration[2] << std::endl; } } TEST_F(PositionControlBasicTest, InvalidState) { _input_setpoint.x = .1f; _input_setpoint.y = .2f; _input_setpoint.z = .3f; PositionControlStates states{}; states.position(0) = NAN; _position_control.setState(states); EXPECT_FALSE(runController()); states.velocity(0) = NAN; _position_control.setState(states); EXPECT_FALSE(runController()); states.position(0) = 0.f; _position_control.setState(states); EXPECT_FALSE(runController()); states.velocity(0) = 0.f; states.acceleration(1) = NAN; _position_control.setState(states); EXPECT_FALSE(runController()); } TEST_F(PositionControlBasicTest, UpdateHoverThrust) { const float hover_thrust = 0.6f; _position_control.setHoverThrust(hover_thrust); _input_setpoint.vx = 0.f; _input_setpoint.vy = 0.f; _input_setpoint.vz = -0.f; EXPECT_TRUE(runController()); EXPECT_EQ(_output_setpoint.thrust[2], -hover_thrust); const float hover_thrust_new = 0.7f; _position_control.updateHoverThrust(hover_thrust_new); EXPECT_TRUE(runController()); EXPECT_EQ(_output_setpoint.thrust[2], -hover_thrust); }
```

速度环控制源码

【mc_rate_control_params.c】

多旋翼姿态控制器参数。

```
PARAM_DEFINE_FLOAT(MC_ROLLRATE_P, 0.15f); PARAM_DEFINE_FLOAT(MC_ROLLRATE_I, 0.2f);
PARAM_DEFINE_FLOAT(MC_RR_INT_LIM, 0.30f); PARAM_DEFINE_FLOAT(MC_ROLLRATE_D, 0.003f);
PARAM_DEFINE_FLOAT(MC_ROLLRATE_FF, 0.0f); PARAM_DEFINE_FLOAT(MC_ROLLRATE_K, 1.0f);
PARAM_DEFINE_FLOAT(MC_PITCHRATE_P, 0.15f); PARAM_DEFINE_FLOAT(MC_PITCHRATE_I, 0.2f);
PARAM_DEFINE_FLOAT(MC_PR_INT_LIM, 0.30f); PARAM_DEFINE_FLOAT(MC_PITCHRATE_D, 0.003f);
PARAM_DEFINE_FLOAT(MC_PITCHRATE_FF, 0.0f); PARAM_DEFINE_FLOAT(MC_PITCHRATE_K, 1.0f);
PARAM_DEFINE_FLOAT(MC_YAWRATE_P, 0.2f); PARAM_DEFINE_FLOAT(MC_YAWRATE_I, 0.1f);
PARAM_DEFINE_FLOAT(MC_YR_INT_LIM, 0.30f); PARAM_DEFINE_FLOAT(MC_YAWRATE_D, 0.0f);
PARAM_DEFINE_FLOAT(MC_YAWRATE_FF, 0.0f); PARAM_DEFINE_FLOAT(MC_YAWRATE_K, 1.0f);
PARAM_DEFINE_FLOAT(MC_ACRO_R_MAX, 720.0f); PARAM_DEFINE_FLOAT(MC_ACRO_P_MAX, 720.0f);
PARAM_DEFINE_FLOAT(MC_ACRO_Y_MAX, 540.0f); PARAM_DEFINE_FLOAT(MC_ACRO_EXPO, 0.69f);
PARAM_DEFINE_FLOAT(MC_ACRO_EXPO_Y, 0.69f); PARAM_DEFINE_FLOAT(MC_ACRO_SUPEXPO, 0.7f);
PARAM_DEFINE_FLOAT(MC_ACRO_SUPEXPOY, 0.7f); PARAM_DEFINE_INT32(MC_BAT_SCALE_EN, 0);
```

【MulticopterRateControl.hpp】

多旋翼速度控制器的接口类。

```
class MulticopterRateControl : public ModuleBase<MulticopterRateControl>, public ModuleParams, public px4::WorkItem { ..... };
```

【MulticopterRateControl.cpp】

多旋翼速度控制器接口类的实现类。

MulticopterRateControl::MulticopterRateControl(bool vtol) : MulticopterRateControl::~MulticopterRateControl() bool MulticopterRateControl::init() void MulticopterRateControl::parameters_updated() void MulticopterRateControl::Run() int MulticopterRateControl::task_spawn(int argc, char *argv[]) int MulticopterRateControl::custom_command(int argc, char *argv[]) int MulticopterRateControl::print_usage(const char *reason) extern "C" __EXPORT int mc_rate_control_main(int argc, char *argv[])

【RateControl/RateControl.hpp】

PID 3轴角速度/角速度控制的接口类。

class RateControl { ..... };

【RateControl/RateControl.cpp】

PID 3轴角速度/角速度控制接口类的实现类。

void RateControl::setGains(const Vector3f &P, const Vector3f &I, const Vector3f &D) void RateControl::setSaturationStatus(const MultirotorMixer::saturation_status &status) Vector3f RateControl::update(const Vector3f &rate, const Vector3f &rate_sp, const Vector3f &angular_accel, const float dt, const bool landed) void RateControl::updateIntegral(Vector3f &rate_error, const float dt) void RateControl::getRateControlStatus(rate_ctrl_status_s &rate_ctrl_status)

【RateControl/RateControlTest.cpp】

PID 3轴角速度/角速度控制接口类的测试代码。

TEST(RateControlTest, AllZeroCase) { RateControl rate_control; Vector3f torque = rate_control.update(Vector3f(), Vector3f(), Vector3f(), 0.f, false); EXPECT_EQ(torque, Vector3f()); }