这是方法，变量太多，注意在msg文件中分析查看有关信息，便于理解

# 位置控制

文件夹：mc\_pos\_control

文件：mc\_pos\_control\_params.c mc\_pos\_control\_main.cpp

两个文件夹一个是位置控制中要使用的参数，另一个是主函数，包含位置控制任务函数。

**参数c文件解析：**

首先看一下都会有哪些参数会被使用到。

所有的参数都是属于这个组Multicopter Position Control。

**有关油门的参数**：

MPC Multicopter Position Control

THR thrust推力（throttle油门即推力）

MIN 最小

MAX 最大

HOVER 盘旋，悬停

MAN manual手动 人工

MPC\_THR\_MIN最小推力（自动模式）：0.05 – 1.0，默认0.12，避免零推力自由落体

MPC\_THR\_HOVER悬停推力:0.2 – 0.8，默认0.5，对应手动模式油门中值，用来表示悬停需要的推力，手动模式转定高模式可触发悬停。

MPC\_ALTCTL\_DZ定高油门曲线断点：0.0 – 0.2，默认0.1，定高油门曲线的中心区域的半宽量，即中心区域=（中点-断点）~（中点+断点），该区域内保持定高，垂直方向不增减推力（*个人理解*）

MPC\_ALTCTL\_DY定高油门曲线断点高度：0.0 – 0.2，默认0.0，表示中心区域油门曲线的斜率

MPC\_THR\_MAX最大推力（自动模式）：0.0 – 0.95，默认0.9，用于限制最大油门，0.8-0.9是推荐值

MPC\_MANTHR\_MIN最小推力（手动模式）：0.0 – 1.0，默认0.08，同自动模式

MPC\_MANTHR\_MAX最大推力（手动模式）：0.0 – 1.0，默认0.9，同自动模式

**有关位置和速度控制**：

MPC\_Z\_VEL\_MAX最大垂直速度：0.0 – 8.0，默认3.0，自动模式下最大垂直速度，自稳模式（包括高度和位置控制模式）下的终点（*不明白*）

MPC\_Z\_FF垂直速度前馈：0.0 – 1.0，默认0.5，自稳模式（包括高度和位置控制模式）下高度控制的前馈权重。0表示慢响应无超调，1表示快响应大超调

MPC\_XY\_VEL\_MAX最大水平速度：0.0 – 20，默认8.0，自动模式下最大垂直速度，自稳模式（包括高度和位置控制模式）下的终点（*不明白*）

MPC\_Z\_FF垂直速度前馈：0.0 – 1.0，默认0.5，位置控制模式下位置控制的前馈权重。0表示慢响应无超调，1表示快响应大超调

MPC\_TILTMAX\_AIR空中最大倾斜角，0.0 – 90，默认45，限制飞行中最大倾斜角（自动模式和位置控制控制）

MPC\_TILTMAX\_LND着陆时最大倾斜角，0.0 – 90，默认12，限制着陆模式下最大倾斜角

MPC\_LAND\_SPEED着陆下降速度，0.0-∞，默认0.5

MPC\_TKO\_SPEED起飞爬升速度，0.0-∞，默认1.5

MPC\_MAN\_R\_MAX最大手动roll，0.0-90，默认35

MPC\_MAN\_P\_MAX最大手动pitch，0.0-90，默认35

MPC\_MAN\_Y\_MAX最大手动yaw速度，0.0-∞，默认200

MPC\_HOLD\_XY\_DZ位置保持XY死区，0.0-1.0，默认0.1，位置保持时XY摇杆死区，即该区域内控制摇杆不会改变位置

MPC\_HOLD\_MAX\_XY位置保持最大XY速度，0.0-∞，默认0.8，位置保持时最大水平方向速度，0则不检测最大速度

MPC\_HOLD\_MAX\_Z位置保持最大Z速度，0.0-∞，默认0.6，位置保持时最大垂直方向速度，0则不检测最大速度

MPC\_VELD\_LP低通滤波器截至频率，0.0 - ∞，默认5.0,用于数字速度微分

MPC\_ACC\_HOR\_MAX最大水平加速度，0.0 - 15，默认10.0,用于速度控制模式中

有关PID的参数：

MPC\_Z\_P垂直位置误差比例增益P：0.0 - ∞，默认1.0

MPC\_Z\_VEL\_P垂直速度误差比例增益P：0.0 - ∞，默认0.2

MPC\_Z\_VEL\_I垂直速度误差积分增益I：0.0 - ∞，默认0.02，非零时在自稳和自动起飞模式下可以进行悬停油门估计

MPC\_Z\_VEL\_D垂直速度误差微分增益D：0.0 - ∞，默认0.0

MPC\_XY\_P水平位置误差比例增益P：0.0 - ∞，默认1.25

MPC\_XY\_VEL\_P水平速度误差比例增益P：0.0 - ∞，默认0.1

MPC\_XY\_VEL\_I水平速度误差积分增益I：0.0 - ∞，默认0.02，非零值可以抗风

MPC\_XY\_VEL\_D水平速度误差微分增益D：0.0 - ∞，默认0.01，小数值可以减少快速震荡，值过大会增加震荡

/\* Transforms a point in the geographic coordinate system to the local

\* azimuthal equidistant plane using the projection given by the argument

\* @param x north

\* @param y east

\* @param lat in degrees (47.1234567掳, not 471234567掳)

\* @param lon in degrees (8.1234567掳, not 81234567掳)

\* @return 0 if map\_projection\_init was called before, -1 else

\*/

\_\_EXPORT int map\_projection\_project

(const struct map\_projection\_reference\_s \*ref, double lat, double lon, float \*x,float \*y);

根据以下参数，将地理坐标系的点 转化为飞机坐标系下的点

Lat lon地理坐标系下的经度维度

X y

Geographic coordinate system地理坐标系统，是以经纬度为地图的存储单位的球面坐标系统

Projection coordinate system（投影坐标系统）投影坐标系统，实质上便是平面坐标系统，其地图单位通常为米。

将球面坐标转化为平面坐标的过程便称为投影。

用经纬度表示的是地理坐标系，也称大地坐标系。有时候用地理坐标系不够方便，人们比较习惯于使用平面坐标系，平面坐标系用xy表示。

将地理坐标系的点 转化为飞机坐标系下的点，

（这里https://en.wikipedia.org/wiki/Azimuthal\_equidistant\_projection

可以看到 等距方位投影即地图上的所有点都在距中心点比例正确的距离，并且在地图上的所有点都在距中心点的正确方位（方向））

注意：不要混淆了概念

在这个文件中 position | pos 这里位置指的是 水平面，x y

Altitude 指的就是垂直方向 z

爬升率指的是垂直方向z的速度

Velocity这个速度指的是 水平面上x y的速度

所以注意区分这些概念，以免理解错误！

**位置控制主函数解析：**

主函数的入口依然是以往的形式：

**extern** "C" \_\_EXPORT **int** **mc\_pos\_control\_main**(**int** argc, **char** \*argv[]);

在主函数中实例化MulticopterPositionControl类，接着使用类中的start()函数新建操作系统任务task\_main\_trampoline，函数指向任务主函数task\_main()。

在这个任务主函数中，以循环为界限，分为循环前和循环内。

**循环前**主要是mavlink通信，主题订阅，参数初始化，主题数据初始化。

**mavlink通信**需要获取句柄，执行该任务：

\_mavlink\_fd = px4\_open(MAVLINK\_LOG\_DEVICE, 0);

**主题订阅**就是通过orb\_subscribe()函数订阅需要的主题：

\_vehicle\_status\_sub = orb\_subscribe(ORB\_ID(vehicle\_status));

\_ctrl\_state\_sub = orb\_subscribe(ORB\_ID(control\_state));

\_att\_sp\_sub = orb\_subscribe(ORB\_ID(vehicle\_attitude\_setpoint));

\_control\_mode\_sub = orb\_subscribe(ORB\_ID(vehicle\_control\_mode));

\_params\_sub = orb\_subscribe(ORB\_ID(parameter\_update));

\_manual\_sub = orb\_subscribe(ORB\_ID(manual\_control\_setpoint));

\_arming\_sub = orb\_subscribe(ORB\_ID(actuator\_armed));

\_local\_pos\_sub = orb\_subscribe(ORB\_ID(vehicle\_local\_position));

\_pos\_sp\_triplet\_sub = orb\_subscribe(ORB\_ID(position\_setpoint\_triplet));

\_local\_pos\_sp\_sub = orb\_subscribe(ORB\_ID(vehicle\_local\_position\_setpoint));

\_global\_vel\_sp\_sub = orb\_subscribe(ORB\_ID(vehicle\_global\_velocity\_setpoint));

**参数初始化**通过parameters\_update(bool force)函数执行，这里手动执行更新（强制force=true），之后的更新都会根据主题更新的状况：

orb\_check(\_params\_sub, &updated);//用于检测参数是否已经更新

**if** (updated) {//有更新则更新该主题

orb\_copy(ORB\_ID(parameter\_update), \_params\_sub, &param\_upd);

}

接下来就是对参数表的更新，参数表就是之前的那个参数文件中的参数，先通过param\_get()函数获取数据，接着赋值给相应的的全局变量：

**if** (updated || force) {//自动更新通过参数更新消息，手动更新通过true

/\* update C++ param system \*/

updateParams();

//通过param\_get()函数来获取参数数值，句柄到参数，并配置大小限制

/\* update legacy C interface params \*/

param\_get(\_params\_handles.thr\_min, &\_params.thr\_min);

param\_get(\_params\_handles.thr\_max, &\_params.thr\_max);

param\_get(\_params\_handles.thr\_hover, &\_params.thr\_hover);

param\_get(\_params\_handles.alt\_ctl\_dz, &\_params.alt\_ctl\_dz);

param\_get(\_params\_handles.alt\_ctl\_dy, &\_params.alt\_ctl\_dy);

param\_get(\_params\_handles.tilt\_max\_air, &\_params.tilt\_max\_air);

\_params.tilt\_max\_air = math::radians(\_params.tilt\_max\_air);

param\_get(\_params\_handles.land\_speed, &\_params.land\_speed);

param\_get(\_params\_handles.tko\_speed, &\_params.tko\_speed);

param\_get(\_params\_handles.tilt\_max\_land, &\_params.tilt\_max\_land);

\_params.tilt\_max\_land = math::radians(\_params.tilt\_max\_land);

**float** v;

param\_get(\_params\_handles.xy\_p, &v);

\_params.pos\_p(0) = v;

\_params.pos\_p(1) = v;

param\_get(\_params\_handles.z\_p, &v);

\_params.pos\_p(2) = v;

param\_get(\_params\_handles.xy\_vel\_p, &v);

\_params.vel\_p(0) = v;

\_params.vel\_p(1) = v;

param\_get(\_params\_handles.z\_vel\_p, &v);

\_params.vel\_p(2) = v;

param\_get(\_params\_handles.xy\_vel\_i, &v);

\_params.vel\_i(0) = v;

\_params.vel\_i(1) = v;

param\_get(\_params\_handles.z\_vel\_i, &v);

\_params.vel\_i(2) = v;

param\_get(\_params\_handles.xy\_vel\_d, &v);

\_params.vel\_d(0) = v;

\_params.vel\_d(1) = v;

param\_get(\_params\_handles.z\_vel\_d, &v);

\_params.vel\_d(2) = v;

param\_get(\_params\_handles.xy\_vel\_max, &v);

\_params.vel\_max(0) = v;

\_params.vel\_max(1) = v;

param\_get(\_params\_handles.z\_vel\_max, &v);

\_params.vel\_max(2) = v;

param\_get(\_params\_handles.xy\_ff, &v);

v = math::constrain(v, 0.0f, 1.0f);

\_params.vel\_ff(0) = v;

\_params.vel\_ff(1) = v;

param\_get(\_params\_handles.z\_ff, &v);

v = math::constrain(v, 0.0f, 1.0f);

\_params.vel\_ff(2) = v;

param\_get(\_params\_handles.hold\_xy\_dz, &v);

v = math::constrain(v, 0.0f, 1.0f);

\_params.hold\_xy\_dz = v;

param\_get(\_params\_handles.hold\_max\_xy, &v);

\_params.hold\_max\_xy = (v < 0.0f ? 0.0f : v);

param\_get(\_params\_handles.hold\_max\_z, &v);

\_params.hold\_max\_z = (v < 0.0f ? 0.0f : v);

param\_get(\_params\_handles.acc\_hor\_max, &v);

\_params.acc\_hor\_max = v;

\_params.sp\_offs\_max = \_params.vel\_max.edivide(\_params.pos\_p) \* 2.0f;

/\* mc attitude control parameters\*/

/\* manual control scale \*/

param\_get(\_params\_handles.man\_roll\_max, &\_params.man\_roll\_max);

param\_get(\_params\_handles.man\_pitch\_max, &\_params.man\_pitch\_max);

param\_get(\_params\_handles.man\_yaw\_max, &\_params.man\_yaw\_max);

param\_get(\_params\_handles.global\_yaw\_max, &\_params.global\_yaw\_max);

\_params.man\_roll\_max = math::radians(\_params.man\_roll\_max);

\_params.man\_pitch\_max = math::radians(\_params.man\_pitch\_max);

\_params.man\_yaw\_max = math::radians(\_params.man\_yaw\_max);

\_params.global\_yaw\_max = math::radians(\_params.global\_yaw\_max);

param\_get(\_params\_handles.mc\_att\_yaw\_p, &v);

\_params.mc\_att\_yaw\_p = v;

/\* takeoff and land velocities should not exceed maximum \*/

\_params.tko\_speed = fminf(\_params.tko\_speed, \_params.vel\_max(2));

\_params.land\_speed = fminf(\_params.land\_speed, \_params.vel\_max(2));

}

**主题数据初始化poll\_subscriptions()**就是对刚才订阅的主题进行数据获取，通过check和copy函数完成初次主题数据的获取：

bool updated;

orb\_check(\_vehicle\_status\_sub, &updated);

**if** (updated) {

orb\_copy(ORB\_ID(vehicle\_status), \_vehicle\_status\_sub, &\_vehicle\_status);

/\* set correct uORB ID, depending on if vehicle is VTOL or not \*/

**if** (!\_attitude\_setpoint\_id) {

**if** (\_vehicle\_status.is\_vtol) {//根据是否是垂直起降使用不同的姿态参考点配置

\_attitude\_setpoint\_id = ORB\_ID(mc\_virtual\_attitude\_setpoint);

} **else** {

\_attitude\_setpoint\_id = ORB\_ID(vehicle\_attitude\_setpoint);

}

}

}

orb\_check(\_ctrl\_state\_sub, &updated);

**if** (updated) {

orb\_copy(ORB\_ID(control\_state), \_ctrl\_state\_sub, &\_ctrl\_state);

/\* get current rotation matrix and euler angles from control state quaternions \*/

math::Quaternion q\_att(\_ctrl\_state.q[0], \_ctrl\_state.q[1], \_ctrl\_state.q[2], \_ctrl\_state.q[3]);

\_R = q\_att.to\_dcm();//获取当前旋转矩阵

math::Vector<3> euler\_angles;

euler\_angles = \_R.to\_euler();

\_yaw = euler\_angles(2);//获取当前偏航角

}

orb\_check(\_att\_sp\_sub, &updated);

**if** (updated) {

orb\_copy(ORB\_ID(vehicle\_attitude\_setpoint), \_att\_sp\_sub, &\_att\_sp);

}

orb\_check(\_control\_mode\_sub, &updated);

**if** (updated) {

orb\_copy(ORB\_ID(vehicle\_control\_mode), \_control\_mode\_sub, &\_control\_mode);

}

orb\_check(\_manual\_sub, &updated);

**if** (updated) {

orb\_copy(ORB\_ID(manual\_control\_setpoint), \_manual\_sub, &\_manual);

}

orb\_check(\_arming\_sub, &updated);

**if** (updated) {

orb\_copy(ORB\_ID(actuator\_armed), \_arming\_sub, &\_arming);

}

orb\_check(\_local\_pos\_sub, &updated);

**if** (updated) {

orb\_copy(ORB\_ID(vehicle\_local\_position), \_local\_pos\_sub, &\_local\_pos);

}

还有一些循环前的准备工作：

/\*initialize values of critical structs until first regular update \*/

\_arming.armed = **false**; //解锁状态标志位

**bool** reset\_int\_z = **true**; //积分量重置标志位

**bool** reset\_int\_z\_manual = **false**;

**bool** reset\_int\_xy = **true**;

**bool** reset\_yaw\_sp = **true**; //偏航参考点重置标志位

**bool** was\_armed = **false**; //上次解锁状态标志位

//前一次执行时间

hrt\_abstime t\_prev = 0;

math::Vector<3> thrust\_int; //推力积分量，重置

thrust\_int.zero();

//设置为单位矩阵

math::Matrix<3, 3> R; //旋转矩阵，单位化（初始化）

R.identity();

/\* wakeup source \*/相关主题的执行，循环的判断依据来源

px4\_pollfd\_struct\_t fds[1];

fds[0].fd = \_local\_pos\_sub;

fds[0].events = POLLIN;

**循环内**包括主题数据获取情况判断，参数和主题数据更新，解锁配置，参考更新，控制前准备（包括速度微分量更新和保持标志位更新），位置和速度控制主函数，手动和姿态控制主函数，主题发布。

循环以\_task\_should\_exit标志位为判断依据（构造函数置false，析构函数置true），以500ms为超时时间等待数据，对于非正常情况继续下一次循环，正常获取数据的则进行以下数据处理。

**主题数据获取情况**

**int** pret = px4\_poll(&fds[0], (**sizeof**(fds) / **sizeof**(fds[0])), 500);

/\* 超时 \*/

**if** (pret == 0) {

**continue**;

}

/\* 不期望的情况 \*/

**if** (pret < 0) {

warn("poll error %d, %d", pret, errno);

**continue**;

}

**参数和主题数据更新**

主题数据的获取根据类中的成员函数poll\_subscriptions()实现，对订阅的主题数据进行更新,具体的函数和之前初始化主题数据是相同的，详细参照上面的函数。

参数更新还是使用parameters\_update(bool force)函数实现，不过这里使用自动更新force=false，即有新数据的情况才进行更新，具体的函数和之前初始化主题数据是相同的，详细参照上面的函数。

**解锁配置**

首先对运行时间进行配置，接着配置控制块的单位时间，最后对初次解锁配置需要重置的参数标志位，以及垂直起降情况的配置（对于旋翼飞行器不起作用）：

hrt\_abstime t = hrt\_absolute\_time();//绝对执行时间t，运行到当前时间

**float** dt = t\_prev != 0 ? (t - t\_prev) \* 0.000001f : 0.0f;//距上次的相对时间，执行单位时间dt

t\_prev = t;//更新上一次绝对执行时间

// set dt for control blocks

setDt(dt);//为控制块设置单位时间，未分析具体含义

/////初次解锁，重置参考点和积分/////////////////

**if** (\_control\_mode.flag\_armed && !was\_armed) {//was\_armed之后和前者相同

/\* 参考点和积分量需要在解锁时重置 \*/

\_reset\_pos\_sp = **true**;

\_reset\_alt\_sp = **true**;

\_vel\_sp\_prev.zero();

reset\_int\_z = **true**;

reset\_int\_xy = **true**;

reset\_yaw\_sp = **true**;

}

/////垂直起降情况，为固定翼重置偏航和姿态参考点（无用）/////////////////

**if** (\_vehicle\_status.is\_vtol) {

**if** (!\_vehicle\_status.is\_rotary\_wing) {//旋翼为true，四旋翼为true，这里用于固定翼

reset\_yaw\_sp = **true**;

\_reset\_alt\_sp = **true**;

}

}

/////更新前一次解锁情况/////////////////

was\_armed = \_control\_mode.flag\_armed;

**参考更新**

参考更新主要是用位置估计的参数来更新当前位置参考点，使用的是映射函数map\_projection\_reproject()和map\_projection\_project()，这种方式将位置转换为经纬度和高度，然后用位置估计参数来更新经纬度和高度，接着转换回位置参考点，是GPS数据转换的方式。

**if** (\_local\_pos.ref\_timestamp != \_ref\_timestamp) {//参考时间帧改变则更新参考，构造函数置0，该函数最后更新为当前参考时间帧

**double** lat\_sp, lon\_sp;

**float** alt\_sp = 0.0f;

**if** (\_ref\_timestamp != 0) {//获取当前的位置和高度参考数值

/\* calculate current position setpoint in global frame \*/

map\_projection\_reproject(&\_ref\_pos, \_pos\_sp(0), \_pos\_sp(1), &lat\_sp, &lon\_sp);

alt\_sp = \_ref\_alt - \_pos\_sp(2);

}

/\* update local projection reference \*///更新当前的位置和高度参考数值

map\_projection\_init(&\_ref\_pos, \_local\_pos.ref\_lat, \_local\_pos.ref\_lon);

\_ref\_alt = \_local\_pos.ref\_alt;

**if** (\_ref\_timestamp != 0) {

/\* reproject position setpoint to new reference \*///映射当前的位置和高度参考数值到参考系

map\_projection\_project(&\_ref\_pos, lat\_sp, lon\_sp, &\_pos\_sp.data[0], &\_pos\_sp.data[1]);

\_pos\_sp(2) = -(alt\_sp - \_ref\_alt);

}

\_ref\_timestamp = \_local\_pos.ref\_timestamp;//更新参考时间为vehicle\_local\_position中时间帧

}

**控制前准备（包括速度微分量更新和保持标志位更新）**

之后会使用速度微分（PID），这里需要更新速度的微分量，实现方式是通过低通滤波器。

**if** (\_local\_pos.timestamp > 0) {

**if** (PX4\_ISFINITE(\_local\_pos.x) &&

PX4\_ISFINITE(\_local\_pos.y) &&

PX4\_ISFINITE(\_local\_pos.z)) {

\_pos(0) = \_local\_pos.x;//北东地坐标系下位置（数据有效时更新）

\_pos(1) = \_local\_pos.y;

\_pos(2) = \_local\_pos.z;

}

**if** (PX4\_ISFINITE(\_local\_pos.vx) &&

PX4\_ISFINITE(\_local\_pos.vy) &&

PX4\_ISFINITE(\_local\_pos.vz)) {

\_vel(0) = \_local\_pos.vx;//北东地坐标系下速度（数据有效时更新）

\_vel(1) = \_local\_pos.vy;

\_vel(2) = \_local\_pos.vz;

}

//获取速度的微分量，一个低通滤波器的实现

\_vel\_err\_d(0) = \_vel\_x\_deriv.update(-\_vel(0));

\_vel\_err\_d(1) = \_vel\_y\_deriv.update(-\_vel(1));

\_vel\_err\_d(2) = \_vel\_z\_deriv.update(-\_vel(2));

}

接着是对位置保持和高度保持的标志位进行判断，在手动控制中需要使用位置保持和高度保持，默认是不使用（构造函数初始化为false）

**if** (!\_control\_mode.flag\_control\_position\_enabled || !\_control\_mode.flag\_control\_manual\_enabled) {

\_pos\_hold\_engaged = **false**;

//位置保持使用标志位，手动控制中使用，这里禁用，构造函数中置false禁用

}

**if** (!\_control\_mode.flag\_control\_altitude\_enabled || !\_control\_mode.flag\_control\_manual\_enabled) {

\_alt\_hold\_engaged = **false**;

//高度保持使用标志位，手动控制中使用，这里禁用，构造函数中置false禁用

}

**位置和速度控制主函数**

A．这里包含四种方式，高度控制，位置控制，爬升率控制（垂直速度），速度控制（水平速度）。任一种都可以进入控制主函数（通过主题vehicle\_control\_mode获取控制模式），下面是判断是否满足条件：

**if** (\_control\_mode.flag\_control\_altitude\_enabled ||

//高度控制，位置控制，爬升率（垂直速度）控制，速度（水平速度）控制

\_control\_mode.flag\_control\_position\_enabled ||

//以上任一使能就进行以下位置控制

\_control\_mode.flag\_control\_climb\_rate\_enabled ||

\_control\_mode.flag\_control\_velocity\_enabled)

控制中默认开启位置控制和高度控制：

\_vel\_ff.zero();//未使用，构造函数中置0

\_run\_pos\_control = **true**;//位置控制，默认，也可禁用然后直接使用速度控制

\_run\_alt\_control = **true**;//高度控制，默认，也可禁用然后直接使用速度控制

之后是具体实现，首先由三种控制源：手动控制，离线控制和自动控制。这是根据控制模式的主题vehicle\_control\_mode来进行判断的。

**a手动控制：**

/\* manual control \*/

control\_manual(dt);//手动控制\*

\_mode\_auto = **false**;//自动控制模式去能

在手动控制函数中，使用遥控输入的特定比例作为速度参考点，另外还有辅助速度模式，弃用位置（高度）保持，使用速度控制。

math::Vector<3> req\_vel\_sp;

// X,Y in local frame and Z in global (D), in [-1,1] normalized range

req\_vel\_sp.zero();//速度参考初始化，置零

**if** (\_control\_mode.flag\_control\_altitude\_enabled) {//高度控制使能z

/\* set vertical velocity setpoint with throttle stick \*///\_manual.z在0-1，这里dz为0.1，dy为0，死区内返回0，死区外返回-1到1，即表示正反的速度，这里坐标系朝地为正，所以需要一个负号

req\_vel\_sp(2) = -*scale\_control*(\_manual.z - 0.5f, 0.5f, \_params.alt\_ctl\_dz, \_params.alt\_ctl\_dy); // D

}//设置垂直速度参考，通过尺度控制器（将0-1转换为-1——1）\*

**if** (\_control\_mode.flag\_control\_position\_enabled) {//位置控制使能xy

/\* set horizontal velocity setpoint with roll/pitch stick \*/

req\_vel\_sp(0) = \_manual.x;//设置水平速度x参考,\_manual.x在-1——1,stick前后，直接作为参考

req\_vel\_sp(1) = \_manual.y;//设置水平速度y参考,\_manual.y在-1——1,stick左右，直接作为参考

}

**if** (\_control\_mode.flag\_control\_altitude\_enabled) {//高度控制使能z

/\* reset alt setpoint to current altitude if needed \*/

reset\_alt\_sp();//重置高度参考z，使用当前高度\*

}

**if** (\_control\_mode.flag\_control\_position\_enabled) {//位置控制使能xy

/\* reset position setpoint to current position if needed \*/

reset\_pos\_sp();//重置位置参考xy，使用当前位置\*

}

/\* limit velocity setpoint \*///限制速度参考

**float** req\_vel\_sp\_norm = req\_vel\_sp.length();//求平方根，对于大于1的情况进行范围限制

**if** (req\_vel\_sp\_norm > 1.0f) {

req\_vel\_sp /= req\_vel\_sp\_norm;

}

/\* \_req\_vel\_sp scaled to 0..1, scale it to max speed and rotate around yaw \*/

math::Matrix<3, 3> R\_yaw\_sp;

R\_yaw\_sp.from\_euler(0.0f, 0.0f, \_att\_sp.yaw\_body);//NED下机体偏航得到绕z旋转的矩阵

math::Vector<3> req\_vel\_sp\_scaled = R\_yaw\_sp \* req\_vel\_sp.emult(//根据参数和尺度设置实际速度大小，max值xy为8，z为3

\_params.vel\_max); // in NED and scaled to actual velocity

//辅助速度模式，用户控制速度，速度过小位置不变

/\* horizontal axes \*///水平方向xy，这里主要是配置位置保持

**if** (\_control\_mode.flag\_control\_position\_enabled) {//位置控制使能是前提

/\* check for pos. hold \*///速度参考点绝对值（0-1）小于死区（这里为0.1）

**if** (fabsf(req\_vel\_sp(0)) < \_params.hold\_xy\_dz && fabsf(req\_vel\_sp(1)) < \_params.hold\_xy\_dz) {

**if** (!\_pos\_hold\_engaged) {//位置保持未使用，接下来判断参数是否可以开启

**if** (\_params.hold\_max\_xy < FLT\_EPSILON || (fabsf(\_vel(0)) < \_params.hold\_max\_xy

&& fabsf(\_vel(1)) < \_params.hold\_max\_xy)) {//hold\_max\_xy默认为0.8

\_pos\_hold\_engaged = **true**;

} **else** {//当前水平速度大于位置保持的速度最大值限制则不能启用位置保持

\_pos\_hold\_engaged = **false**;

}

}

} **else** {

\_pos\_hold\_engaged = **false**;//速度参考点绝对值（0-1）小于死区（这里为0.1）不满足则不适用位置保持

}

/\* set requested velocity setpoint \*/

**if** (!\_pos\_hold\_engaged) {//位置保持不可用（根据上面的结果）

\_pos\_sp(0) = \_pos(0);//重新配置参考位置，根据当前的位置

\_pos\_sp(1) = \_pos(1);

\_run\_pos\_control = **false**; /\* request velocity setpoint to be used, instead of position setpoint \*/

\_vel\_sp(0) = req\_vel\_sp\_scaled(0);//不使用位置控制，这里配置参考速度，根据之前获取的实际速度

\_vel\_sp(1) = req\_vel\_sp\_scaled(1);

}

}

/\* vertical axis \*///垂直方向z，这里主要是配置高度保持

**if** (\_control\_mode.flag\_control\_altitude\_enabled) {//高度控制使能是前提

/\* check for pos. hold \*///垂直速度上面要求绝对值小于FLT\_EPSILON才启用高度保持

**if** (fabsf(req\_vel\_sp(2)) < FLT\_EPSILON) {

**if** (!\_alt\_hold\_engaged) {//高度保持为启用那么继续判断是否满足要求

**if** (\_params.hold\_max\_z < FLT\_EPSILON || fabsf(\_vel(2)) < \_params.hold\_max\_z) {

\_alt\_hold\_engaged = **true**;//hold\_max\_z默认为0.6

} **else** {//当前垂直速度大于高度保持的速度最大值限制则不能启用高度保持

\_alt\_hold\_engaged = **false**;

}

}

} **else** {//垂直速度不满足要求放弃高度保持

\_alt\_hold\_engaged = **false**;

}

/\* set requested velocity setpoint \*/

**if** (!\_alt\_hold\_engaged) {//高度保持不可用（根据上面的结果）

\_run\_alt\_control = **false**; /\* request velocity setpoint to be used, instead of altitude setpoint \*/

\_vel\_sp(2) = req\_vel\_sp\_scaled(2);//不使用高度控制，这里配置参考速度，根据之前获取的实际速度

\_pos\_sp(2) = \_pos(2);//重新配置参考高度，根据当前的高度

}

}

**b离线模式：**

/\* offboard control \*/

control\_offboard(dt);//离线控制\*

\_mode\_auto = **false**;//自动控制模式去能

离线模式需要三个时刻的位置参考数据，接着根据控制器的不同选择不同的参考点配置方案：

**bool** updated;

orb\_check(\_pos\_sp\_triplet\_sub, &updated);

**if** (updated) {；

orb\_copy(ORB\_ID(position\_setpoint\_triplet), \_pos\_sp\_triplet\_sub, &\_pos\_sp\_triplet);

}//这里获取新内容，前一次，本次和下一次的位置参考数据

**if** (\_pos\_sp\_triplet.current.valid) {//本次参考点有效

**if** (\_control\_mode.flag\_control\_position\_enabled && \_pos\_sp\_triplet.current.position\_valid) {//位置控制

/\* control position \*///本次位置参考点有效并且位置控制使能则将本次位置参考点作为参考NED

\_pos\_sp(0) = \_pos\_sp\_triplet.current.x;

\_pos\_sp(1) = \_pos\_sp\_triplet.current.y;

} **else** **if** (\_control\_mode.flag\_control\_velocity\_enabled && \_pos\_sp\_triplet.current.velocity\_valid) {//速度控制

/\* control velocity \*///本次速度参考点有效并且速度控制使能则将本次速度参考点作为参考NED，位置参考需要重置，并且不使用位置控制

/\* reset position setpoint to current position if needed \*/

reset\_pos\_sp();//重置位置参考xy，使用当前位置\*

/\* set position setpoint move rate \*/

\_vel\_sp(0) = \_pos\_sp\_triplet.current.vx;

\_vel\_sp(1) = \_pos\_sp\_triplet.current.vy;

\_run\_pos\_control = **false**; /\* request velocity setpoint to be used, instead of position setpoint \*/

}

**if** (\_pos\_sp\_triplet.current.yaw\_valid) {//本次偏航角有效，姿态参考的偏航使用本次偏航值（弧度）

\_att\_sp.yaw\_body = \_pos\_sp\_triplet.current.yaw;

} **else** **if** (\_pos\_sp\_triplet.current.yawspeed\_valid) {//本次偏航角速度有效，姿态参考的偏航值自增单位偏航值（本次偏航角速度\*时间）

\_att\_sp.yaw\_body = \_att\_sp.yaw\_body + \_pos\_sp\_triplet.current.yawspeed \* dt;

}

**if** (\_control\_mode.flag\_control\_altitude\_enabled && \_pos\_sp\_triplet.current.position\_valid) {//高度控制使能并且当前位置参考点有效

/\* Control altitude \*///高度控制

\_pos\_sp(2) = \_pos\_sp\_triplet.current.z;//高度参考使用本次高度值NED

} **else** **if** (\_control\_mode.flag\_control\_climb\_rate\_enabled && \_pos\_sp\_triplet.current.velocity\_valid) {//爬升率控制使能并且本次速度参考点有效

/\* reset alt setpoint to current altitude if needed \*///爬升率控制，同位置先重置高度参考

reset\_alt\_sp();//重置高度参考z，使用当前高度\*

/\* set altitude setpoint move rate \*///垂直速度参考使用本次z轴速度，并且不适用高度控制

\_vel\_sp(2) = \_pos\_sp\_triplet.current.vz;

\_run\_alt\_control = **false**; /\* request velocity setpoint to be used, instead of position setpoint \*/

}

} **else** {//本次参考点无效那么重新配置参考点

reset\_pos\_sp();//重置位置参考xy，使用当前位置\*

reset\_alt\_sp();//重置位置参考z，使用当前高度\*

}

**c自动模式:**

/\* AUTO \*/

control\_auto(dt);//自动控制\*

自动模式同样需要连续的三个位置参考点数据，并且本次的参考点数据必须有效才能够执行相关配置：

//非手动非离线就进入自动模式

/\* reset position setpoint on AUTO mode activation or if we are not in MC mode \*/

**if** (!\_mode\_auto || !\_vehicle\_status.is\_rotary\_wing) {//重新进入自动模式的情况或者非多旋翼的情况（垂直起降VTOL中就是多旋翼）

**if** (!\_mode\_auto) {//非自动模式下需要改为自动模式

\_mode\_auto = **true**;

}

///需要重置位置和高度参考点

\_reset\_pos\_sp = **true**;

\_reset\_alt\_sp = **true**;

reset\_pos\_sp();//重置位置参考xy，使用当前位置\*

reset\_alt\_sp();//重置高度参考z，使用当前高度\*

}

//Poll position setpoint//同离线模式需要前一次，本次和下一次的参考点数据

**bool** updated;

orb\_check(\_pos\_sp\_triplet\_sub, &updated);

**if** (updated) {

orb\_copy(ORB\_ID(position\_setpoint\_triplet), \_pos\_sp\_triplet\_sub, &\_pos\_sp\_triplet);

//Make sure that the position setpoint is valid

**if** (!PX4\_ISFINITE(\_pos\_sp\_triplet.current.lat) ||//本次纬度（°）

!PX4\_ISFINITE(\_pos\_sp\_triplet.current.lon) ||//本次经度（°）

!PX4\_ISFINITE(\_pos\_sp\_triplet.current.alt)) {//本次高度（m）

//对于任一数据无效的情况，本次参考点的数据就无效\_pos\_sp\_triplet.current.valid = **false**;

}

}

**bool** current\_setpoint\_valid = **false**;

**bool** previous\_setpoint\_valid = **false**;

math::Vector<3> prev\_sp;

math::Vector<3> curr\_sp;

**if** (\_pos\_sp\_triplet.current.valid) {//本次参考点数据有效

/\* project setpoint to local frame \*/

map\_projection\_project(&\_ref\_pos, \_pos\_sp\_triplet.current.lat, \_pos\_sp\_triplet.current.lon, &curr\_sp.data[0], &curr\_sp.data[1]);//位置xy使用经纬度映射关系获得

curr\_sp(2) = -(\_pos\_sp\_triplet.current.alt - \_ref\_alt);//高度为相对高度

**if** (PX4\_ISFINITE(curr\_sp(0)) && PX4\_ISFINITE(curr\_sp(1)) && PX4\_ISFINITE(curr\_sp(2))) {//对于获得的本次参考点数据有效的情况则标志为有效

current\_setpoint\_valid = **true**;

}

}

**if** (\_pos\_sp\_triplet.previous.valid) {//前一次参考点有效

map\_projection\_project(&\_ref\_pos, \_pos\_sp\_triplet.previous.lat, \_pos\_sp\_triplet.previous.lon, &prev\_sp.data[0], &prev\_sp.data[1]);//位置xy使用经纬度映射关系获得

prev\_sp(2) = -(\_pos\_sp\_triplet.previous.alt - \_ref\_alt);//高度为相对高度

**if** (PX4\_ISFINITE(prev\_sp(0)) && PX4\_ISFINITE(prev\_sp(1)) && PX4\_ISFINITE(prev\_sp(2))) {//对于获得的前一次参考点数据有效的情况则标志为有效

previous\_setpoint\_valid = **true**;

}

}

/////本次参考点数据有效（根据以上的判断）///////////////////////////

**if** (current\_setpoint\_valid) {

/\* scaled space: 1 == position error resulting max allowed speed \*///尺度变换，速度最大值vel\_max为[8 8 3]',位置控制P参数pos\_p为[1.25 1.25 1]'

math::Vector<3> scale = \_params.pos\_p.edivide(\_params.vel\_max); // **TODO** add mult param here//用\_params.pos\_p的每一个元素除以\_params.vel\_max对应位置的每一个元素

/\* convert current setpoint to scaled space \*/

math::Vector<3> curr\_sp\_s = curr\_sp.emult(scale);//用curr\_sp的每一个元素乘以scale对应位置的每一个元素

/\* by default use current setpoint as is \*///位置参考点（尺度变换后的）默认使用本次参考点（上面得到的尺度变换值）

math::Vector<3> pos\_sp\_s = curr\_sp\_s;

**if** (\_pos\_sp\_triplet.current.type == position\_setpoint\_s::*SETPOINT\_TYPE\_POSITION* && previous\_setpoint\_valid) {

/\* follow "previous - current" line \*///本次参考点为位置类型并且前一次参考点数据有效

**if** ((curr\_sp - prev\_sp).length() > MIN\_DIST) {//三维距离大于最小距离（0.01）

/\* find X - cross point of unit sphere and trajectory \*///单位球面和轨迹的交叉点

math::Vector<3> pos\_s = \_pos.emult(scale);//位置（构造函数置0，初始化使用消息中位置xyz）的尺度变换值

math::Vector<3> prev\_sp\_s = prev\_sp.emult(scale);//前一次参考点的尺度变换值

math::Vector<3> prev\_curr\_s = curr\_sp\_s - prev\_sp\_s;//本次与前一次参考点尺度变换值的差值

math::Vector<3> curr\_pos\_s = pos\_s - curr\_sp\_s;//位置参考点尺度变换值差值（初始和本次）

**float** curr\_pos\_s\_len = curr\_pos\_s.length();//本次参考点距初始点的尺度变换值大小（用于之后的判断，作出不同设置）

**if** (curr\_pos\_s\_len < 1.0f) {//本次距离小于1

/\* copter is closer to waypoint than unit radius \*///路径点在单位半径内

/\* check next waypoint and use it to avoid slowing down when passing via waypoint \*///检查下个路径点并使用，避免过路径点时减速

**if** (\_pos\_sp\_triplet.next.valid) {//对于下一次参考点有效的

math::Vector<3> next\_sp;//首先得到下一次参考点数据

map\_projection\_project(&\_ref\_pos, \_pos\_sp\_triplet.next.lat, \_pos\_sp\_triplet.next.lon, &next\_sp.data[0], &next\_sp.data[1]);

next\_sp(2) = -(\_pos\_sp\_triplet.next.alt - \_ref\_alt);

**if** ((next\_sp - curr\_sp).length() > MIN\_DIST) {//距离大于0.01（下一次参考点到本次参考点）

math::Vector<3> next\_sp\_s = next\_sp.emult(scale);//同样获得下一次参考点的尺度变换值

/\* calculate angle prev - curr - next \*/

math::Vector<3> curr\_next\_s = next\_sp\_s - curr\_sp\_s;//未来差距

math::Vector<3> prev\_curr\_s\_norm = prev\_curr\_s.normalized();//前次差距归一化（平方和为1，相当于两个cos值）

/\* cos(a) \* curr\_next, a = angle between current and next trajectory segments \*///两个向量之间的夹角（本次与下一次）

**float** cos\_a\_curr\_next = prev\_curr\_s\_norm \* curr\_next\_s;

/\* cos(b), b = angle pos - curr\_sp - prev\_sp \*///三个点之间的夹角

**float** cos\_b = -curr\_pos\_s \* prev\_curr\_s\_norm / curr\_pos\_s\_len;

**if** (cos\_a\_curr\_next > 0.0f && cos\_b > 0.0f) {//角度小于90

**float** curr\_next\_s\_len = curr\_next\_s.length();//求未来差距长度

/\* if curr - next distance is larger than unit radius, limit it \*/

**if** (curr\_next\_s\_len > 1.0f) {//未来差距长于单位1，限制大小

cos\_a\_curr\_next /= curr\_next\_s\_len;

}

/\* feed forward position setpoint offset \*///将位置参考点偏差前馈到位置参考点尺度值上面

math::Vector<3> pos\_ff = prev\_curr\_s\_norm \* cos\_a\_curr\_next \* cos\_b \* cos\_b \* (1.0f - curr\_pos\_s\_len) \* (1.0f - expf(-curr\_pos\_s\_len \* curr\_pos\_s\_len \* 20.0f));//这个求偏差值（不明白具体含义）

pos\_sp\_s += pos\_ff;

}

}

}

} **else** {//本次距离大于1（飞行器离路径点比较远）

**bool** near = cross\_sphere\_line(pos\_s, 1.0f, prev\_sp\_s, curr\_sp\_s, pos\_sp\_s);

**if** (near) {//单位球与轨迹有交，即飞行器在轨迹附近

/\* unit sphere crosses trajectory \*/

} **else** {//距离轨迹太远

/\* copter is too far from trajectory \*/

/\* if copter is behind prev waypoint, go directly to prev waypoint \*/

**if** ((pos\_sp\_s - prev\_sp\_s) \* prev\_curr\_s < 0.0f) {//满足此将前一次参考点作为位置参考

pos\_sp\_s = prev\_sp\_s;

}

/\* if copter is in front of curr waypoint, go directly to curr waypoint \*/

**if** ((pos\_sp\_s - curr\_sp\_s) \* prev\_curr\_s > 0.0f) {//满足此将本次参考点作为位置参考

pos\_sp\_s = curr\_sp\_s;

}

pos\_sp\_s = pos\_s + (pos\_sp\_s - pos\_s).normalized();//更新位置参考点（根据位置）

}

}

}

}

/\* move setpoint not faster than max allowed speed \*///限制最大速度

math::Vector<3> pos\_sp\_old\_s = \_pos\_sp.emult(scale);//位置参考尺度值

/\* difference between current and desired position setpoints, 1 = max speed \*/

math::Vector<3> d\_pos\_m = (pos\_sp\_s - pos\_sp\_old\_s).edivide(\_params.pos\_p);//根据以上的位置参考点来求得偏差

**float** d\_pos\_m\_len = d\_pos\_m.length();//偏差的长度

**if** (d\_pos\_m\_len > dt) {//偏差大于dt，获取先位置参考尺度值

pos\_sp\_s = pos\_sp\_old\_s + (d\_pos\_m / d\_pos\_m\_len \* dt).emult(\_params.pos\_p);

}

/\* scale result back to normal space \*/

\_pos\_sp = pos\_sp\_s.edivide(scale);//将尺度去掉，回到原值

/\* update yaw setpoint if needed \*///对于本次偏航值有效的情况可以将本次偏航值更新为姿态参考中的偏航值

**if** (PX4\_ISFINITE(\_pos\_sp\_triplet.current.yaw)) {

\_att\_sp.yaw\_body = \_pos\_sp\_triplet.current.yaw;

}

/\*

\* if we're already near the current takeoff setpoint don't reset in case we switch back to posctl.

\* this makes the takeoff finish smoothly.

\*/

**if** ((\_pos\_sp\_triplet.current.type == position\_setpoint\_s::*SETPOINT\_TYPE\_TAKEOFF*//起飞参考点

|| \_pos\_sp\_triplet.current.type == position\_setpoint\_s::*SETPOINT\_TYPE\_LOITER*)//悬停参考点

&& \_pos\_sp\_triplet.current.acceptance\_radius > 0.0f

/\* need to detect we're close a bit before the navigator switches from takeoff to next waypoint \*/

&& (\_pos - \_pos\_sp).length() < \_pos\_sp\_triplet.current.acceptance\_radius \* 1.2f) {//距离小于1.2半径

\_reset\_pos\_sp = **false**;//不进行位置和高度参考点的重新设置（号称平滑）

\_reset\_alt\_sp = **false**;

/\* otherwise: in case of interrupted mission don't go to waypoint but stay at current position \*///任务中断，停留

} **else** {

\_reset\_pos\_sp = **true**;

\_reset\_alt\_sp = **true**;

}

} **else** {//没有本次参考点，那就啥都不做

/\* no waypoint, do nothing, setpoint was already reset \*/

}

接着是控制实现，有三种情况idle state，landed和其他情况。

这里有一个风向标模式，有关偏航控制的配置，不清楚具体作用：

/\* weather-vane mode for vtol: disable yaw control \*///风向标模式，垂直起降，禁用偏航控制

**if** (!\_control\_mode.flag\_control\_manual\_enabled && \_pos\_sp\_triplet.current.disable\_mc\_yaw\_control == **true**) {

\_att\_sp.disable\_mc\_yaw\_control = **true**;//非手动控制并且允许使能多旋翼偏航控制

} **else** {

/\* reset in case of setpoint updates \*///其他情况要重置为false

\_att\_sp.disable\_mc\_yaw\_control = **false**;

}

**idle state：**不使用控制器

**if** (!\_control\_mode.flag\_control\_manual\_enabled && \_pos\_sp\_triplet.current.valid&& \_pos\_sp\_triplet.current.type == position\_setpoint\_s::*SETPOINT\_TYPE\_IDLE*) {

/\* idle state, don't run controller and set zero thrust \*///非手动模式&本次参考数据有效&idle，那么不用控制器，推力为0

R.identity();//重置为单位矩阵，然后内存拷贝到姿态参考消息中的R\_body

memcpy(&\_att\_sp.R\_body[0], R.data, **sizeof**(\_att\_sp.R\_body));

\_att\_sp.R\_valid = **true**;//姿态参考消息中数据初始化——旋转矩阵有效性，

\_att\_sp.roll\_body = 0.0f;//欧拉角和推力

\_att\_sp.pitch\_body = 0.0f;

\_att\_sp.yaw\_body = \_yaw;

\_att\_sp.thrust = 0.0f;

\_att\_sp.timestamp = hrt\_absolute\_time();//时间帧

/\* publish attitude setpoint \*/

**if** (\_att\_sp\_pub != **nullptr**) {//对于非空数据进行发布\_attitude\_setpoint\_id

orb\_publish(\_attitude\_setpoint\_id, \_att\_sp\_pub, &\_att\_sp);

} **else** **if** (\_attitude\_setpoint\_id) {//否则对于有数据的情况进行公告\_attitude\_setpoint\_id

\_att\_sp\_pub = orb\_advertise(\_attitude\_setpoint\_id, &\_att\_sp);

}

**landed:**也不使用控制器

} **else** **if** (\_control\_mode.flag\_control\_manual\_enabled

&& \_vehicle\_status.condition\_landed) {//手动控制&着陆状态

/\* don't run controller when landed \*///着陆不调用控制器

\_reset\_pos\_sp = **true**;//位置和高度参考点重置标志位置true

\_reset\_alt\_sp = **true**;

\_mode\_auto = **false**;//自动模式关闭

reset\_int\_z = **true**;//xyz的积分要重置

reset\_int\_xy = **true**;

R.identity();//接下来同idle模式，对旋转矩阵，欧拉角，推力和时间帧进行处理

memcpy(&\_att\_sp.R\_body[0], R.data, **sizeof**(\_att\_sp.R\_body));

\_att\_sp.R\_valid = **true**;

\_att\_sp.roll\_body = 0.0f;

\_att\_sp.pitch\_body = 0.0f;

\_att\_sp.yaw\_body = \_yaw;

\_att\_sp.thrust = 0.0f;

\_att\_sp.timestamp = hrt\_absolute\_time();

/\* publish attitude setpoint \*///此处发布姿态参考点数据，同idle

**if** (\_att\_sp\_pub != **nullptr**) {

orb\_publish(\_attitude\_setpoint\_id, \_att\_sp\_pub, &\_att\_sp);

} **else** **if** (\_attitude\_setpoint\_id) {

\_att\_sp\_pub = orb\_advertise(\_attitude\_setpoint\_id, &\_att\_sp);

}

**其他情况**,这也就是正常需要调用控制器的情况啦。相对代码量大了些，毕竟位置控制和速度控制在一块，每个还分为水平和垂直两块。此外还需要考虑着陆，起飞等特殊情况。整个控制环节发布的数据除了之后的位置参考数据外，还有速度参考点数据(处理数据：位置，速度，推力等)。

} **else** {//剔除两种特殊情况后（idle和land），其他情况是需要位置控制和姿态控制器的（去能情况下使用已有速度参考点）

/\* run position & altitude controllers, if enabled (otherwise use already computed velocity setpoints) \*/

**if** (\_run\_pos\_control) {//位置控制，默认使用，以下获取速度参考点（位置差\*P）

\_vel\_sp(0) = (\_pos\_sp(0) - \_pos(0)) \* \_params.pos\_p(0);

\_vel\_sp(1) = (\_pos\_sp(1) - \_pos(1)) \* \_params.pos\_p(1);

}

**if** (\_run\_alt\_control) {//高度控制，默认使用，以下获取速度参考点（位置差\*P）

\_vel\_sp(2) = (\_pos\_sp(2) - \_pos(2)) \* \_params.pos\_p(2);

}

/\* make sure velocity setpoint is saturated in xy\*/

**float** vel\_norm\_xy = sqrtf(\_vel\_sp(0) \* \_vel\_sp(0) +

\_vel\_sp(1) \* \_vel\_sp(1));

**if** (vel\_norm\_xy > \_params.vel\_max(0)) {//这里假设vel\_max(0) == vel\_max(1)，对于xy平面速度平方根过大的需要进行限制

/\* note assumes vel\_max(0) == vel\_max(1) \*/

\_vel\_sp(0) = \_vel\_sp(0) \* \_params.vel\_max(0) / vel\_norm\_xy;

\_vel\_sp(1) = \_vel\_sp(1) \* \_params.vel\_max(1) / vel\_norm\_xy;

}

/\* make sure velocity setpoint is saturated in z\*///同样的方式处理垂直方向的速度，保证其在限定范围内

**float** vel\_norm\_z = sqrtf(\_vel\_sp(2) \* \_vel\_sp(2));

**if** (vel\_norm\_z > \_params.vel\_max(2)) {

\_vel\_sp(2) = \_vel\_sp(2) \* \_params.vel\_max(2) / vel\_norm\_z;

}

**if** (!\_control\_mode.flag\_control\_position\_enabled) {//位置控制非使能的需要重置位置参考点（通过其他控制器进入控制环节）

\_reset\_pos\_sp = **true**;

}

**if** (!\_control\_mode.flag\_control\_altitude\_enabled) {//高度控制非使能的需要重置高度参考点（通过其他控制器进入控制环节）

\_reset\_alt\_sp = **true**;

}

**if** (!\_control\_mode.flag\_control\_velocity\_enabled) {//速度控制非使能的需要重置速度参考点（通过其他控制器进入控制环节）

\_vel\_sp\_prev(0) = \_vel(0);

\_vel\_sp\_prev(1) = \_vel(1);

\_vel\_sp(0) = 0.0f;

\_vel\_sp(1) = 0.0f;

control\_vel\_enabled\_prev = **false**;

}

**if** (!\_control\_mode.flag\_control\_climb\_rate\_enabled) {//爬升率控制非使能的需要重置爬升率参考点（通过其他控制器进入控制环节）

\_vel\_sp(2) = 0.0f;

}

/\* use constant descend rate when landing, ignore altitude setpoint \*///着陆情况：忽略高度参考点，使用常量作为降落速率

**if** (!\_control\_mode.flag\_control\_manual\_enabled && \_pos\_sp\_triplet.current.valid&& \_pos\_sp\_triplet.current.type == position\_setpoint\_s::*SETPOINT\_TYPE\_LAND*) {//非手动模式&本次参考数据有效&着陆

\_vel\_sp(2) = \_params.land\_speed;//参考速度为0.5

}

/\* special thrust setpoint generation for takeoff from ground \*///从地面起飞，设置专用推力参考点

**if** (!\_control\_mode.flag\_control\_manual\_enabled && \_pos\_sp\_triplet.current.valid&& \_pos\_sp\_triplet.current.type == position\_setpoint\_s::*SETPOINT\_TYPE\_TAKEOFF*&& \_control\_mode.flag\_armed) {//非手动模式&本次参考数据有效&起飞&解锁

// check if we are not already in air.

// if yes then we don't need a jumped takeoff anymore//

**if** (!\_takeoff\_jumped && !\_vehicle\_status.condition\_landed && fabsf(\_takeoff\_thrust\_sp) < FLT\_EPSILON) {//非着陆状态&非起跳&起飞推力参考为0

\_takeoff\_jumped = **true**;//默认是false

}

**if** (!\_takeoff\_jumped) {// ramp thrust setpoint up

**if** (\_vel(2) > -(\_params.tko\_speed / 2.0f)) {//速度小于起飞速度0.75

\_takeoff\_thrust\_sp += 0.5f \* dt;//起飞参考推力缓慢增加

\_vel\_sp.zero();//速度参考重置（本次和上次）

\_vel\_prev.zero();

} **else** {//已经达到起飞速度，推力参考分为积分和比例两项

// copter has reached our takeoff speed. split the thrust setpoint up

// into an integral part and into a P part

thrust\_int(2) = \_takeoff\_thrust\_sp - \_params.vel\_p(2) \* fabsf(\_vel(2));

thrust\_int(2) = -math::constrain(thrust\_int(2), \_params.thr\_min, \_params.thr\_max);//限制在0.12-0.9（默认推力积分）

\_vel\_sp\_prev(2) = -\_params.tko\_speed;

\_takeoff\_jumped = **true**;

reset\_int\_z = **false**;

}

}

**if** (\_takeoff\_jumped) {

\_vel\_sp(2) = -\_params.tko\_speed;//起飞爬升速率作为参考，默认1.5

}

} **else** {//非手动模式&本次参考数据有效&起飞&解锁，不全部满足，重置参数

\_takeoff\_jumped = **false**;

\_takeoff\_thrust\_sp = 0.0f;

}

// limit total horizontal acceleration//限制水平加速度

math::Vector<2> acc\_hor;

acc\_hor(0) = (\_vel\_sp(0) - \_vel\_sp\_prev(0)) / dt;//水平速度微分，即加速度

acc\_hor(1) = (\_vel\_sp(1) - \_vel\_sp\_prev(1)) / dt;

**if** (acc\_hor.length() > \_params.acc\_hor\_max) {//acc\_hor\_max默认为10

acc\_hor.normalize();//限制加速度

acc\_hor \*= \_params.acc\_hor\_max;

math::Vector<2> vel\_sp\_hor\_prev(\_vel\_sp\_prev(0), \_vel\_sp\_prev(1));

math::Vector<2> vel\_sp\_hor = acc\_hor \* dt + vel\_sp\_hor\_prev;//重新获取参考速度

\_vel\_sp(0) = vel\_sp\_hor(0);//将新的参考速度作为参考值

\_vel\_sp(1) = vel\_sp\_hor(1);

}

// limit vertical acceleration//限制垂直方向加速度，同水平速度

**float** acc\_v = (\_vel\_sp(2) - \_vel\_sp\_prev(2)) / dt;//获取加速度

**if** (fabsf(acc\_v) > 2 \* \_params.acc\_hor\_max) {//acc\_hor\_max默认为10

acc\_v /= fabsf(acc\_v);

\_vel\_sp(2) = acc\_v \* 2 \* \_params.acc\_hor\_max \* dt + \_vel\_sp\_prev(2);//获取新的垂直速度参考

}

\_vel\_sp\_prev = \_vel\_sp;//更新前一次速度参考（使用本次的结果）

\_global\_vel\_sp.vx = \_vel\_sp(0);//更新消息中的参考速度值（之后发布）

\_global\_vel\_sp.vy = \_vel\_sp(1);

\_global\_vel\_sp.vz = \_vel\_sp(2);

/\* publish velocity setpoint \*///发布上面的消息，对于首次先通告

**if** (\_global\_vel\_sp\_pub != **nullptr**) {

orb\_publish(ORB\_ID(vehicle\_global\_velocity\_setpoint), \_global\_vel\_sp\_pub, &\_global\_vel\_sp);

} **else** {

\_global\_vel\_sp\_pub = orb\_advertise(ORB\_ID(vehicle\_global\_velocity\_setpoint), &\_global\_vel\_sp);

}

**if** (\_control\_mode.flag\_control\_climb\_rate\_enabled || \_control\_mode.flag\_control\_velocity\_enabled) {//爬升率控制|速度控制

/\* reset integrals if needed \*/

**if** (\_control\_mode.flag\_control\_climb\_rate\_enabled) {//爬升率控制

**if** (reset\_int\_z) {//重置z轴积分

reset\_int\_z = **false**;

**float** i = \_params.thr\_min;//thr\_min默认为0.12

**if** (reset\_int\_z\_manual) {//手动重置z积分

i = \_params.thr\_hover;//thr\_hover默认为0.5

**if** (i < \_params.thr\_min) {//悬停推力范围限制在最小和最大推力之间0.12——0.9

i = \_params.thr\_min;

} **else** **if** (i > \_params.thr\_max) {

i = \_params.thr\_max;

}

}

thrust\_int(2) = -i;//设置z推力积分为-i（朝地面为正）

}

} **else** {//速度控制，下次进入爬升率控制需要重置z轴积分

reset\_int\_z = **true**;

}

**if** (\_control\_mode.flag\_control\_velocity\_enabled) {//速度控制

**if** (reset\_int\_xy) {//这里对平面速度积分进行重置

reset\_int\_xy = **false**;

thrust\_int(0) = 0.0f;//积分量归零

thrust\_int(1) = 0.0f;

}

} **else** {//爬升率控制情况，下次进入速度控制需要重置平面速度积分量

reset\_int\_xy = **true**;

}

/\* velocity error \*/

math::Vector<3> vel\_err = \_vel\_sp - \_vel;//速度误差，参考点-实际

// check if we have switched from a non-velocity controlled mode into a velocity controlled mode

// if yes, then correct xy velocity setpoint such that the attitude setpoint is continuous

**if** (!control\_vel\_enabled\_prev && \_control\_mode.flag\_control\_velocity\_enabled) {//前一次为非速度控制，本次为速度控制，校正平面速度参考点来使姿态参考点连续

// choose velocity xyz setpoint such that the resulting thrust setpoint has the direction

// given by the last attitude setpoint//PID结果==推力，这里用来求参考点（根据上一次的姿态参考点来求速度参考点）

\_vel\_sp(0) = \_vel(0) + (-PX4\_R(\_att\_sp.R\_body, 0, 2) \* \_att\_sp.thrust - thrust\_int(0) - \_vel\_err\_d(0) \* \_params.vel\_d(0)) / \_params.vel\_p(0);

\_vel\_sp(1) = \_vel(1) + (-PX4\_R(\_att\_sp.R\_body, 1, 2) \* \_att\_sp.thrust - thrust\_int(1) - \_vel\_err\_d(1) \* \_params.vel\_d(1)) / \_params.vel\_p(1);

\_vel\_sp(2) = \_vel(2) + (-PX4\_R(\_att\_sp.R\_body, 2, 2) \* \_att\_sp.thrust - thrust\_int(2) - \_vel\_err\_d(2) \* \_params.vel\_d(2)) / \_params.vel\_p(2);

\_vel\_sp\_prev(0) = \_vel\_sp(0);//更新前一次速度参考点

\_vel\_sp\_prev(1) = \_vel\_sp(1);

\_vel\_sp\_prev(2) = \_vel\_sp(2);

control\_vel\_enabled\_prev = **true**;//更新前一次速度控制标志位

// compute updated velocity error

vel\_err = \_vel\_sp - \_vel;//用新的速度参考点获取新的速度误差

}

/\* thrust vector in NED frame \*///获取新的推力参考（NED坐标系）

math::Vector<3> thrust\_sp = vel\_err.emult(\_params.vel\_p) + \_vel\_err\_d.emult(\_params.vel\_d) + thrust\_int;

**if** (\_pos\_sp\_triplet.current.type == position\_setpoint\_s::*SETPOINT\_TYPE\_TAKEOFF*

&& !\_takeoff\_jumped && !\_control\_mode.flag\_control\_manual\_enabled) {//起飞&非起跳&非手动模式

// for jumped takeoffs use special thrust setpoint calculated above

thrust\_sp.zero();

thrust\_sp(2) = -\_takeoff\_thrust\_sp;//这种起飞情况只需要垂直的推力参考就行

}

**if** (!\_control\_mode.flag\_control\_velocity\_enabled) {//非速度控制

thrust\_sp(0) = 0.0f;//平面推力参考置0

thrust\_sp(1) = 0.0f;

}

**if** (!\_control\_mode.flag\_control\_climb\_rate\_enabled) {//非爬升率控制

thrust\_sp(2) = 0.0f;//垂直推力参考置0

}

/\* limit thrust vector and check for saturation \*///保证推力在限定范围内

**bool** saturation\_xy = **false**;

**bool** saturation\_z = **false**;

/\* limit min lift \*///推力最小默认为0.12

**float** thr\_min = \_params.thr\_min;

**if** (!\_control\_mode.flag\_control\_velocity\_enabled && thr\_min < 0.0f) {//非速度控制&最小推力<0

/\* don't allow downside thrust direction in manual attitude mode \*///禁止下降推力，最小推力置0

thr\_min = 0.0f;

}

**float** thrust\_abs = thrust\_sp.length();

**float** tilt\_max = \_params.tilt\_max\_air;//默认45

**float** thr\_max = \_params.thr\_max;//默认0.9

/\* filter vel\_z over 1/8sec \*/

\_vel\_z\_lp = \_vel\_z\_lp \* (1.0f - dt \* 8.0f) + dt \* 8.0f \* \_vel(2);//通过低通滤波器，z轴速度

/\* filter vel\_z change over 1/8sec \*/

**float** vel\_z\_change = (\_vel(2) - \_vel\_prev(2)) / dt;//z轴加速度

\_acc\_z\_lp = \_acc\_z\_lp \* (1.0f - dt \* 8.0f) + dt \* 8.0f \* vel\_z\_change;//通过低通滤波器，z轴加速度

/\* adjust limits for landing mode \*///为着陆模式调整限制

**if** (!\_control\_mode.flag\_control\_manual\_enabled && \_pos\_sp\_triplet.current.valid &&

\_pos\_sp\_triplet.current.type == position\_setpoint\_s::*SETPOINT\_TYPE\_LAND*) {//非手动模式&本次参考数据有效&着陆

/\* limit max tilt and min lift when landing \*/

tilt\_max = \_params.tilt\_max\_land;//默认12

**if** (thr\_min < 0.0f) {//限制最小推力

thr\_min = 0.0f;

}

/\* descend stabilized, we're landing \*///判断是否要下降

**if** (!\_in\_landing && !\_lnd\_reached\_ground

&& (**float**)fabs(\_acc\_z\_lp) < 0.1f

&& \_vel\_z\_lp > 0.5f \* \_params.land\_speed) {

\_in\_landing = **true**;

}

/\* assume ground, cut thrust \*///下降中判断是否着陆，着陆的情况最大推力置0

**if** (\_in\_landing&& \_vel\_z\_lp < 0.1f) {

thr\_max = 0.0f;

\_in\_landing = **false**;

\_lnd\_reached\_ground = **true**;

}

/\* once we assumed to have reached the ground always cut the thrust.

Only free fall detection below can revoke this\*/

**if** (!\_in\_landing && \_lnd\_reached\_ground) {//保证着陆情况最大推力为0

thr\_max = 0.0f;

}

/\* if we suddenly fall, reset landing logic and remove thrust limit \*///着陆的情况是假定的，还有自由落体包含在内，要避免自由落体，所以有以下设定

**if** (\_lnd\_reached\_ground//假定自由落体加速度大于4 && (\_acc\_z\_lp > 4.0f//加速度>4|速度>2着陆速度（默认0.5） || \_vel\_z\_lp > 2.0f \* \_params.land\_speed)) {//恢复最大推力，取消着陆和正在着陆

thr\_max = \_params.thr\_max;

\_in\_landing = **false**;

\_lnd\_reached\_ground = **false**;

}

} **else** {//非手动模式&本次参考数据有效&着陆，不满足的情况取消着陆和正在着陆

\_in\_landing = **false**;

\_lnd\_reached\_ground = **false**;

}

/\* limit min lift \*///这里限制升力，升力需要大于最小推力（绝对值，方向）

**if** (-thrust\_sp(2) < thr\_min) {

thrust\_sp(2) = -thr\_min;

saturation\_z = **true**;//判断超出范围，饱和

}

**if** (\_control\_mode.flag\_control\_velocity\_enabled) {//速度控制

/\* limit max tilt \*///限制最大倾斜角

**if** (thr\_min >= 0.0f && tilt\_max < M\_PI\_F / 2 - 0.05f) {//最小推力>0&最大倾斜角<90(不到90)

/\* absolute horizontal thrust \*/

**float** thrust\_sp\_xy\_len = math::Vector<2>(thrust\_sp(0), thrust\_sp(1)).length();

**if** (thrust\_sp\_xy\_len > 0.01f) {//参考水平推力平方和>0.01

/\* max horizontal thrust for given vertical thrust\*///根据垂直推力来决定水平推力最大值

**float** thrust\_xy\_max = -thrust\_sp(2) \* tanf(tilt\_max);

**if** (thrust\_sp\_xy\_len > thrust\_xy\_max) {//当参考水平推力>以上水平推力，限制参考水平推力

**float** k = thrust\_xy\_max / thrust\_sp\_xy\_len;

thrust\_sp(0) \*= k;

thrust\_sp(1) \*= k;

saturation\_xy = **true**;//同垂直方向，饱和

}

}

}

}

**if** (\_control\_mode.flag\_control\_altitude\_enabled) {//姿态控制

/\* thrust compensation for altitude only control modes \*///只有姿态控制情况补偿推力

**float** att\_comp;

**if** (\_R(2, 2) > TILT\_COS\_MAX) {

att\_comp = 1.0f / \_R(2, 2);

} **else** **if** (\_R(2, 2) > 0.0f) {

att\_comp = ((1.0f / TILT\_COS\_MAX - 1.0f) / TILT\_COS\_MAX) \* \_R(2, 2) + 1.0f;

saturation\_z = **true**;

} **else** {//R（2，2）<0情况

att\_comp = 1.0f;

saturation\_z = **true**;

}

thrust\_sp(2) \*= att\_comp;//补偿（不明白）

}

/\* limit max thrust \*///限制最大推力

thrust\_abs = thrust\_sp.length(); /\* recalculate because it might have changed \*///参考推力可能改变故重新计算

**if** (thrust\_abs > thr\_max) {//0.9，只有过大才限制推力

**if** (thrust\_sp(2) < 0.0f) {//上升

**if** (-thrust\_sp(2) > thr\_max) {//升力>最大推力（0.9）

/\* thrust Z component is too large, limit it \*///限制水平推力，垂直推力为最大推力值，饱和标志位置true

thrust\_sp(0) = 0.0f;

thrust\_sp(1) = 0.0f;

thrust\_sp(2) = -thr\_max;

saturation\_xy = **true**;

saturation\_z = **true**;

} **else** {//升力未饱和

/\* preserve thrust Z component and lower XY, keeping altitude is more important than position \*/

**float** thrust\_xy\_max = sqrtf(thr\_max \* thr\_max - thrust\_sp(2) \* thrust\_sp(2));

**float** thrust\_xy\_abs = math::Vector<2>(thrust\_sp(0), thrust\_sp(1)).length();//abs>max，限制水平推力，高度要比位置更重要

**float** k = thrust\_xy\_max / thrust\_xy\_abs;

thrust\_sp(0) \*= k;

thrust\_sp(1) \*= k;

saturation\_xy = **true**;

}

} **else** {//下降

/\* Z component is negative, going down, simply limit thrust vector \*/

**float** k = thr\_max / thrust\_abs;//下降的情况直接限制大小（上面对水平限制）

thrust\_sp \*= k;

saturation\_xy = **true**;

saturation\_z = **true**;

}

thrust\_abs = thr\_max;

}

/\* update integrals \*///更新积分

**if** (\_control\_mode.flag\_control\_velocity\_enabled && !saturation\_xy) {//速度控制&xy积分不饱和，饱和的情况说明值比较大，不需要再累加

thrust\_int(0) += vel\_err(0) \* \_params.vel\_i(0) \* dt;

thrust\_int(1) += vel\_err(1) \* \_params.vel\_i(1) \* dt;

}

**if** (\_control\_mode.flag\_control\_climb\_rate\_enabled && !saturation\_z) {//爬升率控制&z积分不饱和，饱和的情况说明值比较大，不需要再累加

thrust\_int(2) += vel\_err(2) \* \_params.vel\_i(2) \* dt;

/\* protection against flipping on ground when landing \*/

**if** (thrust\_int(2) > 0.0f) {//垂直推力保证<=0，遇到着陆情况可以避免弹跳

thrust\_int(2) = 0.0f;

}

}

/\* calculate attitude setpoint from thrust vector \*///从推力向量求姿态参考点

**if** (\_control\_mode.flag\_control\_velocity\_enabled) {//速度控制

/\* desired body\_z axis = -normalize(thrust\_vector) \*/

math::Vector<3> body\_x;

math::Vector<3> body\_y;

math::Vector<3> body\_z;

**if** (thrust\_abs > SIGMA) {

body\_z = -thrust\_sp / thrust\_abs;//归一化

} **else** {

/\* no thrust, set Z axis to safe value \*///没有推力的情况，[0 0 1]

body\_z.zero();

body\_z(2) = 1.0f;

}

/\* vector of desired yaw direction in XY plane, rotated by PI/2 \*/

math::Vector<3> y\_C(-sinf(\_att\_sp.yaw\_body), cosf(\_att\_sp.yaw\_body), 0.0f);

**if** (fabsf(body\_z(2)) > SIGMA) {

/\* desired body\_x axis, orthogonal to body\_z \*/

body\_x = y\_C % body\_z;//叉乘，由z和期望y获得

/\* keep nose to front while inverted upside down \*///方向校正

**if** (body\_z(2) < 0.0f) {

body\_x = -body\_x;

}

body\_x.normalize();//归一化

} **else** {

/\* desired thrust is in XY plane, set X downside to construct correct matrix,

\* but yaw component will not be used actually \*///z轴为0，不能叉乘，直接用单位

body\_x.zero();

body\_x(2) = 1.0f;

}

/\* desired body\_y axis \*/

body\_y = body\_z % body\_x;//叉乘，由z和x求y

/\* fill rotation matrix \*/

**for** (**int** i = 0; i < 3; i++) {//获得旋转矩阵

R(i, 0) = body\_x(i);

R(i, 1) = body\_y(i);

R(i, 2) = body\_z(i);

}

/\* copy rotation matrix to attitude setpoint topic \*///更新姿态参考的数据（旋转矩阵及其有效性）

memcpy(&\_att\_sp.R\_body[0], R.data, **sizeof**(\_att\_sp.R\_body));

\_att\_sp.R\_valid = **true**;

/\* copy quaternion setpoint to attitude setpoint topic \*///更新四元数参考，并更新到姿态参考中

math::Quaternion q\_sp;

q\_sp.from\_dcm(R);

memcpy(&\_att\_sp.q\_d[0], &q\_sp.data[0], **sizeof**(\_att\_sp.q\_d));

/\* calculate euler angles, for logging only, must not be used for control \*///根据旋转矩阵求欧拉角，

math::Vector<3> euler = R.to\_euler();

\_att\_sp.roll\_body = euler(0);

\_att\_sp.pitch\_body = euler(1);

/\* yaw already used to construct rot matrix, but actual rotation matrix can have different yaw near singularity \*

} **else** **if** (!\_control\_mode.flag\_control\_manual\_enabled) {//非速度控制&非手动控制

/\* autonomous altitude control without position control (failsafe landing), \* force level attitude, don't change yaw \*///没有位置控制的自动高度控制（失控保护 着陆），强制姿态水平，不改变偏航

R.from\_euler(0.0f, 0.0f, \_att\_sp.yaw\_body);//根据偏航获得旋转矩阵

/\* copy rotation matrix to attitude setpoint topic \*///更新姿态参考中的旋转矩阵及其有效性

memcpy(&\_att\_sp.R\_body[0], R.data, **sizeof**(\_att\_sp.R\_body));

\_att\_sp.R\_valid = **true**;

/\* copy quaternion setpoint to attitude setpoint topic \*///由旋转矩阵获取四元数参考，并更新到姿态参考中

math::Quaternion q\_sp;

q\_sp.from\_dcm(R);

memcpy(&\_att\_sp.q\_d[0], &q\_sp.data[0], **sizeof**(\_att\_sp.q\_d));

\_att\_sp.roll\_body = 0.0f;//保持水平

\_att\_sp.pitch\_body = 0.0f;

}

\_att\_sp.thrust = thrust\_abs;//更新姿态期望中的推力（推力平方根）

/\* save thrust setpoint for logging \*///保存数据用于记录，推力->加速度

\_local\_pos\_sp.acc\_x = thrust\_sp(0) \* ONE\_G;

\_local\_pos\_sp.acc\_y = thrust\_sp(1) \* ONE\_G;

\_local\_pos\_sp.acc\_z = thrust\_sp(2) \* ONE\_G;

\_att\_sp.timestamp = hrt\_absolute\_time();

} **else** {//爬升率控制|速度控制，非速度控制的情况，重置z积分

reset\_int\_z = **true**;

}

}

最后是位置参考vehicle\_local\_position\_setpoint更新，对位置和速度参考点的数据进行更新并发布：

/\* fill local position, velocity and thrust setpoint \*///将得到的数据更新到本地位置参考消息中（之后会发布）

\_local\_pos\_sp.timestamp = hrt\_absolute\_time();

\_local\_pos\_sp.x = \_pos\_sp(0);

\_local\_pos\_sp.y = \_pos\_sp(1);

\_local\_pos\_sp.z = \_pos\_sp(2);

\_local\_pos\_sp.yaw = \_att\_sp.yaw\_body;

\_local\_pos\_sp.vx = \_vel\_sp(0);

\_local\_pos\_sp.vy = \_vel\_sp(1);

\_local\_pos\_sp.vz = \_vel\_sp(2);

/\* publish local position setpoint \*///这里发布位置参考点数据，首次发布需要公告

**if** (\_local\_pos\_sp\_pub != **nullptr**) {

orb\_publish(ORB\_ID(vehicle\_local\_position\_setpoint), \_local\_pos\_sp\_pub, &\_local\_pos\_sp);

} **else** {

\_local\_pos\_sp\_pub = orb\_advertise(ORB\_ID(vehicle\_local\_position\_setpoint), &\_local\_pos\_sp);

}

B．不满足该循环要求的需要重置位置参考，放弃自动控制，重置积分量，并更新速度参考：

} **else** {//高度控制，位置控制，爬升率（垂直速度）控制，速度（水平速度）控制一个都不满足，即不用控制器

/\* position controller disabled, reset setpoints \*///不能使用位置控制器的情况，需要重置参考点

\_reset\_alt\_sp = **true**;//位置和高度参考点重置，自动模式去能，xyz积分重置

\_reset\_pos\_sp = **true**;

\_mode\_auto = **false**;

reset\_int\_z = **true**;

reset\_int\_xy = **true**;

control\_vel\_enabled\_prev = **false**;

/\* store last velocity in case a mode switch to position control occurs \*///保存速度数据用于下次位置控制

\_vel\_sp\_prev = \_vel;

}

**手动和姿态控制主函数**

**A．手动控制和姿态控制与上面的控制是并列的，可以同时实现：**

//并列于“高度控制、位置控制、爬升速率控制、速度控制”，混控

/\* generate attitude setpoint from manual controls \*///由手动控制获得姿态参考点

**if** (\_control\_mode.flag\_control\_manual\_enabled && \_control\_mode.flag\_control\_attitude\_enabled) {//手动控制&姿态控制

首先要判断偏航参考点是否需要重置，需要则将当前偏航作为参考点：

/\* reset yaw setpoint to current position if needed \*///偏航参考重置有效，更新姿态参考中的偏航

**if** (reset\_yaw\_sp) {

reset\_yaw\_sp = **false**;

\_att\_sp.yaw\_body = \_yaw;

}

其他情况中着陆状态不改变偏航：

/\* do not move yaw while sitting on the ground \*///在地面时不改变偏航

**else** **if** (!\_vehicle\_status.condition\_landed &&

!(!\_control\_mode.flag\_control\_altitude\_enabled && \_manual.z < 0.1f)) {//不需要重置偏航的情况，满足非着陆&（姿态控制|油门摇杆>0.1）

/\* we want to know the real constraint, and global overrides manual \*/

**const** **float** yaw\_rate\_max = (\_params.man\_yaw\_max < \_params.global\_yaw\_max) ? \_params.man\_yaw\_max : \_params.global\_yaw\_max;//选取两个中小的一个

**const** **float** yaw\_offset\_max = yaw\_rate\_max / \_params.mc\_att\_yaw\_p;//（比例\*e==输出）

//\_manual.r偏航摇杆的位置\*偏航角速度最大值

\_att\_sp.yaw\_sp\_move\_rate = \_manual.r \* yaw\_rate\_max;

//期望偏航值（期望参考+角速度\*dt，最后限制在+-PI）

**float** yaw\_target = \_wrap\_pi(\_att\_sp.yaw\_body + \_att\_sp.yaw\_sp\_move\_rate \* dt);

**float** yaw\_offs = \_wrap\_pi(yaw\_target - \_yaw);//偏航的误差（期望-当前）

// If the yaw offset became too big for the system to track stop

// shifting it

// **XXX** this needs inspection - probably requires a clamp, not an if

**if** (fabsf(yaw\_offs) < yaw\_offset\_max) {

//偏航误差<求得的最大偏航误差，就进行期望偏航角的更新

\_att\_sp.yaw\_body = yaw\_target;//更新姿态期望中的偏航角

}

}

不适用速度控制的情况（水平速度和垂直速度），手动模式输出直接作为姿态参考点：

/\* control roll and pitch directly if we no aiding velocity controller is active \*/

**if** (!\_control\_mode.flag\_control\_velocity\_enabled) {//非速度控制，直接控制roll和pitch

\_att\_sp.roll\_body = \_manual.y \* \_params.man\_roll\_max;//姿态期望的roll和pitch更新（摇杆位置\*手动模式角度最大值）

att\_sp.pitch\_body = -\_manual.x \* \_params.man\_pitch\_max;

}

/\* control throttle directly if no climb rate controller is active \*/

**if** (!\_control\_mode.flag\_control\_climb\_rate\_enabled) {//非爬升率控制，直接控制油门，同上

**float** thr\_val = *throttle\_curve*(\_manual.z, \_params.thr\_hover);//配置油门曲线（0-0.5，0.5-1）\*

\_att\_sp.thrust = math::min(thr\_val, \_manual\_thr\_max.get());

/\* enforce minimum throttle if not landed \*///着陆前姿态期望中的油门保证较大

**if** (!\_vehicle\_status.condition\_landed) {//非着陆状态

\_att\_sp.thrust = math::max(\_att\_sp.thrust, \_manual\_thr\_min.get());

}

}

接下来根据这列得到的姿态参考数据获取姿态参考旋转矩阵：

math::Matrix<3, 3> R\_sp;

/\* construct attitude setpoint rotation matrix \*///根据获得的姿态参考中的欧拉角获得旋转矩阵

R\_sp.from\_euler(\_att\_sp.roll\_body, \_att\_sp.pitch\_body, \_att\_sp.yaw\_body);

memcpy(&\_att\_sp.R\_body[0], R\_sp.data, **sizeof**(\_att\_sp.R\_body));//更新旋转矩阵数值到姿态参考中

接着完成推力数据更新：

/\* reset the acceleration set point for all non-attitude flight modes \*///为非姿态飞行模式重置加速度参考点

**if** (!(\_control\_mode.flag\_control\_offboard\_enabled &&

!(\_control\_mode.flag\_control\_position\_enabled ||

\_control\_mode.flag\_control\_velocity\_enabled))) {//非离线模式|位置控制|速度控制

\_thrust\_sp\_prev = R\_sp \* math::Vector<3>(0, 0, -\_att\_sp.thrust);//更新前一次推力参考点

}

最后从旋转矩阵获得参考四元数，并更新执行时间：

/\* copy quaternion setpoint to attitude setpoint topic \*///从旋转矩阵获得四元数参考，然后更新到姿态参考中，

math::Quaternion q\_sp;

q\_sp.from\_dcm(R\_sp);

memcpy(&\_att\_sp.q\_d[0], &q\_sp.data[0], **sizeof**(\_att\_sp.q\_d));

\_att\_sp.timestamp = hrt\_absolute\_time();

B．不满足手动控制这个循环要求的直接重置偏航参考标志位：

} **else** {//手动控制&姿态控制，不满足这种情况，就重置偏航参考

reset\_yaw\_sp = **true**;

}

**主题发布**

完成以上的控制操作之后需要对速度更新：

///前一次速度更新，用于速度控制器的D（微分）

\_vel\_prev = \_vel;

数据发布需要在特定模式下，在非离线模式或位置控制模式或者速度控制模式对姿态参考数据进行发布：

**if** (!(\_control\_mode.flag\_control\_offboard\_enabled &&

!(\_control\_mode.flag\_control\_position\_enabled ||

\_control\_mode.flag\_control\_velocity\_enabled))) {//非离线模式|位置控制|速度控制

**if** (\_att\_sp\_pub != **nullptr**) {//发布姿态参考数据，首次发布使用通告

orb\_publish(\_attitude\_setpoint\_id, \_att\_sp\_pub, &\_att\_sp);

} **else** **if** (\_attitude\_setpoint\_id) {

\_att\_sp\_pub = orb\_advertise(\_attitude\_setpoint\_id, &\_att\_sp);

} }

手动控制之后需要重置高度控制积分量，保证高度上面的精度:

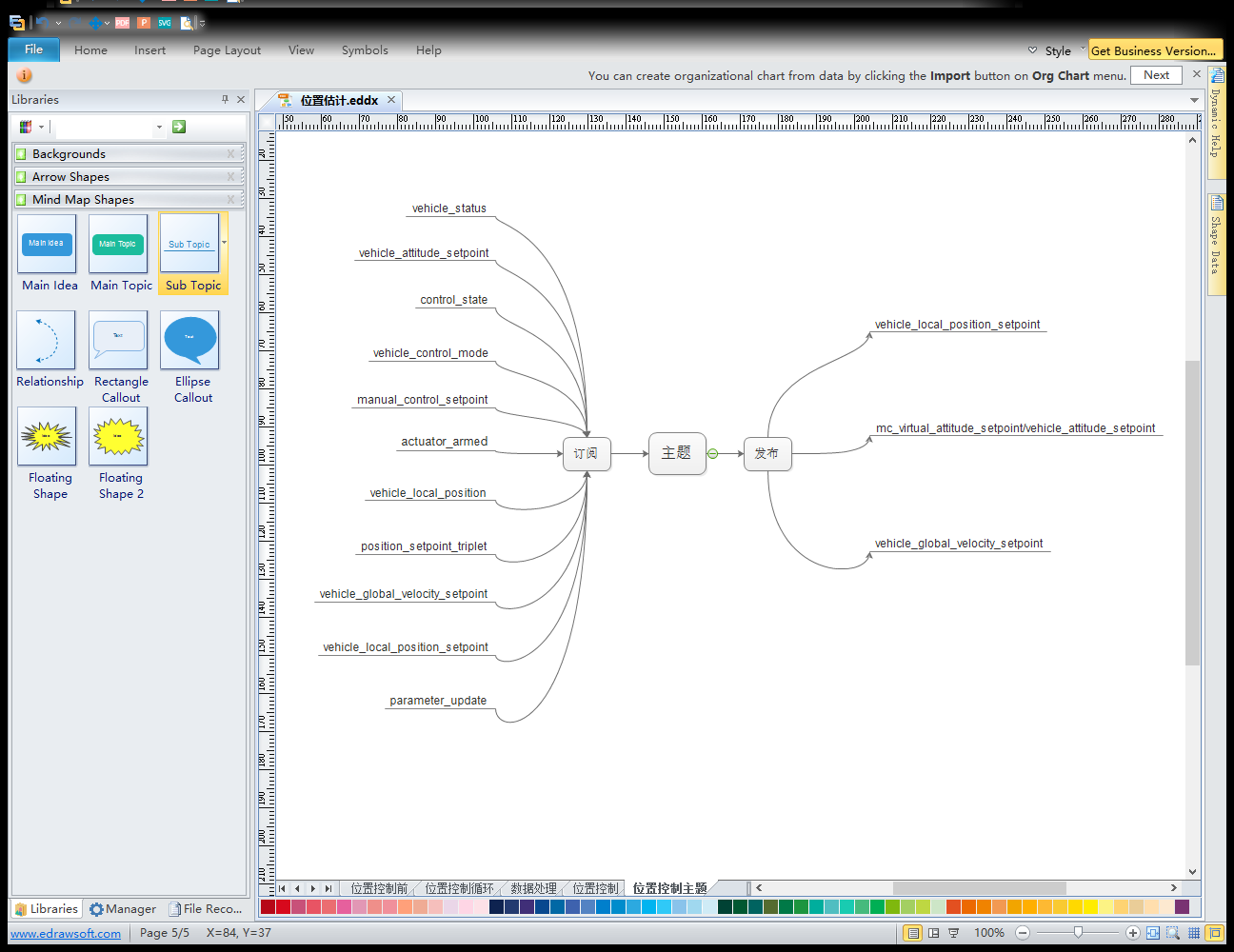
/\* reset altitude controller integral (hovering throttle) to manual throttle after manual throttle control \*///手动控制油门后要重置高度控制器积分（悬停油门）

reset\_int\_z\_manual = \_control\_mode.flag\_armed && \_control\_mode.flag\_control\_manual\_enabled && !\_control\_mode.flag\_control\_climb\_rate\_enabled;//解锁&手动控制&非爬升率控制

循环结束，终止mavlink，结束任务：

mavlink\_log\_info(\_mavlink\_fd, "[mpc] stopped");

\_control\_task = -1;

关于主题订阅和发布的情况，可以用下图简单描述：