[mc\_pos\_control.cpp](http://blog.csdn.net/czyv587/article/details/51878546)

1. 位置控制的大体过程是什么？

（1）copy commander和navigator产生的期望位置-----\_pos\_sp\_triplet结构体

（2）产生位置/速度设定值(期望值)-----\_pos\_sp<3>向量和\_vel\_sp<3>向量

（3）产生可利用的速度设定值(期望值)-----\_vel\_sp<3>向量

（4）产生可利用的推力定值(期望值)-----thrust\_sp<3>向量

（5）根据推力向量计算姿态设定值(期望姿态)-----q\_sp四元数矩阵和R\_sp旋转矩阵

（6）将之前程序得到的各种信息填充\_local\_pos\_sp结构体，并发布出去-----\_local\_pos\_sp（第2、3步得到的）

（7）根据具体应用更改之前得到的姿态设定值(期望姿态)，并发布出去-----\_att\_sp（第5步得到的）

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

int mc\_pos\_control\_main(int argc, char \*argv[]){

……

pos\_control::g\_control = new MulticopterPositionControl;//构造函数

……

if (OK != pos\_control::g\_control->start())

{……}

……

}

进入start()

MulticopterPositionControl::start(){

ASSERT(\_control\_task == -1);

/\* start the task \*/

\_control\_task = px4\_task\_spawn\_cmd("mc\_pos\_control",

SCHED\_DEFAULT,

SCHED\_PRIORITY\_MAX - 5,

1900,

(px4\_main\_t)&MulticopterPositionControl::task\_main\_trampoline,//创建线程

nullptr);

if (\_control\_task < 0) {

warn("task start failed");

return -errno;

}

return OK;

}

void MulticopterPositionControl::task\_main\_trampoline(int argc, char \*argv[]){

pos\_control::g\_control->task\_main();

}

接下来进入task\_main()，task\_main()特别长/情况也特别复杂，所以需要分清楚层次/程序运行的条件

void MulticopterPositionControl::task\_main(){

init部分(只运行一次)

while (!\_task\_should\_exit) {

……;//获取传感器数据/ \_vel(i)赋值/标志位赋值等

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

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

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

\_control\_mode.flag\_control\_velocity\_enabled) {//基本上运行到这都会满足这个条件

**/\*\*\*\*\*\*\*\*\*\*高度控制、位置控制、爬升速率控制、速度控制的相关程序开始\*\*\*\*\*\*\*\*\*\*\*/**

control\_manual(dt);/ control\_offboard(dt);/ control\_auto(dt);

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

&& \_pos\_sp\_triplet.current.type == position\_setpoint\_s::SETPOINT\_TYPE\_IDLE) {……}

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

&& \_vehicle\_status.condition\_landed) {……}

else {

……//飞行器起飞/降落等情况的处理

……//飞行器优化处理为了得到更好的\_vel\_sp，比如利用限制水平方向加速度等

//这部分为了得到\_vel\_sp(i)

if (\_control\_mode.flag\_control\_climb\_rate\_enabled ||\_control\_mode.flag\_control\_velocity\_enabled) {

if (\_control\_mode.flag\_control\_climb\_rate\_enabled) {……}

if (\_control\_mode.flag\_control\_velocity\_enabled) {……}

if (!control\_vel\_enabled\_prev && \_control\_mode.flag\_control\_velocity\_enabled) {……}

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

//推力设定值(三维)=速度差\*P+速度差的差\*D+积分

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

&& !\_takeoff\_jumped && !\_control\_mode.flag\_control\_manual\_enabled) {……}

if (!\_control\_mode.flag\_control\_velocity\_enabled) {……}

if (!\_control\_mode.flag\_control\_climb\_rate\_enabled) {……}

\_vel\_z\_lp = \_vel\_z\_lp \* (1.0f - dt \* 8.0f) + dt \* 8.0f \* \_vel(2);//垂直速度低通滤波

\_acc\_z\_lp = \_acc\_z\_lp \* (1.0f - dt \* 8.0f) + dt \* 8.0f \* vel\_z\_change;//垂直加速度低通滤波

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

\_pos\_sp\_triplet.current.type == position\_setpoint\_s::SETPOINT\_TYPE\_LAND) {……}

//着陆处理

if (\_control\_mode.flag\_control\_velocity\_enabled) {……}//限制最大斜率(xy方向推力限幅)

if (\_control\_mode.flag\_control\_altitude\_enabled) {……}//推力补偿，用于高度控制

if (thrust\_abs > thr\_max) {……}//推力限幅

//经过之前的处理，得到合适的thrust\_sp

if (\_control\_mode.flag\_control\_velocity\_enabled) {

//body\_x、body\_y、body\_z应该是方向余弦矩阵的三个列向量

body\_z = -thrust\_sp / thrust\_abs;//body\_z矩阵是推力设定值矩阵的标准化

y\_C(-sinf(\_att\_sp.yaw\_body), cosf(\_att\_sp.yaw\_body), 0.0f);

//y\_C相当于是矩阵(-sin(偏航角),cos(偏航角),0)

body\_x = y\_C % body\_z;//%是求叉积运算

body\_y = body\_z % body\_x;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*再求出R<3,3>矩阵\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* fill rotation matrix \*/

for (int i = 0; i < 3; i++) {

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

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

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

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*将R<3,3>矩阵copy到\_att\_sp.R\_body[]\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

/\*\*\*\*由方向余弦旋转矩阵R得到四元数，并copy到att\_sp.q\_d[]\*\*\*\*\*\*\*\*\*/

math::Quaternion q\_sp;

q\_sp.from\_dcm(R);

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

/\*\*\*\*\*\*由旋转矩阵R得到姿态设置欧拉角，只是log调试用，不是给控制用的\*\*/

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

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

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

//yaw已经用于构建原始矩阵

}

else if (!\_control\_mode.flag\_control\_manual\_enabled) {……}

//没有位置控制的高度控制(故障安全降落)，固定水平姿态，不改变yaw角

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*用于log，方便调试\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

\_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;

}

}

/\*\*\*\*\*\*\*\*\*将之前程序得到的各种信息填充\_local\_pos\_sp结构体，并发布出去\*\*\*\*\*\*\*\*\*\*/

/\* fill local position, velocity and thrust setpoint \*/

\_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);

//第二部分第一步：产生位置/速度设定值(期望值)

\_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);

//第二部分第二步的重点(1):产生可利用的速度设定值(期望值)

/\* publish local position setpoint \*/

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);

}

**/\*\*\*\*\*\*\*\*\*\*高度控制、位置控制、爬升速率控制、速度控制的相关程序结束\*\*\*\*\*\*\*\*\*\*\*\*/**

else {……}

if (\_control\_mode.flag\_control\_manual\_enabled && \_control\_mode.flag\_control\_attitude\_enabled) {……}

/\*\*\*\*\*\*\*\*\*\*此判断是并列于“高度控制、位置控制、爬升速率控制、速度控制”的判断\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*所以会出现混控现象，在执行任务的时候还可以遥控控制\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*手动控制和姿态控制都使能，则运行以下程序产生姿态设定值\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*刷新之前求得的\_att\_sp\*\*\*\*\*\*\*\*\*\*/

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

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

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

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);

}

}//发布姿态设定值，如果位置/速度失能而机外(offboard)使能，则不发布姿态设定值，因为这种情况//姿态设定值是通过mavlink应用发布的，飞机工作于垂直起降或者做一个过渡，也不发布，因为此

//时由垂直起降控制部分发布

reset\_int\_z\_manual = \_control\_mode.flag\_armed && \_control\_mode.flag\_control\_manual\_enabled

&& !\_control\_mode.flag\_control\_climb\_rate\_enabled;

//手动控制后复位高度控制的积分(悬停油门)，以便更好的转变为手动模式

}

}

1. 控制环是什么样子的？

当需要位置控制时，用P-PID控制环；当不需要位置控制时，不要外环，只用速度环(内环) PID。

程序中需要注意\_run\_pos\_control/\_run\_alt\_control标志位，而\_run\_pos\_control/\_run\_alt\_control标志位的改变又涉及\_pos\_hold\_engaged/\_alt\_hold\_engaged标志位

//飞行器的位置或者速度(local\_position\_estimator或者position\_estimator\_inav中的)

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

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

if (\_params.alt\_mode == 1 && \_local\_pos.dist\_bottom\_valid) {

\_pos(2) = -\_local\_pos.dist\_bottom;

} else {

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

}

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

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

if (\_params.alt\_mode == 1 && \_local\_pos.dist\_bottom\_valid) {

\_vel(2) = -\_local\_pos.dist\_bottom\_rate;

} else {

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

}

……

//速度的微分

\_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));

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

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

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

\_control\_mode.flag\_control\_velocity\_enabled ||

\_control\_mode.flag\_control\_acceleration\_enabled) {

……

\_run\_pos\_control = true;//用于判断是否需要外环，当为true则需要外环，当为false则不需要外环

\_run\_alt\_control = true;//在后面会根据此标志位判断

if (\_control\_mode.flag\_control\_manual\_enabled) {

/\* manual control \*/

control\_manual(dt);

\_mode\_auto = false;

} else if (\_control\_mode.flag\_control\_offboard\_enabled) {

/\* offboard control \*/

control\_offboard(dt);

\_mode\_auto = false;

} else {

/\* AUTO \*/

control\_auto(dt);

}

……

if (\_run\_pos\_control) {

\_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) {

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

}

……

void MulticopterPositionControl::control\_manual(float dt)

{

//req\_vel\_sp来自于遥控

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();

if (\_control\_mode.flag\_control\_altitude\_enabled) {

/\* set vertical velocity setpoint with throttle stick \*/

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

}

if (\_control\_mode.flag\_control\_position\_enabled) {

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

req\_vel\_sp(0) = \_manual.x;

req\_vel\_sp(1) = \_manual.y;

}

if (\_control\_mode.flag\_control\_altitude\_enabled) {

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

reset\_alt\_sp();

}

if (\_control\_mode.flag\_control\_position\_enabled) {

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

reset\_pos\_sp();

}

/\* limit velocity setpoint \*/

float req\_vel\_sp\_norm = req\_vel\_sp.length();

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);

math::Vector<3> req\_vel\_sp\_scaled = R\_yaw\_sp \* req\_vel\_sp.emult(

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

/\*

\* assisted velocity mode: user controls velocity, but if velocity is small enough, position

\* hold is activated for the corresponding axis

\*/

/\* horizontal axes \*/

if (\_control\_mode.flag\_control\_position\_enabled) {

/\* check for pos. hold \*/

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

//遥控没有拨动

if (!\_pos\_hold\_engaged) {

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

&& fabsf(\_vel(1)) < \_params.hold\_max\_xy)) {

//飞机速度小于一个阈值

\_pos\_hold\_engaged = true;//激活位置控制，需要外环

} else {

\_pos\_hold\_engaged = false;//不激活位置控制，不需要外环

}

}

} else {

\_pos\_hold\_engaged = false;//不激活位置控制，不需要外环

}

/\* set requested velocity setpoint \*/

if (!\_pos\_hold\_engaged) {//不激活位置控制，不需要外环

\_pos\_sp(0) = \_pos(0);//期望位置跟随实际位置

\_pos\_sp(1) = \_pos(1);//期望位置跟随实际位置

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

//判断是否需要外环

\_vel\_sp(0) = req\_vel\_sp\_scaled(0);//更新期望速度

\_vel\_sp(1) = req\_vel\_sp\_scaled(1);//更新期望速度

}

}

/\* vertical axis \*/

if (\_control\_mode.flag\_control\_altitude\_enabled) {

/\* check for pos. hold \*/

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

if (!\_alt\_hold\_engaged) {

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

\_alt\_hold\_engaged = true;//激活位置控制，需要外环

} else {

\_alt\_hold\_engaged = false;//不激活位置控制，不需要外环

}

}

} else {

\_alt\_hold\_engaged = false;//不激活位置控制，不需要外环

}

/\* set requested velocity setpoint \*/

if (!\_alt\_hold\_engaged) {

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

\_vel\_sp(2) = req\_vel\_sp\_scaled(2);//更新期望速度

\_pos\_sp(2) = \_pos(2);//期望高度跟随实际高度

}

}

}

由于\_run\_pos\_control = true; \_run\_alt\_control = true一直都在程序前面并且循环，所以默认是需要外环的，外环计算出来的期望速度会替代遥控的产生的期望速度；只有当拨动了遥控或者速度比较大时\_run\_pos\_control = false;\_run\_alt\_control = false才只有速度控制环

control\_offboard(dt)和control\_auto(dt)是一样的分析

control\_offboard(dt);里

\_run\_pos\_control/\_run\_alt\_control可为false或者不改

control\_auto(dt);里

\_run\_pos\_control/\_run\_alt\_control没有改动

同时可以解释一个现象：当飞行器是position(定点)模式下，已经定好点了，此时拨动遥控，期望位置一直跟随实际位置，当不拨动遥控时，飞行器会重新定到新的位置而不返回之前的定点位置。

**顺着程序再来介绍内环PID**

……

比例

math::Vector<3> vel\_err = \_vel\_sp - \_vel;

if (!control\_vel\_enabled\_prev && \_control\_mode.flag\_control\_velocity\_enabled) {

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

// given by the last attitude setpoint

//矫正xy速度设定值

\_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);

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

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

control\_vel\_enabled\_prev = true;

// compute updated velocity error

//用矫正后的速度设定值-实际速度，跟新速度误差

vel\_err = \_vel\_sp - \_vel;

}

……

PID公式

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

//推力设定值(三维)=速度差\*P+速度差的差\*D+积分

……

积分

if (\_control\_mode.flag\_control\_velocity\_enabled && !saturation\_xy) {

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) {

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

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

if (thrust\_int(2) > 0.0f) {

thrust\_int(2) = 0.0f;

}

}

……