```
 3 *   Copyright (c) 2013, 2014 PX4 Development Team. All rights reserved.
33
34/**
35 * @file mc_pos_control_main.cpp
36 * Multicopter position controller. 多旋翼位置控制器
37 *
38 * Original publication for the desired attitude generation: 期望姿态的原始出版物
39 * Daniel Mellinger and Vijay Kumar. Minimum Snap Trajectory Generation and Control for Quadrotors.
40 * 多旋翼的 最小单元轨迹生成和控制。
41 * Int. Conf. on Robotics and Automation, Shanghai, China, May 2011
42 *
43 * Also inspired by https://pixhawk.org/firmware/apps/fw_pos_control_l1
44 *
45 * The controller has two loops: P loop for position error and PID loop for velocity error.
46 * 控制器有两个loop：P loop 位置误差，PID loop为速度误差
47 * Output of velocity controller is thrust vector that splitted to thrust direction/////////
48 * 速度控制器输出的是 推力向量 ， 为什么是向量 以分离到不同的电机上
49 * (i.e. rotation matrix for multicopter orientation) and thrust module (i.e. multicopter thrust itself).
50 * Controller doesn't use Euler angles for work, they generated only for more human-friendly control and logging.
51 *控制器不适用 欧拉角工作，他们只产生更多的人为控制和记录
52 * @author Anton Babushkin <anton.babushkin@me.com>
53 *
54 * 注意：不要混淆了概念
55 * 在这个文件中 position | pos 这里位置指的是 水平面，x y
56 *            Altitude |alt 高度 指的就是垂直方向 z
57 * 爬升率指的是垂直方向z的速度
58 * Velocity这个速度指的是 水平面上x y的速度
59 * 所以注意区分这些概念，以免理解错误！
60 */
61
62#include <px4_config.h>
63#include <px4_defines.h>
64#include <px4_tasks.h>
65#include <px4_posix.h>
66#include <stdio.h>
67#include <stdlib.h>
68#include <string.h>
69#include <unistd.h>
70#include <fcntl.h>
71#include <errno.h>
72#include <math.h>
73#include <poll.h>
74#include <functional>
75#include <drivers/drv_hrt.h>
```

```
 76 #include <arch/board/board.h>
 77
 78 #include <uORB/topics/manual_control_setpoint.h>
 79 #include <uORB/topics/actuator_controls.h>
 80 #include <uORB/topics/vehicle_rates_setpoint.h>
 81 #include <uORB/topics/control_state.h>
 82 #include <uORB/topics/vehicle_control_mode.h>
 83 #include <uORB/topics/actuator_armed.h>
 84 #include <uORB/topics/parameter_update.h>
 85 #include <uORB/topics/vehicle_local_position.h>
 86 #include <uORB/topics/position_setpoint_triplet.h>
 87 #include <uORB/topics/vehicle_global_velocity_setpoint.h>
 88 #include <uORB/topics/vehicle_local_position_setpoint.h>
 89
 90 #include <systemlib/systemlib.h>
 91 #include <mathlib/mathlib.h>
 92 #include <lib/geo/geo.h>
 93 #include <mavlink/mavlink_log.h>
 94 #include <platforms/px4_defines.h>
 95
 96 #include <controllib/blocks.hpp>
 97 #include <controllib/block/BlockParam.hpp>
 98
 99 #define TILT_COS_MAX    0.7f      //倾斜的最大值
100 #define SIGMA           0.000001f
101 #define MIN_DIST        0.01f     //最小距离
102 #define MANUAL_THROTTLE_MAX_MULTICOPTER 0.9f //旋翼手工操作最大的油门
103
104 /**
105  * Multicopter position control app start / stop handling function
106  *多旋翼位置控制应用 启动/停止处理函数
107  * @ingroup apps
108  */
109 //申明这个main函数 这是这个文件程序的入口
110 extern "C" __EXPORT int mc_pos_control_main(int argc, char *argv[]);//这是
    有分号 只是申明
111
112
113 class MulticopterPositionControl : public control::SuperBlock
114 {
115 public:
116     /**
117      * Constructor
118      */
119     MulticopterPositionControl();
120
121     /**
122      * Destructor, also kills task.
123      */
124     ~MulticopterPositionControl();
```

```
125
126     /**
127      * Start task.
128      * @return      OK on success.
129      */
130     int     start();//start task, return OK on success
131
132 private://私有变量 都是为自己所用
133     //sub都是订阅subscribe pub是publish
134     const float alt_ctl_dz = 0.1f;  //这里是一个const常量 不可变的
135
136     bool    _task_should_exit;      /**< if true, task should exit */
137     int     _control_task;          /**< task handle for task */
138     int     _mavlink_fd;            /**< mavlink fd */
139
140     int     _vehicle_status_sub;    /**< vehicle status subscription 车辆状
    态订阅*/
141     int     _ctrl_state_sub;        /**< control state subscription */
142     int     _att_sp_sub;            /**< vehicle attitude setpoint 车辆姿态
    设定 */
143     int     _control_mode_sub;      /**< vehicle control mode subscription
    */
144     int     _params_sub;            /**< notification of parameter updates
    */
145     int     _manual_sub;            /**< notification of manual control
    updates */
146     int     _arming_sub;            /**< arming status of outputs */
147     int     _local_pos_sub;         /**< vehicle local position */
148     int     _pos_sp_triplet_sub;    /**< position setpoint triplet */
149     int     _local_pos_sp_sub;      /**< offboard local position setpoint
    */
150     int     _global_vel_sp_sub;     /**< offboard global velocity setpoint
    */
151
152     orb_advert_t      _att_sp_pub;          /**< attitude setpoint
    publication */
153     orb_advert_t  _local_pos_sp_pub;        /**< vehicle local position
    setpoint publication */
154     orb_advert_t  _global_vel_sp_pub;       /**< vehicle global velocity
    setpoint publication */
155
156     struct vehicle_status_s             _vehicle_status;    /**< vehicle
    status */
157     struct control_state_s              _ctrl_state;        /**< vehicle
    attitude */
158     struct vehicle_attitude_setpoint_s      _att_sp;        /**< vehicle
    attitude setpoint */
159     struct manual_control_setpoint_s        _manual;        /**< r/c
    channel data */
160     struct vehicle_control_mode_s           _control_mode;  /**< vehicle
```

```
   control mode */
161     struct actuator_armed_s                 _arming;        /**< actuator
   arming status */
162     struct vehicle_local_position_s         _local_pos;     /**< vehicle
   local position */
163     struct
   position_setpoint_triplet_s     _pos_sp_triplet;//previous;current;
   next;/**< vehicle global position setpoint triplet */
164     struct vehicle_local_position_setpoint_s    _local_pos_sp;      /**<
   vehicle local position setpoint */
165     struct vehicle_global_velocity_setpoint_s   _global_vel_sp;// (vx vy
   vz) /**< vehicle global velocity setpoint */
166
167     control::BlockParamFloat _manual_thr_min;//manual throttle min
168     control::BlockParamFloat _manual_thr_max;//manual throttle max
169
170     control::BlockDerivative _vel_x_deriv; //velocity x deriv导数
171     control::BlockDerivative _vel_y_deriv; //velocity y deriv导数
172     control::BlockDerivative _vel_z_deriv; //velocity x deriv导数
173
174     struct {
175         param_t thr_min;
176         param_t thr_max;
177         param_t z_p;
178         param_t z_vel_p;
179         param_t z_vel_i;
180         param_t z_vel_d;
181         param_t z_vel_max;
182         param_t z_ff;
183         param_t xy_p;
184         param_t xy_vel_p;
185         param_t xy_vel_i;
186         param_t xy_vel_d;
187         param_t xy_vel_max;
188         param_t xy_ff;
189         param_t tilt_max_air;
190         param_t land_speed;
191         param_t tilt_max_land;
192         param_t man_roll_max;
193         param_t man_pitch_max;
194         param_t man_yaw_max;
195         param_t mc_att_yaw_p;
196         param_t hold_xy_dz;
197         param_t hold_z_dz;
198         param_t hold_max_xy;
199         param_t hold_max_z;
200     }       _params_handles;        /**< 参数处理handles for interesting
   parameters */
201
202     struct {
```

```
203         float thr_min;
204         float thr_max;
205         float tilt_max_air;
206         float land_speed;
207         float tilt_max_land;
208         float man_roll_max;
209         float man_pitch_max;
210         float man_yaw_max;
211         float mc_att_yaw_p;
212         float hold_xy_dz;
213         float hold_z_dz;
214         float hold_max_xy;
215         float hold_max_z;
216 
217         math::Vector<3> pos_p;
218         math::Vector<3> vel_p;
219         math::Vector<3> vel_i;
220         math::Vector<3> vel_d;
221         math::Vector<3> vel_ff;
222         math::Vector<3> vel_max;
223         math::Vector<3> sp_offs_max;
224     }       _params;//上一个是_params_handles 这个是_params
225 
226     struct map_projection_reference_s _ref_pos;//reference position
227     float _ref_alt; //reference altitude
228     hrt_abstime _ref_timestamp;//reference 时间戳
229 
230     bool _reset_pos_sp; //SP setpoint 预置值或者在这里就是目标值
231     bool _reset_alt_sp;
232     bool _mode_auto;
233     bool _pos_hold_engaged;
234     bool _alt_hold_engaged;
235     bool _run_pos_control;
236     bool _run_alt_control;
237 
238     math::Vector<3> _pos;
239     math::Vector<3> _pos_sp;
240     math::Vector<3> _vel;
241     math::Vector<3> _vel_sp;
242     math::Vector<3> _vel_prev; /**<  前一次的速度velocity on previous step
    */
243     math::Vector<3> _vel_ff;   //ff feed forward 前馈
244 
245     math::Matrix<3, 3> _R;  /**< 姿态的转换矩阵 四元数表示rotation matrix
    from attitude quaternions */
246     float _yaw;             /**< 偏航角 欧拉角yaw angle (euler) */
247 
248     /**
249      * Update our local parameter cache.更新缓存的局部参数
250      */
```

```
251     int     parameters_update(bool force);
252
253     /**
254      * Update control outputs更新控制输出
255      */
256     void        control_update();
257
258     /**
259      *这个函数用来检查订阅主题是否更新，更新了就把内容从_sub备份中 拷到类中定义的参数中
260      */
261     void        poll_subscriptions();
262
263     static float    scale_control(float ctl, float end, float dz);
264
265     /**
266      * Update reference for local position projection更新本地位置投影的参考
267      */
268     void        update_ref();
269     /**
270      * Reset position setpoint to current position重置位置预置值为当前位置
271      */
272     void        reset_pos_sp();
273
274     /**
275      * Reset altitude setpoint to current altitude
276      */
277     void        reset_alt_sp();
278
279     /**
280      * Check if position setpoint is too far from current position and adjust it if needed.
281      *检查位置设置值 是否过多的偏离当前位置，如果有需要调整它
282      */
283     void        limit_pos_sp_offset();//limit position setpoint offset限制位置预置值的偏移，检查是否与当前位置偏差太大，需要时调整它
284
285     /**
286      * Set position setpoint using manual control手动设置位置预置值 在这里猜测所谓的setpoint可能是目前位置的设定值
287      */
288     void        control_manual(float dt);
289
290     /**
291      * Set position setpoint using offboard control
292      */
293     void        control_offboard(float dt);
294
295     bool        cross_sphere_line(const math::Vector<3> &sphere_c, float sphere_r,
```

```
296                     const math::Vector<3> line_a, const math::Vector<3>
   line_b, math::Vector<3> &res);
297
298     /**
299      * Set position setpoint for AUTO
300      */
301     void        control_auto(float dt);
302
303     /**
304      * Select between barometric and global (AMSL) altitudes选择高度
305      */
306     void        select_alt(bool global);
307
308     /**
309      * Shim for calling task_main from task_create.
310      */
311     static void task_main_trampoline(int argc, char *argv[]);
312
313     /**
314      * Main sensor collection task.
315      */
316     void        task_main();
317 };
318
319 namespace pos_control
320 {
321
322 /* oddly, ERROR is not defined for c++ */
323 #ifdef ERROR
324 # undef ERROR
325 #endif
326 static const int ERROR = -1;
327
328 MulticopterPositionControl  *g_control;
329 }
330
331 //1、构造函数
332 MulticopterPositionControl::MulticopterPositionControl() :
333     SuperBlock(NULL, "MPC"),
334     _task_should_exit(false),
335     _control_task(-1),
336     _mavlink_fd(-1),
337
338     /* subscriptions */
339     _ctrl_state_sub(-1),
340     _att_sp_sub(-1),
341     _control_mode_sub(-1),
342     _params_sub(-1),
343     _manual_sub(-1),
344     _arming_sub(-1),
```

```
345     _local_pos_sub(-1),
346     _pos_sp_triplet_sub(-1),
347     _global_vel_sp_sub(-1),
348
349     /* publications */
350     _att_sp_pub(nullptr),
351     _local_pos_sp_pub(nullptr),
352     _global_vel_sp_pub(nullptr),
353     _manual_thr_min(this, "MANTHR_MIN"),
354     _manual_thr_max(this, "MANTHR_MAX"),
355     _vel_x_deriv(this, "VELD"),
356     _vel_y_deriv(this, "VELD"),
357     _vel_z_deriv(this, "VELD"),
358     _ref_alt(0.0f),
359     _ref_timestamp(0),
360
361     _reset_pos_sp(true),
362     _reset_alt_sp(true),
363     _mode_auto(false),
364     _pos_hold_engaged(false),
365     _alt_hold_engaged(false),
366     _run_pos_control(true),
367     _run_alt_control(true),
368     _yaw(0.0f)
369 {
370     memset(&_vehicle_status, 0, sizeof(_vehicle_status));
371     memset(&_ctrl_state, 0, sizeof(_ctrl_state));
372     memset(&_att_sp, 0, sizeof(_att_sp));
373     memset(&_manual, 0, sizeof(_manual));
374     memset(&_control_mode, 0, sizeof(_control_mode));
375     memset(&_arming, 0, sizeof(_arming));
376     memset(&_local_pos, 0, sizeof(_local_pos));
377     memset(&_pos_sp_triplet, 0, sizeof(_pos_sp_triplet));
378     memset(&_local_pos_sp, 0, sizeof(_local_pos_sp));
379     memset(&_global_vel_sp, 0, sizeof(_global_vel_sp));
380
381     memset(&_ref_pos, 0, sizeof(_ref_pos));
382
383     _params.pos_p.zero();
384     _params.vel_p.zero();
385     _params.vel_i.zero();
386     _params.vel_d.zero();
387     _params.vel_max.zero();
388     _params.vel_ff.zero();
389     _params.sp_offs_max.zero();
390
391     _pos.zero();
392     _pos_sp.zero();
393     _vel.zero();
394     _vel_sp.zero();
```

```
395     _vel_prev.zero();
396     _vel_ff.zero();
397 
398     _R.identity();
399 
400     _params_handles.thr_min     = param_find("MPC_THR_MIN");
401     _params_handles.thr_max     = param_find("MPC_THR_MAX");
402     _params_handles.z_p         = param_find("MPC_Z_P");
403     _params_handles.z_vel_p     = param_find("MPC_Z_VEL_P");
404     _params_handles.z_vel_i     = param_find("MPC_Z_VEL_I");
405     _params_handles.z_vel_d     = param_find("MPC_Z_VEL_D");
406     _params_handles.z_vel_max   = param_find("MPC_Z_VEL_MAX");
407     _params_handles.z_ff        = param_find("MPC_Z_FF");
408     _params_handles.xy_p        = param_find("MPC_XY_P");
409     _params_handles.xy_vel_p    = param_find("MPC_XY_VEL_P");
410     _params_handles.xy_vel_i    = param_find("MPC_XY_VEL_I");
411     _params_handles.xy_vel_d    = param_find("MPC_XY_VEL_D");
412     _params_handles.xy_vel_max  = param_find("MPC_XY_VEL_MAX");
413     _params_handles.xy_ff       = param_find("MPC_XY_FF");
414     _params_handles.tilt_max_air    = param_find("MPC_TILTMAX_AIR");
415     _params_handles.land_speed  = param_find("MPC_LAND_SPEED");
416     _params_handles.tilt_max_land   = param_find("MPC_TILTMAX_LND");
417     _params_handles.man_roll_max = param_find("MPC_MAN_R_MAX");
418     _params_handles.man_pitch_max = param_find("MPC_MAN_P_MAX");
419     _params_handles.man_yaw_max = param_find("MPC_MAN_Y_MAX");
420     _params_handles.mc_att_yaw_p = param_find("MC_YAW_P");
421     _params_handles.hold_xy_dz = param_find("MPC_HOLD_XY_DZ");
422     _params_handles.hold_z_dz = param_find("MPC_HOLD_Z_DZ");
423     _params_handles.hold_max_xy = param_find("MPC_HOLD_MAX_XY");
424     _params_handles.hold_max_z = param_find("MPC_HOLD_MAX_Z");
425 
426 
427     /* fetch initial parameter values */
428     parameters_update(true);//更新参数 更新了以上这些参数 把以上这些自定义的私有
    变量通过param_find函数与系统参数相匹配
429                             //估计对他们的操作 就是对系统参数的操作
430 }
431 
432 //2、解析函数
433 MulticopterPositionControl::~MulticopterPositionControl()
434 {
435     if (_control_task != -1) {
436         /* task wakes up every 100ms or so at the longest */
437         _task_should_exit = true;
438 
439         /* wait for a second for the task to quit at our request */
440         unsigned i = 0;
441 
442         do {
443             /* wait 20ms */
```

```
444             usleep(20000);
445
446             /* if we have given up, kill it */
447             if (++i > 50) {
448                 px4_task_delete(_control_task);
449                 break;
450             }
451         } while (_control_task != -1);
452     }
453
454     pos_control::g_control = nullptr;
455 }
456
457 //3、参数更新函数 在1、构造函数中调用
458 //Update our local parameter cache.更新缓存的局部参数 //这个函数在构造函数的最后进行调用 更新参数
459 int MulticopterPositionControl::parameters_update(bool force)
460 {
461     bool updated;
462     struct parameter_update_s param_upd;
463
464     orb_check(_params_sub, &updated);
465
466     if (updated) {
467         orb_copy(ORB_ID(parameter_update), _params_sub, &param_upd);
468     }
469
470     if (updated || force) {
471         /* update C++ param system */
472         updateParams();
473
474         /* update legacy C interface params */
475         param_get(_params_handles.thr_min, &_params.thr_min);
476         param_get(_params_handles.thr_max, &_params.thr_max);
477         param_get(_params_handles.tilt_max_air, &_params.tilt_max_air);
478         _params.tilt_max_air = math::radians(_params.tilt_max_air);
479         param_get(_params_handles.land_speed, &_params.land_speed);
480         param_get(_params_handles.tilt_max_land, &_params.tilt_max_land);
481         _params.tilt_max_land = math::radians(_params.tilt_max_land);
482
483         float v;
484         param_get(_params_handles.xy_p, &v);
485         _params.pos_p(0) = v;
486         _params.pos_p(1) = v;
487
488         param_get(_params_handles.z_p, &v);
489         _params.pos_p(2) = v;
490         param_get(_params_handles.xy_vel_p, &v);
491         _params.vel_p(0) = v;
492         _params.vel_p(1) = v;
```

```
493         param_get(_params_handles.z_vel_p, &v);
494         _params.vel_p(2) = v;
495         param_get(_params_handles.xy_vel_i, &v);
496         _params.vel_i(0) = v;
497         _params.vel_i(1) = v;
498         param_get(_params_handles.z_vel_i, &v);
499         _params.vel_i(2) = v;
500         param_get(_params_handles.xy_vel_d, &v);
501         _params.vel_d(0) = v;
502         _params.vel_d(1) = v;
503         param_get(_params_handles.z_vel_d, &v);
504         _params.vel_d(2) = v;
505         param_get(_params_handles.xy_vel_max, &v);
506         _params.vel_max(0) = v;
507         _params.vel_max(1) = v;
508         param_get(_params_handles.z_vel_max, &v);
509         _params.vel_max(2) = v;
510         param_get(_params_handles.xy_ff, &v);
511         v = math::constrain(v, 0.0f, 1.0f);
512         _params.vel_ff(0) = v;
513         _params.vel_ff(1) = v;
514         param_get(_params_handles.z_ff, &v);
515         v = math::constrain(v, 0.0f, 1.0f);
516         _params.vel_ff(2) = v;
517         param_get(_params_handles.hold_xy_dz, &v);
518         v = math::constrain(v, 0.0f, 1.0f);
519         _params.hold_xy_dz = v;
520         param_get(_params_handles.hold_z_dz, &v);
521         v = math::constrain(v, 0.0f, 1.0f);
522         _params.hold_z_dz = v;
523         param_get(_params_handles.hold_max_xy, &v);
524         _params.hold_max_xy = (v < 0.0f ? 0.0f : v);
525         param_get(_params_handles.hold_max_z, &v);
526         _params.hold_max_z = (v < 0.0f ? 0.0f : v);
527 
528         _params.sp_offs_max = _params.vel_max.edivide(_params.pos_p) *
    2.0f;
529 
530         /* mc attitude control parameters*/
531         /* manual control scale */
532         param_get(_params_handles.man_roll_max, &_params.man_roll_max);
533         param_get(_params_handles.man_pitch_max, &_params.man_pitch_max);
534         param_get(_params_handles.man_yaw_max, &_params.man_yaw_max);
535         _params.man_roll_max = math::radians(_params.man_roll_max);
536         _params.man_pitch_max = math::radians(_params.man_pitch_max);
537         _params.man_yaw_max = math::radians(_params.man_yaw_max);
538         param_get(_params_handles.mc_att_yaw_p, &v);
539         _params.mc_att_yaw_p = v;
540     }
541 
```

```
542     return OK;
543 }
544 
545 //4、订阅主题是否更新，更新就把内容从_sub备份中 拷到类中定义的参数
546 void
547 MulticopterPositionControl::poll_subscriptions()
548 {
549     bool updated;
550 
551     orb_check(_vehicle_status_sub, &updated);
552 
553     if (updated) {
554         orb_copy(ORB_ID(vehicle_status), _vehicle_status_sub,
    &_vehicle_status);
555     }
556 
557     orb_check(_ctrl_state_sub, &updated);
558 
559     if (updated) {
560         orb_copy(ORB_ID(control_state), _ctrl_state_sub, &_ctrl_state);
561     }
562 
563     orb_check(_att_sp_sub, &updated);
564 
565     if (updated) {
566         orb_copy(ORB_ID(vehicle_attitude_setpoint), _att_sp_sub,
    &_att_sp);
567     }
568 
569     orb_check(_control_mode_sub, &updated);
570 
571     if (updated) {
572         orb_copy(ORB_ID(vehicle_control_mode), _control_mode_sub,
    &_control_mode);
573     }
574 
575     orb_check(_manual_sub, &updated);
576 
577     if (updated) {
578         orb_copy(ORB_ID(manual_control_setpoint), _manual_sub, &_manual);
579     }
580 
581     orb_check(_arming_sub, &updated);
582 
583     if (updated) {
584         orb_copy(ORB_ID(actuator_armed), _arming_sub, &_arming);
585     }
586 
587     orb_check(_local_pos_sub, &updated);
588 
```

```
589     if (updated) {
590         orb_copy(ORB_ID(vehicle_local_position), _local_pos_sub,
    &_local_pos);
591     }
592 }
593 
594 /**
595  * 4、 和上面4函数在一起定义的 看表面解释“ 控制范围”
596  * 返回是float
597  */
598 float
599 MulticopterPositionControl::scale_control(float ctl, float end, float dz)
600 {
601     if (ctl > dz)
602     {
603         return (ctl - dz) / (end - dz);
604 
605     }
606     else if (ctl < -dz)
607     {
608         return (ctl + dz) / (end - dz);
609 
610     }
611     else
612     {
613         return 0.0f;
614     }
615 }
616 
617 void
618 MulticopterPositionControl::task_main_trampoline(int argc, char *argv[])
619 {
620     pos_control::g_control->task_main();
621 }
622 
623 /**参考更新
624  * 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()，
625  * 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。 */
626 void MulticopterPositionControl::update_ref()
627 {                                       //前面带“_”表示的是上一次的数据
628     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
629         double lat_sp, lon_sp;//longitude经度 latitude纬度 altitude高度
630         float alt_sp = 0.0f;
631 
632         if (_ref_timestamp != 0) {
633             /*获取当前的位置和高度参考数值 calculate current position setpoint
    in global frame */
```

```
634             map_projection_reproject(&_ref_pos, _pos_sp(0), _pos_sp(1),
    &lat_sp, &lon_sp);
635             alt_sp = _ref_alt - _pos_sp(2);
636         }
637 
638         /* 更新当前的位置和高度参考数值update local projection reference */
639         map_projection_init(&_ref_pos, _local_pos.ref_lat,
    _local_pos.ref_lon);
640         _ref_alt = _local_pos.ref_alt;
641 
642         if (_ref_timestamp != 0) {
643             /* 映射当前的位置和高度参考数值到参考系reproject position setpoint
    to new reference */
644             map_projection_project(&_ref_pos, lat_sp, lon_sp,
    &_pos_sp.data[0], &_pos_sp.data[1]);
645             _pos_sp(2) = -(alt_sp - _ref_alt);
646         }
647 
648         _ref_timestamp = _local_pos.ref_timestamp;//更新参考时间
    为vehicle_local_position中时间帧
649     }
650 }
651 
652 void
653 MulticopterPositionControl::reset_pos_sp()
654 {
655     if (_reset_pos_sp) {//检查我们是否将非速度控制模式转变成速度控制模式，如果
    是，那么矫正xy速度设定值，以便姿态设定值是连续的
656         _reset_pos_sp = false;
657         /* shift position setpoint to make attitude setpoint continuous */
658         _pos_sp(0) = _pos(0) + (_vel(0) - PX4_R(_att_sp.R_body, 0, 2) *
    _att_sp.thrust / _params.vel_p(0)
659                     - _params.vel_ff(0) * _vel_sp(0)) / _params.pos_p(0);
660         _pos_sp(1) = _pos(1) + (_vel(1) - PX4_R(_att_sp.R_body, 1, 2) *
    _att_sp.thrust / _params.vel_p(1)
661                     - _params.vel_ff(1) * _vel_sp(1)) / _params.pos_p(1);
662         mavlink_log_info(_mavlink_fd, "[mpc] reset pos sp: %d, %d",
    (int)_pos_sp(0), (int)_pos_sp(1));
663     }
664 }
665 
666 void
667 MulticopterPositionControl::reset_alt_sp()
668 {
669     if (_reset_alt_sp) {//首次进入 进入后就赋值false 应该只进入这一次
670         _reset_alt_sp = false;
671         _pos_sp(2) = _pos(2) + (_vel(2) - _params.vel_ff(2) * _vel_sp(2))
    / _params.pos_p(2);
672         mavlink_log_info(_mavlink_fd, "[mpc] reset alt sp: %d", -
    (int)_pos_sp(2));
```

```
673     }
674 }
675
676 void
677 MulticopterPositionControl::limit_pos_sp_offset()
678 {
679     math::Vector<3> pos_sp_offs;
680     pos_sp_offs.zero();
681
682     if (_control_mode.flag_control_position_enabled) {
683         pos_sp_offs(0) = (_pos_sp(0) - _pos(0)) / _params.sp_offs_max(0);
684         pos_sp_offs(1) = (_pos_sp(1) - _pos(1)) / _params.sp_offs_max(1);
685     }
686
687     if (_control_mode.flag_control_altitude_enabled) {
688         pos_sp_offs(2) = (_pos_sp(2) - _pos(2)) / _params.sp_offs_max(2);
689     }
690
691     float pos_sp_offs_norm = pos_sp_offs.length();
692
693     if (pos_sp_offs_norm > 1.0f) {
694         pos_sp_offs /= pos_sp_offs_norm;
695         _pos_sp = _pos + pos_sp_offs.emult(_params.sp_offs_max);
696     }
697 }
698
699 /**
700  * 简要说明一下程序中用到的变量
701  * math::Vector<3> req_vel_sp; 先理解成是一个数组吧 里面有三个元素，放的是 需求的速度的设定值 这些需求值在manual模式下 来自遥控器的操作杆
702  *                                 即req_vel_sp(2)是 vertical velocity req_vel_sp(0)(1)horizontal velocity
703  * math::Vector<3> req_vel_sp_scaled对这个速度进行缩放 规范化
704  */
705 ///////// Set position setpoint using manual control
706 void
707 MulticopterPositionControl::control_manual(float dt)
708 {
709     math::Vector<3> req_vel_sp; // req_vel_sp需求的速度setpoint X,Y in local frame and Z in global (D), in [-1,1] normalized range
710     req_vel_sp.zero();
711
712     if (_control_mode.flag_control_altitude_enabled) {//高度控制 肯定是vertical velocity
713         /* 使用油门操作杆 设置 垂直速度设定值 set vertical velocity setpoint with throttle stick */
714         //_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
715         req_vel_sp(2) = -scale_control(_manual.z - 0.5f, 0.5f, alt_ctl_dz);//alt_ctl_dz=0.1 设置垂直速度参考，通过尺度控制器（将0-1转换
```

```
为-1——1）
716     }
717             //这是manual手动模式 既然是手动模式 肯定是从遥控器读取信号 转换为相应的设定值 req_vel_sp 需求的速度设定值 从遥控器操作杆读取的
718             //_manual;      r/c channel data
719
720     if (_control_mode.flag_control_position_enabled) {//位置控制肯定是horizontal velocity
721         /* 使用roll/pitch操作杆设置 水平速度设定值 set horizontal velocity setpoint with roll/pitch stick */
722         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
723         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
724     }
725         //高度控制使能z
726     if (_control_mode.flag_control_altitude_enabled) {
727         /*重置高度参考z，使用当前高度 reset alt setpoint to current altitude if needed */
728         reset_alt_sp();//这些函数进去看 发现 首次重置，但是好像只有第一次是重置后面就没了
729     }
730     //位置控制使能xy
731     if (_control_mode.flag_control_position_enabled) {
732         /* 重置位置参考xy，使用当前位置reset position setpoint to current position if needed */
733         reset_pos_sp();
734     }
735
736     /* 限制速度设定值的范围 因为输入肯定是有一个范围 就像再大的电压超过电机的上限也无用 反而引入饱和
737      * limit velocity setpoint
738      * 下面是个“求平均的过程”类似与“规范化”  in [-1,1] normalized range
739     */
740     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
741
742     if (req_vel_sp_norm > 1.0f) {
743         req_vel_sp /= req_vel_sp_norm;
744     }
745
746     //
747     /* _req_vel_sp scaled to 0..1, scale it to max speed and rotate around yaw 绕偏航旋转 */
748     math::Matrix<3, 3> R_yaw_sp;
749     R_yaw_sp.from_euler(0.0f, 0.0f, _att_sp.yaw_body); //from_euler由三个欧拉角产生一个旋转矩阵 这里绕yaw旋转 NED下机体偏航得到绕z旋转的矩阵
750     math::Vector<3> req_vel_sp_scaled = R_yaw_sp * req_vel_sp.emult(
751             _params.vel_max); // in NED and scaled to actual velocity缩放到实际的速度
```

```
752
753     /*
754      * assisted velocity mode: user controls velocity, but if velocity is small enough, position
755      * hold is activated for the corresponding axis
756      * 辅助速度模式，用户控制速度，速度过小位置不变
757      */
758
//flag_control_position_enabled这是（水平）位置控制标识
_pos_hold_engaged（水平）位置保持标志
759     /* horizontal axes 水平方向xy，这里主要是配置位置保持*/
760         if (_control_mode.flag_control_position_enabled) {//位置控制使能是前提
761             /* check for pos. hold  速度参考点绝对值（0-1）小于死区（这里为0.1）*/
762             if (fabsf(req_vel_sp(0)) < _params.hold_xy_dz &&
fabsf(req_vel_sp(1)) < _params.hold_xy_dz)
763             {                           //hold_xy_dz 水平位置保持死区 单位是 %
//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
764                 if (!_pos_hold_engaged)//位置保持未使用，接下来判断参数是否可以开启
765                 {
766                     if ( _params.hold_max_xy < FLT_EPSILON
767                          || ( fabsf(_vel(0)) < _params.hold_max_xy  &&
fabsf(_vel(1)) < _params.hold_max_xy) )//hold_max_xy默认为0.8
768                     {
769                         _pos_hold_engaged = true;
770
771                     } else {
772                         _pos_hold_engaged = false;
773                     }
774                 }
775
776             }
777             else
778             {
779                 _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
780             }
781
782             /* set requested velocity setpoint */
783             if (!_pos_hold_engaged)//位置保持不可用（根据上面的结果）
784             {
785                 _pos_sp(0) = _pos(0);//重新配置参考位置，根据当前的位置
786                 _pos_sp(1) = _pos(1);
787                 _run_pos_control = false; /* request velocity setpoint to be used, instead of position setpoint */
788                 _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
```

```
789                 _vel_sp(1) = req_vel_sp_scaled(1);
790             }
791         }
792 
793     /* vertical axis 垂直方向z，这里主要是配置高度保持*/
794         if (_control_mode.flag_control_altitude_enabled) {//高度控制使能是前提
795             /* 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*/
796             if (fabsf(req_vel_sp(2)) < _params.hold_z_dz) {
797                 if (!_alt_hold_engaged) {//高度保持为启用那么继续判断是否满足要求
798                     if (_params.hold_max_z < FLT_EPSILON || fabsf(_vel(2)) < _params.hold_max_z) {//hold_max_z默认为0.6
799                         _alt_hold_engaged = true;
800 
801                     } else {
802                         _alt_hold_engaged = false;//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
803                     }
804                 }
805 
806             } else {
807                 _alt_hold_engaged = false;//垂直速度不满足要求放弃高度保持
808             }
809 
810             /* set requested velocity setpoint */
811             if (!_alt_hold_engaged) {//高度保持不可用（根据上面的结果）
812                 _run_alt_control = false; /* request velocity setpoint to be used, instead of altitude setpoint */
813                 _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
814                 _pos_sp(2) = _pos(2);
815             }
816         }
817 }
818 
819 
820 void    /////////Set position setpoint using offboard control
821 MulticopterPositionControl::control_offboard(float dt)//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：
822 {
823     bool updated;
824     orb_check(_pos_sp_triplet_sub, &updated);
825 
826     if (updated) {                                  //这里获取新内容，前一次，本次和下一次的位置参考数据// _pos_sp_triplet;//previous;current; next;
827         orb_copy(ORB_ID(position_setpoint_triplet), _pos_sp_triplet_sub, &_pos_sp_triplet);/**< vehicle global position setpoint triplet */
828     }
```

```
829
830
831     if (_pos_sp_triplet.current.valid) {//本次参考点有效
832
833             if (_control_mode.flag_control_position_enabled &&
    _pos_sp_triplet.current.position_valid) {
834                 /* 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*/
835                 _pos_sp(0) = _pos_sp_triplet.current.x;
836                 _pos_sp(1) = _pos_sp_triplet.current.y;
837
838             }
839            else if
840            (_control_mode.flag_control_velocity_enabled &&
    _pos_sp_triplet.current.velocity_valid) {
841                 /* 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*/
842                 /* reset position setpoint to current position if needed
    */
843                 reset_pos_sp();//重置位置参考xy，使用当前位置
844
845                 /* set position setpoint move rate */
846                 _vel_sp(0) = _pos_sp_triplet.current.vx;
847                 _vel_sp(1) = _pos_sp_triplet.current.vy;
848
849                 _run_pos_control = false; /* request velocity setpoint to
    be used, instead of position setpoint */
850             }
851
852
853             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）
854                 _att_sp.yaw_body = _pos_sp_triplet.current.yaw;
855
856             }
857             else if
858             (_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）
859                 _att_sp.yaw_body = _att_sp.yaw_body +
    _pos_sp_triplet.current.yawspeed * dt;
860             }
861
862
863             if (_control_mode.flag_control_altitude_enabled &&
    _pos_sp_triplet.current.position_valid) {
864                 /* Control altitude 高度控制使能并且当前位置参考点有效*/
865                 _pos_sp(2) = _pos_sp_triplet.current.z;//高度参考使用本次高
    度值NED
866
867             }
```

```
868             else if (_control_mode.flag_control_climb_rate_enabled &&
    _pos_sp_triplet.current.velocity_valid) {//
869                 /* 爬升率控制使能并且本次速度参考点有效,爬升率控制，位置先重置高
    度参考reset alt setpoint to current altitude if needed */
870                 reset_alt_sp();
871 
872                 /* 垂直速度参考使用本次z轴速度，并且不使用高度控制set altitude
    setpoint move rate */
873                 _vel_sp(2) = _pos_sp_triplet.current.vz;
874 
875                 _run_alt_control = false; /* request velocity setpoint to
    be used, instead of position setpoint */
876             }
877 
878     }
879     else//本次参考点无效那么重新配置参考点 //if
    (_pos_sp_triplet.current.valid)
880     {
881         reset_pos_sp();//重置位置参考xy，使用当前位置
882         reset_alt_sp();//重置位置参考z，使用当前高度
883         //注意这些重置 进去看代码 好像都只会执行一次 一次过后重置以后 再不重置
884     }
885 }
886 
887 bool
888 MulticopterPositionControl::cross_sphere_line(const math::Vector<3>
    &sphere_c, float sphere_r,
889         const math::Vector<3> line_a, const math::Vector<3> line_b,
    math::Vector<3> &res)
890 {
891     /* project center of sphere on line */
892     /* normalized AB */
893     math::Vector<3> ab_norm = line_b - line_a;
894     ab_norm.normalize();
895     math::Vector<3> d = line_a + ab_norm * ((sphere_c - line_a) *
    ab_norm);
896     float cd_len = (sphere_c - d).length();
897 
898     /* we have triangle CDX with known CD and CX = R, find DX */
899     if (sphere_r > cd_len) {
900         /* have two roots, select one in A->B direction from D */
901         float dx_len = sqrtf(sphere_r * sphere_r - cd_len * cd_len);
902         res = d + ab_norm * dx_len;
903         return true;
904 
905     } else {
906         /* have no roots, return D */
907         res = d;
908         return false;
909     }
```

```
910 }
911
912 ////////Set position setpoint for AUTO
913 void MulticopterPositionControl::control_auto(float dt)//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：
914 {                                                       //非手动非离线就进入自动模式
915     if (!_mode_auto) {
916         _mode_auto = true;
917         /* reset position setpoint on AUTO mode activation */
918         reset_pos_sp();//重置位置参考xy，使用当前位置
919         reset_alt_sp();//重置高度参考z，使用当前高度
920     }
921
922     //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
923     bool updated;
924     orb_check(_pos_sp_triplet_sub, &updated);
925
926     if (updated) {
927         orb_copy(ORB_ID(position_setpoint_triplet), _pos_sp_triplet_sub, &_pos_sp_triplet);
928
929         //Make sure that the position setpoint is valid
930         if (!PX4_ISFINITE(_pos_sp_triplet.current.lat) ||//本次纬度
931             !PX4_ISFINITE(_pos_sp_triplet.current.lon) ||//本次经度
932             !PX4_ISFINITE(_pos_sp_triplet.current.alt)) {//本次高度
933             _pos_sp_triplet.current.valid = false;       //对于任一数据无效的情况，本次参考点的数据就无效
934         }
935     }
936
937     bool current_setpoint_valid = false;
938     bool previous_setpoint_valid = false;
939
940     math::Vector<3> prev_sp;
941     math::Vector<3> curr_sp;
942
943     if (_pos_sp_triplet.current.valid) {//本次参考点数据有效
944
945         /* project setpoint to local frame */
946         map_projection_project(&_ref_pos,
947                                _pos_sp_triplet.current.lat, _pos_sp_triplet.current.lon,
948                                &curr_sp.data[0], &curr_sp.data[1]); //位置xy使用经纬度映射关系获得
949
950         curr_sp(2) = -(_pos_sp_triplet.current.alt - _ref_alt);//高度为相对高度
951
952         if (PX4_ISFINITE(curr_sp(0)) &&
```

```
953             PX4_ISFINITE(curr_sp(1)) &&
954             PX4_ISFINITE(curr_sp(2))) {
955             current_setpoint_valid = true;//对于获得的本次参考点数据有效的情况则标志为有效
956         }
957     }
958 
959     if (_pos_sp_triplet.previous.valid) {//前一次参考点有效
960         map_projection_project(&_ref_pos,
961                                _pos_sp_triplet.previous.lat, _pos_sp_triplet.previous.lon,
962                                &prev_sp.data[0], &prev_sp.data[1]);//位置xy使用经纬度映射关系获得
963         prev_sp(2) = -(_pos_sp_triplet.previous.alt - _ref_alt);//高度为相对高度
964 
965         if (PX4_ISFINITE(prev_sp(0)) &&
966             PX4_ISFINITE(prev_sp(1)) &&
967             PX4_ISFINITE(prev_sp(2))) {
968             previous_setpoint_valid = true;//对于获得的前一次参考点数据有效的情况则标志为有效
969         }
970     }
971 
972     /////本次参考点数据有效（根据以上的判断） 这个条件直到这个函数的结束//////////////////////////
973     if (current_setpoint_valid) {
974         /* in case of interrupted mission don't go to waypoint but stay at current position */
975         _reset_pos_sp = true;
976         _reset_alt_sp = true;
977 
978         /* scaled space: 1 == position error resulting max allowed speed */
979         //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]
980         math::Vector<3> scale = _params.pos_p.edivide(_params.vel_max); // TODO add mult param here
981                                         //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
982 
983         /* convert current setpoint to scaled space */
984         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
985 
986         /* 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）by default use current setpoint as is */
987         math::Vector<3> pos_sp_s = curr_sp_s;
988 
989         if (_pos_sp_triplet.current.type ==
```

```
    position_setpoint_s::SETPOINT_TYPE_POSITION && previous_setpoint_valid) {
990             /* 本次参考点为位置类型并且前一次参考点数据有效 follow "previous - current" line */
991
992             if ((curr_sp - prev_sp).length() > MIN_DIST) {//三维距离大于最小距离（0.01）
993
994                 /* find X - cross point of unit sphere and trajectory 单位球面和轨迹的交叉点*/
995                 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
996                 math::Vector<3> prev_sp_s = prev_sp.emult(scale);//前一次参考点的尺度变换值
997                 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
998                 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）
999                 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）
1000
1001                if (curr_pos_s_len < 1.0f) {//本次距离小于1
1002                    /* 路径点在单位半径内 copter is closer to waypoint than unit radius */
1003                    /* 检查下个路径点并使用，避免过路径点时减速  check next waypoint and use it to avoid slowing down when passing via waypoint */
1004                    if (_pos_sp_triplet.next.valid) {//对于下一次参考点有效
1005                        math::Vector<3> next_sp;     //首先得到下一次参考点数据
1006                        map_projection_project(&_ref_pos,
1007                                               _pos_sp_triplet.next.lat, _pos_sp_triplet.next.lon,
1008                                               &next_sp.data[0], &next_sp.data[1]);
1009                        next_sp(2) = -(_pos_sp_triplet.next.alt - _ref_alt);
1010
1011                        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）
1012                            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
1013
1014                            /* calculate angle prev - curr - next */
1015                            math::Vector<3> curr_next_s = next_sp_s - curr_sp_s; //未来差距
1016                            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）
1017
1018                            /* 两个向量之间的夹角（本次与下一次） cos(a) * curr_next, a = angle between current and next trajectory segments */
1019                            float cos_a_curr_next = prev_curr_s_norm *
```

```
    curr_next_s;
1020
1021                                /*三个点之间的夹角 cos(b), b = angle pos -
    curr_sp - prev_sp */
1022                                float cos_b = -curr_pos_s * prev_curr_s_norm /
    curr_pos_s_len;
1023
1024                                if (cos_a_curr_next > 0.0f && cos_b > 0.0f) {
    //角度小于90
1025                                    float curr_next_s_len =
    curr_next_s.length();//求未来差距长度
1026
1027                                    /* if curr - next distance is larger than
    unit radius, limit it */
1028                                    if (curr_next_s_len > 1.0f) { //未来差距长于
    单位1，限制大小
1029                                        cos_a_curr_next /= curr_next_s_len;
1030                                    }
1031
1032                                    /*将位置参考点偏差前馈到位置参考点尺度值上面
    feed forward position setpoint offset */
1033                                    math::Vector<3> pos_ff = prev_curr_s_norm
    *
1034                                                cos_a_curr_next * cos_b *
    cos_b * (1.0f - curr_pos_s_len) *
1035                                                (1.0f - expf(-curr_pos_s_len
    * curr_pos_s_len * 20.0f));
1036                                    pos_sp_s += pos_ff;
1037                                }
1038                            }
1039                        }
1040
1041                    }
1042                    else {//本次距离大于1（飞行器离路径点比较远）
1043                        bool near = cross_sphere_line(pos_s, 1.0f, prev_sp_s,
    curr_sp_s, pos_sp_s);
1044
1045                        if (near) {//单位球与轨迹有交，即飞行器在轨迹附近
1046                            /* unit sphere crosses trajectory */
1047
1048                        } else {//距离轨迹太远
1049                            /* copter is too far from trajectory */
1050                            /*满足此将前一次参考点作为位置参考 if copter is
    behind prev waypoint, go directly to prev waypoint */
1051                            if ((pos_sp_s - prev_sp_s) * prev_curr_s < 0.0f)
    {//
1052                                pos_sp_s = prev_sp_s;
1053                            }
1054
1055                            /*满足此将本次参考点作为位置参考 if copter is in
```

```
    front of curr waypoint, go directly to curr waypoint */
1056                        if ((pos_sp_s - curr_sp_s) * prev_curr_s > 0.0f)
    {//
1057                            pos_sp_s = curr_sp_s;
1058                        }
1059
1060                        pos_sp_s = pos_s + (pos_sp_s -
    pos_s).normalized();//更新位置参考点（根据位置）
1061                    }
1062                }
1063            }
1064        }
1065
1066        /*限制最大速度  move setpoint not faster than max allowed speed */
1067        math::Vector<3> pos_sp_old_s = _pos_sp.emult(scale);//位置参考尺度值
1068
1069        /* difference between current and desired position setpoints, 1 =
    max speed */
1070        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
1071        float d_pos_m_len = d_pos_m.length();//偏差的长度
1072
1073        if (d_pos_m_len > dt) {//偏差大于dt，获取先位置参考尺度值
1074            pos_sp_s = pos_sp_old_s + (d_pos_m / d_pos_m_len *
    dt).emult(_params.pos_p);
1075        }
1076
1077        /* scale result back to normal space */
1078        _pos_sp = pos_sp_s.edivide(scale);//将尺度去掉，回到原值
1079
1080        /*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
    update yaw setpoint if needed */
1081        if (PX4_ISFINITE(_pos_sp_triplet.current.yaw)) {
1082            _att_sp.yaw_body = _pos_sp_triplet.current.yaw;
1083        }
1084
1085    }
1086    else {//没有本次参考点，那就啥都不做
1087        /* no waypoint, do nothing, setpoint was already reset */
1088    }
1089}
1090
1091void MulticopterPositionControl::task_main()
1092{
1093
1094    _mavlink_fd = px4_open(MAVLINK_LOG_DEVICE, 0);
1095
1096    /*
1097     * do subscriptions
1098     */
```

```
1099    _vehicle_status_sub = orb_subscribe(ORB_ID(vehicle_status));
1100    _ctrl_state_sub = orb_subscribe(ORB_ID(control_state));
1101    _att_sp_sub = orb_subscribe(ORB_ID(vehicle_attitude_setpoint));
1102    _control_mode_sub = orb_subscribe(ORB_ID(vehicle_control_mode));
1103    _params_sub = orb_subscribe(ORB_ID(parameter_update));
1104    _manual_sub = orb_subscribe(ORB_ID(manual_control_setpoint));
1105    _arming_sub = orb_subscribe(ORB_ID(actuator_armed));
1106    _local_pos_sub = orb_subscribe(ORB_ID(vehicle_local_position));
1107    _pos_sp_triplet_sub =
    orb_subscribe(ORB_ID(position_setpoint_triplet));
1108    _local_pos_sp_sub =
    orb_subscribe(ORB_ID(vehicle_local_position_setpoint));
1109    _global_vel_sp_sub =
    orb_subscribe(ORB_ID(vehicle_global_velocity_setpoint));
1110
1111
1112    parameters_update(true);
1113
1114    /* initialize values of critical structs until first regular update */
1115    _arming.armed = false;//解锁状态标志位
1116
1117    /* get an initial update for all sensor and status data */
1118    poll_subscriptions();
1119
1120    bool reset_int_z = true;//积分量重置标志位 int指积分integral
1121    bool reset_int_z_manual = false;
1122    bool reset_int_xy = true;
1123    bool reset_yaw_sp = true;//偏航参考点重置标志位
1124    bool was_armed = false;//上次解锁状态标志位
1125
1126    hrt_abstime t_prev = 0;//前一次执行时间
1127
1128    math::Vector<3> thrust_int;//推力积分量，重置   //速度控制器输出的是推力向
    量，为什么是向量 以分离到不同的电机上
1129    thrust_int.zero();
1130    math::Matrix<3, 3> R;//旋转矩阵，单位化（初始化）
1131    R.identity();
1132
1133    /* wakeup source相关主题的执行，循环的判断依据来源*/
1134    px4_pollfd_struct_t fds[1];
1135
1136    fds[0].fd = _local_pos_sub;
1137    fds[0].events = POLLIN;
1138
1139    while (!_task_should_exit){ //while开始，直到最后才结束，下面的内容基本都
    在while中
1140          /* 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。 */
1141
```

```
1142        /* wait for up to 500ms for data */
1143            int pret = px4_poll(&fds[0], (sizeof(fds) / sizeof(fds[0])),
    500);
1144
1145            /*超时 timed out - periodic check for _task_should_exit */
1146            if (pret == 0) {
1147                continue;
1148            }
1149
1150            /* 不期望this is undesirable but not much we can do */
1151            if (pret < 0) {
1152                warn("poll error %d, %d", pret, errno);
1153                continue;
1154            }
1155
1156
1157           poll_subscriptions();
1158
1159            /* get current rotation matrix and euler angles from control
    state quaternions */
1160            //得到当前的旋转矩阵 和 欧拉角，从control state中 ，这里面的信息一定
    在上面那么多的更新中更新过
1161            math::Quaternion q_att(_ctrl_state.q[0], _ctrl_state.q[1],
    _ctrl_state.q[2], _ctrl_state.q[3]);
1162            _R = q_att.to_dcm();           //q_att 四元数 _R把四元数转换
    为DCM方向余弦矩阵
1163            math::Vector<3> euler_angles;  //欧拉角 Vector<3> 等效
    于（x，y，z）三维坐标
1164            euler_angles = _R.to_euler();  //将方向余弦矩阵转换为欧拉角
1165            _yaw      = euler_angles(2);    //euler_angles中放着三个欧拉角
    roll pitch yaw，且第三位是yaw，估计这里的顺序应该是
1166
1167            parameters_update(false);
1168
1169            /* 解锁配置
1170             * 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）*/
1171
1172            hrt_abstime t = hrt_absolute_time();//绝对执行时间t，运行到当前时
    间
1173            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
1174            t_prev = t;//更新上一次绝对执行时间
1175
1176            //为控制块设置单位时间  给控制块设置dt(这个是操作系统层的)
1177            setDt(dt);
1178
1179            //初次解锁，重置参考点和积分
1180            if (_control_mode.flag_armed && !was_armed) {//这次解锁 上次没解
    锁，即第一次解锁
```

```
1181                /* 参考点和积分量需要在解锁时重置 */
1182                _reset_pos_sp = true;
1183                _reset_alt_sp = true;
1184                reset_int_z = true;
1185                reset_int_xy = true;
1186                reset_yaw_sp = true;
1187            }
1188
1189            //垂直起降情况，为固定翼重置偏航和姿态参考点（无用）
1190            /* reset yaw and altitude setpoint for VTOL which are in fw mode */
1191            if (_vehicle_status.is_vtol) {
1192                if (!_vehicle_status.is_rotary_wing) {//旋翼为true，四旋翼为true，这里用于固定翼
1193                    reset_yaw_sp = true;
1194                    _reset_alt_sp = true;
1195                }
1196            }
1197
1198            //更新前一次解锁情况  这样下一次就不会 if (_control_mode.flag_armed && !was_armed) 再全部重置
1199            was_armed = _control_mode.flag_armed;
1200
1201
1202            /**参考更新
1203             * 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()，
1204             * 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。 */
1205            //是将当前的位置转换为经纬度和高度吗，后面又是更新？
1206            update_ref();//用本地位置更新的  跟新一些地坐标xyz方向基准值
1207
1208            /**位置和速度控制主函数
1209             *
1210             * 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）。
1211             * （方位控制 和 速度控制，分水平方向的x，y和垂直方向的z。在说明一下这里position指的就是水平面 这里的velocity指的就是水平方向的速度
1212             * 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：*/
1213
1214            //以下的_control_mode.xxxxxx均来自commander.cpp
1215            if (_control_mode.flag_control_altitude_enabled ||
1216                _control_mode.flag_control_position_enabled ||
1217                _control_mode.flag_control_climb_rate_enabled ||
1218                _control_mode.flag_control_velocity_enabled)
1219 /*以下代码都在这里面*/{
1220                //以上任一使能就进行以下位置控制
1221
1222                //可见位置控制的对象有二，就是 pos方位和vel速度，如姿态控制的对
```

```
    象外环是角度 内环是角速度
1223                _pos(0) = _local_pos.x;
1224                _pos(1) = _local_pos.y;  //外环是位置（误差）p
1225                _pos(2) = _local_pos.z;
1226
1227                _vel(0) = _local_pos.vx;
1228                _vel(1) = _local_pos.vy; //内环是速度（误差）pid
1229                _vel(2) = _local_pos.vz;
1230
1231                _vel_ff.zero();//未使用，构造函数中置0
1232
1233                // 默认是位置/高度控制器，也可以直接运行速度控制器
1234                 /* the control_* functions can disable this and run
    velocity controllers directly in this cycle
1235                  * 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。 */
1236                _run_pos_control = true;//位置控制，默认，也可禁用然后直接使用
    速度控制
1237                _run_alt_control = true;//高度控制，默认，也可禁用然后直接使用
    速度控制
1238 //第二部分第一步：产生位置/速度设定值(期望值) 选择控制源是手动、机
    外(offboard)、还是自动控制，产生位置/速度设定值(期望值)
1239                /* select control source */
1240                //手动控制 自动模式去能
1241                if (_control_mode.flag_control_manual_enabled) {
1242
1243                    control_manual(dt);//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。
1244                    //此函数很关键，从遥控器读值并赋给 vel_sp,
1245                    //对于自动控制，例如mission模式下，那么pos_sp及vel_sp的值
    通过_pos_sp_triplet得到
1246                    //对于手动控制，_pos_sp_triplet.current.valid =
    false，_pos_sp_triplet由mission.cpp得到，跟正常的遥控器控制无关
1247                    _mode_auto = false;
1248
1249                }
1250                //离线控制 offboard control
1251                else if (_control_mode.flag_control_offboard_enabled) {
1252                    /*  */
1253                    control_offboard(dt);
1254                    _mode_auto = false;
1255
1256                }
1257                //自动控制 AUTO
1258                else {
1259                    control_auto(dt);
1260                }
1261
1262        //空闲状态 推力为0了 那空闲状态飞机就是趴在地上了
1263                 /* 空闲状态 不运行控制器并 设置推力为0 idle state, don't run
```

```
    controller and set zero thrust
1264                 * 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*/
1265                if (!_control_mode.flag_control_manual_enabled &&
    _pos_sp_triplet.current.valid
1266                    && _pos_sp_triplet.current.type ==
    position_setpoint_s::SETPOINT_TYPE_IDLE)
1267                {
1268                                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
1269                                memcpy(&_att_sp.R_body[0], R.data,
    sizeof(_att_sp.R_body));
1270                                _att_sp.R_valid = true;//姿态参考消息中数据初始
    化—旋转矩阵有效性
1271
1272                                _att_sp.roll_body = 0.0f;//欧拉角和推力
1273                                _att_sp.pitch_body = 0.0f;;
1274                               _att_sp.yaw_body = _yaw;
1275                                _att_sp.thrust = 0.0f;
1276
1277                                _att_sp.timestamp = hrt_absolute_time();//时间
    帧
1278
1279                                /* publish attitude setpoint */
1280                                if (_att_sp_pub != nullptr) {//对于非空数据进行
    发布_attitude_setpoint_id
1281
    orb_publish(ORB_ID(vehicle_attitude_setpoint), _att_sp_pub, &_att_sp);
1282
1283                                }
1284                                else
1285                                {//否则对于有数据的情况进行公
    告_attitude_setpoint_id
1286                                    _att_sp_pub =
    orb_advertise(ORB_ID(vehicle_attitude_setpoint), &_att_sp);
1287                                }
1288                }
1289                /* run position & altitude controllers, if enabled
    (otherwise use already computed velocity setpoints) */
1290
1291        //这里进入正常飞行状态的控制  产生可利用的速度设定值(期望值)  这段程序应该
    才是发布正常飞行状态的姿态设定值  否则采用已经计算出来的速度设定值
1292                //对于定高或定点模式，如果摇杆不在死区，那
    么_run_pos_control及_run_alt_control为false
1293                else
1294                {
1295                    if (_run_pos_control) {//位置控制，默认使用，
1296                    //以下获取速度参考点（ 位置差*P  ） 外环p位置 内环pid速度
1297                    _vel_sp(0) = (_pos_sp(0) - _pos(0)) *
    _params.pos_p(0);
1298                    _vel_sp(1) = (_pos_sp(1) - _pos(1)) *
```

```
    _params.pos_p(1);
1299                    }
1300
1301                    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）
1302                        _vel_sp(2) = (_pos_sp(2) - _pos(2)) *
    _params.pos_p(2);
1303                    }
1304
1305                    /* make sure velocity setpoint is saturated（饱和） in
    xy*/
1306                    float vel_norm_xy = sqrtf(_vel_sp(0) * _vel_sp(0) +
1307                                              _vel_sp(1) * _vel_sp(1));
1308
1309                    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
1310                        /* note assumes vel_max(0) == vel_max(1) */
1311                        _vel_sp(0) = _vel_sp(0)/ vel_norm_xy *
    _params.vel_max(0) ;
1312                        _vel_sp(1) = _vel_sp(1)/ vel_norm_xy*
    _params.vel_max(1);
1313                    }
1314
1315                    /*同样的方式处理垂直方向的速度，保证其在限定范围内 make
    sure velocity setpoint is saturated in z*/
1316                    float vel_norm_z = sqrtf(_vel_sp(2) * _vel_sp(2));
1317
1318                    if (vel_norm_z > _params.vel_max(2)) {
1319                        _vel_sp(2) = _vel_sp(2) * _params.vel_max(2) /
    vel_norm_z;
1320                    }
1321                    //上面为限定速度最大值
1322
1323
1324                    //如果这些模式没有使能 就要把它响应的变量赋值为零 重置
1325                    if (!_control_mode.flag_control_position_enabled) {
1326                        //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）
1327                        _reset_pos_sp = true;
1328                    }
1329
1330                    if (!_control_mode.flag_control_altitude_enabled) {
1331                        //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）
1332                        _reset_alt_sp = true;
1333                    }
1334
1335                    if (!_control_mode.flag_control_velocity_enabled) {
1336                        //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）
```

```
1337                        _vel_sp(0) = 0.0f;
1338                        _vel_sp(1) = 0.0f;
1339                    }
1340
1341                    if (!_control_mode.flag_control_climb_rate_enabled) {
1342                        //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）
1343                        _vel_sp(2) = 0.0f;
1344                    }
1345
1346        //降落
1347                    /* use constant descend rate when landing, ignore altitude setpoint */
1348                    //着陆情况：忽略高度参考点，使用常量作为降落速率
1349                    if (!_control_mode.flag_control_manual_enabled && _pos_sp_triplet.current.valid
1350                        && _pos_sp_triplet.current.type == position_setpoint_s::SETPOINT_TYPE_LAND) {
1351                        //非手动模式&本次参考数据有效&着陆
1352                        _vel_sp(2) = _params.land_speed;
1353                    }
1354
1355                    _global_vel_sp.vx = _vel_sp(0);
1356                    _global_vel_sp.vy = _vel_sp(1);
1357                    _global_vel_sp.vz = _vel_sp(2);
1358
1359                    /* publish velocity setpoint 里发布的姿态设定值是和着陆模式有关的，并不是飞行的姿态设定值 */
1360                    if (_global_vel_sp_pub != nullptr) {
1361
1362                        orb_publish(ORB_ID(vehicle_global_velocity_setpoint), _global_vel_sp_pub, &_global_vel_sp);
1363
1364                    } else {
1365                        _global_vel_sp_pub = orb_advertise(ORB_ID(vehicle_global_velocity_setpoint), &_global_vel_sp);
1366                    }
1367
1368
1369        //爬升率控制|速度控制  这个if范围很大，主要是速度控制
1370                    if (_control_mode.flag_control_climb_rate_enabled || _control_mode.flag_control_velocity_enabled) {
1371
1372                        /* reset integrals（积分） if needed */
1373                        if (_control_mode.flag_control_climb_rate_enabled) {
1374                            //爬升率控制
1375                            if (reset_int_z)
1376                            {
```

```
1377                                    //以下都是重置z轴积分
1378                                        reset_int_z = false;
1379                                        float i = _params.thr_min;//thr_min默认为0.12
1380
1381                                        if (reset_int_z_manual) {//手动重置z积分
1382                                            i = _manual.z;
1383
1384                                            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
1385                                                i = _params.thr_min;
1386
1387                                            }
1388                                            else if (i > _params.thr_max) {
1389                                                i = _params.thr_max;
1390                                            }
1391                                        }
1392
1393                                        thrust_int(2) = -i;//设置z推力积分为-i（朝地面为正）向上为负
1394                                    }
1395
1396                                }
1397                                else {//速度控制，下次进入爬升率控制需要重置z轴积分
1398                                    reset_int_z = true;
1399                                }
1400
1401                                if (_control_mode.flag_control_velocity_enabled) {//速度控制（这是指水平速度）
1402                                    if (reset_int_xy) {//这里对平面速度积分进行重置
1403                                        reset_int_xy = false;
1404                                        thrust_int(0) = 0.0f;
1405                                        thrust_int(1) = 0.0f;
1406                                    }
1407                                    //爬升率控制情况，下次进入速度控制需要重置平面速度
1408                                }
1409                                else {
1410                                    reset_int_xy = true;
1411                                }
1412                               //以上这些都在说thrust_int（3）这个推力积分 向量
1413
1414
1415        /* vel_err velocity error 速度误差*/
1416       //内环了vel_err = _vel_sp - _vel; 环外p位置（误差），内环pid速度（误差），联想姿态控制外环角度误差 内环角速度误差
1417                                //上面有外环p位置误差    _vel_sp(0) = (_pos_sp(0) - _pos(0)) * _params.pos_p(0); 位置误差*P
1418                                //这是内环的速度误差了    vel_err = _vel_sp - _vel;
1419                                //可见sp的setpoint期望值的意思 _pos  _vel是当前的位置和速度
```

```
1420                        math::Vector<3> vel_err = _vel_sp - _vel;//速度误差，参考点-实际
1421
1422                        /* 对速度误差进行微分 derivative of velocity error, /
1423                         * does not includes setpoint acceleration */
1424                        math::Vector<3> vel_err_d;  /* derivative of velocity error */
1425                        vel_err_d(0) = _vel_x_deriv.update(-_vel(0));
1426                        vel_err_d(1) = _vel_y_deriv.update(-_vel(1));
1427                        vel_err_d(2) = _vel_z_deriv.update(-_vel(2));
1428
1429
1430                        /* 获取新的推力参考（NED坐标系） thrust vector in NED frame */
1431                        //利用上面的 vel_err = _vel_sp - _vel 经过PID 产生推力期望值thrust_sp
1432                        math::Vector<3> thrust_sp = vel_err.emult(_params.vel_p) + vel_err_d.emult(_params.vel_d) + thrust_int;//pid
1433
1434                        if (!_control_mode.flag_control_velocity_enabled) {//非速度控制
1435                            thrust_sp(0) = 0.0f;//平面推力参考置0
1436                            thrust_sp(1) = 0.0f;
1437                        }
1438
1439                        if (!_control_mode.flag_control_climb_rate_enabled) {//非爬升率控制
1440                            thrust_sp(2) = 0.0f;//垂直推力参考置0
1441                        }
1442
1443            /* 保证推力在限定范围内 限制推力向量 饱和检查 limit thrust vector and check for saturation */
1444                        bool saturation_xy = false;
1445                        bool saturation_z = false;
1446
1447            //限制最小升力     /* limit min lift */
1448                        float thr_min = _params.thr_min;//推力最小默认为0.12
1449
1450                        if (!_control_mode.flag_control_velocity_enabled && thr_min < 0.0f) {
1451                            //非速度控制&最小推力<0
1452                            /*手动模式下 不允许负的推力 don't allow downside thrust direction in manual attitude mode */
1453                            thr_min = 0.0f;
1454                        }
1455                              //tilt倾斜
1456                        float tilt_max = _params.tilt_max_air;//默认45度
1457
1458          /*为着降落式调整推力 限制 adjust limits for landing mode */
```

```
1459                        if (!_control_mode.flag_control_manual_enabled &&
    _pos_sp_triplet.current.valid &&
1460                            _pos_sp_triplet.current.type ==
    position_setpoint_s::SETPOINT_TYPE_LAND) {
1461                            /* limit max tilt and min lift when landing */
1462                            tilt_max = _params.tilt_max_land;//降落时最大的
    倾斜角
1463
1464                            if (thr_min < 0.0f) { //推力向上为正，推力小于0
    即推力向下了
1465                                thr_min = 0.0f;
1466                            }
1467                        }
1468
1469                        /* 这里限制升力，升力需要大于最小推力（绝对值，方向）
    limit min lift举升 刚好（2）是垂直方向 */
1470                        if (-thrust_sp(2) < thr_min) {
1471                            thrust_sp(2) = -thr_min;
1472                            saturation_z = true;//判断超出范围，饱和
1473                        }
1474
1475
1476
1477
1478                        if (_control_mode.flag_control_velocity_enabled)
1479                        {
1480                            /*限制最大倾斜角 limit max tilt 倾斜*/
1481                            if (thr_min >= 0.0f && tilt_max < M_PI_F / 2 -
    0.05f)
1482                            {
1483                                /* 绝对的 水平推力 absolute horizontal thrust
    */
1484                                float thrust_sp_xy_len =
    math::Vector<2>(thrust_sp(0), thrust_sp(1)).length();
1485
1486                                if (thrust_sp_xy_len > 0.01f)//参考水平推力
    平方和>0.01
1487                                {
1488                                    /* 根据垂直推力来决定水平推力最大值 max
    horizontal thrust for given vertical thrust*/
1489                                    float thrust_xy_max = -thrust_sp(2) *
    tanf(tilt_max);
1490
1491                                    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
1492                                    {
1493                                        float k = thrust_xy_max /
    thrust_sp_xy_len;
1494                                        thrust_sp(0) *= k;
1495                                        thrust_sp(1) *= k;
```

```
1496                                        saturation_xy = true;
1497                                    }
1498                                }
1499                            }
1500
1501                        }
1502                        else//姿态控制
1503                        {
1504                            /*只有姿态控制情况补偿推力 thrust compensation for altitude only control mode */
1505                            float att_comp;//姿态补偿 comp compensation
1506
1507                            if (_R(2, 2) > TILT_COS_MAX) {
1508                                att_comp = 1.0f / _R(2, 2);
1509
1510                            }
1511                            else if (_R(2, 2) > 0.0f) {
1512                                att_comp = ((1.0f / TILT_COS_MAX - 1.0f) / TILT_COS_MAX) * _R(2, 2) + 1.0f;
1513                                saturation_z = true;
1514
1515                            }
1516                            else {//R（2，2）<0情况
1517                                att_comp = 1.0f;
1518                                saturation_z = true;
1519                            }
1520
1521                            thrust_sp(2) *= att_comp;//补偿
1522                        }
1523
1524
1525            /* 限制最大推力 limit max thrust */
1526                        float thrust_abs = thrust_sp.length();
    //math::Vector<3> thrust_sp = vel_err.emult(_params.vel_p) + vel_err_d.emult(_params.vel_d) + thrust_int;
1527                        //参考推力可能改变故重新计算
1528                        //推力饱和 超过了最大值 下面分推力向上 和 推力向下进行讨论
1529                        if (thrust_abs > _params.thr_max) {//0.9，只有过大才限制推力
1530                            if (thrust_sp(2) < 0.0f) //上升  thrust_sp(2)向下为正 向上为负
1531                            {
1532                                    if (-thrust_sp(2) > _params.thr_max)//升力饱和
1533                                    {
1534                                        /* Z轴推力太大 thrust Z component is too large, limit it
1535                                         * 限制水平推力，垂直推力为最大推力值，饱和标志位置true*/
```

```
1536                                            thrust_sp(0) = 0.0f;
1537                                            thrust_sp(1) = 0.0f;
1538                                            thrust_sp(2) = -_params.thr_max;
1539                                            saturation_xy = true;
1540                                            saturation_z = true;
1541
1542                                        }
1543                                        //升力未饱和
1544                                        else
1545                                        {//保持z轴推力 降低XY ， 保持高度 比位置更重要
1546                                            /* preserve thrust Z component and lower XY, keeping altitude is more important than position */
1547                                            float thrust_xy_max = sqrtf(_params.thr_max * _params.thr_max - thrust_sp(2) * thrust_sp(2));
1548                                            float thrust_xy_abs = math::Vector<2>(thrust_sp(0), thrust_sp(1)).length();
1549                                            float k = thrust_xy_max / thrust_xy_abs;
1550                                            thrust_sp(0) *= k;
1551                                            thrust_sp(1) *= k;
1552                                            saturation_xy = true;
1553                                        }
1554
1555                                    }
1556                                    else
1557                                    {//推力向下，简单限制推力向量
1558                                        /* Z component is negative, going down, simply limit thrust vector */
1559                                        float k = _params.thr_max / thrust_abs;//下降的情况直接限制大小（上面对水平限制）
1560                                        thrust_sp *= k;
1561                                        saturation_xy = true;
1562                                        saturation_z = true;
1563                                    }
1564
1565                                    thrust_abs = _params.thr_max;
1566                                }
1567
1568                                /* 更新积分 update integrals */
1569                                if (_control_mode.flag_control_velocity_enabled && !saturation_xy) {
1570                                    //速度控制&xy积分不饱和，不饱和时仍继续累加 （速度控制产生推力向量）
1571                                    thrust_int(0) += vel_err(0) * _params.vel_i(0) * dt;
1572                                    thrust_int(1) += vel_err(1) * _params.vel_i(1) * dt;
1573                                }
1574
```

```
1575                        if (_control_mode.flag_control_climb_rate_enabled
    && !saturation_z) {
1576                            //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
1577                            thrust_int(2) += vel_err(2) * _params.vel_i(2)
    * dt;
1578
1579                            /* 着陆时防止轻弹 protection against flipping on
    ground when landing */
1580                            if (thrust_int(2) > 0.0f) {//垂直方向 推力向
    下，遇到着陆情况可以避免弹跳
1581                                thrust_int(2) = 0.0f;
1582                            }
1583                        }
1584
1585
1586
1587
1588
1589            //根据推力向量计算 姿态期望值，这个期望值后面交给姿态控制，结合姿态估
    计att_sp-_att=角度，角速度 又是一个串级pid控制
1590            /* 从推力向量求 姿态期望值 这个期望姿态给姿态控制的 结合姿态估计又是一
    个串级pid控制过程
1591             *            calculate attitude setpoint from thrust vector */
1592                        if (_control_mode.flag_control_velocity_enabled) {
1593                            /* desired body_z axis =
    -normalize(thrust_vector) */
1594                            math::Vector<3> body_x;  //这里x y z是旋转矩阵
    的x列 y列 z列，下面就有 由他们三个构建旋转矩阵
1595                            math::Vector<3> body_y;
1596                            math::Vector<3> body_z;
1597
1598                            if (thrust_abs > SIGMA) {//#define SIGMA
    0.000001f
1599                                body_z = -thrust_sp / thrust_abs;//归一化
1600
1601                            }
1602                            else {
1603                                /*没有推力的情况，[0 0 1]  no thrust, set Z
    axis to safe value */
1604                                body_z.zero();
1605                                body_z(2) = 1.0f;  //z列【0 0 1】这不是重力
    吗，在没有推力时 还有重力呢
1606                            }
1607
1608                            /* vector of desired yaw direction in XY
    plane, rotated by PI/2 */
1609                            //在xy平面上 偏航方向期望的向量，旋转3.14/2
1610                            math::Vector<3> y_C (-sinf(_att_sp.yaw_body),
    cosf(_att_sp.yaw_body), 0.0f);
```

```
1611
1612                            if (fabsf(body_z(2)) > SIGMA) {//#define SIGMA
    0.000001f
1613                                /* desired body_x axis, orthogonal正交 to
    body_z */
1614                                body_x = y_C % body_z;//叉乘，由z和期望y获得
1615
1616                                /* 方向校正 keep nose to front while
    inverted upside down 保持机头向前当倒转时*/
1617                                if (body_z(2) < 0.0f) {
1618                                    body_x = -body_x;
1619                                }
1620
1621                                body_x.normalize();//归一化
1622
1623                            }
1624                            else {
1625                                /* desired thrust is in XY plane, set X
    downside to construct correct matrix,
1626                                 * but yaw component will not be used
    actually */
1627                                body_x.zero();//z轴为0，不能叉乘，直接用单位
1628                                body_x(2) = 1.0f;
1629                            }
1630
1631                            /* desired body_y axis */
1632                            body_y = body_z % body_x;//叉乘，由z和x求y
1633
1634
1635                            /* fill rotation matrix */
1636                            for (int i = 0; i < 3; i++) {
1637                                R(i, 0) = body_x(i);//获得旋转矩阵
1638                                R(i, 1) = body_y(i);
1639                                R(i, 2) = body_z(i);
1640                            }
1641
1642
1643                            /*更新姿态参考的数据（旋转矩阵及其有效性） copy
    rotation matrix to attitude setpoint topic */
1644                            memcpy(&_att_sp.R_body[0], R.data,
    sizeof(_att_sp.R_body));
1645                            _att_sp.R_valid = true;
1646
1647                            /* 更新四元数参考，并更新到姿态参考中 copy
    quaternion setpoint to attitude setpoint topic */
1648                            math::Quaternion q_sp;
1649                            q_sp.from_dcm(R);
1650                            memcpy(&_att_sp.q_d[0], &q_sp.data[0],
    sizeof(_att_sp.q_d));
1651
```

```
1652                                /*根据旋转矩阵求欧拉角 calculate euler angles,
    for logging only, must not be used for control */
1653                                math::Vector<3> euler = R.to_euler();
1654                                _att_sp.roll_body = euler(0);
1655                                _att_sp.pitch_body = euler(1);
1656                                /* yaw already used to construct rot matrix,
    but actual rotation matrix can have different yaw near singularity */
1657
1658                            }
1659
1660
1661                            else if
    (!_control_mode.flag_control_manual_enabled) {
1662                                //非速度控制&非手动控制
1663                                /*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
1664                                 *  autonomous altitude control without
    position control (failsafe landing),
1665                                 * force level attitude, don't change yaw */
1666                                R.from_euler(0.0f, 0.0f, _att_sp.yaw_body);//根
    据偏航获得旋转矩阵
1667
1668                                /* 更新姿态参考中的旋转矩阵及其有效性 copy
    rotation matrix to attitude setpoint topic */
1669                                memcpy(&_att_sp.R_body[0], R.data,
    sizeof(_att_sp.R_body));
1670                                _att_sp.R_valid = true;
1671
1672                                /*由旋转矩阵获取四元数参考，并更新到姿态参考中
    copy quaternion setpoint to attitude setpoint topic */
1673                                math::Quaternion q_sp;
1674                                q_sp.from_dcm(R);
1675                                memcpy(&_att_sp.q_d[0], &q_sp.data[0],
    sizeof(_att_sp.q_d));
1676
1677                                _att_sp.roll_body = 0.0f;//保持水平
1678                                _att_sp.pitch_body = 0.0f;
1679                            }
1680
1681                            _att_sp.thrust = thrust_abs;//更新姿态期望中的推
    力（推力平方根）
1682
1683                            /*保存数据用于记录，推力->加速度 save thrust setpoint
    for logging */
1684                            _local_pos_sp.acc_x = thrust_sp(0);
1685                            _local_pos_sp.acc_y = thrust_sp(1);
1686                            _local_pos_sp.acc_z = thrust_sp(2);
1687
1688                            _att_sp.timestamp = hrt_absolute_time();
1689
```

```
1690
1691                    }
1692                    else {//爬升率控制|速度控制，非速度控制的情况，重置z积分
1693                        reset_int_z = true;
1694                    }
1695                }
1696
1697            //  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： 这个自己发布 其实又被自己订阅了
1698                /*将得到的数据更新到本地位置参考消息中（之后会发布） fill local position, velocity and thrust setpoint */
1699                //填补本地记录 位置 速度 推力 设定值（手动模式下 这个setpoint来自于遥控器）
1700                _local_pos_sp.timestamp = hrt_absolute_time();
1701                _local_pos_sp.x = _pos_sp(0);
1702                _local_pos_sp.y = _pos_sp(1);
1703                _local_pos_sp.z = _pos_sp(2);
1704                _local_pos_sp.yaw = _att_sp.yaw_body;
1705                _local_pos_sp.vx = _vel_sp(0);
1706                _local_pos_sp.vy = _vel_sp(1);
1707                _local_pos_sp.vz = _vel_sp(2);
1708
1709                /* publish local position setpoint */
1710                if (_local_pos_sp_pub != nullptr) {
1711                    orb_publish(ORB_ID(vehicle_local_position_setpoint), _local_pos_sp_pub, &_local_pos_sp);
1712
1713                } else {
1714                    _local_pos_sp_pub = orb_advertise(ORB_ID(vehicle_local_position_setpoint), &_local_pos_sp);
1715                }
1716                  //vehicle_local_position_setpoint搜这个主题 你就会发现_local_pos_sp自己发布 自己订阅
1717                  //内容是 previous;current; next; /**< vehicle global position setpoint triplet */
1718            }
1719
1720
1721
1722
1723
1724
1725
1726
1727            //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：
1728            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
1729                /* 不能使用位置控制器的情况，需要重置参考点 position controller disabled, reset setpoints */
```

```
1730                //位置控制器禁用，重置 设定值
1731                _reset_alt_sp = true;
1732                _reset_pos_sp = true;
1733                _mode_auto = false;
1734                reset_int_z = true;
1735                reset_int_xy = true;
1736            }
1737
1738
1739            //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：
1740            //并列于“高度控制、位置控制、爬升速率控制、速度控制”，混控
1741
1742            /*由手动控制获得姿态参考点 generate attitude setpoint from manual
    controls 手动/定高/定点通用，根据摇杆量计算出期望的旋转矩阵 */
1743                if (_control_mode.flag_control_manual_enabled &&
    _control_mode.flag_control_attitude_enabled)
1744                {
1745                    //手动控制&姿态控制
1746
1747                    //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：其他情况中着陆状态不改变偏航：
1748
1749                    /*偏航参考重置有效，更新姿态参考中的偏航 reset yaw
    setpoint to current position if needed */
1750                    if (reset_yaw_sp) {
1751                        reset_yaw_sp = false;
1752                        _att_sp.yaw_body = _yaw;
1753                    }
1754
1755                    /* do not move yaw while arming */
1756                    else if (_manual.z > 0.1f) {
1757                        const float yaw_offset_max = _params.man_yaw_max /
    _params.mc_att_yaw_p;
1758
1759                        _att_sp.yaw_sp_move_rate = _manual.r *
    _params.man_yaw_max;
1760                        float yaw_target = _wrap_pi(_att_sp.yaw_body +
    _att_sp.yaw_sp_move_rate * dt);
1761                        float yaw_offs = _wrap_pi(yaw_target - _yaw);
1762
1763                        // If the yaw offset became too big for the system
    to track stop如果偏航偏移变得太大以至于跟踪停止
1764                        // shifting it改变
1765                        if (fabsf(yaw_offs) < yaw_offset_max) {
1766                            _att_sp.yaw_body = yaw_target;
1767                        }
1768                    }
1769
1770                    /* 定高或手动直接通过遥控器给出roll/pitch control roll
    and pitch directly if we no aiding velocity controller is active */
```

```
1771                    if (!_control_mode.flag_control_velocity_enabled) {
1772                        _att_sp.roll_body = _manual.y *
    _params.man_roll_max;
1773                        _att_sp.pitch_body = -_manual.x *
    _params.man_pitch_max;
1774                    }
1775
1776                    /* 手动模式直接通过遥控器给推力 control throttle directly
    if no climb rate controller is active */
1777                    if (!_control_mode.flag_control_climb_rate_enabled)
1778                    {
1779                        _att_sp.thrust = math::min(_manual.z,
    _manual_thr_max.get());
1780
1781                        /* enforce minimum throttle if not landed */
1782                        if (!_vehicle_status.condition_landed) {
1783                            _att_sp.thrust = math::max(_att_sp.thrust,
    _manual_thr_min.get());
1784                        }
1785                    }
1786
1787                    /* construct attitude setpoint rotation matrix */
1788                    math::Matrix<3, 3> R_sp;
1789                    // 此处情况比较复杂，感觉写的不清楚，有些地方容易混淆，分三
    种看
1790                    //定点模式：之前已经通过三个方向的thrust_sp计算得到
    了R_sp,之后只需要计算出yaw之后的R_sp即可
1791                    //定高模式：通过摇杆量得到roll+pitch+yaw，计算得到R_sp，推
    力值通过之前的逻辑得到
1792                    //纯手动模式：通过摇杆得到roll + pitch +yaw +thrust，计算
    得到R_sp
1793
1794                    R_sp.from_euler(_att_sp.roll_body, _att_sp.pitch_body,
    _att_sp.yaw_body);
1795                    memcpy(&_att_sp.R_body[0], R_sp.data,
    sizeof(_att_sp.R_body));
1796
1797                    /* copy quaternion setpoint to attitude setpoint topic
    */
1798                    math::Quaternion q_sp;
1799                    q_sp.from_dcm(R_sp);
1800                    memcpy(&_att_sp.q_d[0], &q_sp.data[0],
    sizeof(_att_sp.q_d));
1801                    _att_sp.timestamp = hrt_absolute_time();
1802
1803            }
1804            else
1805            {
1806                reset_yaw_sp = true;
1807            }
```

```
1808
1809            /* update previous velocity for velocity controller D part */
1810            _vel_prev = _vel;
1811
1812            /* 此处的 att_sp.R_body发布后，会被mc_attitude_control调用
    为R_sp，期望的旋转矩阵
1813             * publish attitude setpoint
1814             * Do not publish if offboard is enabled but position/velocity
    control is disabled,
1815             * in this case the attitude setpoint is published by the
    mavlink app
1816             */
1817            if (!(_control_mode.flag_control_offboard_enabled &&
1818                  !(_control_mode.flag_control_position_enabled ||
1819                    _control_mode.flag_control_velocity_enabled)))
1820            {
1821                if (_att_sp_pub != nullptr &&
    (_vehicle_status.is_rotary_wing || _vehicle_status.in_transition_mode)) {
1822                    orb_publish(ORB_ID(vehicle_attitude_setpoint),
    _att_sp_pub, &_att_sp);
1823
1824                }
1825                else if (_att_sp_pub == nullptr &&
    (_vehicle_status.is_rotary_wing || _vehicle_status.in_transition_mode)) {
1826                    _att_sp_pub =
    orb_advertise(ORB_ID(vehicle_attitude_setpoint), &_att_sp);
1827                }
1828            }
1829
1830            /* reset altitude controller integral (hovering throttle) to
    manual throttle after manual throttle control */
1831            reset_int_z_manual = _control_mode.flag_armed &&
    _control_mode.flag_control_manual_enabled
1832                                 &&
    !_control_mode.flag_control_climb_rate_enabled;
1833 /*while结束*/
1834     }
1835
1836     mavlink_log_info(_mavlink_fd, "[mpc] stopped");
1837
1838     _control_task = -1;
1839 }
1840
1841 int
1842 MulticopterPositionControl::start()
1843 {
1844     ASSERT(_control_task == -1);
1845
1846     /* start the task */
1847     _control_task = px4_task_spawn_cmd("mc_pos_control",
```

```
1848                        SCHED_DEFAULT,
1849                        SCHED_PRIORITY_MAX - 5,
1850                        1500,
1851
    (px4_main_t)&MulticopterPositionControl::task_main_trampoline,
1852                        nullptr);
1853
1854    if (_control_task < 0) {
1855        warn("task start failed");
1856        return -errno;
1857    }
1858
1859    return OK;
1860}
1861
1862int mc_pos_control_main(int argc, char *argv[])
1863{
1864    if (argc < 2) {
1865        warnx("usage: mc_pos_control {start|stop|status}");
1866        return 1;
1867    }
1868
1869    if (!strcmp(argv[1], "start")) {
1870
1871        if (pos_control::g_control != nullptr) {
1872            warnx("already running");
1873            return 1;
1874        }
1875
1876        pos_control::g_control = new MulticopterPositionControl;
1877
1878        if (pos_control::g_control == nullptr) {
1879            warnx("alloc failed");
1880            return 1;
1881        }
1882
1883        if (OK != pos_control::g_control->start()) {
1884            delete pos_control::g_control;
1885            pos_control::g_control = nullptr;
1886            warnx("start failed");
1887            return 1;
1888        }
1889
1890        return 0;
1891    }
1892
1893    if (!strcmp(argv[1], "stop")) {
1894        if (pos_control::g_control == nullptr) {
1895            warnx("not running");
1896            return 1;
```

```
1897        }
1898
1899        delete pos_control::g_control;
1900        pos_control::g_control = nullptr;
1901        return 0;
1902    }
1903
1904    if (!strcmp(argv[1], "status")) {
1905        if (pos_control::g_control) {
1906            warnx("running");
1907            return 0;
1908
1909        } else {
1910            warnx("not running");
1911            return 1;
1912        }
1913    }
1914
1915    warnx("unrecognized command");
1916    return 1;
1917}
1918
```