```cpp
/****************************************************************************
 *
 *   Better
 *   QQ: 1751300722
 *
 ****************************************************************************/
//难得就是飞行时的稳定性和对震动和噪声的处理问题，这些都是最细节最重要的部分， 也是最难的部分。
/*
 * @file attitude_estimator_q_main.cpp
 *
 * Attitude estimator (quaternion based)
 *
 * @author Anton Babushkin <anton.babushkin@me.com>
 */

#include <px4_config.h>
#include <px4_posix.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <poll.h>
#include <fcntl.h>
#include <float.h>
#include <errno.h>
#include <limits.h>
#include <math.h>
#include <uORB/uORB.h>
#include <uORB/topics/sensor_combined.h>
#include <uORB/topics/vehicle_attitude.h>
#include <uORB/topics/control_state.h>
#include <uORB/topics/vehicle_control_mode.h>
#include <uORB/topics/vehicle_global_position.h>
#include <uORB/topics/vision_position_estimate.h>
#include <uORB/topics/att_pos_mocap.h>
#include <uORB/topics/parameter_update.h>
#include <drivers/drv_hrt.h>

#include <mathlib/mathlib.h>
#include <mathlib/math/filter/LowPassFilter2p.hpp>
#include <lib/geo/geo.h>
#include <lib/ecl/validation/data_validator_group.h>
#include <mavlink/mavlink_log.h>

#include <systemlib/systemlib.h>
#include <systemlib/param/param.h>
#include <systemlib/perf_counter.h>
#include <systemlib/err.h>

extern "C" __EXPORT int attitude_estimator_q_main(int argc, char *argv[]);

using math::Vector;
using math::Matrix;
using math::Quaternion;

class AttitudeEstimatorQ;

namespace attitude_estimator_q
{
AttitudeEstimatorQ *instance;
}


class AttitudeEstimatorQ
{
public:
	/**
	 * Constructor
	 */
	AttitudeEstimatorQ();
```

```
    /**
     * Destructor, also kills task.
     */
    ~AttitudeEstimatorQ();

    /**
     * Start task.
     *
     * @return      OK on success.
     */
    int     start();

    static void task_main_trampoline(int argc, char *argv[]);

    void        task_main();

    void        print();

private:
    static constexpr float _dt_max = 0.02;
    bool        _task_should_exit = false;      /**< if true, task should exit */
    int     _control_task = -1;         /**< task handle for task */

    int     _sensors_sub = -1;
    int     _params_sub = -1;
    int     _vision_sub = -1;
    int     _mocap_sub = -1;
    int     _global_pos_sub = -1;
    orb_advert_t    _att_pub = nullptr;
    orb_advert_t    _ctrl_state_pub = nullptr;

    struct {
        param_t w_acc;
        param_t w_mag;
        param_t w_ext_hdg;
        param_t w_gyro_bias;
        param_t mag_decl;
        param_t mag_decl_auto;
        param_t acc_comp;
        param_t bias_max;
        param_t vibe_thresh;
        param_t ext_hdg_mode;
    }       _params_handles;        /**< handles for interesting parameters */

    float       _w_accel = 0.0f;
    float       _w_mag = 0.0f;
    float       _w_ext_hdg = 0.0f;
    float       _w_gyro_bias = 0.0f;
    float       _mag_decl = 0.0f;
    bool        _mag_decl_auto = false;
    bool        _acc_comp = false;
    float       _bias_max = 0.0f;
    float       _vibration_warning_threshold = 1.0f;
    hrt_abstime _vibration_warning_timestamp = 0;
    int     _ext_hdg_mode = 0;

    Vector<3>   _gyro;
    Vector<3>   _accel;
    Vector<3>   _mag;

    vision_position_estimate_s _vision = {};
    Vector<3>   _vision_hdg;

    att_pos_mocap_s _mocap = {};
    Vector<3>   _mocap_hdg;

    Quaternion  _q;
    Vector<3>   _rates;
    Vector<3>   _gyro_bias;

    vehicle_global_position_s _gpos = {};
```

```
    Vector<3>    _vel_prev;
    Vector<3>    _pos_acc;

    DataValidatorGroup _voter_gyro;
    DataValidatorGroup _voter_accel;
    DataValidatorGroup _voter_mag;

    /* Low pass filter for attitude rates */
    math::LowPassFilter2p _lp_roll_rate;
    math::LowPassFilter2p _lp_pitch_rate;
    math::LowPassFilter2p _lp_yaw_rate;

    hrt_abstime _vel_prev_t = 0;

    bool         _inited = false;
    bool         _data_good = false;
    bool         _failsafe = false;
    bool         _vibration_warning = false;
    bool         _ext_hdg_good = false;

    int      _mavlink_fd = -1;

    perf_counter_t _update_perf;
    perf_counter_t _loop_perf;

    void update_parameters(bool force);

    int update_subscriptions();

    bool init();

    bool update(float dt);
};


AttitudeEstimatorQ::AttitudeEstimatorQ() :
    _voter_gyro(3),
    _voter_accel(3),
    _voter_mag(3),
    _lp_roll_rate(250.0f, 18.0f),
    _lp_pitch_rate(250.0f, 18.0f),
    _lp_yaw_rate(250.0f, 10.0f)
{
    _voter_mag.set_timeout(200000);

    _params_handles.w_acc        = param_find("ATT_W_ACC");
    _params_handles.w_mag        = param_find("ATT_W_MAG");
    _params_handles.w_ext_hdg    = param_find("ATT_W_EXT_HDG");
    _params_handles.w_gyro_bias = param_find("ATT_W_GYRO_BIAS");
    _params_handles.mag_decl     = param_find("ATT_MAG_DECL");
    _params_handles.mag_decl_auto    = param_find("ATT_MAG_DECL_A");
    _params_handles.acc_comp     = param_find("ATT_ACC_COMP");
    _params_handles.bias_max     = param_find("ATT_BIAS_MAX");
    _params_handles.vibe_thresh = param_find("ATT_VIBE_THRESH");
    _params_handles.ext_hdg_mode     = param_find("ATT_EXT_HDG_M");
}

/**
 * Destructor, also kills task.
 */
AttitudeEstimatorQ::~AttitudeEstimatorQ()
{
    if (_control_task != -1) {
        /* task wakes up every 100ms or so at the longest */
        _task_should_exit = true;

        /* wait for a second for the task to quit at our request */
        unsigned i = 0;

        do {
            /* wait 20ms */
```

```
            usleep(20000);

            /* if we have given up, kill it */
            if (++i > 50) {
                px4_task_delete(_control_task);
                break;
            }
        } while (_control_task != -1);
    }

    attitude_estimator_q::instance = nullptr;
}

int AttitudeEstimatorQ::start()
{
    ASSERT(_control_task == -1);

    /* start the task */
    _control_task = px4_task_spawn_cmd("attitude_estimator_q",
                       SCHED_DEFAULT,
                       SCHED_PRIORITY_MAX - 5,
                       2100,
                       (px4_main_t)&AttitudeEstimatorQ::task_main_trampoline,
                       nullptr);

    if (_control_task < 0) {
        warn("task start failed");
        return -errno;
    }

    return OK;
}

void AttitudeEstimatorQ::print()
{
    warnx("gyro status:");
    _voter_gyro.print();
    warnx("accel status:");
    _voter_accel.print();
    warnx("mag status:");
    _voter_mag.print();
}

void AttitudeEstimatorQ::task_main_trampoline(int argc, char *argv[])
{
    attitude_estimator_q::instance->task_main();
}

void AttitudeEstimatorQ::task_main()
{
    _sensors_sub = orb_subscribe(ORB_ID(sensor_combined));

    _vision_sub = orb_subscribe(ORB_ID(vision_position_estimate));
    _mocap_sub = orb_subscribe(ORB_ID(att_pos_mocap));

    _params_sub = orb_subscribe(ORB_ID(parameter_update));
    _global_pos_sub = orb_subscribe(ORB_ID(vehicle_global_position));

    update_parameters(true);

    hrt_abstime last_time = 0;

    px4_pollfd_struct_t fds[1];
    fds[0].fd = _sensors_sub;
    fds[0].events = POLLIN;

    while (!_task_should_exit) {
        int ret = px4_poll(fds, 1, 1000);

        if (_mavlink_fd < 0) {
            _mavlink_fd = open(MAVLINK_LOG_DEVICE, 0);
```

```
        }

        if (ret < 0) {
            // Poll error, sleep and try again
            usleep(10000);
            continue;

        } else if (ret == 0) {
            // Poll timeout, do nothing
            continue;
        }

        update_parameters(false);

        // Update sensors
        sensor_combined_s sensors;//先有个印象 sensors下面都会用到 是一个很大的结构体
        里面放的都是传感器相关的数据
//!orb_copy开始，对数据做处理 选最优
        if (!orb_copy(ORB_ID(sensor_combined), _sensors_sub, &sensors)) {//如果不成功
        就读取数据
            // Feed validator with recent sensor data
            for (unsigned i = 0; i < (sizeof(sensors.gyro_timestamp) /
            sizeof(sensors.gyro_timestamp[0])); i++)
            {
                //i三次，下面分别读取 陀螺仪 加速度 磁力计      // uint64_t
                gyro_timestamp[3]; 这里就是i<3

                    if (sensors.gyro_timestamp[i] > 0) {

                        float gyro[3];

                        for (unsigned j = 0; j < 3; j++) {
                            if (sensors.gyro_integral_dt[i] > 0) {
                                gyro[j] = (double)sensors.gyro_integral_rad[i * 3 + j] /
                                (sensors.gyro_integral_dt[i] / 1e6);
                                //把陀螺仪的值先积分然后再微分，这个其实就是求平均
                                这样更稳更可靠
                            } else {
                                /* fall back to angular rate */
                                gyro[j] = sensors.gyro_rad_s[i * 3 + j];
                                //为什么这里i都要乘以3，我猜测
                                这里都是【9】，就是每个传感器都有三组数据，然后用put函数对三组
                                数据选最优，那数据的首地址分别是 0 3 6，注意这里传的是地址
                            }                                       //gyro[j]
                            是gyro[3];i=0 赋值【0，1，2】 i=1时赋值【3，4，5】.。。
                        }
                        //那errcount[i],_priority[i]就应该放的三组数据的比较结果
                        //上面读了i=0时gyro[0，1，2]三组数据 一共读了9组数据
                        这个put函数应该是对这些所取的数据做处理的
                        _voter_gyro.put(i, sensors.gyro_timestamp[i], &gyro[0],
                        sensors.gyro_errcount[i], sensors.gyro_priority[i]);
                    }   //执行了3次 这种函数还是实例化分析 现在假设 i=0时

                    /* ignore empty fields */
                    if (sensors.accelerometer_timestamp[i] > 0) {
                        _voter_accel.put(i, sensors.accelerometer_timestamp[i],
                        &sensors.accelerometer_m_s2[i * 3],
                                 sensors.accelerometer_errcount[i],
                                 sensors.accelerometer_priority[i]);
                    }

                    /* ignore empty fields */
                    if (sensors.magnetometer_timestamp[i] > 0) {
                        _voter_mag.put(i, sensors.magnetometer_timestamp[i],
                        &sensors.magnetometer_ga[i * 3],
                                   sensors.magnetometer_errcount[i],
                                   sensors.magnetometer_priority[i]);
                    }
            //如果数据的copy不成功，取陀螺仪 加速度
            磁力计的数据，且取三组，i=0，1，2分别是三组数据。将这三组数据传入put函数
            进行最优的选择，errcount[i] priority[i]这里放的应该就是put所做处理的标志
```

```
    //那put函数都在做什么 对这三组数据做处理 平均数 方差 标准差 滤波等
    //下面的get_best函数应该就是根据这些标志 什么方差 标准差，选择最可靠的数据
    get_best()函数的最后调用该结果(通过比较三组数据的confidence大小决定是否选取)
    }

    int best_gyro, best_accel, best_mag;

    // 得到最好的测量数据 Get best measurement values
    hrt_abstime curr_time = hrt_absolute_time();
    _gyro.set(_voter_gyro.get_best(curr_time, &best_gyro));
    _accel.set(_voter_accel.get_best(curr_time, &best_accel));
    _mag.set(_voter_mag.get_best(curr_time, &best_mag));

    if (_accel.length() < 0.01f || _mag.length() < 0.01f) {
        warnx("WARNING: degenerate accel / mag!");//degenerate 恶化 变质
        就是数据有问题 可能硬件有问题
        continue;
    }

    _data_good = true;

    if (!_failsafe) {
        uint32_t flags = DataValidator::ERROR_FLAG_NO_ERROR;

                //根据_voter_gyro、_voter_accel、_voter_mag三个参数的failover_count函数
                判断是否存在数据获取失误问题，并通过mavlink协议显示错误原因。
            if (_voter_gyro.failover_count() > 0) {
                _failsafe = true;
                flags = _voter_gyro.failover_state();
                mavlink_and_console_log_emergency(_mavlink_fd, "Gyro #%i failure
                :%s%s%s%s%s!",
                                  _voter_gyro.failover_index(),
                                  ((flags & DataValidator::ERROR_FLAG_NO_DATA) ? "
                                  No data" : ""),
                                  ((flags & DataValidator::ERROR_FLAG_STALE_DATA) ?
                                  " Stale data" : ""),
                                  ((flags & DataValidator::ERROR_FLAG_TIMEOUT) ? "
                                  Data timeout" : ""),
                                  ((flags & DataValidator::ERROR_FLAG_HIGH_ERRCOUNT)
                                  ? " High error count" : ""),
                                  ((flags &
                                  DataValidator::ERROR_FLAG_HIGH_ERRDENSITY) ? "
                                  High error density" : ""));
            }

            if (_voter_accel.failover_count() > 0) {
                _failsafe = true;
                flags = _voter_accel.failover_state();
                mavlink_and_console_log_emergency(_mavlink_fd, "Accel #%i failure
                :%s%s%s%s%s!",
                                  _voter_accel.failover_index(),
                                  ((flags & DataValidator::ERROR_FLAG_NO_DATA) ? "
                                  No data" : ""),
                                  ((flags & DataValidator::ERROR_FLAG_STALE_DATA) ?
                                  " Stale data" : ""),
                                  ((flags & DataValidator::ERROR_FLAG_TIMEOUT) ? "
                                  Data timeout" : ""),
                                  ((flags & DataValidator::ERROR_FLAG_HIGH_ERRCOUNT)
                                  ? " High error count" : ""),
                                  ((flags &
                                  DataValidator::ERROR_FLAG_HIGH_ERRDENSITY) ? "
                                  High error density" : ""));
            }

            if (_voter_mag.failover_count() > 0) {
                _failsafe = true;
                flags = _voter_mag.failover_state();
                mavlink_and_console_log_emergency(_mavlink_fd, "Mag #%i failure
                :%s%s%s%s%s!",
                                  _voter_mag.failover_index(),
                                  ((flags & DataValidator::ERROR_FLAG_NO_DATA) ? "
```

```
                                          No data" : ""),
                                          ((flags & DataValidator::ERROR_FLAG_STALE_DATA) ?
                                          " Stale data" : ""),
                                          ((flags & DataValidator::ERROR_FLAG_TIMEOUT) ? "
                                          Data timeout" : ""),
                                          ((flags & DataValidator::ERROR_FLAG_HIGH_ERRCOUNT)
                                          ? " High error count" : ""),
                                          ((flags &
                                          DataValidator::ERROR_FLAG_HIGH_ERRDENSITY) ? "
                                          High error density" : ""));
                    }

                    if (_failsafe) {//如果上面检测真的出了问题 立即要求返回着陆
                        mavlink_and_console_log_emergency(_mavlink_fd, "SENSOR FAILSAFE!
                        RETURN TO LAND IMMEDIATELY");
                    }                                                      //传感器故障, 立即着陆
            }

                //根据_voter_gyro、_voter_accel、_voter_mag三个参数的get_vibration_factor函数
                判断是否有震动现象, 返回值是float型的RSM值, 其代表振动的幅度大小。
            if (!_vibration_warning && (_voter_gyro.get_vibration_factor(curr_time) >
            _vibration_warning_threshold ||
                                _voter_accel.get_vibration_factor(curr_time) >
                                _vibration_warning_threshold ||
                                _voter_mag.get_vibration_factor(curr_time) >
                                _vibration_warning_threshold))
            {

                    if (_vibration_warning_timestamp == 0) {
                        _vibration_warning_timestamp = curr_time;

                    } else if (hrt_elapsed_time(&_vibration_warning_timestamp) > 10000000) {
                        _vibration_warning = true;
                        mavlink_and_console_log_critical(_mavlink_fd, "HIGH VIBRATION! g: %d
                        a: %d m: %d",
                                                 (int)(100 *
                                                 _voter_gyro.get_vibration_factor(curr_time)),
                                                 (int)(100 *
                                                 _voter_accel.get_vibration_factor(curr_time)),
                                                 (int)(100 *
                                                 _voter_mag.get_vibration_factor(curr_time)));
                    }

            }
            else {
                _vibration_warning_timestamp = 0;
            }
        }
//!orb_copy结束, 上面是!orb_copy不成功时 每个传感器去三组数据 用put函数对三组数据进行处理
求他们的方差 标准差
滤波, 然后用get_best函数取最优的一组数据。最后的一部分就是根据数据看传感器是否正常 抖动

        //更新视觉 和 捕捉航向  Update vision and motion capture heading航向
        bool vision_updated = false;
        orb_check(_vision_sub, &vision_updated);

        bool mocap_updated = false;
        orb_check(_mocap_sub, &mocap_updated);

        if (vision_updated) //如果视觉更新  拷出视觉信息 , 视觉也是一个结构体里面有很多信息
        包括想x,y,x,vx,vy,vz,q[4] 就是根据视觉也可以获取位置信息
        {                   //下面就是根据视觉 来更新姿态
                orb_copy(ORB_ID(vision_position_estimate), _vision_sub, &_vision);
                math::Quaternion q(_vision.q);

                math::Matrix<3, 3> Rvis = q.to_dcm();  // R-vision 基于视觉 得到的转换矩阵
                math::Vector<3> v(1.0f, 0.0f, 0.4f);

                // Rvis is Rwr (robot respect to world) while v is respect to world.
                // Hence Rvis must be transposed having (Rwr)' * Vw
                // Rrw * Vw = vn. This way we have consistency
```

```
            _vision_hdg = Rvis.transposed() * v;//transposed转置 因为R是标准正交阵
            转置=逆  原来由地理->机体, 转置后就是 机体->地理
        }                                       //猜测应该类似与重力的处理 *【0 0
        1】, 无关怎么转换 最后得到的hdg应该就是用此种传感器校准的误差

        if (mocap_updated)
        {
            orb_copy(ORB_ID(att_pos_mocap), _mocap_sub, &_mocap);
            math::Quaternion q(_mocap.q);
            math::Matrix<3, 3> Rmoc = q.to_dcm(); //R-mocap 基于动作捕捉 得到的转换矩阵

            math::Vector<3> v(1.0f, 0.0f, 0.4f);

            // Rmoc is Rwr (robot respect to world) while v is respect to world.
            // Hence Rmoc must be transposed having (Rwr)' * Vw
            // Rrw * Vw = vn. This way we have consistency
            _mocap_hdg = Rmoc.transposed() * v;
        }

        // Check for timeouts on data
        if (_ext_hdg_mode == 1) {
            _ext_hdg_good = _vision.timestamp_boot > 0 &&
            (hrt_elapsed_time(&_vision.timestamp_boot) < 500000);

        } else if (_ext_hdg_mode == 2) {
            _ext_hdg_good = _mocap.timestamp_boot > 0 &&
            (hrt_elapsed_time(&_mocap.timestamp_boot) < 500000);
        }

        bool gpos_updated;//global position 全球位置
        orb_check(_global_pos_sub, &gpos_updated);

        if (gpos_updated) {//如果位置已经更新 就取出位置数据
            orb_copy(ORB_ID(vehicle_global_position), _global_pos_sub, &_gpos);

            if (_mag_decl_auto && _gpos.eph < 20.0f && hrt_elapsed_time(&_gpos.timestamp) <
            1000000) {
                /* set magnetic declination automatically 自动获取磁偏角 因为不同重力
                不同位置磁偏角不同 根据经度和维度 自动获取磁偏角*/
                _mag_decl = math::radians(get_mag_declination(_gpos.lat,
                _gpos.lon));//latitude纬度 longitude经度
            }
        }

        if (_acc_comp && _gpos.timestamp != 0 && hrt_absolute_time() < _gpos.timestamp +
        20000 && _gpos.eph < 5.0f && _inited)
        {
            /* 实际的位置数据 在 北 东 地 坐标系下 position data is actual */
            if (gpos_updated) {
                Vector<3> vel(_gpos.vel_n, _gpos.vel_e, _gpos.vel_d);// 在 北 东 地
                坐标系下 的速度 下面根据它求加速度

                /* velocity updated */
                if (_vel_prev_t != 0 && _gpos.timestamp != _vel_prev_t) {
                    float vel_dt = (_gpos.timestamp - _vel_prev_t) / 1000000.0f;
                    /* 计算载体坐标系上的加速度calculate acceleration in body frame */
                    _pos_acc = _q.conjugate_inversed((vel - _vel_prev) / vel_dt);
                }

                _vel_prev_t = _gpos.timestamp;
                _vel_prev = vel;
            }

        } else {
            /* position data is outdated, reset acceleration */
            _pos_acc.zero();
            _vel_prev.zero();
            _vel_prev_t = 0;
        }

        // Time from previous iteration
```

```
	hrt_abstime now = hrt_absolute_time();
	float dt = (last_time > 0) ? ((now  - last_time) / 1000000.0f) : 0.00001f;
	last_time = now;

	if (dt > _dt_max) {
		dt = _dt_max;
	}
	/*! ! update 就是姿态更新的函数，先利用视觉 mcap 加速度 磁力计
	修正陀螺仪，再利用四元数的微分方程 实时更新解算姿态信息，
	此函数后就是得到更新后的姿态信息了  */
	if (!update(dt)) {
		continue;
	}

	Vector<3> euler =
	_q.to_euler();//用更新的四元数（_q）求出欧拉角，以便在控制过程中实现完美的控制，控制还
	是需要用直接明了的欧拉角。

	struct vehicle_attitude_s att = {};
	att.timestamp = sensors.timestamp;

	att.roll = euler(0); //获取的欧拉角赋值给roll、pitch、yaw
	att.pitch = euler(1);
	att.yaw = euler(2);

	att.rollspeed = _rates(0); //获取roll、pitch、yaw得旋转速度
	att.pitchspeed = _rates(1);
	att.yawspeed = _rates(2);

	for (int i = 0; i < 3; i++) {
		att.g_comp[i] = _accel(i) -
		_pos_acc(i);////获取导航坐标系的重力加速度，前面介绍过
		加速度测量值-运动加速度= 重力加速度
	}

	//下面就是对更新后的姿态信息 进行重新写出系统 方便下一次的使用
	/* copy offsets */
	memcpy(&att.rate_offsets, _gyro_bias.data, sizeof(att.rate_offsets));

	Matrix<3, 3> R = _q.to_dcm();

	/* copy rotation matrix */
	memcpy(&att.R[0], R.data, sizeof(att.R));
	att.R_valid = true;

	att.rate_vibration = _voter_gyro.get_vibration_factor(hrt_absolute_time());
	att.accel_vibration = _voter_accel.get_vibration_factor(hrt_absolute_time());
	att.mag_vibration = _voter_mag.get_vibration_factor(hrt_absolute_time());

	if (_att_pub == nullptr) {
		_att_pub = orb_advertise(ORB_ID(vehicle_attitude), &att);

	} else {
		orb_publish(ORB_ID(vehicle_attitude), _att_pub, &att);
	}

	struct control_state_s ctrl_state = {};

	ctrl_state.timestamp = sensors.timestamp;

	/* Attitude quaternions for control state */
	ctrl_state.q[0] = _q(0);

	ctrl_state.q[1] = _q(1);

	ctrl_state.q[2] = _q(2);

	ctrl_state.q[3] = _q(3);

	/* Attitude rates for control state */
	ctrl_state.roll_rate = _lp_roll_rate.apply(_rates(0));
```

```
        ctrl_state.pitch_rate = _lp_pitch_rate.apply(_rates(1));

        ctrl_state.yaw_rate = _lp_yaw_rate.apply(_rates(2));

        /* Publish to control state topic */
        if (_ctrl_state_pub == nullptr) {
            _ctrl_state_pub = orb_advertise(ORB_ID(control_state), &ctrl_state);

        } else {
            orb_publish(ORB_ID(control_state), _ctrl_state_pub, &ctrl_state);
        }
    }
}

void AttitudeEstimatorQ::update_parameters(bool force)
{
    bool updated = force;

    if (!updated) {
        orb_check(_params_sub, &updated);
    }

    if (updated) {
        parameter_update_s param_update;
        orb_copy(ORB_ID(parameter_update), _params_sub, &param_update);

        param_get(_params_handles.w_acc, &_w_accel);
        param_get(_params_handles.w_mag, &_w_mag);
        param_get(_params_handles.w_ext_hdg, &_w_ext_hdg);
        param_get(_params_handles.w_gyro_bias, &_w_gyro_bias);
        float mag_decl_deg = 0.0f;
        param_get(_params_handles.mag_decl, &mag_decl_deg);
        _mag_decl = math::radians(mag_decl_deg);
        int32_t mag_decl_auto_int;
        param_get(_params_handles.mag_decl_auto, &mag_decl_auto_int);
        _mag_decl_auto = mag_decl_auto_int != 0;
        int32_t acc_comp_int;
        param_get(_params_handles.acc_comp, &acc_comp_int);
        _acc_comp = acc_comp_int != 0;
        param_get(_params_handles.bias_max, &_bias_max);
        param_get(_params_handles.vibe_thresh, &_vibration_warning_threshold);
        param_get(_params_handles.ext_hdg_mode, &_ext_hdg_mode);
    }
}

//构建一个初始的四元数，用四元数来表示姿态时 通过微分方程实时更新解算姿态
但是这种求解也是需要一个初值啊，至于是行是列不重要，这里加速度的测量值作为第三行
说明这个四元数是由 载体转换到地理，如果是第三列还记得之前的[0 0
1]g得到的第三列吗，这里行不就是转置吗
//i= （_mag - k * (_mag * k))之所以这么复杂 只是在强制使k i叉乘为零，相互垂直。
// k= -_accel
然后归一化k，k为加速度传感器测量到加速度方向向量，由于第一次测量数据时无人机一般为平稳状态无运
动状态，所以可以直接将测到的加速度视为重力加速度g，以此作为dcm旋转矩阵的第三行k（这个介绍过了
）。
// i= （_mag - k * (_mag * k)) _mag向量指向正北方，k*(_mag*k)
正交correction值，对于最终的四元数归一化以后的范数可以在正负5%以内；感觉不如《DCM
IMU:Theory》中提出的理论“强制正交化”修正的好，Renormalization算法在ardupilot的上层应用AP_AHRS_
DCM中使用到了。
// j= k%i ：外积、叉乘。关于上面的Vector<3>k =
-_accel，Vector<3>相当于一个类型（int）定义一个变量k，然后把-_accel负值给k，在定义_accel时也是
使用Vector<3>，属于同一种类型的，主要就是为了考虑其实例化过程（类似函数重载）。
bool AttitudeEstimatorQ::init()
{
    // Rotation matrix can be easily constructed from acceleration and mag field vectors
    // 'k' is Earth Z axis (Down) unit vector in body frame
    Vector<3> k = -_accel;
    k.normalize();

    // 'i' is Earth X axis (North) unit vector in body frame, orthogonal with 'k'
    Vector<3> i = (_mag - k * (_mag * k));
```

```
    i.normalize();

    // 'j' is Earth Y axis (East) unit vector in body frame, orthogonal with 'k' and 'i'
    Vector<3> j = k % i;

    // Fill rotation matrix
    Matrix<3, 3> R;
    R.set_row(0, i);
    R.set_row(1, j);
    R.set_row(2, k);

    // Convert to quaternion
    _q.from_dcm(R);
    _q.normalize();

    if (PX4_ISFINITE(_q(0)) && PX4_ISFINITE(_q(1)) &&
        PX4_ISFINITE(_q(2)) && PX4_ISFINITE(_q(3)) &&
        _q.length() > 0.95f && _q.length() < 1.05f) {
        _inited = true;

    } else {
        _inited = false;
    }

    return _inited;
}

bool AttitudeEstimatorQ::update(float dt) //用于对四元数向量_q进行初始化赋值 或者 更新。
{
    if (!_inited) {// 首先判断是否是第一次进入该函数，第一次进入该函数先进入init函数初始化

        if (!_data_good) {
            return false;
        }

        return init();
    }

    //如果不是第一次进入该函数，则判断是使用什么mode做修正的，比如vision、mocap、acc和mag（DJI
    精灵4用的视觉壁障应该就是这个vision），Hdg就是heading。
    Quaternion q_last = _q;//这里_q就是init()初始化得到的那个 是由 载体->地理坐标系

    // Angular rate of correction校准
    Vector<3> corr;
    //corr包含磁力计修正、加速度计修正、（vision、mocap修正）、gyro中测量到的角速度偏转量，

                //且因为corr为update函数中定义的变量，所以每次进入update函数中时会刷新corr
                变量的数据； _rate也会刷新其中的数据，含义为三个姿态角的角速度（修正后）；

                //_q为外部定义的变量，在这个函数中只会+=不会重新赋值，如果计算出现错误会返
                回上一次计算出的_q。

    if (_ext_hdg_mode > 0 && _ext_hdg_good) {
        //  _ext_hdg_mode==
        1、2时都是利用vision数据和mocap数据对gyro数据进行修正，下面的global
        frame就是所谓的earthframe。
        if (_ext_hdg_mode == 1) {
            // Vision heading correction
            // Project heading to global frame and extract XY component
            Vector<3> vision_hdg_earth = _q.conjugate(_vision_hdg);
            //_q.conjugate先b系到n系，后conjugate_inversedn系到b系，再补回到b系
            float vision_hdg_err = _wrap_pi(atan2f(vision_hdg_earth(1), vision_hdg_earth(0)));
            // Project correction to body frame
            corr += _q.conjugate_inversed(Vector<3>(0.0f, 0.0f, -vision_hdg_err)) *
            _w_ext_hdg;
        }

        if (_ext_hdg_mode == 2) {
            // Mocap heading correction
            // Project heading to global frame and extract XY component
            Vector<3> mocap_hdg_earth = _q.conjugate(_mocap_hdg);
```

```
        float mocap_hdg_err = _wrap_pi(atan2f(mocap_hdg_earth(1), mocap_hdg_earth(0)));
        // Project correction to body frame
        corr += _q.conjugate_inversed(Vector<3>(0.0f, 0.0f, -mocap_hdg_err)) * _w_ext_hdg;
    }   //计算corr值等于单位化的旋转矩阵R（b系转n系）的转置（可以理解为
    R（n系转b系））乘以（0,0, -mag_err），相当于机体坐标系绕地理坐标系N轴（Z轴）转动arctan
    （mag_earth(1), mag_earth(0)）度。
}

if (_ext_hdg_mode == 0  || !_ext_hdg_good) {
    //_ext_hdg_mode== 0利用磁力计修正。 Magnetometer磁力计 correction
    // Project mag field vector to global frame and extract XY component
    Vector<3> mag_earth = _q.conjugate(_mag); //b系到n系
    float mag_err = _wrap_pi(atan2f(mag_earth(1), mag_earth(0)) -
    _mag_decl);//只考虑Vector<3>
    mag_earth中的前两维的数据mag_earth(1)和mag_earth(0)（即x、y，忽略z轴上的偏移），
                                                        //通过arctan得到的角度和前
                                                        面根据经纬度获取的磁偏角做
                                                        差值得到纠偏误差角度mag_er
                                                        r
                                                        ，_wrap_pi函数是用于限定结
                                                        果-pi到pi的函数
    // Project magnetometer correction to body frame
    corr += _q.conjugate_inversed(Vector<3>(0.0f, 0.0f, -mag_err)) *
    _w_mag;//_w_mag为mag的权重
}           ////n系到b系

// 加速度修正，上面都是用来修正航向的 z轴，但是还有x y靠谁修正，Accelerometer correction
// Project 'k' unit vector of earth frame to body frame
// Vector<3> k = _q.conjugate_inversed(Vector<3>(0.0f, 0.0f, 1.0f));
// 把归一化的n系重力加速度通过旋转矩阵R左乘旋转到b系  Vector<3> k =
_q.conjugate_inversed(Vector<3>(0.0f, 0.0f, 1.0f)); R(n->b) 乘以（0,0, 1）g
Vector<3> k(
    2.0f * (_q(1) * _q(3) - _q(0) * _q(2)),
    2.0f * (_q(2) * _q(3) + _q(0) * _q(1)),
    (_q(0) * _q(0) - _q(1) * _q(1) - _q(2) * _q(2) + _q(3) * _q(3))
);

corr += (k % (_accel - _pos_acc).normalized()) * _w_accel;//w开头都是权重
//这就是mahony算法中的的计算过程，只是换了个包装
{k%（_accel"加速度计的测量值"-位移加速度）的单位化）<约等于重力加速度g>}*权重。
这里考虑了运动加速度，减去它更可靠，之前都忽略了。（加速度计测量的是
重力加速度+运动加速度）

//总加速度（加速度获取）减去机体运动加速度（第五部分）获取重力加速度，然后姿态矩阵的不是行
就是列来与纯重力加速度来做叉积，算出误差。因为运动加速度是有害的干扰，必须减掉。算法的理论
基础是[0,0,1]与姿态矩阵相乘。该差值获取的重力加速度的方向是导航坐标系下的z轴，加上运动加速
度之后，总加速度的方向就不是与导航坐标系的天或地平行了，所以要消除这个误差，即"_accel-_pos
_acc"。
//然后叉乘z轴向量得到误差，进行校准

// 陀螺仪误差估计 Gyro bias estimation
_gyro_bias += _w_gyro_bias * corr *
dt;//PI控制器中的I（积分）效果。然后对_gyro_bias做约束处理。 _w开头都是weigh都是权重
这里类似与KI的效果

for (int i = 0; i < 3; i++) {
    _gyro_bias(i) = math::constrain(_gyro_bias(i), -_bias_max, _bias_max);
}

_rates = _gyro + _gyro_bias;//角速度 = 陀螺仪的测量值 + 误差校准

// 前馈 Feed forward gyro
corr += _rates;

//最后就是使用修正的数据更新四元数，并把_rates和_gyro_bias置零便于下次调用时使用。
// 对状态进行修正，实际上 就是关键的姿态更新 Apply correction to state
_q += _q.derivative(corr) * dt;
    //
    非常重要，又用到了微分方程离散化的思想。以前讲过DCM矩阵更新过程中也是用到了该思想。
    //
```

```
        先看看代码，有点怪，怪就怪在derivative（衍生物）这个名字上，平时一大推的论文和期刊上面
        都是用的omga *Q 的形式，而这里的代码实现确是用的Q *
        omga的形式，所以构造的4*4矩阵的每一列的符号就不一样了。
        //http://blog.csdn.net/qq_21842557/article/details/51058206最后
    _q.normalize();

    if (!(PX4_ISFINITE(_q(0)) && PX4_ISFINITE(_q(1)) &&
          PX4_ISFINITE(_q(2)) && PX4_ISFINITE(_q(3)))) {
        // Reset quaternion to last good state
        _q = q_last;
        _rates.zero();
        _gyro_bias.zero();
        return false;
    }

    return true;
}


int attitude_estimator_q_main(int argc, char *argv[])
{
    if (argc < 1) {
        warnx("usage: attitude_estimator_q {start|stop|status}");
        return 1;
    }

    if (!strcmp(argv[1], "start")) {

        if (attitude_estimator_q::instance != nullptr) {
            warnx("already running");
            return 1;
        }

        attitude_estimator_q::instance = new AttitudeEstimatorQ;

        if (attitude_estimator_q::instance == nullptr) {
            warnx("alloc failed");
            return 1;
        }

        if (OK != attitude_estimator_q::instance->start()) {
            delete attitude_estimator_q::instance;
            attitude_estimator_q::instance = nullptr;
            warnx("start failed");
            return 1;
        }

        return 0;
    }

    if (!strcmp(argv[1], "stop")) {
        if (attitude_estimator_q::instance == nullptr) {
            warnx("not running");
            return 1;
        }

        delete attitude_estimator_q::instance;
        attitude_estimator_q::instance = nullptr;
        return 0;
    }

    if (!strcmp(argv[1], "status")) {
        if (attitude_estimator_q::instance) {
            attitude_estimator_q::instance->print();
            warnx("running");
            return 0;

        } else {
            warnx("not running");
            return 1;
        }
```

```
    }

    warnx("unrecognized command");
    return 1;
}
```