Contents

px4: 代码结构清晰, 比较适合开发 APM: 结构比较乱, 功能比较丰富, 稳定性好启动的时候需要sd卡里的一段引导程序，你要是把sd卡拔下来了飞控就启动不了了！还有就是是记录飞行日志和地形跟随的一些数据

第一章　flyMode

1. 1 fly modes

Table 1.1: fly modes

MANUAL	
ACRO	
ALTCTL	
POSCTL	
OFFBOARD	
STABILIZED	
RATTITUDE	
AUTO.MISSION	
AUTO.LOITER	
AUTO.RTL	
AUTO.LAND	
AUTO.RTGS	
AUTO.READY	
AUTO.TAKEOFF	

1. 2 takeOff

1. 3 mission

1. 4 landing

1.5 offboard

1. 6 ALTCTL

参数 NAV_RCL_ACT, 影响着最后的 Failsafe enabled: no RC 的错误警报.

- $NAV\_RCL\_ACT = 0$: stop the fail safe behavior due to having no RC

第二章 uorb

2. 1 Single uorb

2. 1. 1 advertise

2. 1. 2 publisher

2. 1. 3 subscriber

2. 2 Multiple uorb

2. 2. 1 advertise

2. 2. 2 publisher

2. 2. 3 subscriber

第三章　navigator

3. 1 Waypoints List

3. 1. 1 enu Waypoints List

该文本存放关于enu航点列表的描述

3. 1. 2 wgs84 Waypoints list

3. 2 Waypoints Changed Logic

3. 2. 1 normals to normals

3. 2. 2 normals to loiters

3. 2. 3 loiters to loiters

3. 2. 4 loiters to normals

3. 3 Publishing date to position controller

第四章　position controller

4. 1 Single vehicle

4. 1. 1 offboard control logic

4. 1. 2 vf control logic

4. 2 Multiple vehicles

4. 2. 1 fomation control logic

4. 3 Publishing date to attitude controller

第五章 attitude controller

5. 1 offboard

5. 1. 1 mavros发送offboard数据流

```
1     // 1. offboard publish (mavros topic)
2     fixed_wing_local_att_sp_pub = nh.advertise<mavros_msgs::AttitudeTarget>("
          mavros/setpoint_raw/attitude", 10);
3
4     #define MAVLINK_MSG_ID_SET_ATTITUDE_TARGET 82
5       typedef struct __mavlink_set_attitude_target_t {
6           uint32_t time_boot_ms; /*< [ms] Timestamp (time since system boot).*/
7           float q[4]; /*< Attitude quaternion (w, x, y, z order, zero-rotation is
                1, 0, 0, 0)*/
8           float body_roll_rate; /*< [rad/s] Body roll rate*/
9           float body_pitch_rate; /*< [rad/s] Body pitch rate*/
10          float body_yaw_rate; /*< [rad/s] Body yaw rate*/
11          float thrust; /*< Collective thrust, normalized to 0 .. 1 (-1 .. 1 for
                vehicles capable of reverse trust)*/
12          uint8_t target_system; /*< System ID*/
13          uint8_t target_component; /*< Component ID*/
14          uint8_t type_mask; /*< Mappings: If any of these bits are set, the
                corresponding input should be ignored: bit 1: body roll rate, bit 2:
                 body pitch rate, bit 3: body yaw rate. bit 4-bit 6: reserved, bit
                7: throttle, bit 8: attitude*/
15    }) mavlink_set_attitude_target_t;
```

publish messgae

5. 2 Coordinated Turn 协调转弯

5. 2. 1 not being wind

方向角的变化率是和机体的roll以及倾斜角(bank angle)有关系, 我们需要寻找一个简单的关系来帮助我们研究这种线性传递函数的关系 – 协调转弯.

在协调转弯期间, 飞机在体坐标系下没有横向加速度. 从分析的角度来看, 协调转弯的一个假设运行我们得到一个简单的表达式将 heading rate 和 bank angle 联系起来.

协调转弯时, 为了无人机没有侧向力, bank angle ϕ 被设置. 在图5.1中, 作用在微型飞行器上的离心力与作用在水平方向上的升力的水平分量相等并相反。

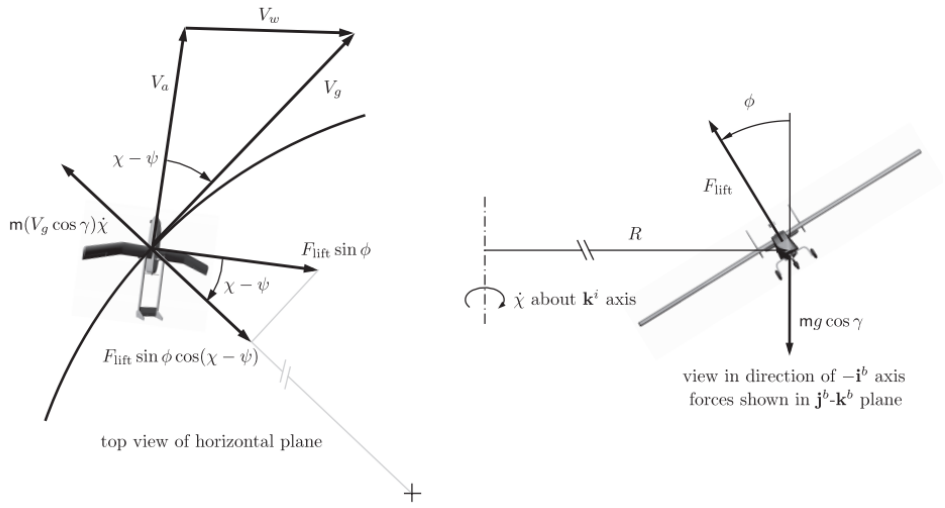

Figure 5.1 Free-body diagrams indicating forces on a MAV in a climbing coordinated turn.

图 5.1. 爬升协调转弯MAV上的力

作用在水平方向力的关系表示如下:

$$\begin{aligned} F_{lift} sin\phi cos(\chi - \psi) &= m\frac{v^2}{R} \\ &= mv\omega \\ &= m(V_g cos\gamma)\dot{\chi} \end{aligned} \tag{5.1}$$

其中, F_{lift}代表的是升力, γ 代表的是飞行轨迹的角度, V_g, χ 分别表示的是地速度以及方向角. 向心加速度的表达式: $a_n = \frac{v^2}{R} = v\omega$

离心力(The centrifugal force)($m(V_g cos\gamma)\dot{\chi}$)计算的时候, 用到了在惯性坐标系$k^i$上的方向角变化率$\dot{\chi}$和速度的水平分量 $V_a cos\gamma$

同样, 升力的垂直分量与重力在 $j^b - k^b$平面上的投影是等大反向的. 垂直方向上的合力为:

$$F_{lift} cos\phi = mgcos\gamma \tag{5.2}$$

将等式5.1除以5.2得的 $\dot{\chi}$

$$\dot{\chi} = \frac{g}{V_g} tan\phi cos(\chi - \psi) \tag{5.3}$$

等式5.3就是协调转弯的表达式.

考虑到转弯半径等于 $R = V_g \frac{cos\gamma}{\dot{\chi}}$, 将上式代入半径中, 得到式子5.4. 在没有风或侧滑的情况下, 有$V_a = V_g$和$\psi = \chi$, 从而得到了式子5.5.

$$R = \frac{V_g^2 cos\gamma}{g tan\phi cos(\chi - \psi)} \tag{5.4}$$

$$\dot{\chi} = \frac{g}{V_g} tan\phi = \dot{\psi} = \frac{g}{V_a} tan\phi \tag{5.5}$$

在 9.2 节中, 我们将要介绍在有风的情况下 $\dot{\psi} = \frac{g}{V_a} tan\phi$ 该式子也成立

5. 2. 2 being wind-Kinematic Model of Controlled Flight

在推导降阶表达式中, 简化的目的是估计运动中力平衡以及动量平衡的关系式(这些包含了 $\dot{u},\dot{v},\dot{\omega},\dot{p},,\dot{q},\dot{r}$), 预估这些变量需要计算复杂的空气动力. 这些变量表达式可以被更简单的动力学表达式替代. 这个更简单的动力学表达式是针对协调转弯和加速爬升的特定飞行条件而导出. 针对图5.2, 飞机相对于惯性系的速

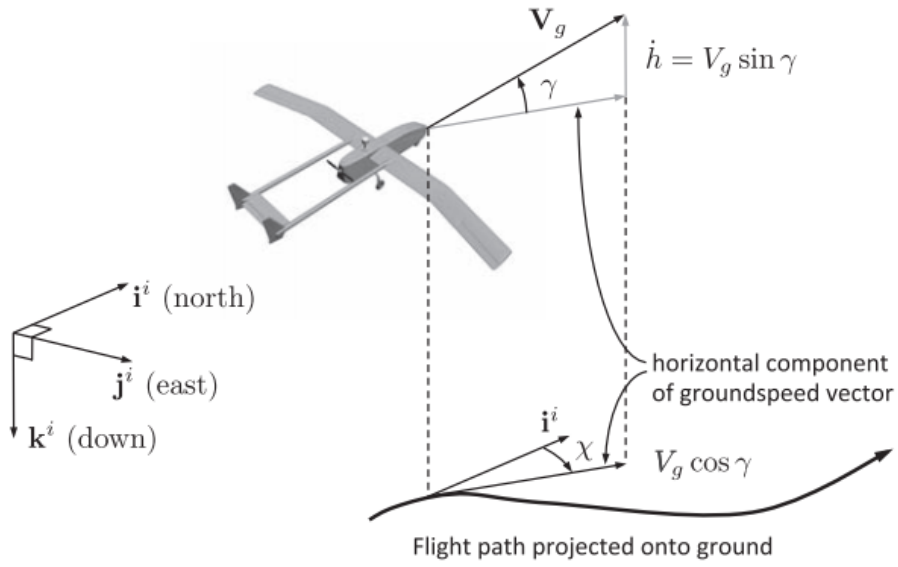

Figure 2.10 The flight path angle γ and the course angle χ.

图 5.2. 航线轨迹角度γ和航向角χ

度矢量可以用航向角和(惯性参考)飞行路径角表示为

$$V_g^i = V_g \begin{pmatrix} cos\chi cos\gamma \\ sin\chi cos\gamma \\ -sin\gamma \end{pmatrix} = \begin{pmatrix} \dot{p_n} \\ \dot{p_e} \\ \dot{h} \end{pmatrix} \tag{5.6}$$

由于控制飞机的航向和空速是很常见的，因此用ψ和V_a表示等式5.6很有用.

$$V_g \begin{pmatrix} cos\chi cos\gamma \\ sin\chi cos\gamma \\ -sin\gamma \end{pmatrix} - \begin{pmatrix} w_n \\ w_e \\ w_d \end{pmatrix} = V_a \begin{pmatrix} cos\psi cos\gamma_a \\ sin\psi cos\gamma_a \\ -sin\gamma_a \end{pmatrix} \tag{5.7}$$

结合风的表达式5.7(地速等于空速加风速, 其中的 γ_a 代表的是空速的方向和水平方向的夹角), 我们可以得到

$$\begin{pmatrix} \dot{p_n} \\ \dot{p_e} \\ \dot{h} \end{pmatrix} = V_a \begin{pmatrix} cos\psi cos\gamma_a \\ sin\psi cos\gamma_a \\ sin\gamma_a \end{pmatrix} + \begin{pmatrix} w_n \\ w_e \\ -w_d \end{pmatrix} \tag{5.8}$$

如果我们假设飞机保持在一个恒定的高度，并且没有向下的风分量，那么运动学表达式简化为5.9, 同样

该模型也是无人机领域中比较常用的模型.

$$\begin{pmatrix} \dot{p_n} \\ \dot{p_e} \\ \dot{h} \end{pmatrix} = V_a \begin{pmatrix} cos\psi \\ sin\psi \\ 0 \end{pmatrix} + \begin{pmatrix} w_n \\ w_e \\ 0 \end{pmatrix} \tag{5.9}$$

5. 2. 3 Coordinated Turn

之前的协调转弯的表达式为 $\dot{\chi} = \frac{g}{V_g} tan\phi cos(\chi - \psi)$. 即使在第6章中描述的自动驾驶回路并没有强制执行协调转弯条件，飞机必须倾斜才能转弯(而不是打滑才能转弯)这个基本条件已经被这个模型捕捉到了。

协调转弯可以被 heading 和空速进行表示. 我们先对5.7两边进行求导, 得到下面的式子5.10

$$\begin{pmatrix} cos\chi cos\gamma & -V_g sin\chi cos\gamma & -V_g cos\chi sin\gamma \\ sin\chi cos\gamma & V_g cos\chi cos\gamma & -V_g sin\chi sin\gamma \\ -sin\gamma & 0 & -cos\gamma \end{pmatrix} \begin{pmatrix} \dot{V_g} \\ \dot{\chi} \\ \dot{\gamma} \end{pmatrix} = \begin{pmatrix} cos\psi cos\gamma_a & -V_a sin\psi cos\gamma_a & -V_a cos\psi sin\gamma_a \\ sin\psi cos\gamma_a & V_a cos\psi cos\gamma_a & -V_a sin\psi sin\gamma_a \\ -sin\gamma_a & 0 & -cos\gamma_a \end{pmatrix} \begin{pmatrix} \dot{V_a} \\ \dot{\psi} \\ \dot{\gamma_a} \end{pmatrix} \tag{5.10}$$

在定高和没有向下风分量的情况下, $\gamma, \gamma_a, \dot{\gamma}, \dot{\gamma_a}$ 和 w_d 都是0, 根据$\dot{V_a}$ 和$\dot{\chi}$求解$\dot{V_g}$ 和$\dot{\psi}$

$$\begin{aligned} \dot{V_g} &= \frac{\dot{V_a}}{cos(\chi - \psi)} + V_g \dot{\chi} tan(\chi - \psi) \\ \dot{\psi} &= \frac{\dot{V_a}}{V_a} tan(\chi - \psi) + \frac{V_g \dot{\chi}}{V_a cos(\chi - \psi)} \end{aligned} \tag{5.11}$$

若假定空速为常数, 那么得5.12 最值得注意的是在有风的情况下，这个等式是成立的。

$$\dot{\chi} = \frac{g}{V_g} tan\phi \tag{5.12}$$

5. 2. 4 px4内部的实现

第一次处理产生5.13, 得到$roll_{constrained}$, 之后在对其进行$(-roll_{setpoint}, roll_{setpoint})$约束. 得出5.14, 进而进行 PID 控制, 产生5.15.

$$roll_{constrained} = \begin{cases} constrained[-80^o, 80^o], & fabs(roll_{current} < 90^o) \\ constrained[100^o, 180^o], & fabs(roll_{current} > 90^o) \& roll_{current} > 0^o \\ constrained[-180^o, -100^o], & fabs(roll_{current} > 90^o) \& roll_{current} < 0^o \end{cases} \tag{5.13}$$

$$roll_{constrained} = roll_{constrained}.constrained[-roll_{setpoint}, roll_{setpoint}] \tag{5.14}$$

$$\begin{aligned} \dot{yaw} &= \frac{tan(roll_{constrained}) * cos(pitch_{current}) * G}{V_{air}}, (V_{air} = V_{air} < V_{air}^{min} ? V_{air}^{min} : V_{air}) \\ \dot{roll} &= \frac{roll_{setpoint} - roll_{current}}{0.1} \\ \dot{pitch} &= \frac{pitch_{setpoint} - pitch_{current}}{0.1} \end{aligned} \tag{5.15}$$

在第一次计算的基础上, 续进行第二手我们继计算, 在px4内部, 首先实现进行参数的设置(见下面的"参数设置"),

$$\begin{aligned} fw\_acro\_x\_max &= 90^o \\ fw\_acro\_y\_max &= 90^o \\ fw\_acro\_z\_max &= 45^o \end{aligned} \tag{5.16}$$

```
_roll_ctrl.set_max_rate(radians(_param_fw_acro_x_max.get()));
_pitch_ctrl.set_max_rate_pos(radians(_param_fw_acro_y_max.get()));
_pitch_ctrl.set_max_rate_neg(radians(_param_fw_acro_y_max.get()));
_yaw_ctrl.set_max_rate(radians(_param_fw_acro_z_max.get()));
```

参数设置

Name	Description	Min > Max (Incr.)	Default	Units
FW_ACRO_X_MAX (FLOAT)	Acro body x max rate **Comment:** This is the rate the controller is trying to achieve if the user applies full roll stick input in acro mode.	45 > 720	90	degrees

(a) x

FW_ACRO_Y_MAX (FLOAT)	Acro body y max rate **Comment:** This is the body y rate the controller is trying to achieve if the user applies full pitch stick input in acro mode.	45 > 720	90	degrees
FW_ACRO_Z_MAX (FLOAT)	Acro body z max rate **Comment:** This is the body z rate the controller is trying to achieve if the user applies full yaw stick input in acro mode.	10 > 180	45	degrees

(b) y and z

图 5.3. parameters

其中各个参数的描述见5.3, 在px4中分别对应的值为5.16. 实现了各个参数的初始化之后, 进行下面的处理

$$\begin{aligned} roll_{bodyrateSetpoint} &= [\dot{roll} - sin(pitch_{current}) * \dot{yaw}] \\ &\quad .constrained[-FW\_ACRO\_X\_MAX, FW\_ACRO\_X\_MAX] \\ pitch_{bodyrateSetpoint} &= [cos(roll_{current}) * \dot{roll} + cos(pitch_{current}) * sin(roll_{current}) * \dot{yaw}] \\ &\quad .constrained[-FW\_ACRO\_Y\_MAX, FW\_ACRO\_Y\_MAX] \\ yaw_{bodyrateSetpoint} &= [-sin(roll_{current}) * \dot{pitch} + cos(roll_{current}) * cos(pitch_{current}) * \dot{yaw}] \\ &\quad .constrained[-FW\_ACRO\_Z\_MAX, FW\_ACRO\_Z\_MAX] \end{aligned} \tag{5.17}$$

最终将上面约束过的体变化率当做姿态的设定值, 发布给下面的控制器以及执行器

$$\begin{aligned} roll_{setpoint} &= roll_{bodyrateSetpoint} \\ pitch_{setpoint} &= pitch_{bodyrateSetpoint} \\ yaw_{setpoint} &= yaw_{bodyrateSetpoint} \end{aligned} \tag{5.18}$$

5.2.5 欧拉角, 四元数的相互转换

欧拉角转换为四元数

```
void euler_2_quaternion(float angle[3], float quat[4])
```

```
2   {
3       // q0 q1 q2 q3
4       // w x y z
5       double cosPhi_2 = cos(double(angle[0]) / 2.0);
6       double sinPhi_2 = sin(double(angle[0]) / 2.0);
7       double cosTheta_2 = cos(double(angle[1]) / 2.0);
8       double sinTheta_2 = sin(double(angle[1]) / 2.0);
9       double cosPsi_2 = cos(double(angle[2]) / 2.0);
10      double sinPsi_2 = sin(double(angle[2]) / 2.0);
11
12      quat[0] = float(cosPhi_2 * cosTheta_2 * cosPsi_2 + sinPhi_2 * sinTheta_2 *
            sinPsi_2);
13      quat[1] = float(sinPhi_2 * cosTheta_2 * cosPsi_2 - cosPhi_2 * sinTheta_2 *
            sinPsi_2);
14      quat[2] = float(cosPhi_2 * sinTheta_2 * cosPsi_2 + sinPhi_2 * cosTheta_2 *
            sinPsi_2);
15      quat[3] = float(cosPhi_2 * cosTheta_2 * sinPsi_2 - sinPhi_2 * sinTheta_2 *
            cosPsi_2);
16  }
```

欧拉角转换为四元数

四元数转换为欧拉角

```
1   Quaternion(const Euler<Type> &euler)
2     {
3         Quaternion &q = *this;
4
5         Type cosPhi_2 = Type(cos(euler.phi() / Type(2)));
6         Type cosTheta_2 = Type(cos(euler.theta() / Type(2)));
7         Type cosPsi_2 = Type(cos(euler.psi() / Type(2)));
8         Type sinPhi_2 = Type(sin(euler.phi() / Type(2)));
9         Type sinTheta_2 = Type(sin(euler.theta() / Type(2)));
10        Type sinPsi_2 = Type(sin(euler.psi() / Type(2)));
11        q(0) = cosPhi_2 * cosTheta_2 * cosPsi_2 +
12               sinPhi_2 * sinTheta_2 * sinPsi_2;
13        q(1) = sinPhi_2 * cosTheta_2 * cosPsi_2 -
14               cosPhi_2 * sinTheta_2 * sinPsi_2;
15        q(2) = cosPhi_2 * sinTheta_2 * cosPsi_2 +
16               sinPhi_2 * cosTheta_2 * sinPsi_2;
17        q(3) = cosPhi_2 * cosTheta_2 * sinPsi_2 -
18               sinPhi_2 * sinTheta_2 * cosPsi_2;
19  }
```

四元数转换为欧拉角

5.2.6 针对offboard对px4的更改

在 *fw_att_control.cpp* 控制器中, 对yaw重新计算的时候, 使用的是 $roll_{setpoint}$ 而不是根据当前的roll进行两次约束之后得到的结果. 同时也去掉$cos(pitch_{current})$这一项.

5. 3 RC control

5. 3. 1 raspberry's handled logic and px4's receiver

树莓派的控制指令以mavlink消息($MAVLINK_MSG_ID_RC_CHANNELS_OVERRIDE = 70$)进入到px4内部, 处理函数名字为:

```
// Firmware/src/modules/mavlink/mavlink_receiver.cpp
void MavlinkReceiver::handle_message(mavlink_message_t *msg);
```

mavlink处理函数声明体

处理函数内部会对信息进行解码处理，得到18个通道(channels)的值，并且赋值给 *input_rc_s* rc变量，之后进行有效性的处理(判断值18个通道的值是否为65535或者是0，若是，则该通道的值变为0；反之，该通道的值不变，且更新*rc.channel_count*的值), 处理之后，使用发布器 *_rc_pub* 将该变量发布到$ORB_ID(input_rc)$话题上(定义发布器的时候，会设置到PublicationMulti机制，实例化会有一个优先级的赋值，可以参见需要补充) 树莓派发送的四个值占用前四个通道，这个四个通道对应的顺序和遥控器对应指令的通道是不一样的 $(roll = 1, pitch = 2, throttle = 3, yaw = 4)$.

```
struct input_rc_s {
    uint64_t timestamp;
    uint64_t timestamp_last_signal;
    uint32_t channel_count;
    int32_t rssi;
    uint16_t rc_lost_frame_count;
    uint16_t rc_total_frame_count;
    uint16_t rc_ppm_frame_length;
    uint16_t values[18];
    bool rc_failsafe;
    bool rc_lost;
    uint8_t input_source;
    uint8_t _padding0[3]; // required for logger
}
// publisher declaration
uORB::PublicationMulti<input_rc_s>      _rc_pub{ORB_ID(input_rc), ORB_PRIO_LOW
    };
```

*input_rc_s*结构体的定义

```
// raspberry
#define PITCH_CHANNEL      1
#define ROLL_CHANNEL       2
#define YAW_CHANNEL        3
#define THROTTLE_CHANNEL   4

// px4
#define PITCH_CHANNEL      1
#define ROLL_CHANNEL       2
#define YAW_CHANNEL        3
#define THROTTLE_CHANNEL   4
```

树莓派以及固件通道定义

5.3.2 其他部分的订阅

mavlink接收函数拿到数据之后，进行一些处理之后，将数据publish出去。在*rc_update*中进行订阅(multi publish 及其 subscribe 机制在后续文章中进行讲解)，订阅器定义如下：

```
// src/modules/sensors/rc_update.h
uORB::Subscription  _rc_sub{ORB_ID(input_rc)};        /**< raw rc channels
    data subscription */
```

订阅器的声明定义

在源文件中*rc_poll api*中进行更新获取该变量的值，再进行一些逻辑标志位的检索，最后对其进行第二次的有效性检测(将每个通道的值约束到某一个固定的区间，类似constrained()函数; trim操作, 准备数据为manual id publish做准备)，再一次更新元素的值. 更新后的值处理.

(1) 发布到 ORB_ID(rc_channels) 话题上面.

(2) 将 struct manual_control_setpoint_s manual = 数据进行约束到[-1.0f, 1.0f]之间更新到该变量中,继而发布到 ORB_ID(manual_control_setpoint) 消息上.

(3) 将第二次发布得到的 manual 变量, 赋值给 actuator_group_3.control 数组, 发布到ORB_ID(actuator_controls_3)消息上.

我们真正关心的消息topic应该是(1, 2), 也就是 $ORB\_ID(rc\_channels)$ 和 $ORB\_ID(manual\_control\_setpoint)$, 其中 $ORB\_ID(rc\_channels)$ 会被作用到PWM(遥控器和接收机的通信方式)上,所以,我们只需要关心的就是 $ORB\_ID(manual\_control\_setpoint)$ 消息topic即可.

上述变量及其函数定义如下:

```
void RCUpdate::rc_poll(const ParameterHandles &parameter_handles);


rc_channels_s _rc {};     /**< r/c channel data */
orb_publish_auto(ORB_ID(rc_channels), &_rc_pub, &_rc, &instance,
    ORB_PRIO_DEFAULT);


struct manual_control_setpoint_s manual = {};
orb_publish_auto(ORB_ID(manual_control_setpoint), &_manual_control_pub, &
    manual, &instance,
            ORB_PRIO_HIGH);


/* copy from mapped manual control to control group 3 */
struct actuator_controls_s actuator_group_3 = {};
/* publish actuator_controls_3 topic */
orb_publish_auto(ORB_ID(actuator_controls_3), &_actuator_group_3_pub, &
    actuator_group_3, &instance,
             ORB_PRIO_DEFAULT);
```

上述变量及其函数定义

5.3.3 ORB_ID(manual_control_setpoint)

这个uORB消息, 在px4内部会被 FixedWing Position Controller , FixedWing Attitude Controller

及其他原件进行订阅使用, 这里我们需要关心的 FixedWing Position Controller , FixedWing Attitude Controller中的使用情况.

FixedWing Position Controller

在制导控制器中, px4会根据当前的throttle期望值, 调用内部的 TECS, 进行新的姿态设定值, 计算期望空速, pitch, 以及使用其他的逻辑来进行计算 yaw, 以及roll的设定值, 赋值给变量*_att_sp*, 从而在最后发布给下一层的姿态控制器.

```
_att_sp.roll_body = _manual.y * _parameters.man_roll_max_rad;
_att_sp.yaw_body = 0;

const float deadBand = 0.06f;
float factor = 1.0f - deadBand;
float pitch = -(_manual.x + deadBand) / factor;

// calculate the demanded airspeed.
float
FixedwingPositionControl::get_demanded_airspeed()
{
  float altctrl_airspeed = 0; // the demanded airspeed.

  // neutral throttle corresponds to trim airspeed
  if (_manual.z < 0.5f) {
    // lower half of throttle is min to trim airspeed
    altctrl_airspeed = _parameters.airspeed_min +
           (_parameters.airspeed_trim - _parameters.airspeed_min) *
           _manual.z * 2;

  } else {
    // upper half of throttle is trim to max airspeed
    altctrl_airspeed = _parameters.airspeed_trim +
           (_parameters.airspeed_max - _parameters.airspeed_trim) *
           (_manual.z * 2 - 1);
  }

  return altctrl_airspeed;
}
```

计算一些姿态的设定值

上述代码公式转换如下

FixedWing Attitude Controller

姿态控制器拿到数据且赋值给 *_manual* 变量.

```
if (_vcontrol_mode.flag_control_rattitude_enabled) {
  if (fabsf(_manual.y) > _parameters.rattitude_thres || fabsf(_manual.x) >
      _parameters.rattitude_thres) {
    _vcontrol_mode.flag_control_attitude_enabled = false;
  }
}
```

高度处理逻辑

- 若该变量的y和x大于 *_parameters.rattitude_thres* 参数的值, 则 $flag\_control\_attitude\_enabled = false$, 若这个时候$flag_control_rates_enabled$ 为真, 那么执行处理逻辑1; 再将值发布到$ORB_ID(vehicle_rates_setpo$ 息上, 进行下一步的处理.

- 若该变量的y和x小于等于 *_parameters.rattitude_thres* 参数的值, 则 $flag\_control\_attitude\_enabled = true$, 执行处理逻辑2, 将值publish到 *_attitude_setpoint_id* 上面(这个topic就对应offboard从mavros发送到px4的控制逻辑层)

- 若不符合上面两个逻辑, 直接执行处理逻辑3, 将控制指令直接发布给执行器. 将值publish到 *_attitude_setpoint_id* 上

```
1     _rates_sp.roll = _manual.y * _parameters.acro_max_x_rate_rad;
2     _rates_sp.pitch = -_manual.x * _parameters.acro_max_y_rate_rad;
3     _rates_sp.yaw = _manual.r * _parameters.acro_max_z_rate_rad;
4     _rates_sp.thrust_body[0] = _manual.z;
```

处理逻辑1

```
1     // STABILIZED mode generate the attitude setpoint from manual user inputs
2           _att_sp.timestamp = hrt_absolute_time();
3
4           // calculate the setpoints
5           _att_sp.roll_body = _manual.y * _parameters.man_roll_max + _parameters.
                rollsp_offset_rad;
6           _att_sp.roll_body = math::constrain(_att_sp.roll_body, -_parameters.
                man_roll_max, _parameters.man_roll_max);
7           _att_sp.pitch_body = -_manual.x * _parameters.man_pitch_max +
                _parameters.pitchsp_offset_rad;
8           _att_sp.pitch_body = math::constrain(_att_sp.pitch_body, -_parameters.
                man_pitch_max, _parameters.man_pitch_max);
9           _att_sp.yaw_body = 0.0f;
10          _att_sp.thrust_body[0] = _manual.z;
11
12          // get the Quatf
13          Quatf q(Eulerf(_att_sp.roll_body, _att_sp.pitch_body, _att_sp.yaw_body))
                ;
14          q.copyTo(_att_sp.q_d);
15                    _att_sp.q_d_valid = true;
16
17                    _attitude_setpoint_id = ORB_ID(vehicle_attitude_setpoint);
```

处理逻辑2

```
1     /* manual/direct control */
2     _actuators.control[actuator_controls_s::INDEX_ROLL] = _manual.y * _parameters.
          man_roll_scale + _parameters.trim_roll;
3     _actuators.control[actuator_controls_s::INDEX_PITCH] = -_manual.x *
          _parameters.man_pitch_scale + _parameters.trim_pitch;
4     _actuators.control[actuator_controls_s::INDEX_YAW] = _manual.r * _parameters.
          man_yaw_scale + _parameters.trim_yaw;
5     _actuators.control[actuator_controls_s::INDEX_THROTTLE] = _manual.z;
6
7     _actuators_id = ORB_ID(actuator_controls_0);
```

处理逻辑3

```
_attitude_setpoint_id = ORB_ID(vehicle_attitude_setpoint);
```

_attitude_setpoint_id

5.3.4 Low pass filter

```
    float LowPassFilter2p::apply(float sample)
{
  // do the filtering
  float delay_element_0 = sample - _delay_element_1 * _a1 - _delay_element_2 * _a2
      ;

  if (!PX4_ISFINITE(delay_element_0)) {
    // don't allow bad values to propagate via the filter
    delay_element_0 = sample;
  }

  const float output = delay_element_0 * _b0 + _delay_element_1 * _b1 +
      _delay_element_2 * _b2;

  _delay_element_2 = _delay_element_1;
  _delay_element_1 = delay_element_0;

  // return the value. Should be no need to check limits
  return output;
}
```

Low pass filter from px4

第六章　前备知识

6. 1 The centrifugal force 离心力

当物体在做非直线运动时（非牛顿环境，例如：圆周运动或转弯运动），因物体一定有本身的质量存在，质量造成的惯性会强迫物体继续朝着运动轨迹的切线方向（原来那一瞬间前进的直线方向）前进，而非顺着接下来转弯过去的方向走。

若这个在做非直线运动的物体（例如：车）上有乘客的话，乘客由于同样随着车子做转弯运动，会受到车子向乘客提供的向心力，但是若以乘客为参照系，由于该参照系为非惯性系，他会受到与他相对静止的车子给他的一个指向圆心的向心力作用，但同时他也会给车子一个反向等大，由圆心指向外的力，就好像没有车子他就要被甩出去一样，这个力就是所谓的离心力。

6.2 航向角和偏航角以及升力

(1) 航向角是质心沿着速度方向在水平面上的投影与预定轨迹的切线方向之间的夹角, 记为χ;
偏航角是质心沿着机头方向在水平面上的投影与预定轨迹的切线方向之间的夹角, 记为 ψ.
航向角χ是地速相对于i^i(正北)方向的偏移角度, 偏航角ψ是空速方向;
在没有风的情况下, 偏航角和航向角是相等的

(2) 升力是垂直副翼向上的

(3) roll的大小和方向: 机体坐标系OZb轴与通过机体OXb轴的铅垂面间的夹角，机体向右滚为正，反之为负。

(4) γ是地速方向和水平方向的夹角

(5) 圆周运动的半径, 线速度, 以及角速度三者之间的关系: $v = r\omega$

6. 3 raspberry

6. 3. 1 开机自启动程序sample

在文件 */.config* 文件下找到autostart文件夹, 如如果没有就新创建一个。在该文件夹下创建一个*xxx.desktop*文件，文件名自拟，后缀必须是desktop，文件内容如下：

```
[Desktop Entry]
Name=test
Comment=Python Program
Exec=python /home/pi/test.py
Icon=/home/pi/python_games/4row_black.png
Terminal=false
MultipleArgs=false
Type=Application
Categories=Application;Development;
StartupNotify=true
```

xxx.desktop

- Desktop Entry: 每个desktop文件都以这个标签开始，说明这是一个Desktop Entry文件
- Name=python(程序名称（必须），这里以创建一个Firefox的快捷方式为例)
- Comment=Python program(程序描述可选)
- Exec=python3 wifitz.py(程序的启动命令（必选），可以带参数运行,当下面的Type为Application，此项有效. 这里是中端执行的命令,路径应该是绝对路径)
- Icon=*/home/pi/python_games/4row*$_a$*rrow.png*(设置快捷方式的图标（可选）,可以从系统其他地方直接法制个图标路径过来)
- Terminal = false 是否在终端中运行（可选），当Type为Application，此项有效
- Type = Application desktop的类型（必选），常见值有“Application”和“Link”
- Categories = GNOME;Application;Network; 注明在菜单栏中显示的类别（可选）

Name、Comment、Icon可以自定，表示启动项的名称、备注和图标。 Exec表示调用的指令，和在终端输入运行脚本的指令格式一致。如果你的树莓派没有png图标，那么就和我一样，找到 *python_game* 文件夹，那里有几个简单的图标可以现成拿来使用。之后再次*sudo reboot*

重启成功后，在linux终端使用命令*ps aux*查看当前运行的所有程序，如果程序正常启动，可以在这里找到.

如果需要启动多个程序，我试过用上述方法添加三个.desktop文件，结果失败了；所以，如果需要启动多个程序，建议创建一个.sh脚本文件，将多个程序的终端启动命令添加到.sh文件中，然后将上述第三步中的第4行改为Exec=./filename.sh。接下来执行第4步和第5步查看执行结果，我这里能够成功启动三个python程序， 本方法是利用树莓派进入桌面后再自动启动程序的方式来实现自动启动，所以需要等桌面加载完成后才启动，等待的时间相对较长一些 这个方法测试失败!!!

6.3.2 几种开机自启动的方法

之前在树莓派上写的程序，都是通过ssh连接后在控制台上用命令行启动的，这种方式适合测试和调试，完善好程序后，比较好的方法是把程序设置为开机自启动，这样树莓派一上电就开始运行程序。查阅网上的资料，主要有三种方法，

博客地址

1. 是在*rc.local*添加启动项
2. 是在 */.config/autostart*中添加桌面启动应用
3. 是在*/etc/init.d/*中添加服务项
4. 使用systemctl设置服务

测试代码如下:

```
#!/usr/bin/python
import socket
import time

HOST='192.168.64.136'
PORT=9999

server=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
server.connect((HOST,PORT))
data='123'

try:
    while True:
        server.sendall(data.encode(encoding='utf-8'))
        time.sleep(1)
except:
    server.close()
```

UDP测试代码