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。

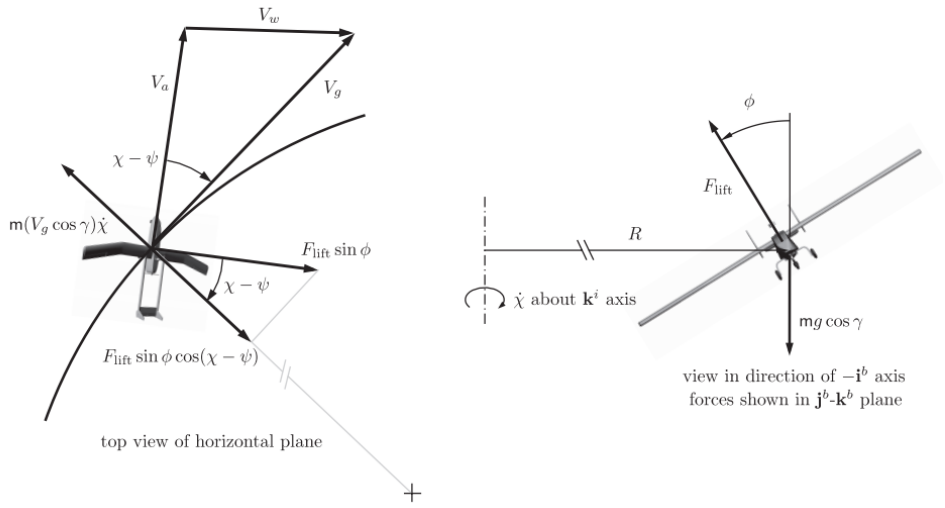

Figure 5.1 Free-body diagrams indicating forces on a MAV in a climbing coordinated turn.

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

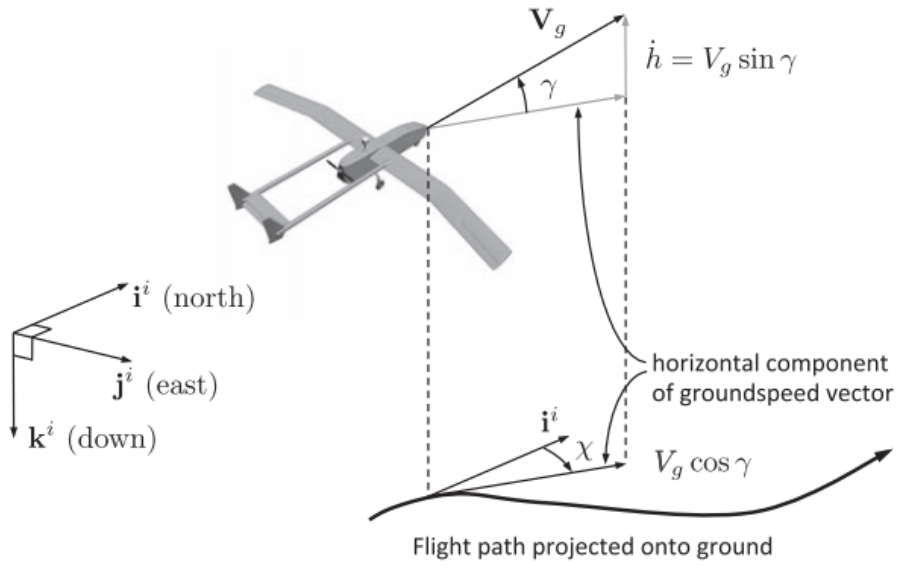

Figure 2.10 The flight path angle γ and the course angle χ.

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 $ORB_ID(vehicle_rates_setpoint)$消息上, 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

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

处理逻辑1

```
    // STABILIZED mode generate the attitude setpoint from manual user inputs
          _att_sp.timestamp = hrt_absolute_time();

          // calculate the setpoints
          _att_sp.roll_body = _manual.y * _parameters.man_roll_max + _parameters.
              rollsp_offset_rad;
          _att_sp.roll_body = math::constrain(_att_sp.roll_body, -_parameters.
              man_roll_max, _parameters.man_roll_max);
          _att_sp.pitch_body = -_manual.x * _parameters.man_pitch_max +
              _parameters.pitchsp_offset_rad;
          _att_sp.pitch_body = math::constrain(_att_sp.pitch_body, -_parameters.
              man_pitch_max, _parameters.man_pitch_max);
          _att_sp.yaw_body = 0.0f;
          _att_sp.thrust_body[0] = _manual.z;

          // get the Quatf
          Quatf q(Eulerf(_att_sp.roll_body, _att_sp.pitch_body, _att_sp.yaw_body))
              ;
          q.copyTo(_att_sp.q_d);
                    _att_sp.q_d_valid = true;

                    _attitude_setpoint_id = ORB_ID(vehicle_attitude_setpoint);
```

处理逻辑2

```
    /* manual/direct control */
    _actuators.control[actuator_controls_s::INDEX_ROLL] = _manual.y * _parameters.
        man_roll_scale + _parameters.trim_roll;
    _actuators.control[actuator_controls_s::INDEX_PITCH] = -_manual.x *
        _parameters.man_pitch_scale + _parameters.trim_pitch;
    _actuators.control[actuator_controls_s::INDEX_YAW] = _manual.r * _parameters.
        man_yaw_scale + _parameters.trim_yaw;
```

```
    _actuators.control[actuator_controls_s::INDEX_THROTTLE] = _manual.z;

    _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 Raspberry

6. 1 raspberry

6. 1. 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!!!

6.1.2 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(已经实现)
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

测试方法代码: *UDP_Send*

通过rc.local自启动

为了在树莓派启动的时候运行一个命令或程序，你需要将命令添加到rc.local文件中。这对于想要在树莓派接通电源后无需配置直接运行程序，或者不希望每次都手动启动程序的情况非常有用。

在你的树莓派上，选择一个文本编辑器编辑/etc/rc.local文件。你必须使用root权限编辑，例如 *sudo vim /etc/rc.local* 在注释后面添加命令, 但是要保证exit 0这行代码在最后, 然后保存文件退出. 如果你的命令需要长时间运行（例如死循环）或者运行后不能退出，那么你必须确保在命令的最后添加&符号让命令运行在其它进程, 否则，这个脚本将无法结束，树莓派就无法启动。这个&符号允许命令运行在一个指定的进程中，然后继续运行启动进程

rc.local添加的执行指令, 会在启动的时候运行, 并且是在其他服务开启之前就启动了.

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
fi

```

```
./home/pi/UDP/UDP_Send &

exit 0
```

/etc/rc.local

执行sudo vim /etc/rc.local 命令编辑rc.local文件, 在exit 0上面一行添加语句 ./home/pi/UDP/UDP_Send &, 表示运行UDP_Send可执行文件，而&符号可以简单理解为让程序运行在后台。然后执行sudo reboot 重启树莓派，记得把自己电脑上的接收程序提前打开，看看能不能接收到数据。

启动之后, 可以在接收器上面执行接收到UDP数据.成功的时候如果输入ps aux 来查看进程情况, 可以查看到

```
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         541  0.7  0.0   3824   944 ?        S    08:38   0:00 ./home/pi/UDP/
    UDP_Send
```

进程执行

Can't execute the scriptes by rc.local

it's reported that not all programs will run reliably, because not all services may be available when rc.local runs.

we would have root ownership to execute availably all commands.

通过桌面应用自启动

待测

6.1.3 /etc/profile.d

自己写一个shell脚本; 将写好的脚本（.sh文件）放到目录 /etc/profile.d/ 下，系统启动后就会自动执行该目录下的所有shell脚本。

将执行脚本放在 /etc/profile.d/ 路径下, 不可以执行, 因为 /etc/profile.d/ 路径下的脚本文件都是在 /etc/profile 脚本启动的时候进行遍历执行的.

```
if [ -d /etc/profile.d ]; then
  for i in /etc/profile.d/*.sh; do
    if [ -r $i ]; then
      . $i
    fi
  done
  unset i
fi
```

遍历执行/etc/profile.d/路径下的脚本文件

待测

通过服务脚本自启动

执行ls /etc/init.d 可以看到该目录下有很多服务程序文件，在这里添加自己的服务文件，就可以对其进行配置从而实现自启动。在该目录下新建文件 Auto_Start_UDP_Send，编辑内容：

```
#!/bin/bash
### BEGIN INIT INFO
# Provides: Auto_Start_Test
# Required-Start: $remote_fs
# Required-Stop: $remote_fs
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# Short-Description: Auto Start Test
# Descrption: This service is used to test auto start service
### END INIT INFO

case "$1" in
    start)
        echo "Start"
        ./home/pi/UDP/UDP_Send &
        ;;
    stop)
        echo "Stop"
        killall ./home/pi/UDP/UDP_Send
        exit 1
        ;;
    *)
        echo "Usage:service Auto_Start_UDP_Send start|stop"
        exit 1
        ;;
esac
exit 0
```

服务文件配置

服务文件是什么: 操作系统中的服务是指执行指定系统功能的程序、例程或进程，以便支持其他程序，尤其是低层(接近硬件)程序。通过网络提供服务时，服务可以在Active Directory(活动目录)中发布，从而促进了以服务为中心的管理和使用。

服务是一种应用程序类型，它在后台运行。服务应用程序通常可以在本地和通过网络为用户提供一些功能，例如客户端/服务器应用程序、Web服务器、数据库服务器以及其他基于服务器的应用程序。(文件的权限r是4, w是2, x是1)

按照下面的执行流程进行操作:

```
    // 1. change the file's Authority 755 or 777
    sudo chmod 755 Auto_Start_UDP_Send

    // 2. add the service to Self-starting, than can be succeed
    sudo update-rc.d Auto_Start_UDP_Send defaults

    // 3. we can start the service by hand(had been verified).
    sudo service Auto_Start_Test start

    // 3. we can stop the service by hand(had been verified).
    sudo service Auto_Start_Test stop

    //
    sudo reboot
```

之后, 重新启动树莓派查看效果. 不成功, 但是手动启动的时候是奏效的.

那么如何将自己写的服务添加到开机自动启动呢?

chkconfig command.如果没有该命令, 我们可以先进行安装

```
// install the chkconfig command
sudo apt-get install chkconfig

// 1. check out the services list and their priority
chkconfig

// 2. add the service to the list
sudo chkconfig --add Auto_Start_UDP_Send

// 3. reboot(not verified)
sudo reboot
```

将启动服务指令放到rc.local文件中

这里开启UDP_Send, 需要网络服务

```
pi@raspberrypi:~ $ sudo service Auto_udp_send status
   Loaded: loaded (/etc/init.d/Auto_udp_send; generated; vendor preset: enabled)
   Active: active (exited) since Fri 2020-08-07 10:07:59 BST; 43s ago
     Docs: man:systemd-sysv-generator(8)
    Tasks: 0 (limit: 4915)
   CGroup: /system.slice/Auto_udp_send.service

Aug 07 10:07:59 raspberrypi Auto_udp_send[393]: Start
Aug 07 10:07:59 raspberrypi systemd[1]: Started LSB: Auto Start Test.
Aug 07 10:07:59 raspberrypi Auto_udp_send[393]: send heading    is:16
Aug 07 10:07:59 raspberrypi Auto_udp_send[393]: send airspeed   is:17
Aug 07 10:07:59 raspberrypi Auto_udp_send[393]: send position_x is:18
Aug 07 10:07:59 raspberrypi Auto_udp_send[393]: send position_y is:19
Aug 07 10:07:59 raspberrypi Auto_udp_send[393]: send position_z is:20
Aug 07 10:07:59 raspberrypi Auto_udp_send[393]: send yaw        is:21
Aug 07 10:07:59 raspberrypi Auto_udp_send[393]: message:1 Parity is:111
Aug 07 10:07:59 raspberrypi Auto_udp_send[393]: sendto error:: Network is
    unreachable
```

查看服务的运行状态

6.1.4 使用rc.local修改

1. 在执行某一个语句之后, 等到几秒再次执行下一条语句
2. 程序后台进行的脚本文件

```
#!/bin/bash
a=0
b=10
echo "Will it execute the source command!"
source ~/fixedWing_ws/devel/setup.bash
echo "Will it execute the UDP_Send command!"
rosrun communication UDP_Send 0 -mt UH -p 8002&
```

```
echo "Will it execute the while()!"
while [ "$a" != "$b" ];
do
#       /home/pi/UDP/UDP_Send &
        sleep 1s
        a=$(($a+1))
        echo "waiting 1s... $a"
done

echo "Will it open the px4.launch!"
roslaunch mavros px4.launch fcu_url:=/dev/serial0:921600 &
```

启动脚本多ROS节点

如果#!/*bin*/*sh*, 会出现*source* : *notfound*!, 解决办法是修改为#!/*bin*/*bash*

在rc.local中执行脚本失败了, 卧槽! 现象是*UDP_Send*是被执行了, 但是后面打开px4.launch文件失败了(会是while导致的吗?)

6.1.5 init.d目录和/etc/rc.local的区别

init.d目录中的脚本都是以服务的形式启动的，顾名思义，服务会在后台一直运行. 所以，系统在执行init.d目录中的服务脚本时，会分别单独为每个服务脚本启动一个非登录非交互式shell来始终在后台运行服务脚本一直到用户退出登录，关闭系统，这些始终运行在各个非登录非交互式的shell中的服务脚本才会停止运行

rc.local也是我经常使用的一个脚本。该脚本是在系统初始化级别脚本运行之后再执行的，因此可以安全地在里面添加你想在系统启动之后执行的脚本, 启动顺序可见7.2。

可以在rc.local中启动一些服务.

6.1.6 查看服务优先顺序

要知道服务的启动顺序，就需要先知道服务如何启动的。 linux有7个运行级别，用户可选择不同的运行级别。进入/etc/rc.d/目录，可查看到对应从rc0.d到rc6.d等7个目录，这些目录即对应7个级别。

```
    pi@raspberrypi:/etc $ who -r
         run-level 5  2020-08-07 11:28
```

查看运行级别

查看自己系统的运行级别指令为who -r. 进入rc3.d目录，可看到各种以K或者S开始的服务，命名都以S（start）、K（kill）或D（disable）开头, 而后面的数字就表示启动顺序。我们以熟悉的network服务为例，这里只是个链接，其实还是指向/etc/init.d/network，其启动值为10。

6.1.7 /etc directory

编写好一个服务之后, 使用下面的指令将(比如名字为start自定义服务)其添加到 linux 系统的启动服务中. 运行后会将 start 文件加入到 /etc/rcX.d/ 目录中。其中: X:0-6, 分别表示不同的启动级别, 3为字符界面启动, 5为GUI启动. 最后的数据 95 表示启动顺序, 最后在rc3.d中生成的文件会变成: S95start(命名规则: S-启动顺序-服务名字).

```
        // 1. add the service to start list. Will generate a file named is "
            S95start"
        cd /etc/init.d
```

```
3         sudo update-rc.d start defaults 95
4
5         // 2. rcX.d
6         drwxr-xr-x  2 root root    4096 Aug  4 10:07 rc0.d
7         drwxr-xr-x  2 root root    4096 Aug  4 10:07 rc1.d
8         drwxr-xr-x  2 root root    4096 Aug  4 10:07 rc2.d
9         drwxr-xr-x  2 root root    4096 Aug  4 10:07 rc3.d
10        drwxr-xr-x  2 root root    4096 Aug  4 10:07 rc4.d
11        drwxr-xr-x  2 root root    4096 Aug  4 10:07 rc5.d
12        drwxr-xr-x  2 root root    4096 Aug  4 10:07 rc6.d
13        -rwxr-xr-x  1 root root     735 Aug  5 09:38 rc.local
14        -rwxr-xr-x  1 root root     420 Apr  8  2019 rc.local.bak
15
16        // 3. add Auto_Start_UDP_Send service to unix system not GUI. So it lies
              in rc3.d/
17        pi@raspberrypi:/etc/rc3.d $ ls
18        S01console-setup.sh      S02rsyslog        S03dphys-swapfile  S03rsync
                  S05plymouth
19        S02Auto_Start_UDP_Send   S02triggerhappy   S03paxctld         S03ssh
20        S02binfmt-support        S03cron           S03raspi-config    S04avahi-
              daemon
21        S02dhcpcd                S03dbus           S03rng-tools       S04bluetooth
```

6.2 raspberry 启动 ssh

如果您没有可用的备用HDMI显示器或键盘来连接Raspberry Pi，则可以通过将名为ssh（无任何扩展名）的空文件放入启动分区中来轻松启用SSH。 博客地址

要在Raspberry Pi上启用SSH，请执行以下步骤：

- 关闭Raspberry Pi的电源，然后卸下SD卡。
- 将SD卡插入计算机的读卡器。 SD卡将自动安装。
- 使用OS文件管理器导航到SD卡引导目录。 Linux和macOS用户也可以从命令行执行此操作。
- 在启动目录中创建一个新的名为ssh的空文件，不带任何扩展名。
- 从计算机上卸下SD卡，然后将其放入Raspberry Pi中。
- 打开您的Pi板上的电源。 Pi会在启动时检查此文件是否存在，如果存在，将启用SSH并删除该文件。

就这样。 Raspberry Pi启动后，您可以SSH进入它。

6.3 开机服务: systemd

待测

为了在树莓派启动时执行一个命令或程序，你可以设置一个服务。一旦你有了一个服务，则可以使用start/stop/enable/disable来控制服务的执行.

6.3.1 创建服务

在Pi上，你需要创建一个 .service 文件来创建服务。如下：

```
[Unit]
Description=BackgroundMusic
After=network.target

[Service]
ExecStart=/usr/bin/python3 -u play_audio.py
WorkingDirectory=/home/pi/myscript
StandardOutput=inherit
StandardError=inherit
Restart=always
User=pi

[Install]
WantedBy=multi-user.target
```

background_music.service

在这个示例中，服务会在使用python3来执行/home/pi/myscript目录下的 *play_audio.py* 脚本。服务不仅限于执行python脚本，修改ExecStart后的命令即可执行程序或者shell命令。以root身份拷贝 *background_music.service*文件到etc/systemd/system目录下即可使用这个服务。

6.3.2 启动/停止服务

```
    // start service
    sudo systemctl start background_music.service

    // stop service
    sudo systemctl stop background_music.service
```

6.3.3 开机启动服务

```
    // 1. set to the auto start when start your machine
    sudo systemctl enable background_music.service

    // 2. cancel the auto start when start your machine
    sudo systemctl disable background_music.service
```

需要注意服务的启动依赖顺序:

- 服务需要在它所依赖的服务启动之后再启动。*background_music.service*服务被指定在网络有效之后才启动（After=network.target）
- 服务的启动顺序和依赖可以在.service文件里配置

更多关于服务控制的细节，可以参考man systemctl.

第七章 Linux

7. 1 the Linux OS Start Process

树莓派(Raspberry Pi) (中文名为“树莓派”,简写为RPi，(或者RasPi / RPI)是为学习计算机编程教育而设计)，只有信用卡大小的微型电脑，其系统基于Linux。随着Windows 10 IoT的发布，我们也将可以用上运行Windows的树莓派。

树莓派系统镜像SD卡的内存分布图7.1.

```
pi@raspberrypi:~ $ cat /proc/device-tree/model
Raspberry Pi 3 Model B Rev 1.2
```

查看树莓派所属型号

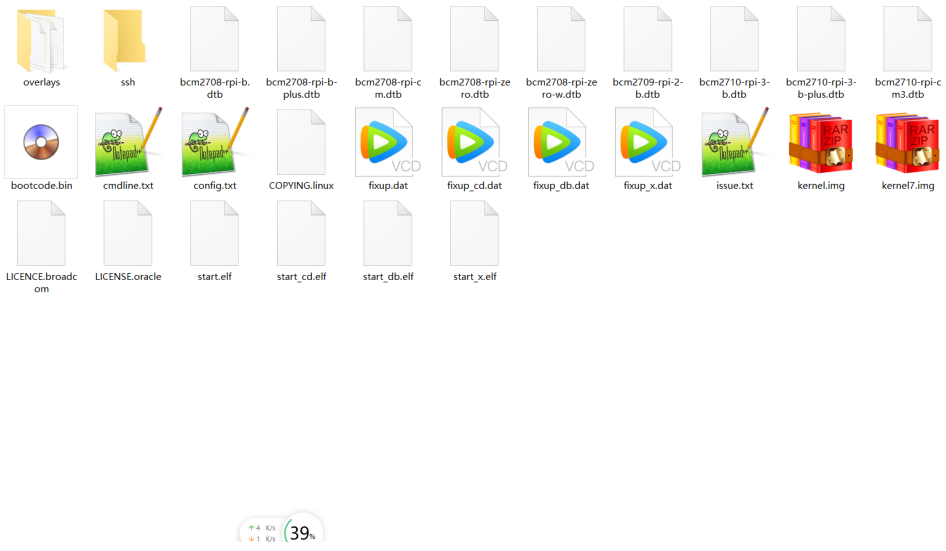

图 7.1. Raspberry Files Structure

其中的:

- bootcode.bin: 启动文件
- start.elf : 类似U-Boot的BootLoader文件
- kernel7.img kernel.img: Linux内核文件
- config.txt: 配置文件

对于树莓派 3B，在编译内核时， kernel7.img 使用的是 armv7h，这可以更好的发挥处理器的性能，当然，kernel.img 也是可以用的，因为 ARM 可以向下兼容，但是无法发挥处理器的全部能力。因此在引导内核时，使用的是 kernel7.img

树莓派的SoC(SoC称为系统级芯片, 片上系统)内部集成了ARM CPU，GPU，ROM，SDRAM，以及其他设备，所以可以把树莓派想象成一股arm系统结构的PC。当给树莓派加电后，最先执行保存在ROM中的代码，这些代码是芯片出厂的时候就设定的，通常被称为 first-stage bootloader，这些代码固化硬件内部，可以认为是SoC硬件的一部分。 first-stage bootloader的主要工作是加载位于SD卡上第一个分区的bootloader（称为second-stage bootloader ），第一个分区必须是FAT32格式。second-stage bootloader 主要是bootloader.bin。可以把SD卡取出，放到Windows或Linux系统中，就可以看到bootloader.bin文件。需要说明的是，上电或者重启后，cpu和ram都没有初始化，因此，执行second-stage bootloader 的实体是GPU，bootcode.bin是加载到GPU的128KB大小的L2Cache中，再执行的。bootcode.bin的主要工作是初始化ram，并把start.elf（也位于SD卡的第一分区）加载到内存中。start.elf就是third-stage bootloader，start.efl从第一个分区中加载config.txt，可以把config.txt想象成bios配置信息，内部的配置都可以改变。

7.1.1 1st stage: power on

GPU读取ROM内容, 且运行GPU运行ROM中代码

7.1.2 2nd stage: run bootcode.bin

GPU读取SD卡第一个FAT分区根目录下bootcode.bin

- GPU将bootcode.bin复制到L2 cache
- GPU执行bootcode.bin

7.1.3 3rd stage: load start.elf

GPU读取SD卡第一个FAT分区根目录中start.elf

- GPU将start.elf加载到内存中
- GPU开始执行start.elf

start.elf把ram空间划分为2部分：CPU访问空间和GPU访问空间。 SoC芯片只访问属于GPU地址空间的内存区，例如，GPU的物理内存地址空间为0x000F000 – 0x0000FFFF， CPU的物理内存地址空间为0x00000000 – 0x0000EFFF，如果GPU访问0x0000008，那么它访问的物理地址为0x000F008。（实际上，ARM处理器的mmu部件把GPU的内存空间映射到0xC0000000 开始）。

config.txt在内存地址空间分配完成后才加载，因此，不可以在config.txt中更改内存地址的配置。然而，可以通过配置多个elf文件来让start.elf和config.txt支持多种配置空间。 start.elf还从SD卡的第一个分区中加载cmdline.txt（如果cmdline.txt存在的话）。该文件保存的是启动kernel（不一定是Linux的内核）的参数。至此，SoC进入了boot的最后阶段， start.efl把kernel.img，ramdisk，dtb加载到内存的预定地址，然后向cpu发出重启信号，因此cpu就可以从内存的预定地址执行kernel的代码，就进入了软件定义的系统启动流程。

7.1.4 4th stage: load linux kernel

GPU读取SD卡第一个FAT分区根目录中的kernel.img(Linux 内核) 到内存

- 唤醒CPU
- CPU 开始运行kernel.img

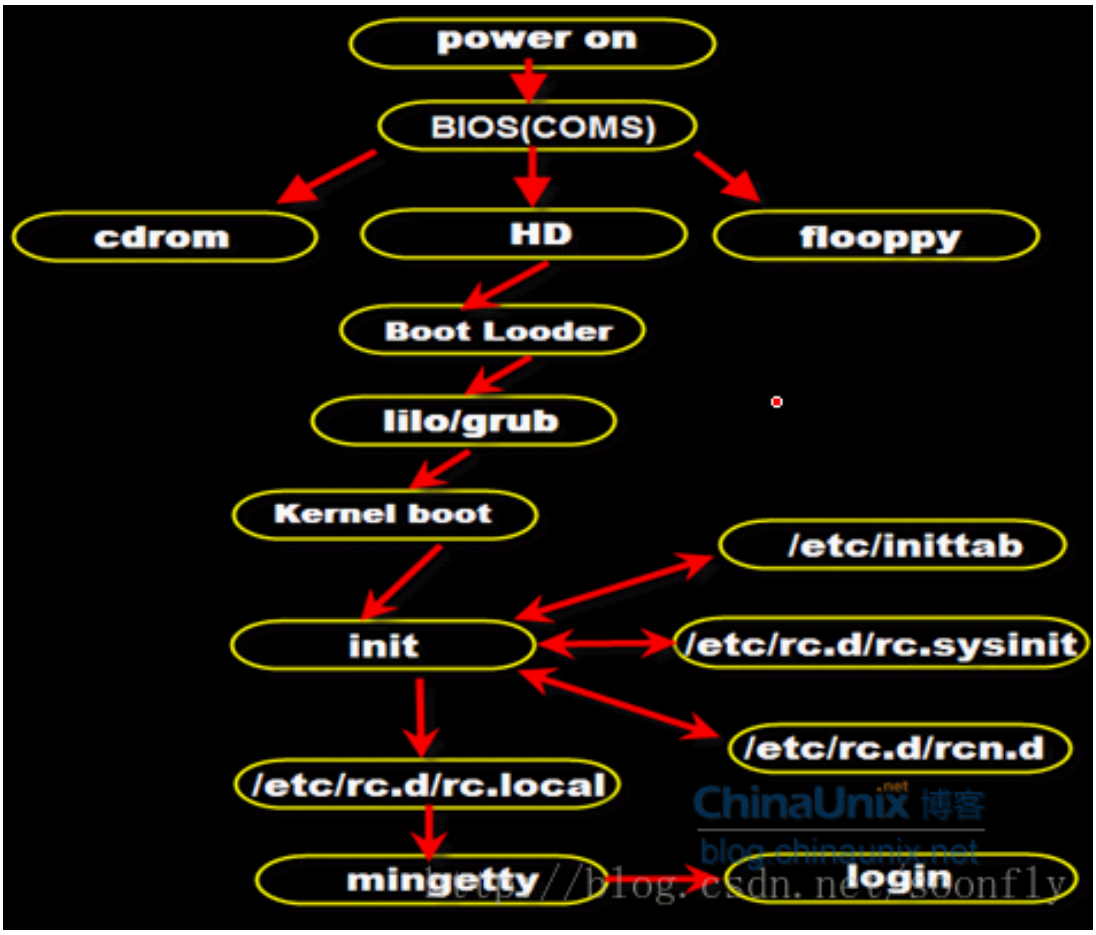

图 7.2. linux启动流程

7. 2 what's the shell?

7. 2. 1 什么是shell?

shell是你（用户）和Linux（或者更准确的说，是你和Linux内核）之间的接口程序。你在提示符下输入的每个命令都由shell先解释然后传给Linux内核.

不论何时你键入一个命令，它都被Linux shell所解释。一些命令，比如打印当前工作目录命令（pwd），是包含在Linux bash内部的（就象DOS的内部命令）。其他命令，比如拷贝命令（cp）和移动命令（rm），是存在于文件系统中某个目录下的单独的程序。而对用户来说，你不知道（或者可能不关心）一个命令是建立在shell内部还是一个单独的程序。

shell 首先检查命令是否是内部命令，不是的话再检查是否是一个应用程序，这里的应用程序可以是Linux本身的实用程序，比如ls 和 rm，也可以是购买的商业程序，比如 xv，或者是公用软件（public domain software），就象 ghostview。然后shell试着在搜索路径($PATH$)里寻找这些应用程序。搜索路径是一个能找到可执行程序的目录列表。如果你键入的命令不是一个内部命令并且在路径里没有找到这个可执行文件，将会显示一条错误信息。而如果命令被成功的找到的话，shell的内部命令或应用程序将被分解为系统调用并传给Linux内核。

shell的另一个重要特性是它自身就是一个解释型的程序设计语言，shell 程序设计语言支持在高级语言里所能见到的绝大多数程序控制结构，比如循环，函数，变量和数组。shell 编程语言很易学，并且一旦掌握后它将成为你的得力工具。任何在提示符下能键入的命令也能放到一个可执行的shell程序里，这意味着用shell语言能简单地重复执行某一任务。

7. 2. 2 shell 如何启动?

shell在你成功地登录进入系统后启动，并始终作为你与系统内核的交互手段直至你退出系统。你系统上的每位用户都有一个缺省的shell。每个用户的缺省shell在系统里的passwd文件里被指定，该文件的路径是/etc/passwd。 passwd文件里还包含有其他东西：每个人的用户ID号，一个口令加密后的拷贝和用户登录后立即执行的程序，（注：为了加强安全性，现在的系统一般都把加密的口令放在另一个文件–shadow中，而passwd中存放口令的部分以一个x字符代替）虽然没有严格规定这个程序必须是某个Linux shell ，但大多数情况下都如此。

7. 3 Linux启动时读取配置文件的顺序

在刚登录Linux时，首先启动 /etc/profile 文件，然后再启动用户目录下的 /.$bash\_profile$、 /.$bash_login$或/.profile文件中的其中一个，执行的顺序为： /.$bash\_profile$、 /.$bash\_login$、 /.profile。(profile: 配置文件)

7. 3. 1 各个配置文件的作用域

/.$bash\_profile$: 是交互式、login 方式进入 bash 运行的

/.bashrc 是交互式 non-login 方式进入 bash 运行的. 通常二者设置大致相同，所以通常前者会调用后者.

/$etc/profile$

此文件为系统的每个用户设置环境信息,当用户第一次登录时,该文件被执行. 并从/etc/profile.d目录的配置文件中搜集shell的设置。

/etc/bashrc

为每一个运行bash shell的用户执行此文件.当bash shell被打开时,该文件被读取（即每次新开一个终端，都会执行bashrc）。

/.bash_profile

每个用户都可使用该文件输入专用于自己使用的shell信息,当用户登录时,该文件仅仅执行一次。默认情况下,设置一些环境变量,执行用户的.bashrc文件。

/.bashrc

该文件包含专用于你的bash shell的bash信息,当登录时以及每次打开新的shell时,该该文件被读取。

/.bash_logout

当每次退出系统(退出bash shell)时,执行该文件. 另外,/etc/profile中设定的变量(全局)的可以作用于任何用户,而 /.bashrc等中设定的变量(局部)只能继承 /etc/profile中的变量,他们是"父子"关系。

7. 4 bash脚本编写

7. 4. 1 后台运行指令

- 在下达的命令后面加上&，就可以使该命令在后台进行工作，这样做最大的好处就是不怕被ctrl-c这个中断指令所中断。
- 在后台执行的程序怎么使它恢复到前台来运行呢？很简单，只用执行fg这个命令，就可以了。
- 已经在前台运行的命令，我能把它放到后台去运行么？当然可以了，只要执行ctrl-z就可以做到了.

```
1 #!/bin/sh
2 /home/pi/UDP/UDP_Send &
```

测试脚本

执行上面的脚本之后, 打印输出会在终端输出.

7. 4. 2 bash和sh的区别

*#!/bin/sh*和*#!/bin/bash*的区别如下, 更加详细的可以见 两者区别

- *#!/bin/bash −−posix* (遵循posix的特定规范，有可能就包括这样的规范: 当某行代码出错时，不继续往下解释)
- *#!/bin/dash* 等价于 *#!/bin/sh*, 执行速度更快.
- 两者的执行标准不一样, *#!/bin/bash* 如果脚本中的某一行代码不能执行, 那么后面的程序也没有办法执行.

```
1 pi@raspberrypi:~/UDP $ ls -la /bin/sh
2 lrwxrwxrwx 1 root root 4 Jan 24  2017 /bin/sh -> dash
```

7.4.3 脚本解释器加上-e参数

e的参数的作用是：每条指令之后后，都可以用#?去判断他的返回值，零就是正确执行，非零就是执行有误，加了e 之后，就不用自己写代码去判断返回值，返回非零，脚本就会退出。

7.5 source, bash, sh, and ./

shell编程中的命令有时和C语言是一样的。&&表示与，|||表示或。把两个命令用&&联接起来，如make mrproper && make menuconfig，表示要第一个命令执行成功才能执行第二个命令。对执行顺序有要求的命令能保证一旦有错误发生，下面的命令不会盲目地继续执行.

7.5.1 source and ./

source命令：
source命令也称为“点命令”，也就是一个点符号（.）,是bash的内部命令。
功能：使Shell读入指定的Shell程序文件并依次执行文件中的所有语句
source命令通常用于重新执行刚修改的初始化文件，使之立即生效，而不必注销并重新登录。
用法：source filename 或 . filename

source filename：这个命令其实只是简单地读取脚本里面的语句依次在当前shell里面执行，没有建立新的子shell 。那么脚本里面所有新建、改变变量的语句都会保存在当前shell里面。

它的作用就是把一个文件的内容当成是shell来执行

7.5.2 bash

7.5.3 sh

sh filename 重新建立一个子shell，在子shell中执行脚本里面的语句，该子shell继承父shell的环境变量，但子shell新建的、改变的变量不会被带回父shell，除非使用export。

当shell脚本具有可执行权限时，用sh filename与./filename执行脚本是没有区别得。 ./filename是因为当前目录没有在PATH中，所有”.”是用来表示当前目录的。

7.5.4 区别

-

7.6 commands

7.6.1 nohup

用nohup运行命令可以使命令永久的执行下去，和用户终端没有关系，例如我们断开SSH连接都不会影响他的运行，注意了nohup没有后台运行的意思；&才是后台运行

&是指在后台运行，但当用户推出(挂起)的时候，命令自动也跟着退出.

- 使用&后台运行程序：
- 结果会输出到终端
- 使用Ctrl + C发送SIGINT信号，程序免疫
- 关闭session发送SIGHUP信号，程序关闭

- 使用nohup运行程序：
- 结果会输出到终端
- 使用Ctrl + C发送SIGINT信号，程序免疫
- 关闭session发送SIGHUP信号，程序关闭

平日, 经常使用nohup和&配合来启动程序: 同时免疫SIGINT和SIGHUP信号.

同时，还有一个最佳实践：不要将信息输出到终端标准输出，标准错误输出，而要用日志组件将信息记录到日志里

第八章 ros

8. 1 ros system

8. 2 ros 设置开机自启动 launch 文件

博客地址

8. 2. 1 *robot_upstart*

该软件包旨在帮助创建特定于平台的简单作业，以机器人PC开机时启动其ROS启动文件。

```
1        rosrun robot_upstart install myrobot_bringup/launch/base.launch
```

该指令将会创建一个名为 myrobot 的 job, 该 job 会执行bash.launch文件. 当你下次开始你的机器的时候, 可以手动打开该服务

```
1        sudo service myrobot start
2        sudo service myrobot stop
```

执行的时候, 打印的输出是在:

```
1        sudo tail /var/log/upstart/myrobot.log -n 30
```

第九章　前备知识

9. 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。

9.2 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: $v = r\omega$