UORB 订阅发布实例

以下例子包含了飞控串口读取外部传感器数据，飞控开启一个进程读取外部传感器发布一个 UORB 主题，另一个进程订阅前一个进程发布的主题，还有就是订阅到的主题通过 mavlink 消息发送到地面站。走了一个完整的飞控数据链路(UORB 链路和 MAVLINK 链路)。

1 新增一个自定义 UORB 主题

在源码的 msg 文件夹下面是飞控所有的 UORB 主题

```
amov01@ubuntu:~/src/Firmware$ ls
build_px4fmu-v2_default  eclipse.project           misc          src
circle.yml               Firmware.sublime-project  msg           test_data
cmake                    Images                    NuttX         Tools
CMakeLists.txt           integrationtests          nuttx-configs unittests
CONTRIBUTING.md          launch                    package.xml   Vagrantfile
Debug                    LICENSE.md                posix-configs
Documentation            Makefile                  README.md
eclipse.cproject         mavlink                   ROMFS
amov01@ubuntu:~/src/Firmware$
```

```
amov01@ubuntu:~/src/Firmware/msg$ ls
actuator_armed.msg                optical_flow.msg
actuator_controls.msg             output_pwm.msg
actuator_direct.msg               parameter_update.msg
actuator_outputs.msg              position_setpoint.msg
adc_report.msg                    position_setpoint_triplet.msg
airspeed.msg                      pwm_input.msg
att_pos_mocap.msg                 qshell_req.msg
battery_status.msg                rc_channels.msg
camera_trigger.msg                rc_parameter_map.msg
CMakeLists.txt                    ros
CMakeLists.txt~                   safety.msg
commander_state.msg               satellite_info.msg
control_state.msg                 sensor_accel.msg
cpuload.msg                       sensor_baro.msg
debug_key_value.msg               sensor_combined.msg
differential_pressure.msg         sensor_gyro.msg
distance_sensor.msg               sensor_mag.msg
ekf2_innovations.msg              servorail_status.msg
ekf2_replay.msg                   subsystem_info.msg
esc_report.msg                    system_power.msg
esc_status.msg                    tecs_status.msg
estimator_status.msg              telemetry_status.msg
fence.msg                         templates
fence_vertex.msg                  test_motor.msg
filtered_bottom_flow.msg          time_offset.msg
follow_target.msg                 transponder_report.msg
fw_pos_ctrl_status.msg            uavcan_parameter_request.msg
```

可以看到 vehicle_global_position.msg 载具全球位置，vehicle_attitude.msg 载具姿态消息，这个姿态消息很重要如下所示:

```
amov01@ubuntu:~/src/Firmware/msg$ cat vehicle_attitude.msg
# This is similar to the mavlink message ATTITUDE, but for onboard use */
# @warning roll, pitch and yaw have always to be valid, the rotation matrix and quaternion are optional
float32 roll            # Roll angle (rad, Tait-Bryan, NED)
float32 pitch           # Pitch angle (rad, Tait-Bryan, NED)
float32 yaw             # Yaw angle (rad, Tait-Bryan, NED)
float32 rollspeed       # Roll body angular rate (rad/s, x forward/y right/z down)
float32 pitchspeed      # Pitch body angular rate (rad/s, x forward/y right/z down)
float32 yawspeed        # Yaw body angular rate (rad/s, x forward/y right/z down)
float32 rollacc         # Roll angular accelration (rad/s^2, x forward/y right/z down)
float32 pitchacc        # Pitch angular acceleration (rad/s^2, x forward/y right/z down)
float32 yawacc          # Yaw angular acceleration (rad/s^2, x forward/y right/z down)
float32 rate_vibration  # Value between 0 and 1 indicating vibration. A value of 0 means no vibration, a v
alue of 1 indicates unbearable vibration levels.
float32 accel_vibration # Value between 0 and 1 indicating vibration. A value of 0 means no vibrat
alue of 1 indicates unbearable vibration levels.
float32 mag_vibration   # Value between 0 and 1 indicating vibration. A value of 0 means no vibration, a v
alue of 1 indicates unbearable vibration levels.
float32[3] rate_offsets # Offsets of the body angular rates from zero
float32[9] R            # Rotation matrix, body to world, (Tait-Bryan, NED)
float32[4] q            # Quaternion (NED)
float32[3] g_comp       # Compensated gravity vector
bool R_valid            # Rotation matrix valid
bool q_valid            # Quaternion valid
```

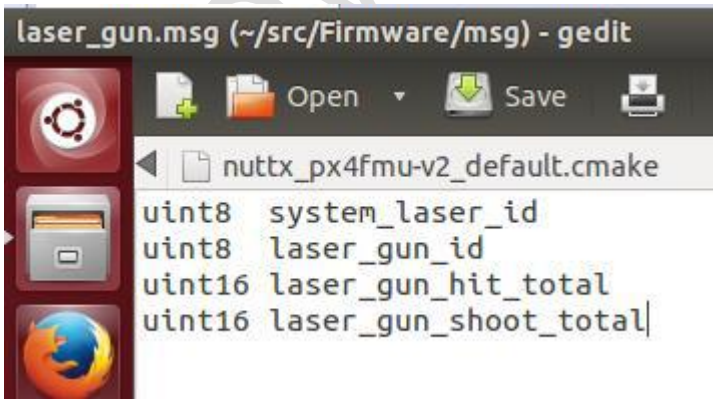
我们可以看到姿态的成员变量，包括横滚 roll，俯仰 pitch,偏航 yaw，横滚速度，俯仰速度等等相关姿态的数据。相关的加速度计，磁力计，陀螺仪经过算法滤波整合之后会发布姿态数据，而姿态控制进程会订阅这个 vehicle_attitude.msg 主题。这个模式很像 linux 的消息队列，或者进程间通信的手段。

1.1 自定义一个主题

我们在 msg 文件夹下面添加添加一个具体的消息,比如我们的消息是

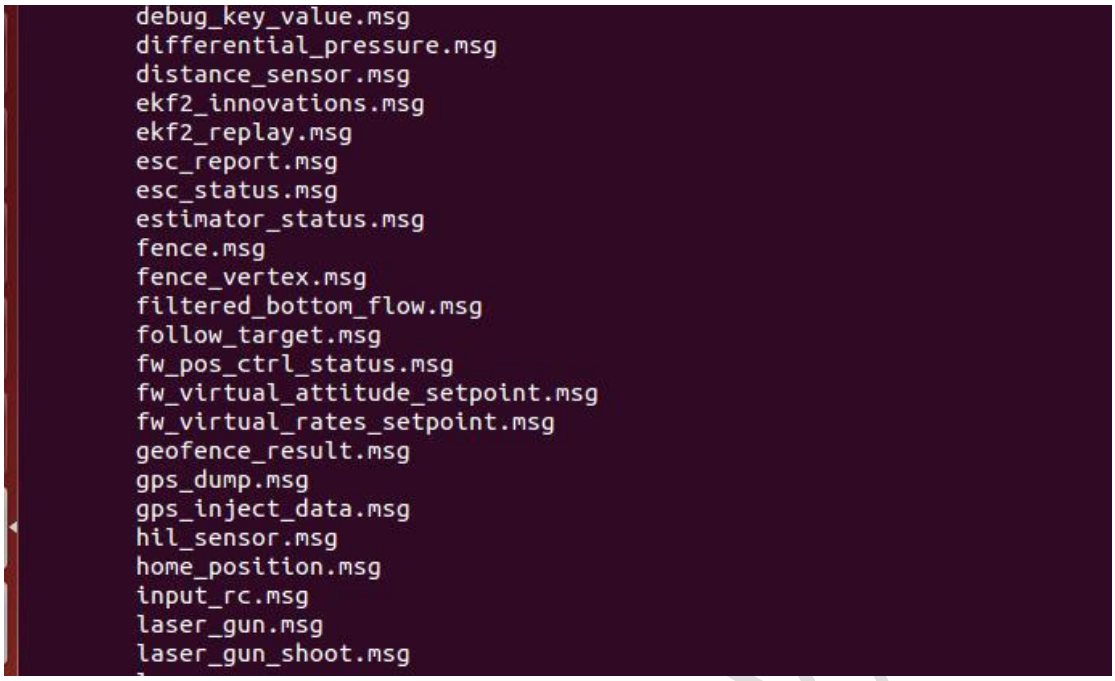
```
amov01@ubuntu:~/src/Firmware/msg$ ls
actuator_armed.msg              gps_inject_data.msg
actuator_controls.msg           hil_sensor.msg
actuator_direct.msg             home_position.msg
actuator_outputs.msg            input_rc.msg
adc_report.msg                  laser_gun.msg
airspeed.msg                    laser_gun_shoot.msg
```

Laser_gun.msg 和 laser_gun_shoot.msg 消息。消息的成员函数可以参照其他原有的消息来写。Msg 文件里面的成员如下:



这个结构体成员是自己定义的。

修改 cmakelist 脚本可以让这个消息生成相应的 uorb 头文件。这个 Cmakelinst 和 msg 消息是同目录的，我们修改如下:



相当于把消息名字添加到这个 Cmakelist 里面，在编译源码的是就把这个消息主题相应的.h 头文件自动写好了，不要我们自己写。到这里我们 make px4fun-v2_default 一把。然后在源码的~/src/Firmware/build_px4fmu-v2_default/src/modules/uORB/topics 这个文件夹下面可以看到我们自定义消息的头文件。

AMOV 阿木(UAV)

```
#include <uORB/uORB.h>


#ifndef __cplusplus

#endif


#ifdef __cplusplus
struct __EXPORT laser_gun_s {
#else
struct laser_gun_s {
#endif
        uint64_t timestamp; // required for logger
        uint16_t laser_gun_hit_total;
        uint16_t laser_gun_shoot_total;
        uint8_t system_laser_id;
        uint8_t laser_gun_id;
        uint8_t _padding0[2]; // required for logger

#ifdef __cplusplus

#endif
};

/* register this as object request broker structure */
ORB_DECLARE(laser_gun);
```

我们可以看到和我们在 msg 文件夹里面自定义成员的一样。而这个结构体名为 laser_gun_s，我们在以后的订阅发布进程里面，包含这个头文件就可以顺利使用了。

2 使用 uorb

2.1 发布一个自定义消息

```
#include <stdio.h>
#include <termios.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <errno.h>
#include <drivers/drv_hrt.h>
#include <systemlib/err.h>
#include <fcntl.h>
#include <uORB/topics/laser_gun.h> //包含消息结构体头文件
static bool thread_should_exit = false;
static bool thread_running = false;
```

```
static int daemon_task;
__EXPORT int laser_gun_main(int argc, char *argv[]);
int laser_gun_thread_main(int argc, char *argv[]);
static int uart_init(const char * uart_name);       //
static int set_uart_baudrate(const int fd, unsigned int baud); //static
static void usage(const char *reason);                    //static
static int shoot_total;

int set_uart_baudrate(const int fd, unsigned int baud)//设置串口波特率
{
    int speed;
    switch (baud) {
        case 9600:    speed = B9600;    break;
        case 19200:   speed = B19200;   break;
        case 38400:   speed = B38400;   break;
        case 57600:   speed = B57600;   break;
        case 115200: speed = B115200; break;
        default:
            warnx("ERR: baudrate: %d\n", baud);
            return -EINVAL;
    }
    struct termios uart_config;
    int termios_state;
    tcgetattr(fd, &uart_config);
    uart_config.c_oflag &= ~ONLCR;
    uart_config.c_cflag &= ~(CSTOPB | PARENB);

    if ((termios_state = cfsetispeed(&uart_config, speed)) < 0) {
        warnx("ERR: %d (cfsetispeed)\n", termios_state);
        return false;
    }
    if ((termios_state = cfsetospeed(&uart_config, speed)) < 0) {
        warnx("ERR: %d (cfsetospeed)\n", termios_state);
        return false;
    }
    if ((termios_state = tcsetattr(fd, TCSANOW, &uart_config)) < 0) {
        warnx("ERR: %d (tcsetattr)\n", termios_state);
        return false;
    }
    return true;
}
int uart_init(const char * uart_name)
{
    int serial_fd = open(uart_name, O_RDWR | O_NOCTTY);
```

```
	if (serial_fd < 0) {
		err(1, "failed to open port: %s", uart_name);
		return false;
	}
	return serial_fd;
}

static void usage(const char *reason)
{
	if (reason) {
		fprintf(stderr, "%s\n", reason);
	}
	fprintf(stderr, "usage: position_estimator_inav {start|stop|status} [param]\n\n");
	exit(1);
}

int laser_gun_main(int argc, char *argv[])
{
	if (argc < 2) {
		usage("[YCM]missing command");
	}
	if (!strcmp(argv[1], "start")) {
		if (thread_running) {
			warnx("[YCM]already running\n");
			exit(0);
		}
		thread_should_exit = false; //初始化这个进程
		daemon_task = px4_task_spawn_cmd("laser_gun",
						 SCHED_DEFAULT,
						 SCHED_PRIORITY_MAX - 5,
						 2000,
						 laser_gun_thread_main,//进程主函数
						 (argv) ? (char * const *)&argv[2] : (char * const *)NULL);
		exit(0);
	}
	if (!strcmp(argv[1], "stop")) {
		thread_should_exit = true;
		exit(0);
	}
	if (!strcmp(argv[1], "status")) {
		if (thread_running) {
			warnx("[YCM]running");
		} else {
```

AMOV 阿木(UAV)社区

```
                warnx("[YCM]stopped");
            }
            exit(0);
        }
        usage("unrecognized command");
        exit(1);
}
int laser_gun_thread_main(int argc, char *argv[])
{
        if (argc < 2) {
            errx(1, "[YCM]need a serial port name as argument");
            usage("eg:");
        }
        const char *uart_name = argv[1];
        warnx("[YCM]opening port %s", uart_name);
        char data = '0';
        int uart_read = uart_init(uart_name);
        if(false == uart_read)return -1;
        if(false == set_uart_baudrate(uart_read,9600)){
            printf("[YCM]set_uart_baudrate is failed\n");
            return -1;
        }
        printf("[YCM]uart init is successful\n");
        shoot_total = 0;
        thread_running = true;
        struct laser_gun_s lasergundate;//定义消息结构体
        memset(&lasergundate, 0 , sizeof(lasergundate));//结构体清零

        orb_advert_t laser_gun_pub = orb_advertise(ORB_ID(laser_gun), &lasergundate);//公告这
个主题,同一个进程只用公告一次。
        while(!thread_should_exit)
       {
            read(uart_read,&data,1);
            lasergundate.laser_gun_shoot_total = data;//
            orb_publish(ORB_ID(laser_gun), laser_gun_pub, &lasergundate);//发布主题(串口读取
到的数据)
            warnx("data == %d\n",data);//打印从串口读取到的数据
        }
        warnx("[YCM]exiting");
        thread_running = false;
        close(uart_read);
        fflush(stdout);
        return 0;
}
```

2.2 消息订阅:

```
#include <stdio.h>
#include <termios.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <errno.h>
#include <drivers/drv_hrt.h>
#include <systemlib/err.h>
#include <fcntl.h>
#include <uORB/topics/laser_gun_shoot.h>
#include <uORB/topics/laser_gun.h>
static bool thread_should_exit = false;
static bool thread_running = false;
static int daemon_task;
static int hit_total;
__EXPORT int laser_gun_shoot_main(int argc, char *argv[]);
int laser_gun_shoot_thread_main(int argc, char *argv[]);
static int uart_init(const char * uart_name);       //
static int set_uart_baudrate(const int fd, unsigned int baud); //static
static void usage(const char *reason);                  //static
int set_uart_baudrate(const int fd, unsigned int baud)
{
    int speed;
    switch (baud) {
        case 9600:    speed = B9600;    break;
        case 19200:   speed = B19200;   break;
        case 38400:   speed = B38400;   break;
        case 57600:   speed = B57600;   break;
        case 115200: speed = B115200; break;
        default:
            warnx("ERR: baudrate: %d\n", baud);
            return -EINVAL;
    }
    struct termios uart_config;

    int termios_state;
    tcgetattr(fd, &uart_config);
    uart_config.c_oflag &= ~ONLCR;
    uart_config.c_cflag &= ~(CSTOPB | PARENB);
    if ((termios_state = cfsetispeed(&uart_config, speed)) < 0) {
        warnx("ERR: %d (cfsetispeed)\n", termios_state);
```

```
            return false;
        }

        if ((termios_state = cfsetospeed(&uart_config, speed)) < 0) {
            warnx("ERR: %d (cfsetospeed)\n", termios_state);
            return false;
        }

        if ((termios_state = tcsetattr(fd, TCSANOW, &uart_config)) < 0) {
            warnx("ERR: %d (tcsetattr)\n", termios_state);
            return false;
        }
        return true;
}
int uart_init(const char * uart_name)
{
        int serial_fd = open(uart_name, O_RDWR | O_NOCTTY);
        if (serial_fd < 0) {
            err(1, "failed to open port: %s", uart_name);
            return false;
        }
        return serial_fd;
}

static void usage(const char *reason)
{
        if (reason) {
            fprintf(stderr, "%s\n", reason);
        }
        fprintf(stderr, "usage: position_estimator_inav {start|stop|status} [param]\n\n");
        exit(1);
}
int laser_gun_shoot_main(int argc, char *argv[])
{
        if (argc < 2) {
            usage("[YCM]missing command");
        }

        if (!strcmp(argv[1], "start")) {
            if (thread_running) {
                warnx("[YCM]already running\n");
                exit(0);
            }
            thread_should_exit = false;
```

AMOV 阿木(UAV)社区

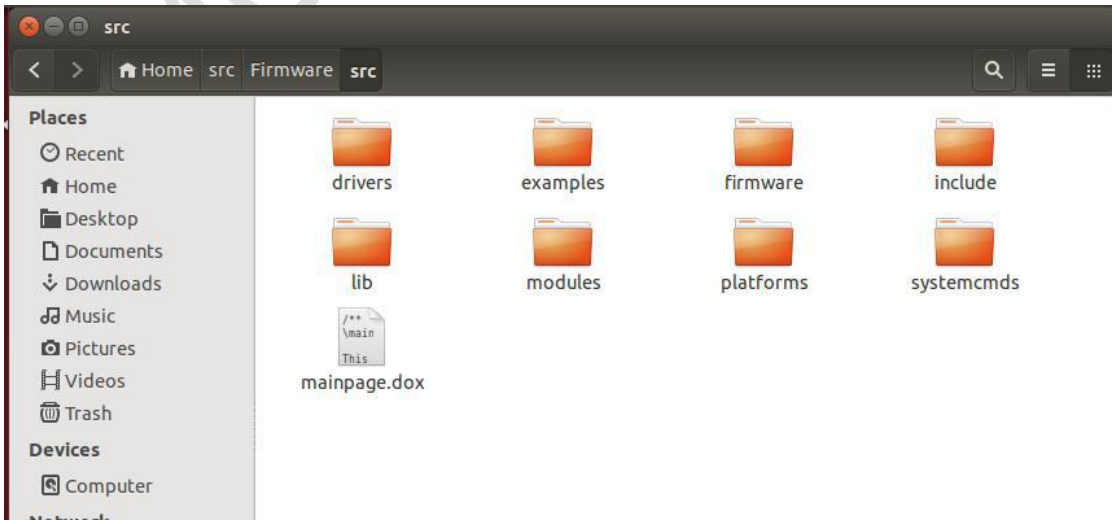
```
            daemon_task = px4_task_spawn_cmd("laser_gun_shoot",//初始化进程
                                    SCHED_DEFAULT,
                                    SCHED_PRIORITY_MAX - 5,
                                    2000,
                                    laser_gun_shoot_thread_main,
                                    (argv) ? (char * const *)&argv[2] : (char * const *)NULL);
            exit(0);
        }

        if (!strcmp(argv[1], "stop")) {
            thread_should_exit = true;
            exit(0);
        }
        if (!strcmp(argv[1], "status")) {
            if (thread_running) {
                warnx("[YCM]running");
            } else {
                warnx("[YCM]stopped");
            }
            exit(0);
        }
        usage("unrecognized command");
        exit(1);
}

int laser_gun_shoot_thread_main(int argc, char *argv[])
{
        if (argc < 2) {
            errx(1, "[YCM]need a serial port name as argument");
            usage("eg:");
        }
        const char *uart_name = argv[1];
        warnx("[YCM]opening port %s", uart_name);
        char data[5];
        int uart_read = uart_init(uart_name);
        if(false == uart_read)return -1;
        if(false == set_uart_baudrate(uart_read,9600)){
            printf("[YCM]set_uart_baudrate is failed\n");
            return -1;
        }
        printf("[YCM]uart init is successful\n");
        thread_running = true;
        int laser_gun_sub;
        laser_gun_sub = orb_subscribe(ORB_ID(laser_gun));//订阅 laser_gun 主题，返回消息句柄
```

```
    struct laser_gun_s lasergundate;
    memset(&lasergundate, 0 , sizeof(lasergundate));
    while(!thread_should_exit)
   {
        bool changed;
        orb_check(laser_gun_sub,&changed);//检查主题数据是否有更新
        if(changed)//如果数据有更新
        {
            orb_copy(ORB_ID(laser_gun),laser_gun_sub,&lasergundate);//拷贝数据到结构体
            data[0] = lasergundate.laser_gun_shoot_total;//使用这个消息的成员变量
            write(uart_read,&data,5);//从串口发布数据出去
            warnx("lasergundate.laser_gun_shoot_total
= %d\n",lasergundate.laser_gun_shoot_total);
        }
        usleep(100000);
   }
    warnx("[YCM]exiting");
    thread_running = false;
    orb_unsubscribe(laser_gun_sub);//取消订阅消息
    close(uart_read);
    fflush(stdout);
    return 0;
}
```

3 编译订阅和发布进程

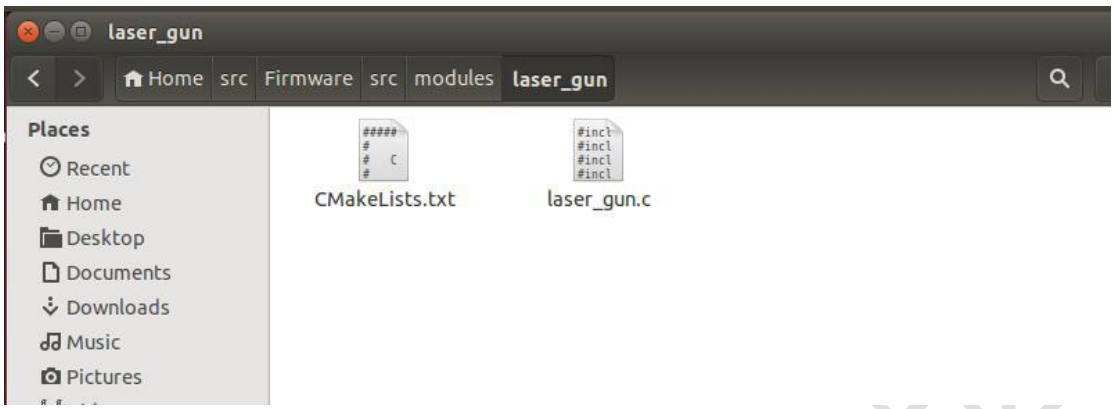
3.1 修改编译脚本

我们要在飞控内部添加，可执行应用，就是要修改编译脚本，把我们自己的应用代码编译到飞控里面去，让飞控可以执行我们自己写的代码。这部分和嵌入式 linux 下的开发非常类似。



我们可以在 deivers 里面添加传感器驱动应用，lib 是一些库文件，modules 是系统模块

应用，比如位置估计，位置控制等关键程序都在这个文件夹下面。在 pixhawk 上做二次开发，都是在这个文件夹里面做添加。我们也是在这个文件夹里面新建我们的应用程序。如下：



其中 laser_gun.c 是上面的发布一个组题的进程。里面还有个 CmakeList.txt 这个编辑脚本是我们要自己添加的，它会指导 makefile，编译出一个在飞控的 nuttx 系统中可以运行的可执行程序了，其中 CMakeLists.txt 内容如下:

```
#
############################################################################
px4_add_module(
        MODULE modules__laser_gun
        MAIN laser_gun
        STACK 3000
        SRCS
                 laser_gun.c
        DEPENDS
                platforms__common
        )
# vim: set noet ft=cmake fenc=utf-8 ff=unix :
```

CMake ▾ Tab Width: 8

就是向编译系统说明了，模块名字，栈大小，源文件名字等。编译 makefile 就会更据中国 cmakelist.txt 说明来编译这个可执行程序。

我们还要在总的编译配置文件夹下面修改
nuttx_px4fmu-v2_default.cmake 这个文件
具体的目录为: /src/Firmware/cmake/configs
我们要把自定义的模块程序的目录添加到 nuttx_px4fmu-v2_default.cmake 这个文件夹里面。

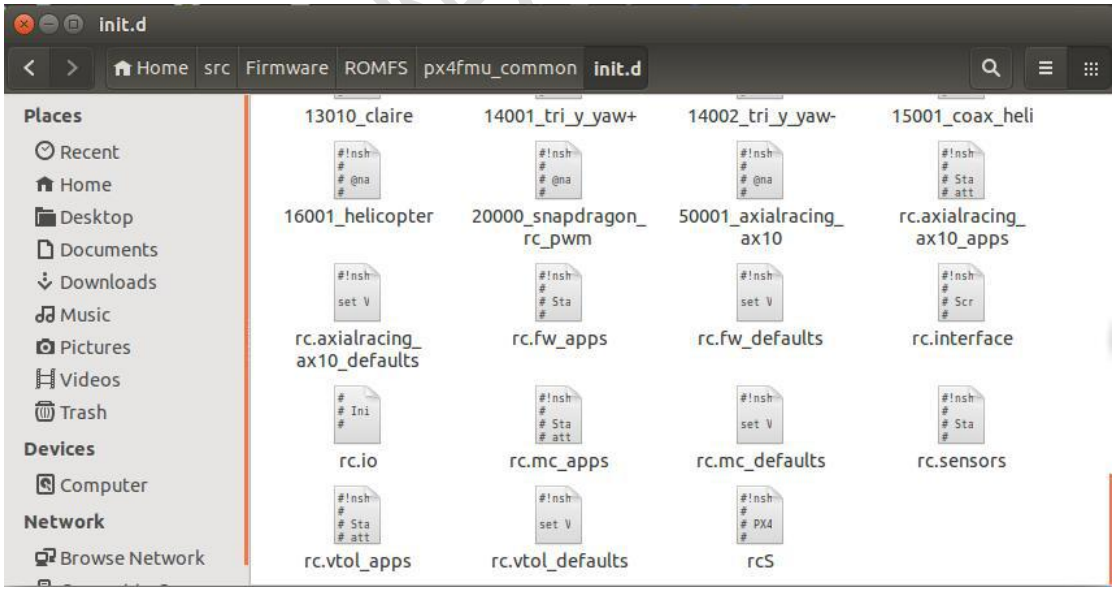
nuttx_px4fmu-v2_default.cmake × laser_gun.c × laser_gun_shoot.c × rcS × lase

```
include(nuttx/px4_impl_nuttx)

set(CMAKE_TOOLCHAIN_FILE ${PX4_SOURCE_DIR}/cmake/toolchains/Toolchain-arm-none-eabi.cmake)

set(config_uavcan_num_ifaces 2)

set(config_module_list
        #
        # Board support modules
        #
        drivers/device
        drivers/stm32
        drivers/stm32/adc
        drivers/stm32/tone_alarm
        drivers/led
        drivers/px4fmu
        drivers/px4io
        drivers/boards/px4fmu-v2
        drivers/rgbled
        drivers/mpu6000
        drivers/mpu9250
        drivers/lsm303d
        drivers/l3gd20
```

我们只要把我们自己定义的 laser_gun.c 所在的目录添加到这个类表即可。修改好了之后，我们 make px4fmu-v2_default,在飞控的文件目录里面，就有了我们自定义的可执行进程。这个程序会在 nsh 终端里面手动执行的应用程序了。具体的执行可以参考 PIX 原生固件调试技巧这篇文章。
还有订阅程序的编译，可以参考如上代码编译到飞控里面。

3.2 自定义应用进程的自启动

我们写好的 laser_gun 程序是可以飞控上电自启动的，和飞控本身的功能模块一样，上电自启动。
修改自启动是在



这个 ROMFS/px4fmu_common/init.d 下面这些文件都是和系统启动有关系的，其中 rcS 是总的启动脚本，启动了很多通用模块比如(GPS，MAVLINK 等等)，rc.mc_apps 是多旋翼的固件特有的应用模块比如(多旋翼姿态固件模块等等)，如果我们是多旋翼固件，我们就在 rc.mc_apps 添加 laser_gun start,就可以飞控上电自启动了。

AMOV 阿木(UAV)社区