源码分析—
Commander模块

Author: Qyp
2017.5.1

Commander模块介绍

整个系统的过程实现,比如起飞前的各 sensor 的校准算法实现、 安全开关是否使能、起飞前检查、飞行模式的切换、初始化LED灯和蜂鸣器、各种情况下LED灯的颜色定义及蜂鸣器定义等等。

个人感觉最重要的是执行来自vehicle_command中的命令。

模块文件分析

文件名字	作用
test文件夹	测试例程，可不用理会
accelerometer_calibration.cpp; accelerometer_calibration.h	加速度计校准程序函数和库函数
airspeed_calibration.cpp; airspeed_calibration.h	空速计校准程序函数和库函数
baro_calibration.cpp; baro_calibration.h	气压计校准程序函数和库函数
calibration_message.h	?
calibration_routines.cpp; calibration_routines.h	?
commander_help.cpp; commander_help.h	commander中用到的一些辅助函数，如LED、蜂鸣器相关函数
commander_params.c	储存commander.cpp中所需参数数值
commander.cpp	主函数
esc_calibration.cpp; esc_calibration.h	电调校准程序函数和库函数
gyro_calibration.cpp; gyro_calibration.h	陀螺仪校准程序函数和库函数
mag_calibration.cpp,mag_calibration.h	磁力计校准程序函数和库函数
PreflightCheck.cpp; PreflightCheck.h	起飞检查函数和库函数
px4_custom_mode.h	自定义飞行模式
rc_calibration.cpp; rc_calibration.h	遥控器校准程序函数和库函数
state_machine_helper.cpp; state_machine_helper.cpp	状态机帮助函数——里面有状态切换函数
CMakeLists.txt	堆栈大小：4096

commander.cpp源码分析

——订阅Topic汇总

订阅Topic[按照拷贝顺序排列]	后续用途
parameter_updated.msg 参数是否更新	进入主循环while时，更新参数
manual_control_setpoint.msg 遥控器的输入	
offboard_control_mode.msg offboard模式下的模式设置	检测是否超时(根据时间戳)
telemetry_status.msg 数传状态，最多有四个	检测连接情况
snesor_combined.msg 陀螺仪，加速度计，磁力计，气压计数据	检测气压计数值是否健康
differential_pressure.msg 气压变化值?	
system_power.msg 5V电压 是否连接USB	
safety.msg 安全开关	打开/关闭安全开关相关操作
vtol_vehicle_status.msg 垂直起降，固定翼相关	旋翼不需要理会
vehicle_global_position.msg 全局位置(经纬高以及NED方向的速度)	检查全局位置是否有效
vehicle_local_position.msg 本地位置(NED系下的位置和速度)	检查本地位置是否有效
vehicle_attitude.msg 姿态（四元数表示）和角速度	
vehicle_land_detected.msg 是否在地上，是否处于free-fall	
cpu_load.msg cpu负载和内存使用情况	
battery_status.msg 电池状态	电池电压过低的操作
subsystem_info	子系统信息，更新vehichle_status
position_setpoint_triplet.msg 位置期望值（3）	
vehicle_gps_position.msg GPS原始数据	
mission_result.msg 任务的执行结果	
geofence_result.msg 地理栅栏	超出地理栅栏相关操作
actuator_controls.msg ??	
vehicle_command.msg	handle_command()

commander.cpp源码分析

——发布Topic汇总

发布Topic	在哪发布
home_position	第一次进入commander时；handle_command()执行相关指令时
vehicle_roi	handle_command()执行相关指令时
offboard_mission	读取misision信息后，可是根本找不到这个topic？？
vehicle_control_mode	循环里 5hz
vehicle_status	循环里 5hz
actuator_armed	循环里 5hz
commander_state	循环里，这里只做log
vehicle_command_ack	执行相关command后

commander.cpp源码分析

——主函数执行流程

```
int commander_main(int argc, char *argv[])
```

·commander.cpp的主函数，根据不同输入参数执行相关指令。

```
usage: commander {start|stop|status|calibrate|check|arm|disarm|takeoff|land|transition|mode}
```

1.**start**：启动commander_thread_main线程

2.**stop**：中止commander_thread_main线程

3.**status**：打印当前相关状态，比如：系统基本信息、home点位置、data_link_loss状态

4.**calibrate+(mag/accel/gyro/level/esc/airspeed)**：执行相应校准程序

5.**check**：运行preflight_check()函数，打印是否返回值

6.**arm**：调用arm_disarm()函数来解锁

7.**disarm**：调用arm_disarm()函数来上锁

8.**takeoff**：解锁并公告一个vehicle_command(VEHICLE_CMD_NAV_TAKEOFF)命令来起飞

9.**land**：公告一个vehicle_command(VEHICLE_CMD_NAV_LAND)命令来降落

10.**transition**：公告一个vehicle_command(VEHICLE_CMD_DO_VTOL_TRANSITION)命令来转换

11.**mode+(manual/altctl/posctl/auto:mission/auto:loiter/auto:rtl/acro/offboard/stabilized/rattitude/auto:takeoff/auto:land)**：调用main_state_transition()来改变状态

commander.cpp源码分析

——主线程执行流程

```
int commander_thread_main(int argc, char *argv[])
```

1. 从commander_params.c中读取参数

2. 定义低优先级线程commander_low_prio_thread

3. 初始化LED灯和蜂鸣器

4. 公告vehicle_status、actuator_armed、vehicle_control_mode、home_postion、vehicle_roi、vehicle_command_ack、offboard_mission、commander_state，方便以后发布

5. 订阅safety、mission_result、geofence_result、manual_control_setpoint、offboard_control_mode、vehicle_global_position、vehicle_local_position、vehicle_attitude、vehicle_land_detected、vehicle_gps_position、sensor_combined、differential_pressure、vehicle_command、parameter_update、battery_status、subsystem_info、position_setpoint_triplet、system_power、actuator_controls、vtol_vehicle_status、cpuload。

6. 根据当前状态更新LED灯。至此初始化完成commander_initialized = true

7. PreflightCheck()函数，检测是否传感器是否校准等等

8. 设置通过遥控器摇杆解锁/上锁所需时间

9. 启动commander_low_prio_thread

10. 进入while (!thread_should_exit) 主循环

commander.cpp源码分析

——主循环while执行流程

```
while (!thread_should_exit)
```

1.检查是否需要更新参数及参数更新函数

2.检查各个订阅话题是否更新，若更新则拷贝

3.检测遥控器输入是否超时及遥控器解锁/上锁操作定义

4.调用set_main_state_rc()，根据RC输入来切换状态

5.检测和地面站是否有连接？没看

6.检查vehicle_command是否更新，若更新则拷贝并执行handle_command()函数

7.检测几种是否需要直接kill的情况

8.设置home_position，调用commander_set_home_position()函数。设置home_position需要GPS有效

9.调用set_nav_state()函数

10.在大约5Hz的频率内：1、set_control_mode(); 2、发布vehicle_control_mode 3、发布vehicle_status 4、发布actuator_armed

11.各种情况下LED和蜂鸣器相关操作

12.发布commander_state用于LOG

commander.cpp源码分析

——执行遥控器切换模式命令流程

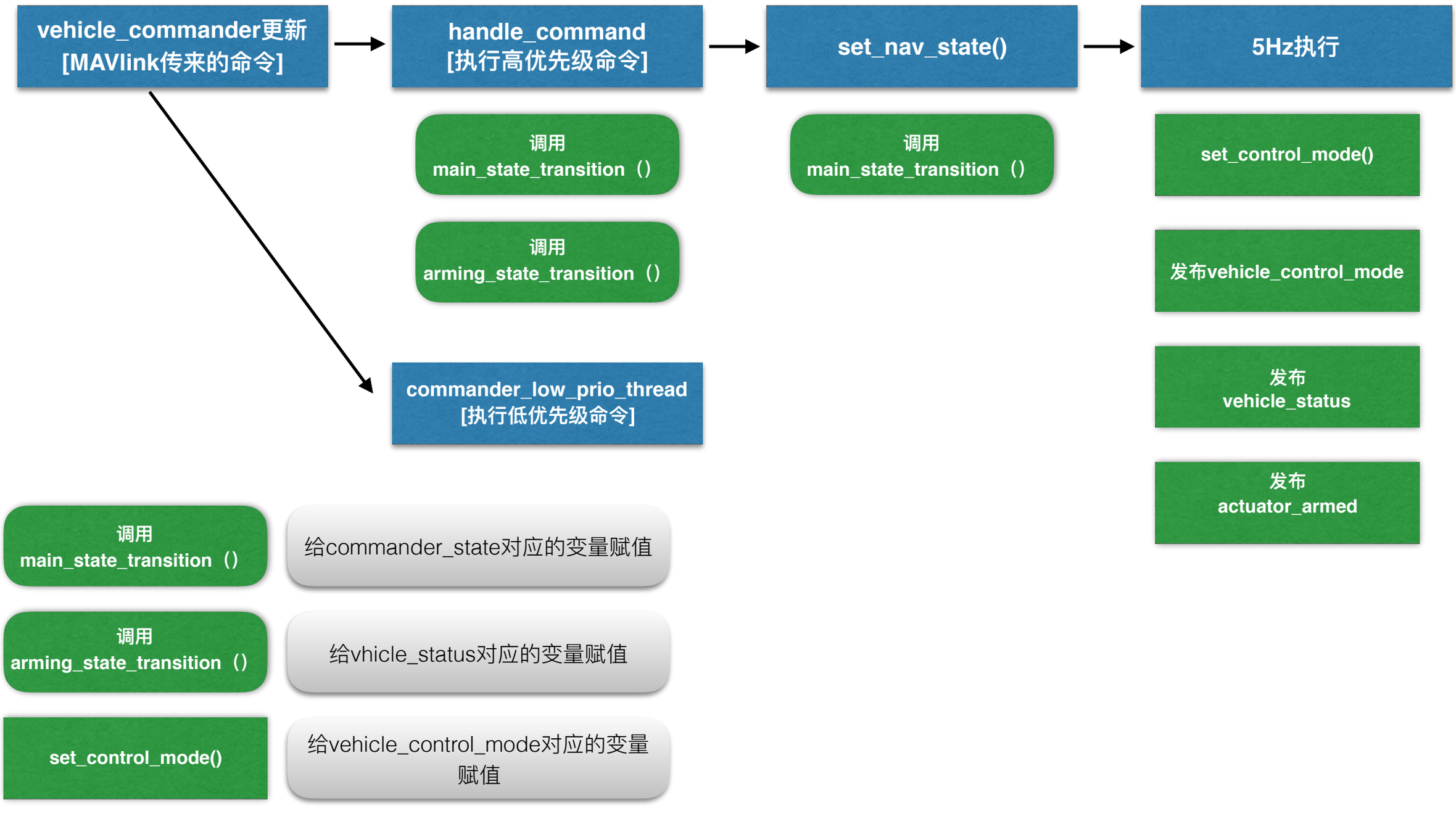
set_main_state_rc()

调用
main_state_transition（）

给commander_state对应的变量赋值

commander.cpp源码分析

——执行vehicle_command命令流程



几个重要的topic解释

Msg	重要变量	用途	发布者
vehicle_status	• nav_state • arm_state • system_type • system_id	当前的vehicle的arming状态和nav状态（手动、位置、高度、Auto、offboard模式....）	commander.cpp
vehicle_command	• command	由mavlink传输过来的命令（起飞、降落、校准....）；通过handle_command()来执行该命令	猜测是mavlink_receiver.cpp发布的
vehicle_command_ack	• command • result	vehicle_command执行的结果，由commander.cpp执行命令后发布	commander.cpp
commander_state	• main_state	通过RC或地面站传来的控制模式（手动、位置、offboard.....等13种模式），只用作logging	commander.cpp
vehicle_control_mode	• flag_control_manual_enabled • flag_control_offboard_enabled	和commander_state相对应的13种模式使能标志量，用于后续模块状态判断	commander.cpp

起飞检查函数—preflightCheck()

```
bool preflightCheck(orb_advert_t *mavlink_log_pub, /* 发送mavlink以及log的发布句柄 */
                    bool checkMag, /* 是否检测磁力计 */
                    bool checkAcc, /* 是否检测加速度计 */
                    bool checkGyro, /* 是否检测陀螺仪 */
                    bool checkBaro, /* 是否检测气压计 */
                    bool checkAirspeed, /* 是否检测空速计 */
                    bool checkRC, /* 是否检测遥控 */
                    bool checkGNSS, /* 是否检测GNSS */
                    bool checkDynamic, /* 是否动态(和加速度计校准相关) */
                    bool isVTOL, /* 是否是VTOL(旋翼不需要理会这个) */
                    bool reportFailures, /* 是否汇报失败?  */
                    bool prearm, /* 是否?(和空速计校准相关) */
                    hrt_abstime time_since_boot) /* 上电时间 */
```

- **功能**：检查各个传感器是否校准、IMU的一致性、遥控器是否校准、EKF2是否启用

- **调用情况**（每次调用时可能输入参数不同）：第一次调用—进入commander函数的时候；进入主循环后也会循环调用；执行命令vehicle_command.command = VEHICLE_CMD_PREFLIGHT_CALIBRATION时也会调用一次，用于检查是否校准成功。

起飞点设置函数—commander_set_home_position()

- 待写。。。

main_state_transition()函数

```
transition_result_t main_state_transition
                    (struct vehicle_status_s *status,  //vehicle_status
                     main_state_t new_main_state,      //要转换到的状态
                     uint8_t &main_state_prev,         //上一个状态
                    status_flags_s *status_flags,     //status_flags
                     struct commander_state_s *internal_state  //commander_status
                     )
```

- 根据需要设置的状态，给commander_state对应的变量赋值。

arming_state_transition()函数

```
transition_result_t arming_state_transition(struct vehicle_status_s *status,
                struct battery_status_s *battery,
                const struct safety_s *safety,
                arming_state_t new_arming_state,
                struct actuator_armed_s *armed,
                bool fRunPreArmChecks,
                orb_advert_t *mavlink_log_pub,     ///< uORB handle for mavlink log
                status_flags_s *status_flags,
                float avionics_power_rail_voltage,
                bool can_arm_without_gps,
                hrt_abstime time_since_boot)
```

- 很迷，vehicle_status里的arming_state是什么意思，为什么会有7个状态?
- 这个函数就是根据目前状态切换这个状态，没发布，然后给vehicle_status对应的变量赋值

起飞检查函数—preflightCheck()

```
bool preflightCheck(orb_advert_t *mavlink_log_pub, /* 发送mavlink以及log的发布句柄 */
                    bool checkMag, /* 是否检测磁力计 */
                    bool checkAcc, /* 是否检测加速度计 */
                    bool checkGyro, /* 是否检测陀螺仪 */
                    bool checkBaro, /* 是否检测气压计 */
                    bool checkAirspeed, /* 是否检测空速计 */
                    bool checkRC, /* 是否检测遥控 */
                    bool checkGNSS, /* 是否检测GNSS */
                    bool checkDynamic, /* 是否动态(和加速度计校准相关) */
                    bool isVTOL, /* 是否是VTOL(旋翼不需要理会这个) */
                    bool reportFailures, /* 是否汇报失败?  */
                    bool prearm, /* 是否?(和空速计校准相关) */
                    hrt_abstime time_since_boot) /* 上电时间 */
```

- **功能**：检查各个传感器是否校准、IMU的一致性、遥控器是否校准、EKF2是否启用

- **调用情况**（每次调用时可能输入参数不同）：第一次调用—进入commander函数的时候；进入主循环后也会循环调用；执行命令vehicle_command.command = VEHICLE_CMD_PREFLIGHT_CALIBRATION时也会调用一次，用于检查是否校准成功。

遥控器控制函数—set_main_state_rc()

- 根据遥控器的输入，调用main_state_transition()函数来切换状态

执行命令函数—handle_command()

```
bool handle_command(
                struct vehicle_status_s *status_local,
                const struct safety_s *safety_local,
                struct vehicle_command_s *cmd,
                struct actuator_armed_s *armed_local,
                struct home_position_s *home,
                struct vehicle_global_position_s *global_pos,
                struct vehicle_local_position_s *local_pos,
                struct vehicle_attitude_s *attitude,
                orb_advert_t *home_pub,
                orb_advert_t *command_ack_pub,
                struct vehicle_command_ack_s *command_ack,
                struct vehicle_roi_s *roi,
                orb_advert_t *roi_pub)
```

- **功能：**执行vehicle_command中的命令，比如起飞降落模式切换（注：有一些命令会被忽略，会在low prio loop里执行）

- **调用情况：**在主循环中当检测到vehicle_command更新时执行

- **输出：**发布command_ack.msg，可能会发布home_position.msg、vehicle_roi.msg

handle_command()-源码分析

1. 确认system_id 和 component_id

2. 判断vehicle_command.command命令类型，并执行对应指令

3. 如果是切换模式命令VEHICLE_CMD_DO_SET_MODE：解锁相关操作，设置并发布home_position，调用main_state_transition()函数，该函数将改变的状态赋值到commander_status对应的变量中。

4. 如果是解锁/上锁命令VEHICLE_CMD_COMPONENT_ARM_DISARM：先调用arm_disarm()函数，arm_disarm()函数调用arming_state_transition()函数，该函数讲改变的状态赋值到vehicle_status对应的变量中。如果解锁会设置并发布home_position。

5. 还有很多别的命令，这里不一一详述，具体看源码。

set_nav_state()函数

```
bool set_nav_state(struct vehicle_status_s *status,
                   struct commander_state_s *internal_state,
                   orb_advert_t *mavlink_log_pub,
                   const bool data_link_loss_enabled,
                   const bool mission_finished,
                   const bool stay_in_failsafe,
                   status_flags_s *status_flags,
                   bool landed,
                   const bool rc_loss_enabled,
                   const int offb_loss_act,
                   const int offb_loss_rc_act)
```

- 根据commander_status(其实就是根据main_state_transition()这个函数的赋值)来设置状态，不发布，赋值到vehicle_status对应的变量中

- 还没细看，有各种判断条件。真正的状态如何转换应该在个函数里面

set_control_mode()函数

- 根据vehicle_status来赋值vehicle_control_mode对应的变量

commander_low_prio_thread

- 执行低优先级的vehicle_command，比如请求校准传感器等等命令。