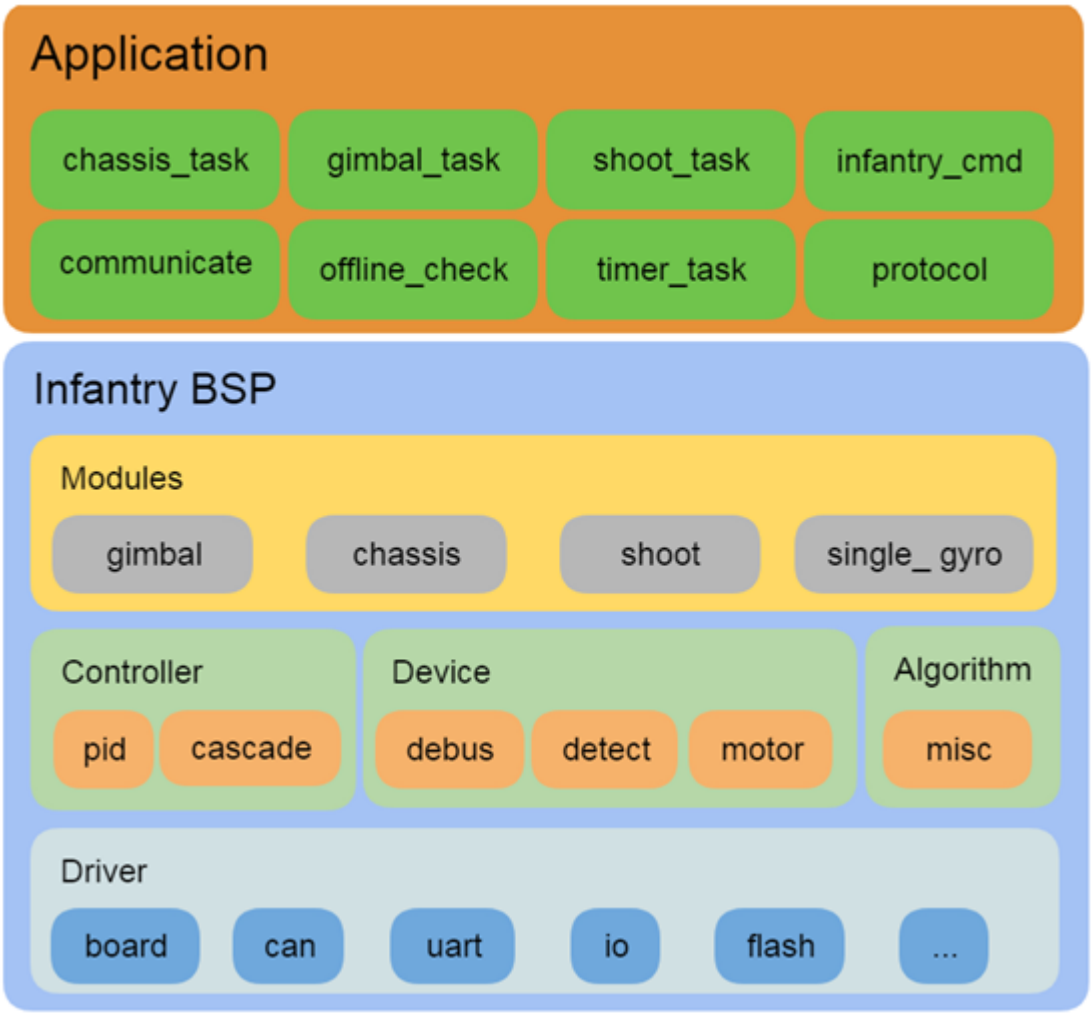

modules

gimbal.h

1. `struct gimbal_param`：云台yaw、pitch编码器中点
2. `struct gimbal_p_y`：用于定义yaw、pitch陀螺仪、编码器角度值
3. `struct gimbal_rate`：用于定义yaw、pitch速度值
4. `struct gimbal_sensor`：定义了陀螺仪角度和速度
5. `struct gimbal`：云台结构体（包括上述结构体、电机以及控制相关的参数）
6. `struct gimbal_info`：云台信息（云台模式、两轴编码器角度、陀螺仪角度、速率）

gimbal.c

1. `int32_t gimbal_cascade_register(struct gimbal *gimbal, const char *name, enum device_can can)`：云台双环控制初始化（yaw、pitch电机接入、设定为编码器模式并进行pid参数初始化）
2. `int32_t gimbal_set_pitch_delta(struct gimbal *gimbal, float pitch)`：设置pitch位移增量（陀螺仪、编码器两种模式调用gimbal_set_pitch_angle传入**输入角度**）

   yaw同理

3. `int32_t gimbal_set_pitch_speed(struct gimbal *gimbal, float pitch)`：设置pitch速度(两种模式下调用gimbal_set_pitch_angle) 用于infanty_cmd.c

   yaw同理

4. `int32_t gimbal_set_pitch_angle(struct gimbal *gimbal, float pitch)`：设置pitch角度

   yaw同理

5. `int32_t gimbal_set_pitch_mode(struct gimbal *gimbal, uint8_t mode)`：设置pitch模式：陀螺仪模式、编码器模式

   yaw同理

6. `int32_t gimbal_set_offset(struct gimbal *gimbal, uint16_t yaw_ecd, uint16_t pitch_ecd)`：设置云台编码器中点值

7. `int32_t gimbal_pitch_enable(struct gimbal *gimbal)`：pitch使能开启

   `int32_t gimbal_pitch_disable(struct gimbal *gimbal)`：pitch使能关闭

   yaw轴同理

8. `int32_t gimbal_execute(struct gimbal *gimbal)`：云台运行

9. `int32_t gimbal_rate_update(struct gimbal *gimbal, float yaw_rate, float pitch_rate)`：传入imu速率值作为云台两轴的速率

10. `int32_t gimbal_pitch_gyro_update(struct gimbal *gimbal, float pitch)`：传入陀螺仪pitch角度值作为云台pitch角度

    yaw同理（陀螺仪+编码器）

11. `int32_t gimbal_get_info(struct gimbal *gimbal, struct gimbal_info *info)`：云台获取角度、速率等信息存入gimbal_info

12. `gimbal_t gimbal_find(const char *name)`：判断是否连接云台

13. `static int16_t gimbal_get_ecd_angle(int16_t raw_ecd, int16_t center_offset)`：获取云台编码器角度

14. `static int32_t gimbal_set_yaw_gyro_angle(struct gimbal *gimbal, float yaw, uint8_t mode)`：获取云台yaw陀螺仪角度

chassis.h

1. `struct chassis_acc`：底盘加速度
2. `struct chassis`：

```
struct object parent;
struct mecanum mecanum;
struct chassis_acc acc;
struct motor_device motor[4];
struct pid motor_pid[4];
struct pid_feedback motor_feedback[4];
struct controller ctrl[4];
```

3. `struct chassis_info`：底盘速度、位置、角度轮速等信息

chassis.c

1. `int32_t chassis_pid_register(struct chassis *chassis, const char *name, enum device_can can)`：底盘pid参数、电机信息、麦轮信息等初始化
2. `int32_t chassis_execute(struct chassis *chassis)`：底盘运行
3. `int32_t chassis_gyro_updata(struct chassis *chassis, float yaw_angle, float yaw_rate)`：从陀螺仪更新角度和速度数据并存入底盘结构体的麦轮陀螺仪信息中
4. `int32_t chassis_set_speed(struct chassis *chassis, float vx, float vy, float vw)`：设置chassis->mecanum.speed(vx, vy, vw)
5. `int32_t chassis_set_acc(struct chassis *chassis, float ax, float ay, float wz)`：设置底盘加速度，chassis->acc
6. `int32_t chassis_set_vw(struct chassis *chassis, float vw)`：单独设置底盘麦轮自转速度（未使用）
7. `int32_t chassis_set_vx_vy(struct chassis *chassis, float vx, float vy)`：仅设置底盘x, y方向速度（未使用）
8. `int32_t chassis_set_offset(struct chassis *chassis, float offset_x, float offset_y)`：设置x, y轴底盘中心
9. `int32_t chassis_get_info(struct chassis *chassis, struct chassis_info *info)`：获取底盘参数存入`struct chassis_info`
10. `chassis_t chassis_find(const char *name)`：调用`object_find`寻找底盘，判断其是否连接
11. `int32_t chassis_enable(struct chassis *chassis)`：底盘使能开启

    `int32_t chassis_disable(struct chassis *chassis)`：底盘使能关闭
12. `static int32_t motor_pid_input_convert(struct controller *ctrl, void *input)`：将电机的速度反馈作为pid的反馈值，即此处使用速度pid

shoot.h

1. `struct shoot_param`：射击参数
2. `struct shoot_target`：射击次数、两个摩擦轮速度、电机速度
3. `struct shoot`：

```
struct object parent;
struct shoot_param param;
enum shoot_state state;
uint8_t cmd;
uint8_t trigger_key;
uint16_t fric_spd[2];
```

> uint32_t shoot_num;
>
> uint32_t block_time;
>
> struct shoot_target target;
>
> struct motor_device motor;
>
> struct pid motor_pid;
>
> struct pid_feedback motor_feedback;
>
> struct controller ctrl;

shoot.c

1. `int32_t shoot_pid_register(struct shoot *shoot, const char *name, enum device_can can)`：射击pid参数、电机信息、摩擦轮转速、卡弹转速等参数初始化
2. `int32_t shoot_set_fric_speed(struct shoot *shoot, uint16_t fric_spd1, uint16_t fric_spd2)`：设置摩擦轮速度
3. `int32_t shoot_get_fric_speed(struct shoot *shoot, uint16_t *fric_spd1, uint16_t *fric_spd2)`：获取摩擦轮速度
4. `int32_t shoot_set_cmd(struct shoot *shoot, uint8_t cmd, uint32_t shoot_num)`：cmd为单发时目标发射数加一
5. `int32_t shoot_execute(struct shoot *shoot)`：射击运行
    -
6. `int32_t shoot_state_update(struct shoot *shoot)`：射击状态更新，检查是否触发射击
7. `int32_t shoot_set_turn_speed(struct shoot *shoot, uint16_t speed)`：设置射击频率
8. `shoot_t shoot_find(const char *name)`：调用`object_find`寻找射击部分，判断其是否连接
9. `int32_t shoot_enable(struct shoot *shoot)`：射击使能开启

   `int32_t shoot_disable(struct shoot *shoot)`：射击使能关闭
10. `static int32_t shoot_block_check(struct shoot *shoot)`：卡弹检测
11. `static int32_t shoot_cmd_ctrl(struct shoot *shoot)`：射击各模式下的工作状态（初始化、单发、连发、停止、卡弹)
12. `static int32_t shoot_fric_ctrl(struct shoot *shoot)`：摩擦轮射速控制
13. `static int32_t shoot_pid_input_convert(struct controller *ctrl, void *input)`：摩擦轮电机速度反馈作为pid的反馈

single_gyro.h

1. `struct single_gyro`：

> uint32_t std_id;
>
> float yaw_gyro_angle;
>
> float yaw_gyro_rate;

single_gyro.c

1. `int32_t single_gyro_can_send_register(gyro_can_send_t send)`：底盘陀螺仪can通信初始化
2. `int32_t single_gyro_update(struct single_gyro *gyro, uint32_t std_id, uint8_t can_rx_data[])`：底盘陀螺仪信息传入
3. `int32_t single_gyro_reset(struct single_gyro *gyro)`：陀螺仪重置
4. `int32_t single_gyro_adjust(struct single_gyro *gyro)`：

application

gimbal_task.c

1. `void gimbal_task(void const *argument)`：云台主程序

   > 找到云台
   >
   > 设置补偿
   >
   > 云台软实时操控初始化
   >
   > imu温度控制初始化
   >
   > while
   >
   > > 遥控器状态接收并作为输入操控机器人
   > >
   > > 初始状态云台回中
   > >
   > > 云台自动寻找中点
   > >
   > > 陀螺仪数据实时更新
   > >
   > > 云台运行

2. `static int32_t gimbal_imu_updata(void *argc)`：陀螺仪数据实时更新
3. `static void imu_temp_ctrl_init(void)`：陀螺仪温度pid
4. `static int32_t imu_temp_keep(void *argc)`：温度保持
5. `void send_gimbal_current(int16_t iq1, int16_t iq2, int16_t iq3)`：直接向电机发送电流值
6. `static void auto_gimbal_adjust(gimbal_t pgimbal)`：云台自动寻找中点
7. `void gimbal_auto_adjust_start(void)`：自动调整中点的flag置1
8. `uint8_t get_gimbal_init_state(void)`：获取云台状态（是否为初始状态）
9. `void gimbal_init_state_reset(void)`：重置为初始状态，同时初始化yaw、pit斜坡函数参数
10. `static void gimbal_state_init(gimbal_t pgimbal)`：初始状态云台回中

chassis_task.c

1. `void chassis_task(void const *argument)`：底盘主程序

   > 寻找底盘
   >
   > 软实时初始化

> 底盘跟随pid初始化
>
> while
>
> > 遥控器状态接收并作为输入操控机器人
> >
> > 底盘运行

2. `int32_t chassis_set_relative_angle(float angle)`：设置底盘跟随角度

shoot_task.c

1. `void shoot_task(void const *argument)`：射击主程序

> 寻找射击装置
>
> while
>
> > 遥控器状态接收并作为输入操控机器人
> >
> > 射击运行

2. `int32_t shoot_firction_toggle(shoot_t pshoot)`：切换摩擦轮速度

timer_task.h

1. `struct soft_timer`：软实时

> uint8_t id;
>
> uint32_t ticks;
>
> void *argc;
>
> int32_t (*soft_timer_callback)(void *argc);

timer_task.c

1. `void timer_task(void const *argument)`：实时控制主程序
2. `int32_t soft_timer_register(int32_t (*soft_timer_callback)(void *argc), void *argc, uint32_t ticks)`：实时参数初始化

object

object.h

1. `struct object`：

> char name[OBJECT_NAME_MAX_LEN];
>
> enum object_class_type type;
>
> uint8_t flag;
>
> list_t list;

2. `struct object_information`：设备信息（设备类别）

object.c

1. `struct object_information * object_get_information(enum object_class_type type)`：获取对象信息
2. `int32_t object_init(struct object *object, enum object_class_type type, const char *name)`：对象初始化
3. `object_t object_find(const char *name, enum object_class_type type)`：寻找对象
4. `void object_detach(object_t object)`：对象分离

device

device.h

1. `struct device`：

> struct object parent;
>
> enum device_type type;
>
> uint16_t flag;
>
> uint16_t open_flag;
>
> uint8_t ref_count;
>
> uint8_t device_id;
>
> void *user_data;

device.c

1. `int32_t device_register(struct device *dev,const char *name,uint16_t flags)`：设备初始化
2. `int32_t device_unregister(struct device *dev)`：设备注销
3. `device_t device_find(const char *name)`：调用`object_find`寻找设备

motor.h

1. `struct motor_data`：

> uint16_t ecd;
>
> uint16_t last_ecd;
>
> int16_t speed_rpm;
>
> int16_t given_current;
>
> int32_t round_cnt;
>
> int32_t total_ecd;
>
> int32_t total_angle;
>
> int32_t ecd_raw_rate;

> uint32_t msg_cnt;
>
> uint16_t offset_ecd;

2. `struct can_msg`： ?
3. `struct motor_device`：

> struct device parent;
>
> struct motor_data data;
>
> enum device_can can_periph;
>
> uint16_t can_id;
>
> uint16_t init_offset_f;
>
> int16_t current;
>
> void (*get_data)(motor_device_t, uint8_t*);

motor.c

1. `int32_t motor_device_register(motor_device_t motor_dev,const char *name,uint16_t flags)`：电机设备初始化

> 将编码器信息存入电机设备指针
>
> 设备初始化

2. `void motor_device_can_send_register(fn_can_send fn)`：电机can通信初始化
3. `motor_device_t motor_device_find(const char *name)`：寻找电机
4. `motor_data_t motor_device_get_data(motor_device_t motor_dev)`：获取电机数据
5. `int32_t motor_device_set_current(motor_device_t motor_dev, int16_t current)`：设置对应电机电流

algorithm

mecanum.h

1. `struct mecanum_structure`：麦轮结构参数
2. `struct mecanum_position`：麦轮位置参数
3. `struct mecanum_speed`：麦轮速度参数
4. `struct mecanum_gyro`：麦轮陀螺仪角度和速度
5. `struct mecanum`： 定义麦轮结构体，包含上述四项参数
6. `struct mecanum_motor_fdb`：底盘电机编码器、速度返回值

mecanum.c

1. `void mecanum_calculate(struct mecanum *mec)`：四个麦轮电机速度解算
2. `void mecanum_position_measure(struct mecanum *mec, struct mecanum_motor_fdb wheel_fdb[])`：

pid.h

1. `struct pid_param`：pid参数（p、i、d、最大误差、最大输出、积分限）
2. `struct pid`：pid结构体

pid.c

1. `static void pid_param_init( struct pid *pid, float maxout, float inte_limit, float kp, float ki, float kd)`：pid参数初始化
2. `static void pid_reset(struct pid *pid, float kp, float ki, float kd)`：代码运行时可进行pid参数的修改
3. `float pid_calculate(struct pid *pid, float get, float set)`：计算增量pid和位置pid
4. `void pid_struct_init( struct pid *pid, float maxout, float inte_limit, float kp, float ki, float kd)`：pid结构初始化

ramp.h

1. `typedef struct ramp_t`：定义了斜坡函数参数

ramp.c

1. `void ramp_init(ramp_t *ramp, int32_t scale)`：斜坡函数初始化，定义scale
2. `float ramp_calculate(ramp_t *ramp)`：生成斜坡函数

controller

controller.h

1. `struct controller`：

```
struct object parent;
enum controller_type type;
uint8_t enable;
void *param;
void *feedback;
float input;
float output;
int32_t (*convert_feedback)(struct controller *ctrl, void *feedback);
int32_t (*control)(struct controller *ctrl, void *param, void *feedback, float input);
```

controller.c

1. `int32_t controller_register(struct controller *ctrl, const char *name, enum controller_type type, void *param, void *feedback, uint8_t enable)`：控制器初始化
2. `int32_t controller_unregister(struct controller *ctrl)`：控制器注销
3. `controller_t controller_find(const char *name)`：调用`object_find`寻找控制器
4. `int32_t controller_set_param(struct controller *ctrl, void *param)`：控制器设置参数（未使用）
5. `int32_t controller_execute(struct controller *ctrl, void *feedback)`：控制器运行
6. `int32_t controller_set_input(struct controller *ctrl, float input)`：传入输入值
7. `float controller_get_output(struct controller *ctrl, float *out)`：获取返回值
8. `int32_t controller_enable(struct controller *ctrl)`：使能开启
9. `int32_t controller_disable(struct controller *ctrl)`：使能关闭

pid_controller.h

1. `struct pid_feedback`：pid的反馈值
2. `struct cascade`：双环级联pid

   > struct pid outer; struct pid inter;

3. `struct cascade_feedback`：双环pid的两个反馈

pid_controller.c

1. `int32_t pid_controller_register(struct controller *ctrl, const char *name, struct pid *param, struct pid_feedback *feedback, uint8_t enable)`：

   调用`controller_register`创建控制器

2. `int32_t pid_control(struct controller *ctrl, void *param, void *feedback, float input)`：单环pid参数、反馈值、pid计算、输出
3. `int32_t cascade_controller_register(struct controller *ctrl, const char *name, struct cascade *param, struct cascade_feedback *feedback, uint8_t enable)`：

   调用`controller_register`创建双环控制器

4. `int32_t cascade_control(struct controller *ctrl, void *param, void *feedback, float input)`：双环pid参数、输入、两次pid计算、输出