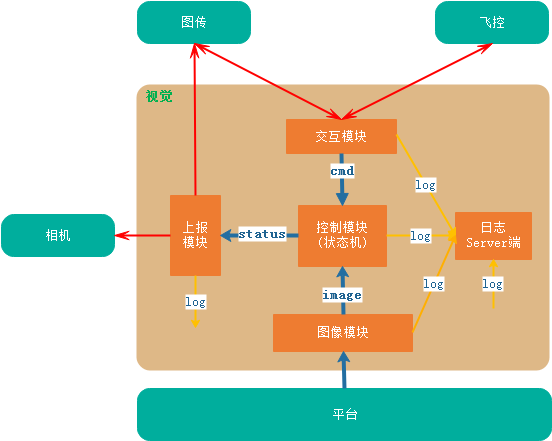
视觉控制模块关键设计说明

1. 视觉模块整体架构

视觉模块整体框架如下图所示:



1.整个视觉与外部交互的模块有平台、相机、图传、飞控.

2.视觉根椐功能划分以下模块:

a.上报模块:

1)负责上报控制模块状态信息给app

2)负责上报IMU信息给相机

b.交互模块:负责与图传、飞控进行信息交互

c.图像模块:负责从平台端获取图像

d.日志server端:负责视觉各模块的打印

d.控制模块:负责控制视觉的状态

1. 通用状态机设计
2. 关键数据描述

2.状态转换原理

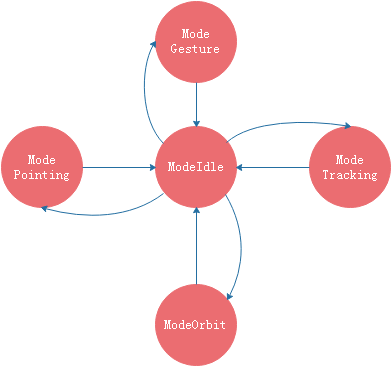
1.1 状态初始化原理

1.2 状态转换原理

1. vision状态机设计

1.模式

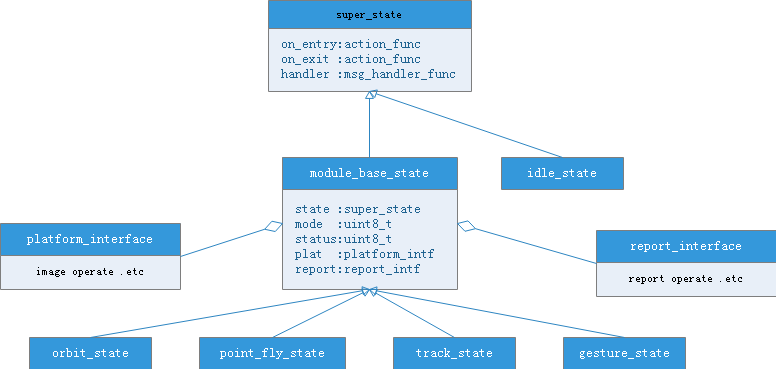
1.1 模式转换  
 飞机目前有以下模式:idle,tracking,orbit,pointing,gesture,其中Idle作为中转状态，各模式转换如下图所示:



1.2 模式类图  
 对于每一个模式,继承状态机超类,实现状态机超类定义的on\_entry,on\_exit,handler函数.

由于idle模式主要作用用于中转,所以直接继承状态机超类,而其它模式依赖于平台提供图像数据并实时上传状态信息给APP,故定义平台接口与上报接口,对模式状态机与平台、上报模块进行解耦。

类图如下:



2.模式下的操作  
2.1操作接口  
 用户切换模式后,可以操控飞机,有以下基本操作:开始,停止,暂停,恢复。虽然每个模式的基本操作一样,但是具体行为可能不同,所以抽象出行为接口类:

struct base\_action {

int (\*init)(void\* arg);

int (\*deinit)(void\* arg);

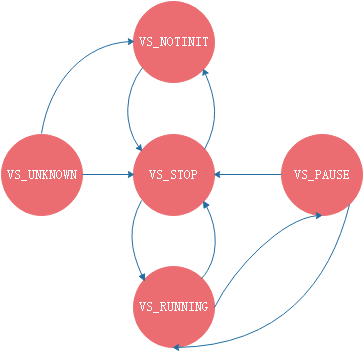
int (\*start)(void\* arg);

int (\*stop)(void\* arg);

int (\*pause)(void\* arg);

int (\*resume)(void\* arg);

}

2.2操作状态转换  
 飞机进入模式以后有以下状态:vs\_not\_init,vs\_stop,vs\_pause,vs\_running,vs\_unknown,它们的转换关系如下:  
  
a.状态较少,转换简单  
b.后期增加的状态的需求概率较小  
采用状态设计模式,后期若有需求可快速迁移到状态机实现.

又由于请求是由app下发,目前针对每个模式的请求命令是不一样的,但实际行为是一致的,可增加适配接口实现,将协议请求转换成内部请求,从而达到行为与协议分离。

整个类图设计如下:  
