Contents

第一章 引言 3
1.1 项目背景 3
1.2 总体要求 3
第二章 软件开发 5
2.1 需求分析 5
2.1.1 引言 5
2.1.2 功能需求 5
2.2 总体设计 7
2.2.1 处理流程 7
2.2.2 总体结构和模块外部设计 7
2.3 详细设计 11
2.3.1 实时系统 11
2.3.2 主控模块 11
2.3.3 通信模块 12
2.3.4 导航控制器 12
2.3.5 制导控制器 12
2.3.6 姿态控制器 14
2.4 编码和单元测试 15
第三章 算法 17
3.1 导航控制-路径管理 17
3.1.1 航点控制逻辑1 17
3.1.2 航点控制逻辑2 18
3.1.3 算法优化 19
3.2 制导控制 22
3.2.1 集群(主机)控制逻辑 22
3.2.2 编队控制逻辑 24
3.3 自组织算法 25
第四章 仿真 27
4.1 软件仿真效果展示 27
4.2 硬件仿真效果展示 27

第一章 引言

1.1 项目背景

无人飞行器(Unmanned Air Vehicle. UAV)具有广阔的应用前景，是近年来高技术研究的热点目标之一。随着社会的发展和经济的进步，无人机的发展逐步趋于成熟。无人机的应用有不同的场景和方式，例如，应用于军事勘探和追踪，农药的无人喷洒，民用影像航拍等。以 2017 年发生飓风的波多黎各为例，利用最合理的方法向该地投放物资. 同时也伴随着计算机技术，通信技术，传感器技术，电池技术等的飞速发展，开展微型UAV研究并把它运用到军事或民用中已经成为可能。本论文以PX4控制逻辑为基础，进行了一定的改进，实现了更加平滑的转弯控制，使得无人机能以更地高效率执行航线。

无人机系统通常较有人机复杂。系统的基本组成主要包括无人机、地面控制站、发射回收装置及地面数据终端。控制站通常提供三个工作站：完成任务控制、无人机控制及图像分析。地面数据终端完成与无人机的信息交流。另外，有些无人机系统还增加了卫星作为其系统的组成部分，增加GPS以提高无人机定位的精度。无人机研究的关键技术包括：飞行器系统技术一用于预测故障的综合性管理系统和满足故障容错要求的制导、导航和控制系统；人工智能技术一用于解决无人机的自主程度问题；通信技术一宽带、大数据量的数据链技术可以使无人机远距离快速传递信息，实施超视距控制.

无人机已被各国广泛用于国防建设和民用领域，随着无人机技术的深入研究，无人机自主集群系统也有理一定的发展, 目前已经达到了能够通过紧密的协作完成各种复杂多变的任务，并且具备卓越的协调性、智能性和自主性，同时无人机集群已成为无人机研究的一个重要方向。

无人机集群作战是指一组具备部分自主能力的无人机通过相关的辅助操作，在作战指挥系统监控下，完成作战任务的过程。美国国防部在《无人机系统路线图2005-2030》中指出，到2025年，集群无人机将具备战场认知能力，能够实现完全自组织作战。

1.2 总体要求

国外对无人机集群技术的研究开始较早，侧重于无人机集群技术的整体性研究。主要对无人机集群技术中的无人机集群结构框架、控制与优化技术、任务管理与协同等进行深入研究，并且取得了一定的成效。如美国国防部高级研究计划局主导的自主编队混合控制项目（MICA），该项目对协同任务分配、协同路径规划、混合主动与自主编队控制、协同跟踪、信息共享等有关无人机集群的技术进行全面的研究。美国广域搜索弹药项目（Wide Area Search Munitions，WASM）通过建立Multi UAV协同控制仿真平台，采用分层控制与优化技术，研究了复杂任务背景下如何增强无人机集群协同全域搜索与打击能力。 2006年，美国空军技术研究院基于进化机制的同构或异构的无人机集群自组织行为，建立自组织框架，使集群无人机通过自然选择和遗传变异，实现自身和行为的不断优化，产生对环境和作战任务的自适应能力。

国内由于现有技术的限制，无人机集群技术整体研究处于起步阶段，但对多无人机自主协同控制中的信息感知与传输、编队与队形、避障与避碰等技术研究较为深入，理论成果较多。如在多机协同方面，基于分层递阶控制思想，研究了多机任务分配、多机航迹规划、多机编队控制等内容。在群体智能研究方面，北京航空航天大学段海滨教授长期从事基于仿生智能的无人机自主控制研究，研究成果显著，主

要研究了基于生物群集行为特性，建立了鸽群、雁群、狼群等典型生物群体智能模型，研究了从生物群体智能到无人机集群控制的理论映射。

对无人机集群实施有效的控制是完成各种复杂集群任务的基础。无人机集群任务规划问题分解为决策层、路径规划层、轨迹生成层和控制层，其中，决策层负责无人机集群系统中的任务规划与分配、避碰和任务评估等；路径规划层负责将任务决策数据转换成航路点，以引导无人机完成任务、规避障碍；轨迹生成层根据无人机姿态信息、环境感知信息生成无人机通过航路点的可飞路径；控制层控制无人机按照生成的轨迹飞行。无人机集群任务规划分层结构如图1.1所示.

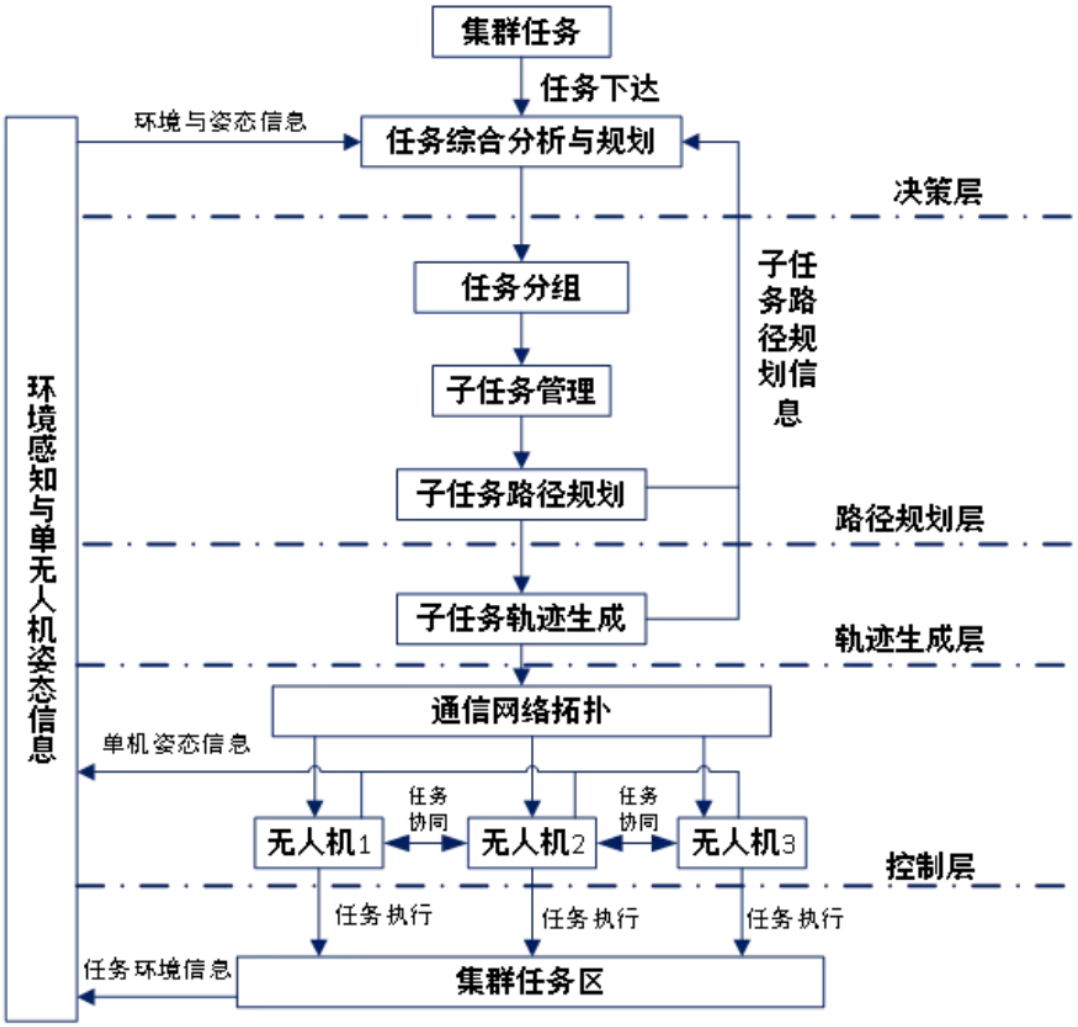


图 1.1. 无人机集群任务规划分层结构

第二章 软件开发

2.1 需求分析

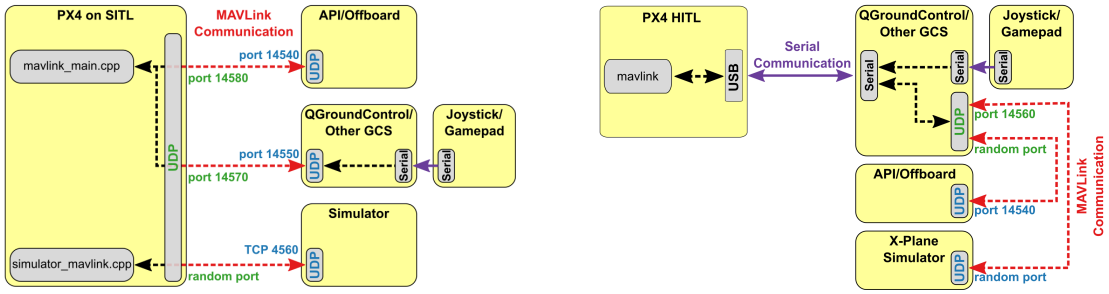
2.1.1 引言

编写目的

本需求的编写目的在于研究无人机集群编队任务的开发途径和应用方法, 为以后的开发工作提供可靠的依据.

项目背景

本课题的研究开发依赖于ROS平台, PX4开源飞控代码, QGC, 以及软件仿真平台(Gazebo), 硬件仿真平台(XPlane). 各自对应的关系如图2.1所示.



(a) Software In the Loop (b) Software In the Loop

图 2.1. 仿真平台

数据字典

(1) QGC_COMMAND指令:

- 用处: 集群地面站为每一架飞机发送的指令
- 组成: 飞机的id编号 + 飞机分组之后的组编号 + 主从机标志位 + 任务类型 + 其他的一些数据

2.1.2 功能需求

把系统主要功能包括实时子系统, 导航子系统, 制导子系统, 姿态子系统, 通信子系统, 主控, 以及飞行日志记录. 其中实时子系统主要负责和PX4进行交互, 获取当前系统的实时位姿数据, 供其他子系统进行使用; 导航子系统包含了路径管理和期望航点的获取, 将其下发给制导子系统; 制导子系统包含了直线控制逻辑和盘旋控制逻辑, 并依托于控制逻辑计算期望姿态数据, 最后下发给姿态子系统; 姿态子系统负责

将制导子系统获得的期望姿态进行PID控制,最终以某种方式将控制量发送给PX4; 通信子系统在集群系统数据流下担任了主要角色, 它是无人机群组之间, 无人机相互之间, 无人机与集群地面站之间的沟通桥梁; 飞行日志记录主要担任着飞行数据的保存.

系统功能图如下:

2.2 总体设计

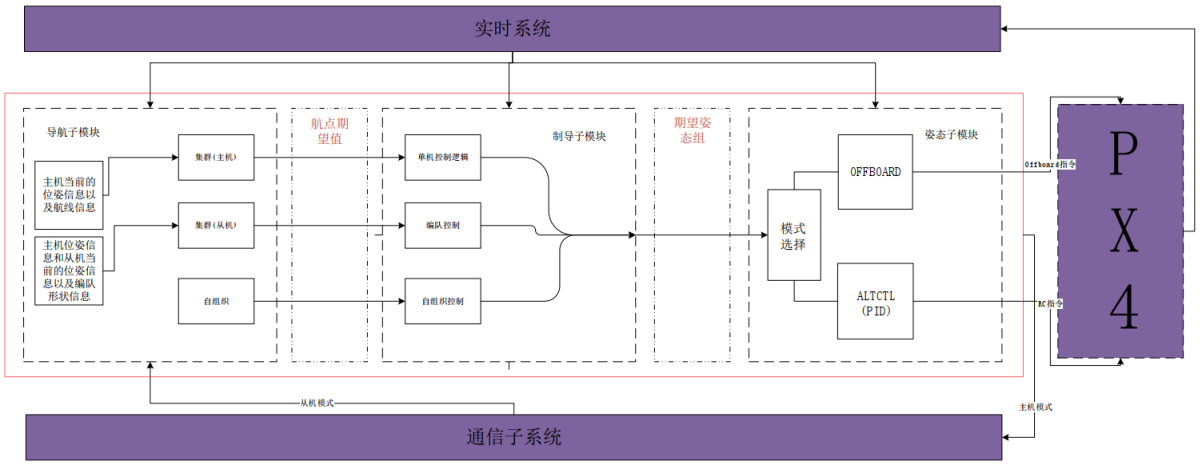


图 2.2. UAV逻辑图

2.2.1 处理流程

集群编队, 当所有飞机全部上电起飞后(2.3), 等待集群地面站为每一架飞机发送指令(QGC_COMMNAD), 对应无人机接收到集群地面站发送给自己的指令, 进行解析, 指定无人机当前的执行任务以及一个所属属性.

若指定当前无人机为主机, 那么主机执行特定算法, 计算期望姿态组, 按照某种特定方式发布给PX4, 进而交给PX4的执行器进行执行, 同时广播自己特定的位姿数据到同组的无人机, 为后序的编队或者自组织服务, 数据流图见2.4; 若指定当前无人机为从机, 那么接收其他无人机广播过来的位姿信息, 筛选出同组主机位姿信息, 进行计算, 完成特定任务, 见数据流图2.5.

2.2.2 总体结构和模块外部设计

因每一架飞机根据任务分配的不同, 所属于的组名, 以及在组内所担任的角色也是变化的, 所以需要将主机和从机的两套执行控制逻辑整合成为一套, 在内部根据飞机控制模式进行判断选择所要执行的逻辑. 所以当前无人机的执行控制逻辑如图2.2所示.

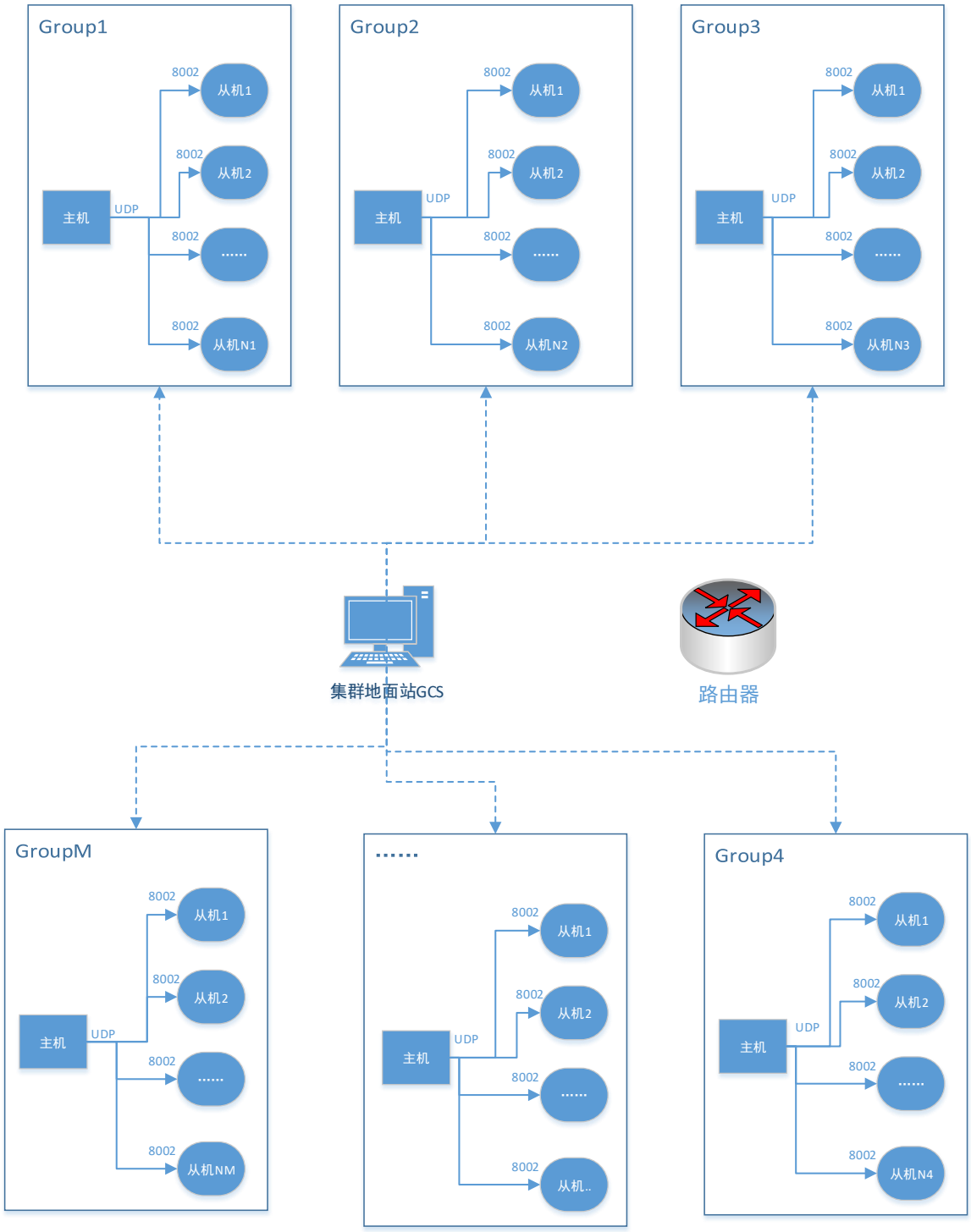

Group1
主机
UDP
8002
从机1
8002
从机2
8002
.....
8002
从机N1
Group2
主机
UDP
8002
从机1
8002
从机2
8002
.....
8002
从机N2
Group3
主机
UDP
8002
从机1
8002
从机2
8002
.....
8002
从机N3
集群地面站GCS
路由器
GroupM
主机
UDP
8002
从机1
8002
从机2
8002
.....
8002
从机NM
.....
主机
UDP
8002
从机1
8002
从机2
8002
.....
8002
从机..
Group4
主机
UDP
8002
从机1
8002
从机2
8002
.....
8002
从机N4

图 2.3. 组间分布图

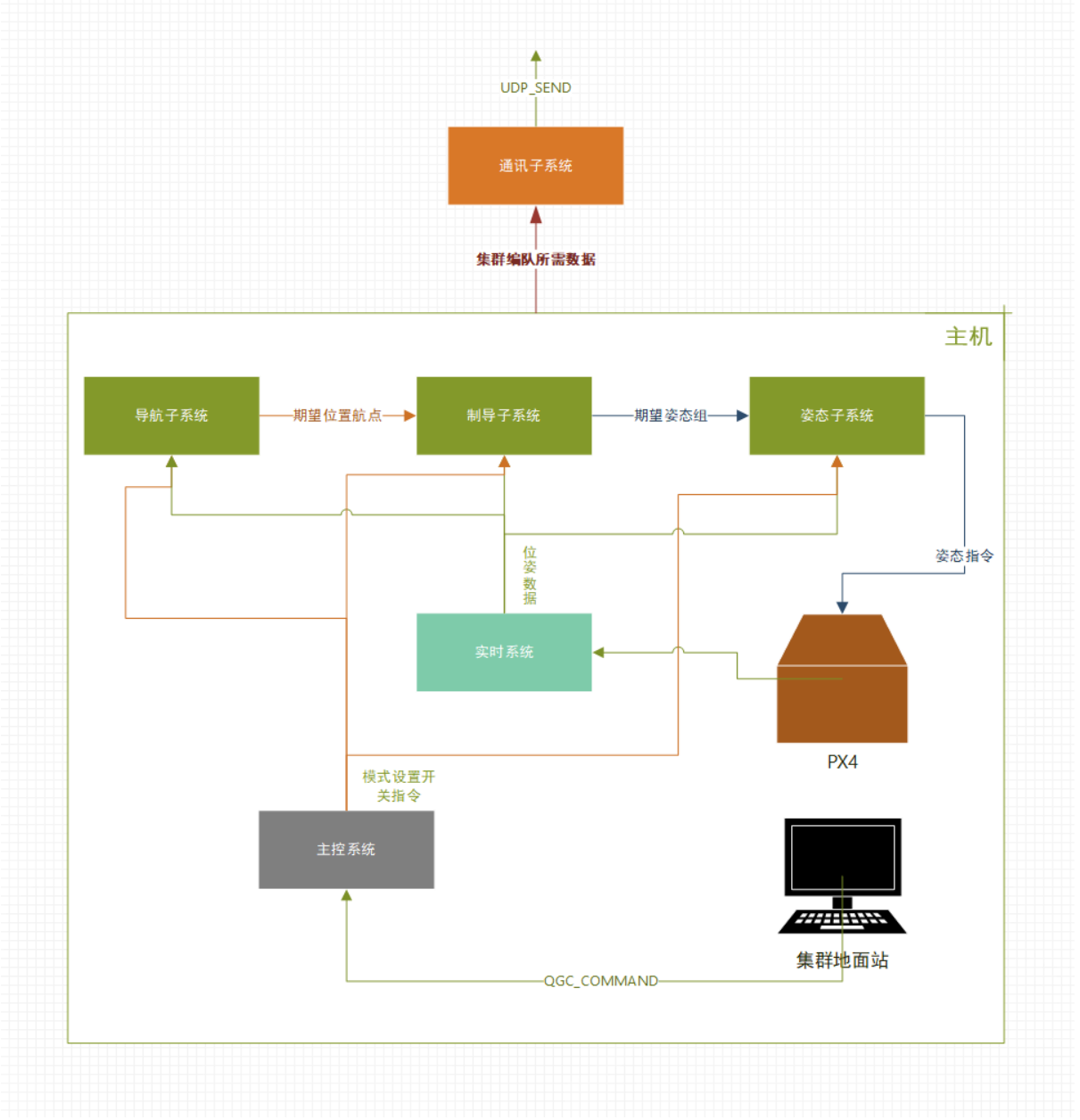

UDP_SEND
通讯子系统
集群编队所需数据
主机
导航子系统
期望位置航点
制导子系统
期望姿态组
姿态子系统
位姿数据
姿态指令
实时系统
PX4
模式设置开关指令
主控系统
集群地面站
QGC_COMMAND

图 2.4. 主机数据流图

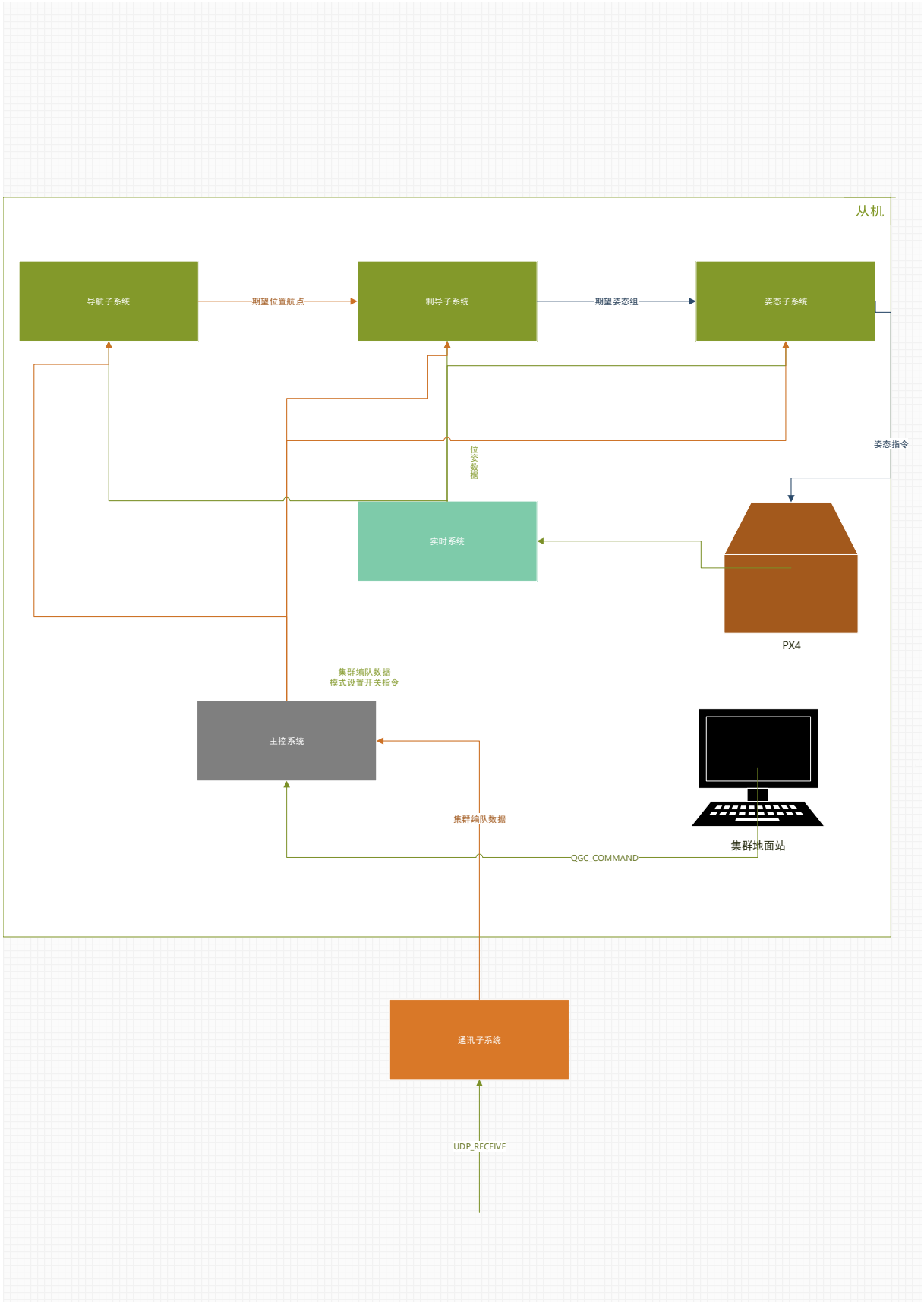

从机
导航子系统
期望位置航点
制导子系统
期望姿态组
姿态子系统
姿态指令
位姿数据
实时系统
PX4
集群编队数据
模式设置开关指令
主控系统
集群编队数据
集群地面站
QGC_COMMAND
通讯子系统
UDP_RECEIVE

图 2.5. 从机数据流图

2.3 详细设计

2.3.1 实时系统

实时系统担任的主要角色是和PX4进行交互. 如图2.6所示, 实时系统从px4中读取到当前系统的实时位姿信息之后, 将其分别下发到下面几个控制器进行一些逻辑方面的计算.

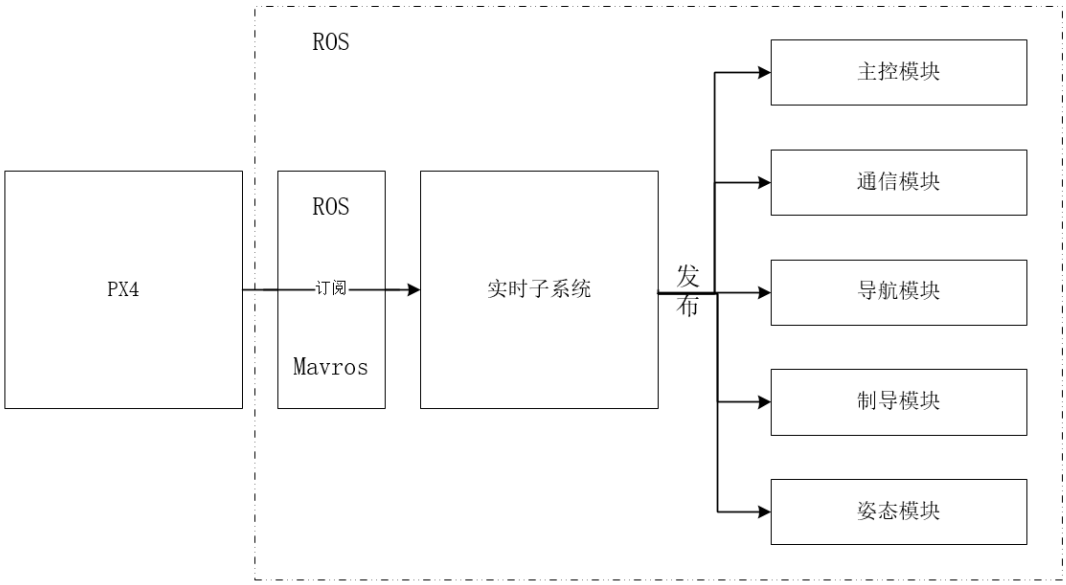


图 2.6. 实时子系统

2.3.2 主控模块

主控模块在系统中担任着主要控制的作用, 任何模块之间的相互控制, 如图2.7.

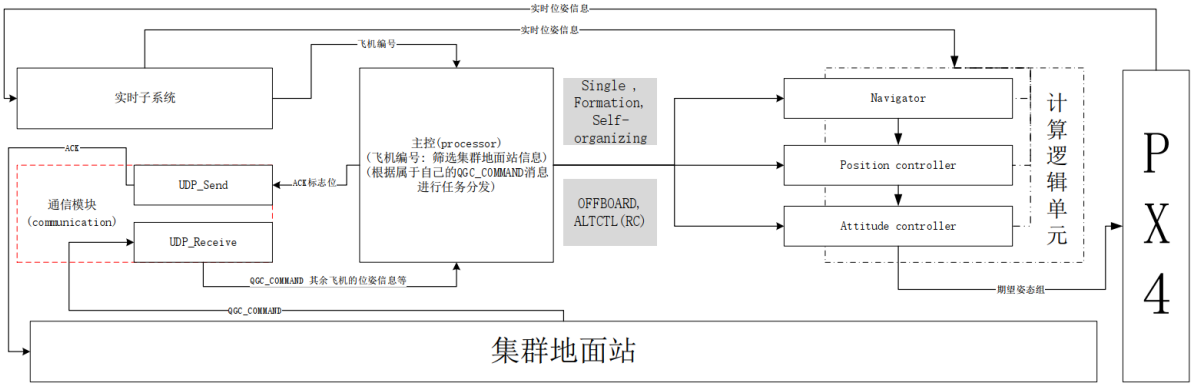


图 2.7. processor

首先, 集群地面站以UDP的方式向每一架无人机广播飞行指令(内部有指定接收的飞机编号), 由通信模块(communication)下的UDP_Receive子功能进行接收, 同时将其发送到主控(processor). processor实例化之后, 会从实时子系统中获取当前无人机的编号id, 并据此id从对个QGC_COMMAND中筛选出属于自己的指令. 获取任务指令之后, 进行具体的解析, 获取飞行的任务指令, 同时判断数据的有效性, 且为UDP_Send发送一个ACK标志位, 来对集群地面站产生一个应答响应. 获取有效的飞行控制指令(single,

formation, or self-organizing), 且指定与硬件的交互方式(OFFBOARD or ALTCTL(RC)), 将指令整合发布给计算逻辑单元下的各个控制器, 进行计算, 最后将计算结果从串口发送给硬件PX4.

2.3.3 通信模块

通信模块在整个集群之间担任着非常重要的地位, 有了它我们就可以完成多个个体之间的交流配合.

如图2.8所示内部主要功能有 UDP_Send 和 UDP_Receive 两大功能, 其中 UDP_Send 主要担任着广播自己的位姿信息, 供机间之间配合调用. 有了send, receive 也是必不可少的. UDP_Receive 主要是用来接收集群地面站的指令信息以及其余飞机广播的位姿信息, 进而数据转发给processor进行处理.

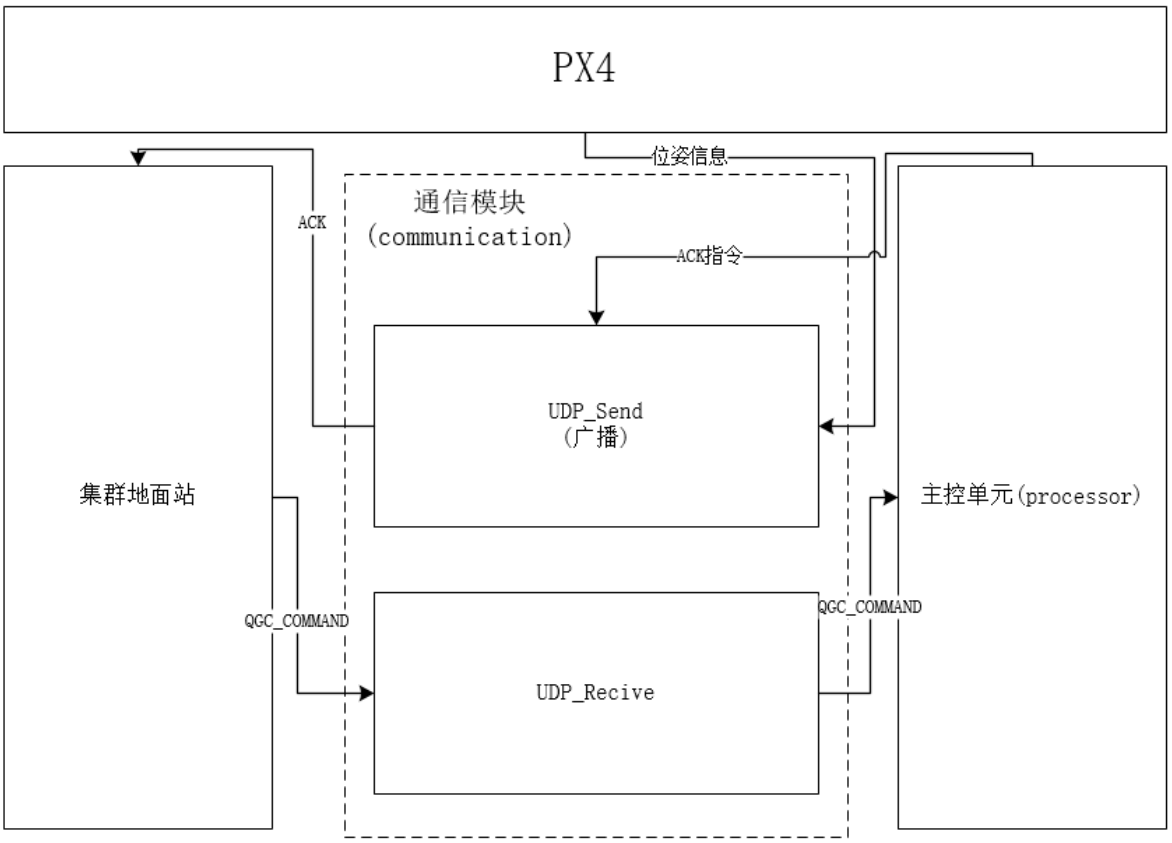


图 2.8. communication

2.3.4 导航控制器

导航就字面上说，就是引导航行的意思，而其确切的定义可表述为：导航是有目的地、安全有效地引导运动体(船只、潜艇、地面车辆以及飞机、宇宙飞船等). 也就是在当前位置已知的前提下，计算如何才能到达目的地，也即路径规划。本系统设计的导航子系统(navigator)执行流程如图2.9所示. 路径管理内部有三处控制逻辑, 分别对应着直线切换, 转弯圆角, 以及dubins曲线三种控制算法. 会在后面”算法”一节中进行介绍.

2.3.5 制导控制器

制导是导引和控制飞行器按一定规律飞向目标或预定轨道的技术和方法。制导过程中，系统不断无人机与目标或预定轨道的相对位置关系，发出制导信息传递给姿态控制器，以控制飞行. 控制方式如图2.10所示.

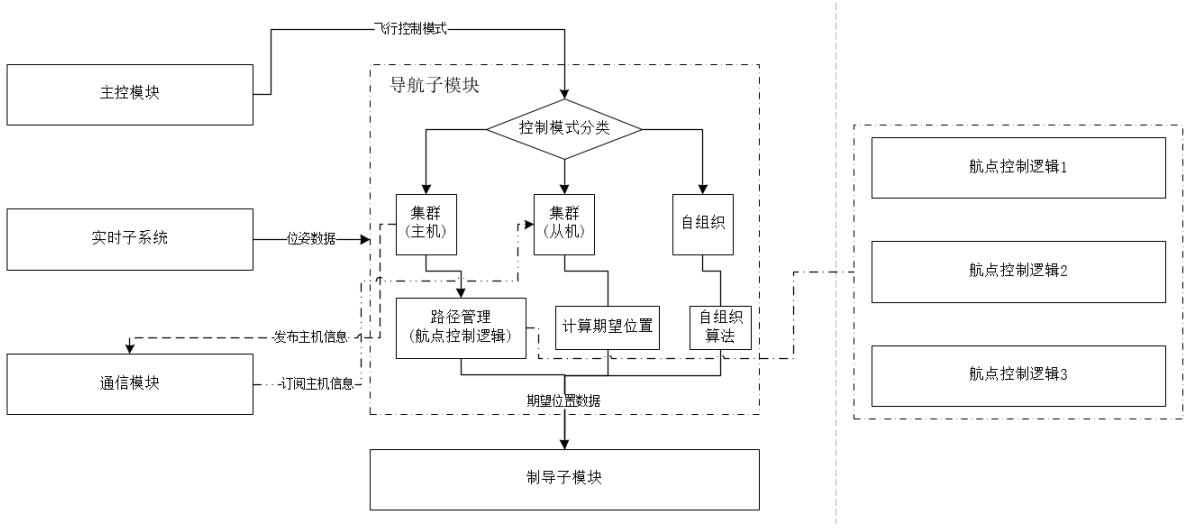


图 2.9. navigator

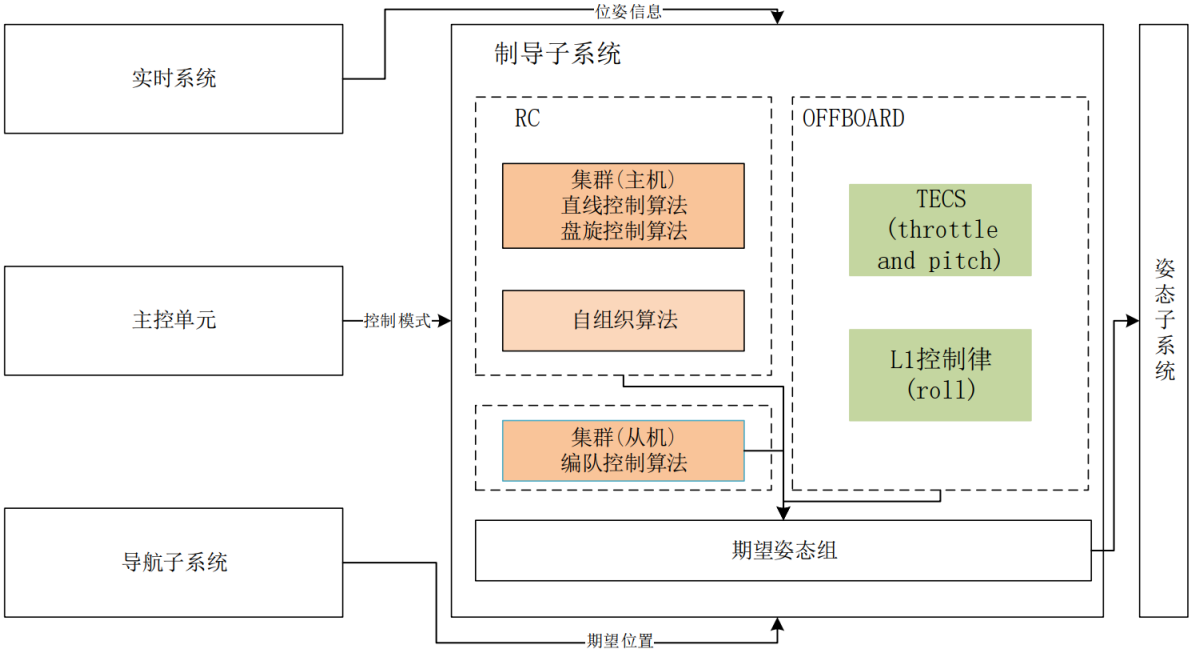


图 2.10. position controller

数据准备阶段有从实时系统获取当前的位姿信息, 从主控模块获取控制模式, 以及从导航子系统中获取期望位置. 之后的数据处理阶段主要有两套逻辑: 单机和集群两种处理模式; 其中单机处理模式又分为RC处理模式(ALTCTL), 以及OFFBOARD处理模式; 集群有编队控制模式.

RC处理模式下, 有直线控制逻辑, 盘旋控制逻辑, 以及自组织控制逻辑; OFFBOARD处理模式下, 有总能量守恒(TECS)以及L1控制律. 在集群编队控制中, 有编队控制算法. 在各种算法的计算下, 可以得到期望姿态组, 有欧拉角(roll, pitch, yaw)设定值, 以及油门(throttle)设定值. 进而下发给姿态子系统.

2.3.6 姿态控制器

姿态控制器获取到制导控制器计算得到的期望姿态数据组之后, 按照ALTCTL(RC)和OFFBOARD两种控制方式进行PID控制处理, 如图2.11所示.

OFFBOARD模式下, 将计算得到的欧拉姿态设定值转化为四元数, 以及根据自己欲控制的量为type_mask进行赋值, 完成姿态的封装, 发布给PX4; RC模式下, 会根据当前无人机的欧拉姿态与期望欧拉姿态计算一个ERROR, 根据ERROR, 进行PID控制计算, 得到四个RC通道的值(roll, pitch, yaw, throttle), 发布给PX4.

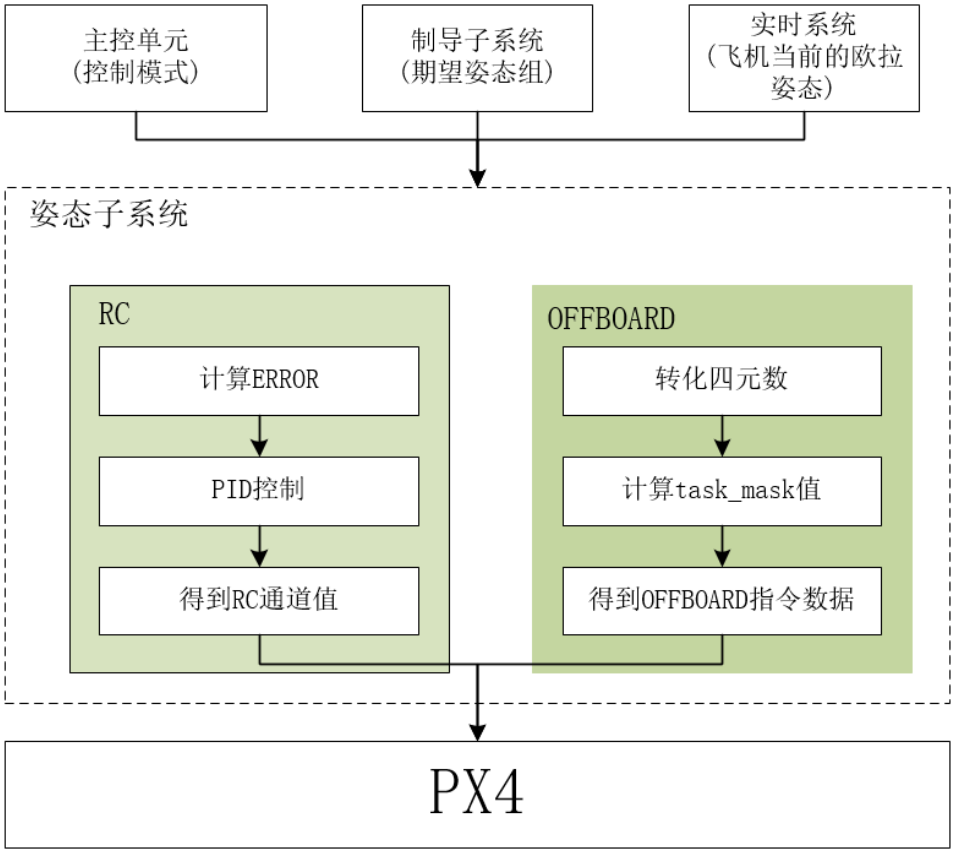


图 2.11. attitude controller

2. 4 编码和单元测试

本项目代码采用面向对象高级编程语言C++编写而成. 采用CMake编译构建系统.

第三章 算法

3.1 导航控制-路径管理

3.1.1 航点控制逻辑1

在引入算法之前，我们需要先了解一个概念"半平面(half-plane)"，半平面是被用来指定是否航点切换的核心条件，通过无人机当前位置和半平面的几何关系，来判定航点是到达。具体的几何关系式如下：

$$H(r,n)=\left\{p\in R^3:(p-r)^T n\geq 0\right\} \tag{3.1}$$

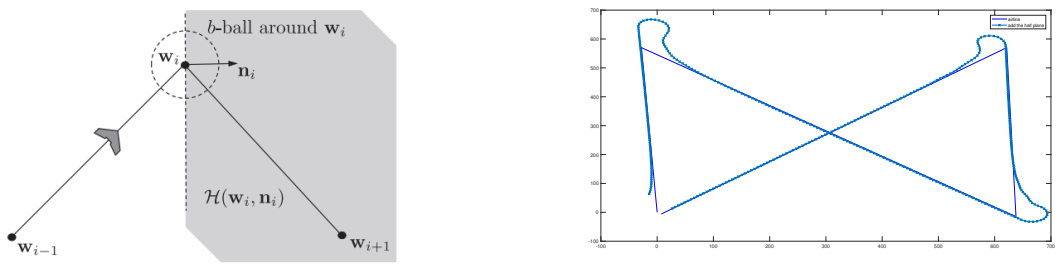
其中的 r 为正在执行的目标航点，n为该半平面的法向量，p为无人机的当前位置，物理模型如图3.1(a)所示。定义了一个单位向量q_i和 n_i, q_i 是航线 $\overline{w_iw_{i+1}}$ 的单位方向向量，所以存在一个三维向量n_i，可以把航线 $\overline{w_{i-1}w_i}$ 和 $\overline{w_iw_{i+1}}$ 分隔开.

$$\begin{cases} q_i \triangleq \frac{w_{i+1}-w_i}{\|w_{i+1}-w_i\|} \\ n_i \triangleq \frac{q_{i-1}+q_i}{\|q_{i-1}+q_i\|} \end{cases} \tag{3.2}$$

无人机从航点w_{i-1}到w_i的飞行过程中，若到达两条航线所生成的半平面，则认为到达了目标航点，航线进行切换。飞行控制逻辑包含直线飞行控制和盘旋飞行控制，在直线飞行的过程中，航点切换依附着算法1(航点控制逻辑1)。

第一次执行该算法时候，需要初始化i的值，让其等于2，见2；UAV会执行航线$\overline{w_{i-1}w_i}$，在算法第4、5行分别定义针对当前执行航点的r和q，第6行定义了下一条航线的单位方向向量，第7行是航线 $\overline{w_{i-1}w_i}$ 和 $\overline{w_iw_{i+1}}$ 所生成半平面的法向量，8行到10行进行位置判定，若满足某个关系式，即若无人机穿过半平面，那么就要切换航点，执行下一条航线，直到最后一个航点序列号减一。

算法1产生的飞行效果如图3.1(b)所示，很明显，但在直线飞行航点之间切换的时候，因忽略转弯时无人机初始动量的问题，造成了一段很长的转换缓冲距离，致使飞行效果很差，航线不平滑。



(a) (b)

图 3.1. straight line

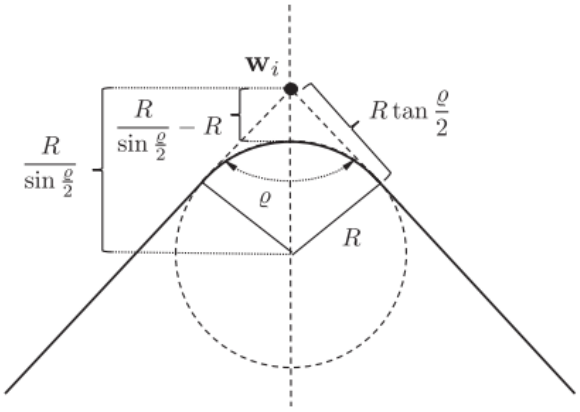


图 3.2. 插入fillet

Algorithm 1 Follow Waypoints:(r, q)=followWpp(W, p)

Input: Waypoints path $W = \{w_1, \ldots, w_N\}$, MAV position p=$(p_n, p_e, p_d)^T$.
Require: N ≥ 3
1: **if** New waypoints path W is received **then**
2: Initialize waypoint index: $i \leftarrow 2$
3: **end if**
4: $r \leftarrow w_{i-1}$
5: $q_{i-1} \leftarrow \frac{w_i - w_{i-1}}{\|w_i - w_{i-1}\|}$
6: $q_i \leftarrow \frac{w_{i+1} - w_i}{\|w_{i+1} - w_i\|}$
7: $n_i \leftarrow \frac{q_{i-1} + q_i}{\|\|q_{i-1} + q_i\|\|}$
8: **if** $p \in H(w_i, n_i)$ **then**
9: Increment $i \leftarrow (i+1)$ until $i = N$ - 1
10: **end if**
11: **return** r, q=q_{i-1} at each time step.

3. 1. 2 航点控制逻辑2

对其优化有两种方法, 一是当无人机当前的位置在当前执行的航线上的投影到该航线的起点的距离占整个比重的90%的时候, 进行航点切换;

另外一个较优的改进方法是为其增加一个圆角(fillet)，如图3.3所示，在航点 w_i 附近，生成一个盘旋中心，让其使得无人机在执行该航点的时候，以圆弧的方式进入到下一条航线，这么航线飞行出来的效果较好。定义航线$\overline{w_{i-1}w_i}$ 和$\overline{w_iw_{i+1}}$的角度为 ϱ:

$$\varrho \triangleq cos^{-1}(-q_{i-1}^T q_i) \tag{3.3}$$

在图3.2中，假定fillet的半径为R，那么圆弧和两条航线的切点距航点w_i的距离为$\frac{R}{tan(\frac{\varrho}{2})}$ 且航点w_i和fillet的中心的距离是$\frac{R}{sin(\frac{\varrho}{2})}$，所以航点$w_i$距圆角fillet的最短距离为$\frac{R}{sin(\frac{\varrho}{2})}$ - R. 为了让其直线控制逻辑和圆弧控制逻辑 [?] 更好的结合，对此也引进来了"半平面"。在图3.3中，定义了两个半平面 H_1 和 H_2。飞机在未进入半平面 H_1 的时候，执行直线控制逻辑；当进入到 H_1 的时候，执行圆弧控制逻辑，此时，圆弧的中心是 c，半径是 ρ；直到无人机进入 H_2 ，切换到直线控制逻辑。在算法2(航点控制逻辑2)中，给出了

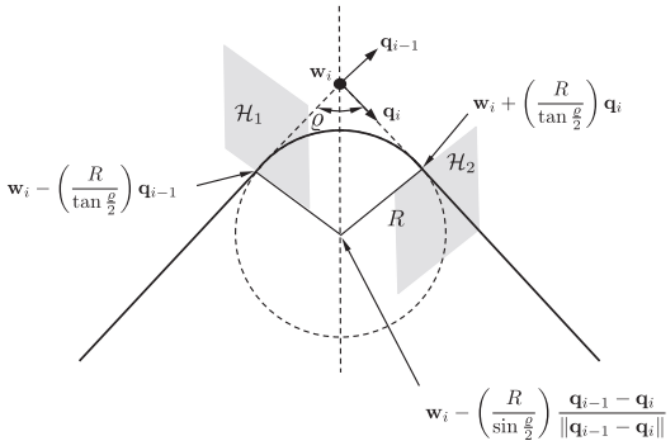


图 3.3. 将半平面和fillet进行结合

详细的逻辑梳理。圆角的中心 c, 第一个半平面、第二半平面与圆角相交的点 r_1 和 r_2 定义如下：

$$c = w_i - \frac{R}{sin(\frac{\varrho}{2})}\frac{q_{i-1}-q_i}{\|q_{i-1}-q_i\|} \tag{3.4}$$

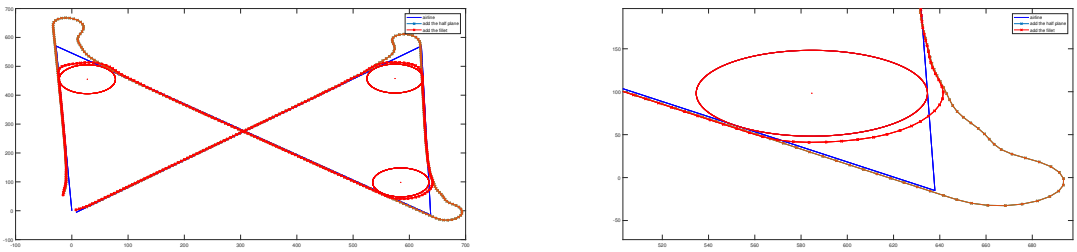
$$r_1 = w_i - \left(\frac{R}{tan(\frac{\varrho}{2})}\right)q_{i-1} \tag{3.5}$$

$$r_2 = w_i + \left(\frac{R}{tan(\frac{\varrho}{2})}\right)q_i \tag{3.6}$$

其中的向量 q_{i-1} 和 q_i 分别为两条航线的单位方向向量。算法2中，若一条新的航线被接收，那么航点序列号和state的值也会在第2行进行更新，向量q_{i-1} 和 q_i，以及ϱ在4到6行计算；当state = 1的时候，调用直线控制逻辑，执行航线 $\overline{w_{i-1}w_i}$，同时计算相对应的 r 和 q，在第11到14行判断是否到达第一个半平面；若无人机穿过第一个半平面，朝着第二个半平面飞去的时候，state = 2，执行曲线控制逻辑，对应的圆心为c，半径为ρ，飞行的方向(逆时针，或顺时针)为λ；若无人机穿过第二个半平面，则state赋值为1，执行直线控制逻辑。

3. 1. 3 算法优化

算法1和算法2执行效果对比如图3.4(a)所示, 图3.4(b)是将右下角转弯轨迹放大之后得到的. 很显然, 从总体路径长度最优的角度来看, 后者的总体路径长度要明显比前者总体路径长度短很多, 执行效率也会优于前者. 尽管如此, 算法2在转弯的时候, 对初试惯性动量考虑的不太周到, 存在一些瑕疵, 如图3.5(a)所



(a) (b)

图 3.4. comparing

示. 虽然在算法2中, 模型建立是以fillet和航线相切的情况来考虑的, 但在仿真平台中, 无人机的并不是以

Algorithm 2 Follow Waypoints with Fillet:(flag, r, q, c, ρ, λ)=followWppFillet(W, p, R)

Input: Waypoints path $W = \{w_1, \ldots, w_N\}$, MAV position p=$(p_n, p_e, p_d)^T$, fillet radius R
Require: N $\geq$ 3
1: **if** New waypoints path W is received **then**
2: Initialize waypoint index: $i \leftarrow 2$, and state machine: state $\leftarrow 1$
3: **end if**
4: $q_{i-1} \leftarrow \frac{w_i - w_{i-1}}{\|w_i - w_{i-1}\|}$
5: $q_i \leftarrow \frac{w_{i+1} - w_i}{\|w_{i+1} - w_i\|}$
6: $\varrho \leftarrow cos^{-1}(-q_{i-1}^T q_i)$
7: **if** state $= 1$ **then**
8: flag $\leftarrow 1$
9: $r \leftarrow w_{i-1}$
10: $q \leftarrow q_{i-1}$
11: $z \leftarrow w_i - \frac{R}{tan(\frac{\varrho}{2})} q_{i-1}$
12: **if** $p \in H(z, q_{i-1})$ **then**
13: state $\leftarrow 2$
14: **end if**
15: **else if** state $= 2$ **then**
16: flag $\leftarrow 2$
17: $c \leftarrow w_i - \frac{R}{sin(\frac{\varrho}{2})} \frac{q_{i-1} - q_i}{\|q_{i-1} - q_i\|}$
18: $\rho \leftarrow R$
19: $\lambda \leftarrow sign(q_{i-1,n} q_{i,e} - q_{i-1,e} q_{i,n})$
20: $z \leftarrow w_i + \frac{R}{tan(\frac{\varrho}{2})} q_i$
21: **if** $p \in H(z, q_i)$ **then**
22: $i \leftarrow (i+1)$ until $i = N$ - 1
23: state $\leftarrow 1$
24: **end if**
25: **end if**
26: **return** flag, r, q, c, ρ,λ.

期望空速来飞行, 而是以期望空速(airspeed)和风速(wind speed)叠加之后形成的地速(ground speed)来飞行. 由于风速大小的不确定性, 造成了飞行效果差强人意.

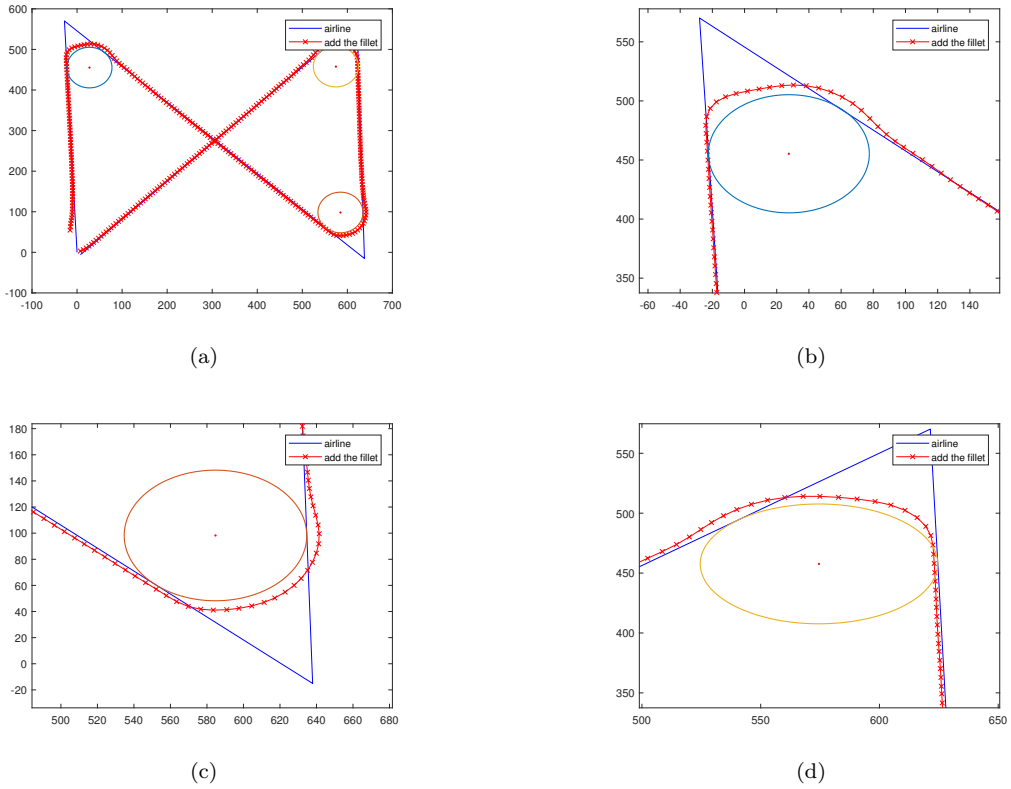
对此提出了一个算法的改进. 根据图3.5(b), 3.5(c), 3.5(d)显示, 超出航线的那一小部分还是由于惯性动量导致的, 所以在这里我们需要提前来进行航先切换, 从而留出一部分来减弱惯性动量, 让转弯轨迹更加的圆滑, 航线执行总的距离更小. 故对算法2进行了改进, 从而得到了算法3.

算法中, 在12行, 保存第一个半平面与航线$\overline{w_{i-1}w_i}$的交点z为z_1, 当进入第一个半平面的时候，state = 2，执行曲线控制逻辑，对应的圆心为第19行所示的c, 其中半径进行了一定比例的缩放，已到达保留一定的距离来消耗无人机初始惯性动量的目的。第20行和22行，对应的R都进行了比例缩放，其中第20行是为fillet center设置半径，第22行是计算新的第二个半平面和下一条航线的新交点；第23行判断是否到达第二个半平面，若到达则切换航线，state重新设置为1，执行直线控制逻辑；反之继续执行当前航线。前后算法比对的效果图如图3.6(a)所示。

误差分析

Table 3.1: 误差分析表

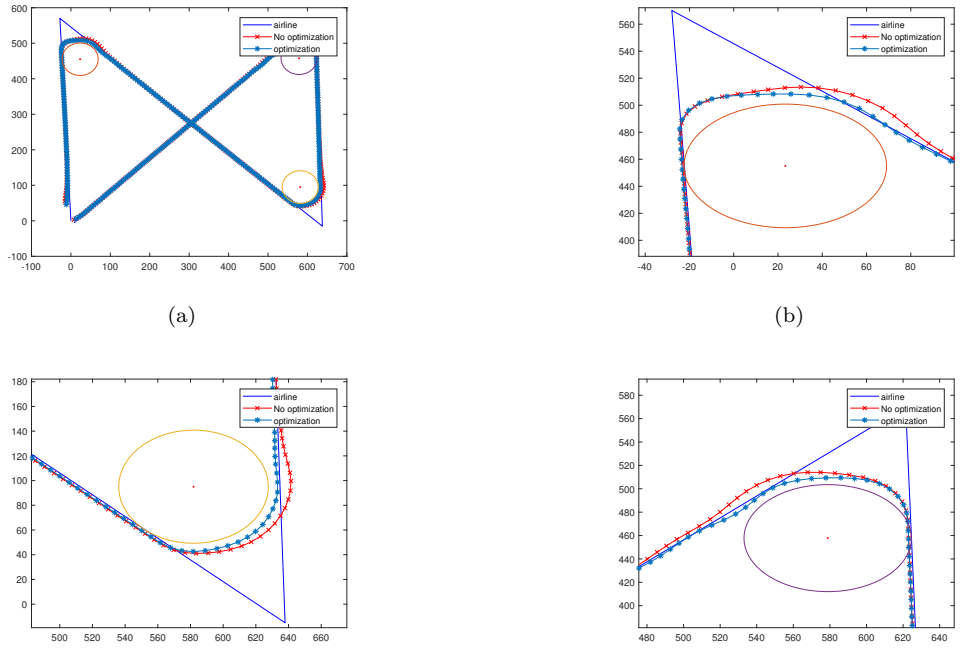
	第一段(*150)	第二段((*130))	第三段(*120)	总和
优化之前	9.506485437269829	13.459338133308067	12.517388189403794	11.694433389122446
优化之后	9.307437579575690	2.896129449605189	7.097844734906930	6.560884583934650



(a) (b)

(c) (d)

图 3.5. algorithm6



(a) (b)

(c) (d)

图 3.6. algorithm6 comparing 1

Algorithm 3 Follow Waypoints with Fillet:(flag, r, q, c, ρ, λ)=followWppFillet(W, p, R)

Input: Waypoints path $W = \{w_1, \ldots, w_N\}$, MAV position p=$(p_n, p_e, p_d)^T$, fillet radius R
Require: N $\geq$ 3
if New waypoints path W is received **then**
  Initialize waypoint index: $i \leftarrow 2$, and state machine: state $\leftarrow 1$
end if
$q_{i-1} \leftarrow \frac{w_i - w_{i-1}}{\|w_i - w_{i-1}\|}$
$q_i \leftarrow \frac{w_{i+1} - w_i}{\|w_{i+1} - w_i\|}$
$\varrho \leftarrow cos^{-1}(-q_{i-1}^T q_i)$
if state $= 1$ **then**
  flag $\leftarrow 1$
  $r \leftarrow w_{i-1}$
  $q \leftarrow q_{i-1}$
  $z \leftarrow w_i - \frac{R}{tan(\frac{\varrho}{2})} q_{i-1}$
  $z_1 \leftarrow z$
  if $p \in H(z, q_{i-1})$ **then**
    state $\leftarrow 2$
  end if
else if state $= 2$ **then**
  flag $\leftarrow 2$
  $c \leftarrow w_i - \frac{R}{sin(\frac{\varrho}{2})} \frac{q_{i-1} - q_i}{\|q_{i-1} - q_i\|}$
  $c \leftarrow z_1 + \frac{c - z_1}{\|c - z_1\|} * 0.915R$
  $\rho \leftarrow 0.915R$
  $\lambda \leftarrow sign(q_{i-1,n} q_{i,e} - q_{i-1,e} q_{i,n})$
  $z_2 \leftarrow w_i + \frac{1}{tan(\frac{\varrho}{2})} * 0.915R * q_i$
  if $p \in H(z, q_i)$ **then**
    $i \leftarrow (i + 1)$ until $i = N$ - 1
    state $\leftarrow 1$
  end if
end if
return flag, r, q, c, ρ,λ.

通过计算超出航线部分的点到航线的垂直距离，即偏离航线的误差，来判定算法是否优化。算法优化之前和优化之后误差分析如3.1所示。最终计算得到的总的偏差由11.7减少到6.6，效果优化了78%，足以可见算法3较之前算法具有很好的效用性。

3. 2 制导控制

3. 2. 1 集群(主机)控制逻辑

OFFBOARD

OFFBOARD模式控制下主要存在两种控制算法: TECS 和 L1控制律, 其中TECS就是利用势能和动能总和不变来根据期望高度和空速计算出期望pitch值以及期望throttle值; L1制导算法，就是在航线上选择与无人机距离为L1的参考点，然后根据速度方向与到参考点连线方向之间的夹角计算横向加速度, 进而求得滚转roll, 实现航线跟踪。

这两者的算法都是从PX4内部剥离出来, 在OFFBOARD模式下进行的逻辑控制.

RC

直线控制逻辑3.7

Algorithm 1 Vector Field Construction Algorithm (Constant altitude).

$\chi^f \leftarrow atan2(w_{2y}-w_{1y}, w_{2x}-w_{1x})$ { Calculate heading from waypoint 1 to waypoint 2. }
$s^* \leftarrow \frac{(z-w_1)^T(w_2-w_1)}{\|w_2-w_1\|^2}$ { Calculate position of MAV along path. }
$\epsilon \leftarrow \|z-(s^*(w_2-w_1)+w_1)\|$ { Calculate distance of MAV from path. }
$\rho \leftarrow sign\left[(w_2-w_1)\times(z-w_1)\right]$ { Calculate which side of path MAV is on. }
if $s^* > 1$ **then** {MAV is past second waypoint}
  Switch to next waypoint
else
  $\epsilon \leftarrow \rho\epsilon$
  if $|\epsilon| > \tau$ **then** {Distance from path is greater than threshhold.}
    $\chi^d \leftarrow \chi^f - \rho\chi^e$ {Set vector field heading.}
    $\chi^c \leftarrow \chi^d$ {Set commanded heading to the autopilot.}
  else
    $\chi^d \leftarrow \chi^f - (\chi^e)(\frac{\epsilon}{\tau})^k$ {Set vector field heading.}
    $\chi^c \leftarrow \chi^d - \left(\frac{k\chi^e S}{\alpha\tau^k}\right)\epsilon^{k-1}\sin\chi$ {Set commanded heading to the autopilot.}
  end if
end if

图 3.7. 直线控制逻辑

盘旋控制逻辑3.8

Algorithm 2 Orbit Following Vector Field Algorithm (Counter-Clockwise Direction)

Obtain current position z
$d \leftarrow \|z-c\|$ { Calculate distance to center of orbit }
if $d > 2r$ **then**
  $\chi^d \leftarrow \gamma - \frac{5\pi}{6}$
  $\chi^c \leftarrow \gamma - \frac{5\pi}{6} + \frac{S}{d}\sin(\chi-\gamma)$
else
  $\chi^d \leftarrow \gamma - \frac{\pi}{2} - \left(\frac{\pi}{3}\right)\left(\frac{d-r}{r}\right)^k$
  $\chi^c \leftarrow \chi^d - \frac{S}{\alpha d}\sin(\chi-\gamma) - \frac{kS\pi}{3r^k\alpha}\tilde{d}^{k-1}\cos(\chi-\gamma).$
end if

图 3.8. 盘旋控制逻辑

3. 2. 2 编队控制逻辑

从机的导航子系统中, 会根据当前飞机的位置, 以及主机的当前位置以及编队的gap值, 计算出从机期望位置点. 下发给从机的制导控制器, 进行编队, 其中的算法逻辑如下:

Algorithm 4 Formation control:(ψ, throttle)

Input: (followerFinalPoint, followerPosition, followerHeading), (leaderHeading, leaderHeadingCommanded, groundSpeedLeader)

$followerFinalPoint_{Proj}.x \leftarrow followerFinalPoint.x + cos(leaderHeadingCommanded)$
$followerFinalPoint_{Proj}.y \leftarrow followerFinalPoint.y + sin(leaderHeadingCommanded)$
$A \leftarrow (followerFinalPoint.x - followerPosition.x) * cos(followerHeading)$
$B \leftarrow (followerFinalPoint.y - followerPosition.y) * sin(followerHeading)$
$x_e \leftarrow A + B$
$\chi_e \leftarrow leaderHeading - followerHeading$
$followerDistance \leftarrow followerFinalPoint - followerPosition$
if $followerDistance > threshold$ **then**
  $headingSetpoint \leftarrow atan2(\frac{followerFinalPoint.y - followerPosition.y}{followerFinalPoint.x - followerPosition.x})$
  $groundSpeedSetpoint \leftarrow 2 * groundSpeedLeader$
else if $followerDistance \leq threshold$ **then**
  $groundSpeedSetpoint \leftarrow c1 * x_e + groundSpeedLeader * cos(chi_e)$
  $headingSetpoint \leftarrow leaderHeadingCommanded + side * \chi_{\inf} * (\frac{followerDistance}{threshold}^k)$;
end if
$throttle \leftarrow groundSpeedSetpoint$
$\psi \leftarrow headingSetpoint$
return (ψ, $throttle$).

3.3 自组织算法

第四章 仿真

4.1 软件仿真效果展示

视频演示(基于ros-gazebo-qgc三者平台进行的软件仿真展示)

- 视频SITL_FORMATION: linux系统下, 基于ros-gazebo-qgc三者平台进行的软件仿真展示, 期间进行三种编队的飞行(T字型编队, Y字型编队, V字型编队), 并且完成了普通航线, 八字形航线, 操场跑道航线的飞行任务, 最后回到起飞Home点处.

4.2 硬件仿真效果展示

视频演示(基于XPlane10-QGC平台进行的硬件仿真展示), 因为XPlane硬件在环仿真平台数据链路很不稳定, 只能勉强测一台机子, 所以只能采用将主机飞行数据保存下来, 从机读取文本数据的方法进行测试.

- 视频HITL_LEADER: 主机按照航线飞行, 使用的是RC飞行控制模式(集群(主机)), 且在飞行途中将飞行期间的姿态数据(编队所需)保存到文本文件中, 供从机编队测试时使用.
- 视频HITL_FOLLOWER: 从机从文本文件中获取到主机的飞行数据, 进行编队飞行控制.