文章编号:1671-4598(2022)12-0297-09 DOI:10.16526/j.cnki.11-4762/tp.2022.12.045 中图分类号:TP311 文献标识码:A

基于 OSG 和 OSDK 的无人机标校系统设计及实现

虞炳文，蔡红维，丁思炜，杨 波，王 康
（西昌卫星发射中心，四川 西昌 615000）

摘要：通过对无人机标校过程中遇到的精度不高、稳定性较差、可操作性较低、易受环境因素干扰、专业标校软件缺失等问题进行分析，设计一套包含硬件及软件在内的无人机标校方案，硬件设计包含对机载应答机的指标设计、需求设计及选型，对无人机的指标设计、需求设计及选型，以及配套硬件设备的选型，设计一套可行性高，功能完整的无人机标校软件，设计部分主要包含三部分功能，分别为基于 OSG 三维建模的标校无人机飞行轨迹生成软件、基于大疆 OSDK 开发包开发的飞行轨迹导入并启动飞行任务软件、基于远控模块实现机载应答机参数设置的控制软件，并提出无人机标校中的主要技术指标参数，最终形成一套切实可行的无人机标校系统，增加无人机标校的可行性及可靠性，为无人机标校软件的开发及无人机标校实施打下基础。

关键词：标校；硬件设计；软件设计；三维建模；OSDK

Design and Implementation of UAV Calibration System Based on OSG and OSDK

YU Bingwen，CAI Hongwei，DING Siwei，YANG Bo，WANG Kang
（Xichang Satellite Launch Center，Xichang 615000，China）

Abstract：By analyzing the problems encountered in UAV calibration，such as low accuracy，poor stability，low operability，easy environmental influence，and lack of professional calibration software，a set of UAV calibration scheme including hardware and software is designed. The hardware design includes the index design，demand design and selection of airborne transponder and UAV，As well as the selection of supporting hardware and equipment，this paper designs a set of UAV calibration software with high feasibility and complete function. The design part mainly includes three functions，namely，the flight trajectory generation software of calibration UAV based on OSG 3D modeling，the flight trajectory import and start flight mission software based on Dajiang OSDK development package，and the control software based on remote control module to achieve the airborne transponder parameters setting，And the main technical index parameters in UAV calibration are put forward，finally the scheme forms a set of practical UAV calibration system，the system can increase the feasibility and reliability of the UAV calibration，and lay a foundation for the software development and implementation of the UAV calibration.

Keywords：calibration；hardware design；software design；3D modeling；OSDK

0 引言

目前的标校，根据跟踪目标的属性，可以区分为对静态目标的标校以及对动态目标的标校。

对静态目标的跟踪标校实现较为简单，但是需要标校塔或者标校杆配合进行，因此受到场地限制的约束较为明显。

通过对动态目标的跟踪标校，较为常见的有三种方式，分别是利用标校卫星、利用飞机、利用无人机进行标校。

利用标校卫星，即利用天平卫星进行标校，优点是简单可行，但是此种方法只能对测控设备的跟踪性能（基带指标）进行标校，对设备的动态性能（伺服指标）则无法进行有效标校。

利用飞机标校，优点是对设备的跟踪性能和动态性能都能进行有效的检验，但缺点也很明显，就是成本高昂。

利用无人机进行标校，优点是成本较低，起飞在一定程度上不受场地限制，对测控设备的跟踪和动态性能在理论上都能进行有效的验证，但是缺点是，目前所使用的无人机上可携带的应答机较为简单，以及无人机容易受到天气等因素影响，验证精度无法得到有效保证。

而文中通过对无人机、应答机等的硬件指标设计，对标校相关软件设计，提高无人机的标校可靠性，保证其精度，从而保证无人机标校的可行性。通过对无人机标校系统进行功能需求分析，提出了对于标校软件功能和无人机、应答机等硬件指标的设计与实现，对于进一步提高无人机标校精度与可靠性，推动无人机标校的应用有较强的现实意义。

1 无人机标校存在的问题

在前期开展的无人机标校过程中，暴露出一些问题。

1）无人机在高空中稳定性差，容易受到高空风等因素影响，导致不能按照既定飞行轨迹飞行。

2）目前基地正在使用的标校无人机，携带的是信标机，并非应答机，无应答模式。

收稿日期：2022-09-08； **修回日期**：2022-09-15。
作者简介：虞炳文(1993-)，男，浙江东阳人，大学本科，助理工程师，主要从事航天测控方向的研究。
引用格式：虞炳文，蔡红维，丁思炜，等. 基于 OSG 和 OSDK 的无人机标校系统设计及实现[J]. 计算机测量与控制，2022，30(12)：297-305.

3）无人机携带的信标机的发射频点与信号强度不支持修改。

4）因机载信标机较为简易，未配备频标设备，频率的稳定性较差。

5）飞行轨迹的设计依靠无人机产家提供的商业软件，例如 DJI Pilot，并非专业的标校软件，因此使用上有很多限制，例如在手动添加飞行轨迹的路径点时，难以避免地会出现点密度较低，点分布不均匀等情况。另外，在飞行轨迹的导入导出格式上，也会出现不匹配的情况。

6）专业的无人机标校软件的缺失，导致无人机的飞行轨迹的设计无法考虑到很多重要因素，例如被标校设备的指标参数，当时当地的自然因素等。

2 无人机标校软硬件分析设计

在此对无人机标校系统进行一个分析及设计，涉及硬件及软件内容。

2.1 应答机需求分析及设计

应答机应包含六个模块，依次为：控制天线、无线通信模块、控制模块、应答模块、射频天线、供电模块，应答机示意图见图 1。

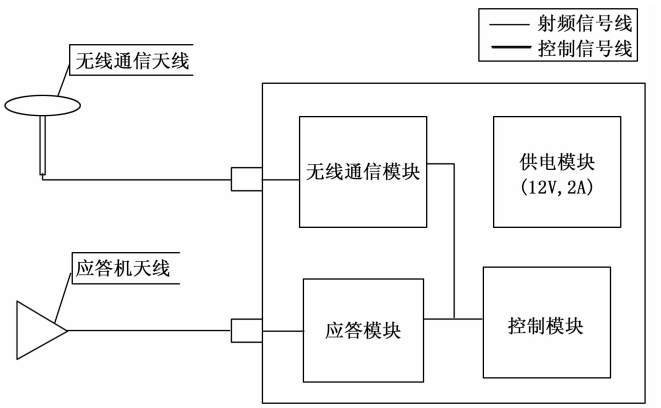

图 1 应答机平面示意图

对机载应答机作如下要求：

1）应答机的重量主要来源于应答机保护外壳、无线通信模块、应答模块、电池。应答机总重量小于 15 kg。

2）供电模块的输入端电压为 12 V，电流为 2 A，即功率为 24 W。

3）应答机外接 3S 电池。

4）可通过地面控制台无线控制应答机信号开关，调整信号频点、调整信号强度，切换应答及信标模式等。

2.2 标校无人机需求分析及设计

标校无人机的重要指标为：无人机载重与飞行稳定性。

1）无人机载重量与无人机的电机、螺旋桨桨叶长度、螺距等密切相关。电池、电机、螺旋桨选型与搭配很重要，一般情况下，整机重量，应该小于电机最大动力的 2/5。

2）飞行稳定性受飞控影响较大。选择大疆 A3 Pro 飞控，搭载 D-RTK-GNSS，可使得飞行悬停精度达到垂直方向约 0.02 m，水平方向约 0.01 m。

3）机架选用六旋翼机架，材质轻且坚固。

4）电池的选择需要综合考虑电池的重量及容量。无人机电池容量保证载重 10 kg 下，可飞行 30 min。

5）无人机遥控天线与应答机射频天线、应答机通信控制天线区分开。

6）无人机可同时接收来自遥控器及地面控制台的控制信号。遥控器控制无人机进行飞行操作，地面控制台通过上传规划好的航线驱动无人机启动航线飞行。

无人机示意图见图 2。

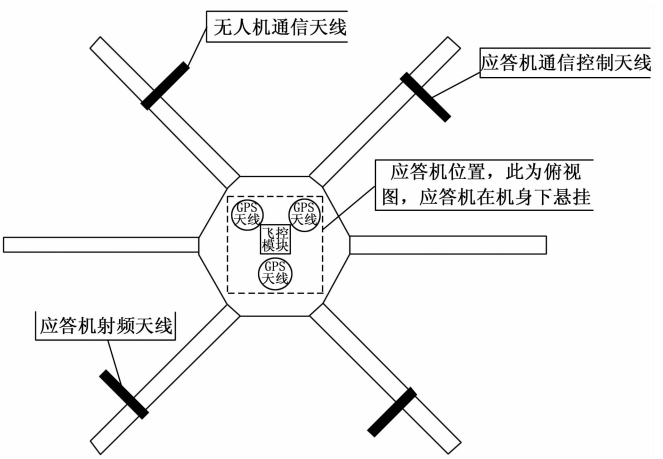

图 2 无人机俯视图

2.3 无人机标校软件需求分析及设计

无人机标校软件是为标校无人机专用设计的桌面端应用软件，运行在地面控制台服务器上，其作用是能够根据测控设备及机载应答机的性能参数、指标要求以及当地环境因素，综合生成一条满足标校需求的飞行轨迹，并且能够将飞行轨迹上传至无人机飞控，驱动标校无人机实施飞行计划。同时，无人机标校软件可以实现对机载应答机的远程控制。

2.3.1 软件功能

无人机标校软件需要实现以下三个功能：

1）生成一条可靠的飞行轨迹；

2）将飞行轨迹导入飞控，并实施；

3）远程控制机载应答机。

2.3.2 开发环境

软件部分使用较为成熟的 C++开发框架：

1）操作系统：Linux 操作系统。

2）开发语言：C++。

3）开发工具：Qt 及 QtCreator。

4）编译器：GCC 5.4.0/5.5.0 以上版本。

2.3.3 功能设计

软件功能设计分为三部分进行阐述。

2.3.3.1 生成飞行轨迹功能

生成飞行轨迹的功能是整个软件部分的基础内容。

2.3.3.1.1 功能分析

飞行轨迹的规划，需要考虑以下几个方面的情况：

1）标校的目的。跟踪静态目标与跟踪动态目标所要达成的目标是不同的。跟踪静态目标的通常是为了检查测控设备的极性、AGC 曲线标定。跟踪动态目标通常是为了测试设备的伺服性能，即设备方位俯仰上的速度、加速度、加加速度等，能否达到指标要求，因此在设计飞行轨迹时，需考虑其极限值。

2）飞行轨迹的约束条件。标校无人机允许飞行的最大高度与起飞点位置相关。测控设备与应答机之间的最小距离与电磁波传输特性有关，即电磁波远场距离公式，见公式（1），其中 D 为天线口径，λ 为电磁波频率；最远距离与测控设备与应答机的发射信号强度、接收灵敏度有关。

3）无人机飞行的干扰因素。主要是环境因素，包括高空风、大气折射率、温湿压、太阳夹角、月球夹角等。

4）飞行轨迹的模拟仿真。搭建三维立体场景进行模拟仿真，一方面实现飞行轨迹可视化，一方面验证各参数的正确性，可选用 OpenSceneGraph 及 osgEarth 函数库实现。

2.3.3.1.2　程序设计

根据上述影响分析，对飞行轨迹的功能作如下设计。

1）设计飞行轨迹生成函数类，QGenerateTraj。

2）设计一个结构体 trajStruct，包含每个飞行轨迹迹点的数据格式值，见代码 1，其中 T 为相对时间，X、Y、Z 分别为地心固定坐标系下的 X 轴值、Y 轴值、Z 轴值，Vx、Vy、Vz 分别为在 X 轴、Y 轴、Z 轴方向的速度分量。

代码 1：飞行轨迹点结构体

```
struct trajStruct{
    int T;
    int X;
    int Y;
    int Z;
    int Vx;
    int Vy;
    int Vz;
}
```

3）函数 emulateTrajectory（），无返回值，功能为根据生成的飞行轨迹在三维环境中进行仿真。

4）函数 getFlyTrajectory（），返回值为保存为 QList＜trajStruct＞的飞行轨迹，即保存为每个迹点的数据的容器，功能为根据输入的参数生成飞行轨迹，该函数在 emulateTrajectory（）中被调用。

5）setOsgSceneParaAndDisplay（QList＜trajStruct＞traj）函数，返回值为 QList＜trajStruct＞的飞行轨迹，输入值为 getFlyTrajectory（）输出的结果数据，功能为在三维空间中演示输入的飞行轨迹。该函数在 emulateTrajectory（）中被调用。

6）函数 setFlyTrajPara（），无返回值，输入值为各指标参数，功能为设置飞行轨迹设计中需要用到的基本参数，如测站地理坐标、测试指标、飞行限制条件、环境干扰因素、数据输出格式等。该函数在 getFlyTrajectory（）函数中被调用。

7）函数 setFlyTrajPara（），无返回值，输入值为测试指标参数，功能为设置测试期望达到的指标参数。该函数在 setFlyTrajPara（）被调用。在该函数下，根据指标参数的不同，分别调用 setElSpeed（int val）设置方位速度，调用 setElAcceleration（int val）设置方位加速度，调用 setElJerk（int val）设置方位加加速度，调用 setAzSpeed（int val）设置方位速度，调用 setAzAcceleration（int val）设置方位加速度，调用 setAzJerk（int val）设置方位加加速度。

8）函数 setFlyLimitPara（），无返回值，输入值为飞行轨迹的限制条件，功能为设置因设备性能带来的限制条件。该函数在 setFlyTrajPara（）被调用。在该函数下，根据指标参数的不同，分别调用 setRadarEmitPower（int val）设置测控设备的发送功率，调用 setRadarRecvMinVal（int val）设置测控设备的接收灵敏度，调用 setRespEmitPower（int val）设置应答机的发送功率，调用 setRespRecvMinVal（int val）设置应答机的接收灵敏度，调用 setRemoteDist（int val，int fr）设置并计算远场距离，其中 val 为天线口径，fr 为电磁波频率。

9）函数 setFlyInterPara（），无返回值，输入值为环境因素带来的干扰参数，功能为设置因环境因素带来的干扰。该函数在 setFlyTrajPara（）被调用。在该函数下，根据指标参数的不同，分别调用 ifUpperAirWinds（bool ok）设置是否启用高空风误差修正，调用 ifAtmosRef（bool ok）设置是否启用大气折射修正，调用 ifTempHumiPressure（bool ok）设置是否启用温湿压修正，调用 ifSunAngle（bool ok）设置是否避开太阳夹角干扰，调用 ifMoonAngle（bool ok）设置是否避开月球夹角干扰。

函数调用关系见图 3。

2.3.3.1.3　关键技术实现

关键技术主要围绕与 OSG 相关的内容。

2.3.3.1.3.1　三维环境的搭建

为搭建一个高还原度、可测量、准确可靠的三维世界，选用开源的 osgEarth 函数库来实现。osgEarth 是基于 OSG 函数库二次开发的，专门用于地理信息系统建模的函数库，使用范围广，可配置性强。选用 osgEarth2.10 版本及 OSG3.6.4 版本进行二次开发。

通过 osgEarth 实现三维建模主要有两种方法，一种是纯代码形式，所有的配置项通过代码中调用函数库实现；另一种是代码加配置文件形式，三维地理模型的绝大部分配置内容，都可以通过 *.earth 文件中的配置项实现配置。在无人机标校软件的设计中使用第二种方式。

2.3.3.1.3.1.1　编写 *.earth 文件

编写 *.earth 文件，可以将其命名为 simple.earth 配置文件，earth 后缀的文件格式，类似于 *.xml 文件，是用作配置三维地理模型的配置文件。定义地理信息模型，需要最少包含五部分内容，一是确定空间参考系，二是确定地形渲染参数，三是加载高程数据，四是加载纹理信息，五是确定数据缓存方式。

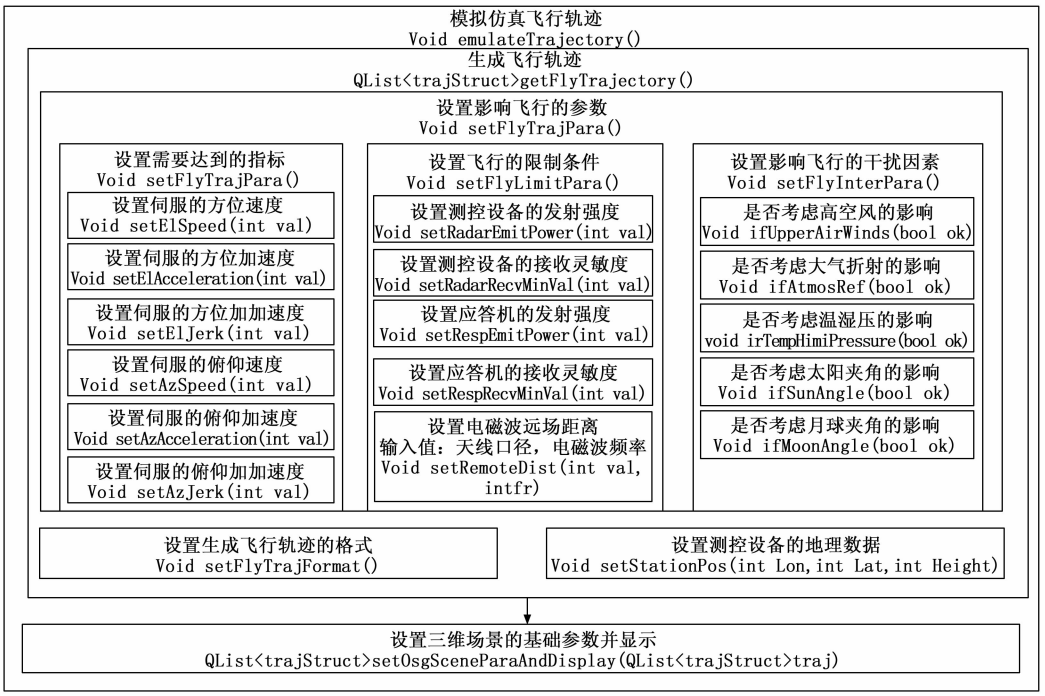

图 3 生成飞行轨迹函数设计

1）确定空间参考系。空间参考系配置选用 geocentric，地固系坐标系，即地心为质心的空间直角坐标系。

代码 2：空间参考系配置项

```
<map name="Globle" type="geocentric" version="2">
```

2）定义地形引擎如何渲染影像数据和高程数据。可对地理模型的插值方式、是否开启地形表面的光照、加载地形数据（瓦片）的策略、多影像数据叠加时集成最终影像数据的方式、地形瓦片分割的最大层数、瓦片范围系数、瓦片的采样率、瓦片边缘率、高程夸张系数、边缘缓冲率等参数进行设置。

代码 3：地形渲染方式配置项

```
<terrain>
<！--插值方法-->
<elevation_interpolation>triangulate</elevation_interpolation>
<！--定义是否开启地形表面的光照-->
<lighting>true</lighting>
<！--定义加载地形数据(瓦片)的策略-->
<loading_policy mode="preemptive" loading_threads_per_core
="2" compile_threads="8"></loading_policy>
<！--多影像数据叠加时集成最终影像数据的方式-->
<compositor>auto</compositor>
<！--定义地形瓦片分割的最大层数-->
<max_lod>10</max_lod>
<！--定义瓦片范围系数-->
<min_tile_range_factor>3</min_tile_range_factor>
<！--定义瓦片的采样率-->
<sample_radio>1.2</sample_radio>
<！--定义瓦片边缘率-->
<skirt_ratio>0.05</skirt_ratio>
<！--高程夸张系数-->
<vertical_scale>2.0</vertical_scale>
<！--边缘缓冲率-->
<edge_buffer_ratio>0.02</edge_buffer_ratio>
</terrain>
```

3）加载高程数据。使用加载 DEM 图的方式加载高程数据，DEM 图是一种每个图元像素都包含了地理信息，如经纬度、高程信息的 TIF 格式图。在此使用的 DEM 图为包含了中国范围内的 1Km 精度的高程信息。加载高程数据的操作，可以先贴上一张高程精度较低的 DEM 图，然后在此基础上，根据需求在特定区域贴上精度更高的 DEM 图，如下图所示。之所以要选择不同精度的不同范围的 DEM 图，是因为精度如果太高，就会导致 DEM 图非常大，加载起来就会非常困难，因此要合理安排 DEM 图的精度。

代码 4：加载高程数据配置项

```
<！--全国的高程,1KM-->
<heightfield name="china_DEM_1KM" driver="gdal">
<url>/media/ybw/SSD/OSG_DATA/MAIN_DATA/eleva-
tions/dem_1KM.tif</url>
<tile_size>32</tile_size>
</heightfield>
<！--局部全国的高程,90M-->
<heightfield name="detail_dem" driver="gdal">
<url>/media/ybw/SSD/OSG_DATA/MAIN_DATA/detail/
srtm_90M_S.tif</url>
<tile_size>32</tile_size>
</heightfield>
```

4）加载纹理信息，即贴图操作。是为了使得地理模型可以更直观且美观的展现，如果没有贴图操作，则显示的地理模型就是一个密密麻麻的点云。贴图操作可以先贴上

一张分辨率较低的全球影像图，然后在此基础上，根据需求在特定区域贴上分辨率更高的影像图。原因同高程数据。

代码 5：加载纹理信息配置项

```
<！ －－定义全球影像图,1KM－－>
<image name="World_Pic" driver="gdal">
<url>/media/ybw/SSD/OSG_DATA/MAIN_DATA/images/world_1KM.tif</url>
</image>
<！ －－部分区域高精度影像－－>
<image name="detail－image" driver="composite">
<image name="i－8" driver="gdal">
<url>/media/ybw/SSD/OSG_DATA/MAIN_DATA/detail/china_8_1KM.tif</url>
</image>
<image name="i－9" driver="gdal">
<url>/media/ybw/SSD/OSG_DATA/MAIN_DATA/detail/china_9_600M.tif</url>
</image>
<image name="i－10" driver="gdal">
<url>/media/ybw/SSD/OSG_DATA/MAIN_DATA/detail/china_10_300M.tif</url>
</image>
<image name="i－11" driver="gdal">
<url>/media/ybw/SSD/OSG_DATA/MAIN_DATA/detail/china_11_150M.tif</url>
</image>
</image>
```

5）确定数据缓存的方式和存储位置。缓存在三维建模中显得较为重要，因为加载一次高程和纹理图的数据量较大，需要较长时间，如果在每一次运行时都去重新加载，那显然是不可接受的，因此 osgEarth 给出的一种方案是，第一次加载完成后，就会生成缓存数据，以后每次加载，则从缓存数据进行加载，这样会显著提升效率。

代码 6：数据缓存的配置项

```
<！ －－文件缓存－－>
<options>
<cache type="filesystem">
<path>/media/ybw/SSD/OSG_DATA/MAIN_DATA/FileCache</path>
</cache>
<lighting>true</lighting>
</options>
```

2.3.3.1.3.1.2　运行并显示三维世界

因为选用 Qt 作为开发工具，要将 osgEarth 显示的地理模型，显示在 Qt 的窗口中。在 osgEarth 开发中，至少需要配置两部分内容，一是加载三维模型，二是设置相机操作器。

1）加载 *.earth 模型。

代码 7：读取模型的代码

```
osg::Node* pNode =
    osgDB::readNodeFile("./Data/simple.earth");
    osgEarth::MapNode* mapNode =
    osgEarth::MapNode::findMapNode(pNode.get());
```

2）初始化相机操作器，即相机参数。osgEarth 的界面显示，实际上是将显示界面当作一个摄像头，显示的是摄像机视角。

代码 8：设置相机操作器的代码

```
osgEarth::Util::EarthManipulator * em = new osgEarth::Util::EarthManipulator;
if(mapNode.valid())
{
  em.get()－>setNode(mapNode);
}
em.get()－>getSettings()－>setArcViewpointTransitions(true);
osgViewer::Viewer * pViewer=this－>getOsgViewer();
pViewer－>setCameraManipulator(em.get());
pViewer－>setSceneData(mRoot.get());//添加到场景
pViewer－>realize();
```

2.3.3.2　航线导入飞控并实施功能

在生成了航线之后，就需要考虑如何将该航线转换为无人机可识别的格式并上传使用。

2.3.3.2.1　功能分析

实现飞行轨迹的导入，需要解决几方面的问题。

1）将生成的飞行轨迹数据整理成飞控程序能识别的格式。

2）将整理后的飞行轨迹数据上传至飞控。

3）驱动飞行器按既定飞行轨迹飞行。

2.3.3.2.2　程序设计

大疆无人机设备基于 C＋＋的二次开发，大疆官方提供了 OSDK 开发工具包，目前最新的版本为 V4.0.0。

2.3.3.2.2.1　DJI OSDK 简介

在此介绍 OSDK 开发包。

1）DJI OSDK 是由大疆自主开发的一个运行于桌面端的软件库，贴合用户自主进行二次开发的需要，可个性定制对无人机在飞行中与自动化相关的的需求，其提供了一些基础的 API，用于实现基本的运动规划功能；开发者在使用调用规划相关的库的同时，可以根据个人需要，对功能进一步扩展，可根据实际的使用需求，设计并编写相应的航点任务以及热点任务，制定一套个性化控制无人机自动化飞行的控制逻辑。

2）在无人机标校中主要用到的功能是航点规划的功能。航点规划，简单来说就是通过设置一系列的轨迹点，连接成线，然后将航迹上传给无人机，指令控制无人机按照指定的航线启动飞行，实现无人机根据既定航线自动化飞行的控制功能。开发者通过调用 DJI OSDK 的 API 接口，能够控制无人机以指定的高度、方向飞往既定的位置，并且执行约定好的动作，根据任务需求，还可以编写更多的任务动作，控制无人机多次重复执行该任务，以此实现多轮次标校的功能。

3）开发者在代码开发中，调用航点任务相关的 API，需要指定的参数为航点数量和对应的航点类型。航点数量在最新版本的 API 中，最多可设置 65535 个点。航点类型主要包含三种飞行模式，分别为曲率飞行、直线飞行和协调转弯。

4）OSDK 库文件同时还为开发者提供了更多方便简洁的功能，以便直接调用，比如对速度的控制功能，开发者能够按照既定任务要求，为不同的航点配置不同的速度，当然，也可以为同一个航点设置多个速度。另外，在无人机执行航线飞行任务时，修改或查询无人机全局巡航速度等的状态数据的函数，也是事先写好，可以直接调用的。

2.3.3.2.2.2 主要函数类

为实现航点任务功能，利用 OSDK 函数库，主要将设计到以下几个代码文件：

1）dji _ mission _ type.hpp 文件。该文件主要用于定义在航线自动飞行中会用到的一些数据的结构体文件，主要的内容为 WaypointV2 结构体，其中主要涉及到的内容有：longitude 为精度值，latitude 为纬度值，relativeHeight 为高程值，maxFlightSpeed 为最大速度值，autoFlightSpeed 为自动巡航速度值。

代码 9：WaypointV2 结构体内容

```
typedef struct WaypointV2
{
float64_t longitude;
float64_t latitude;
float32_t relativeHeight; /*! relative to takeoff height*/
DJIWaypointV2FlightPathMode waypointType;
DJIWaypointV2HeadingMode headingMode;
WaypointV2Config config;
uint16_t dampingDistance;
float32_t heading;
DJIWaypointV2TurnMode turnMode;
RelativePosition pointOfInterest;
float32_t maxFlightSpeed;
float32_t autoFlightSpeed;
}WaypointV2;
```

2）dji _ mission _ base.hpp 文件。在这部分代码中，主要实现的是对航线任务的控制函数，start（）实现启动任务，stop（）实现停止任务，pause（）实现暂停任务，resume（）实现重启任务。

代码 10：dji _ mission _ base.hpp 主要代码

```
virtual void start(VehicleCallBack callback = 0, UserData userData=0)=0;
virtual ACK::ErrorCode start(int timer) = 0;
virtual void stop(VehicleCallBack callback = 0, UserData userData = 0) = 0;
virtual ACK::ErrorCode stop(int timer) = 0;
virtual void pause(VehicleCallBack callback = 0, UserData userData = 0) = 0;
virtual ACK::ErrorCode pause(int timer) = 0;
virtual void resume(VehicleCallBack callback = 0, UserData userData = 0)=0;
virtual ACK::ErrorCode resume(int timer) = 0;
```

3）dji _ waypoint _ v2.hpp、dji _ waypoint _ v2.cpp 代码文件。其中的 WaypointV2MissionOperator 类为主要功能实现的类，在该类中主要涉及到的函数有，init（WayPointV2InitSettings *info，int timeout）函数为实现该类函数的初始化操作，其中 WayPointV2InitSettings 为航线任务结构体，在该结构体中主要规定了任务 ID 值，missionID，任务重复次数 repeatTimes，航线结束时的动作 finishedAction，最大飞行速度 maxFlightSpeed，自动飞行速度 autoFlightSpeed，前往第一个任务点的方式 gotoFirstWaypointMode，以及装载了任务航点的容器 mission。downloadInitSetting（）即从飞控设备下载航迹任务的初始化配置信息。start（）函数用于启动航线任务，stop（）为停止航线任务，pause（）函数用于暂停飞行任务，resume（）用于重启飞行任务，注意到，start（）、stop（）、pause（）、resume（）四个函数都继承自 dji _ mission _ base.hpp 代码。uploadMission（）函数用于上传任务航线，downloadMission（）用于下载在飞控中的任务航线。uploadAction（）用于上传飞行器动作。RegisterMissionEventCallback（）和 RegisterMissionStateCallback（）分别用于反馈目前的动作和状态信息。

代码 11：dji _ waypoint _ v2.hpp 主要代码：

```
class WaypointV2MissionOperator
{
public:
const uint16_t MAX_WAYPOINT_NUM_SIGNAL_PUSH = 260;
WaypointV2MissionOperator(Vehicle* vehiclePtr);
~WaypointV2MissionOperator();
ErrorCode::ErrorCodeType init(WayPointV2InitSettings *info, int timeout);
ErrorCode::ErrorCodeType downloadInitSetting(WayPointV2InitSettingsInternal &info, int timeout);
ErrorCode::ErrorCodeType start(int timeout);
ErrorCode::ErrorCodeType stop(int timeout);
ErrorCode::ErrorCodeType pause(int timeout);
ErrorCode::ErrorCodeType resume(int timeout);
ErrorCode::ErrorCodeType uploadMission(int timeout);
ErrorCode::ErrorCodeType downloadMission(std::vector<WaypointV2>&mission, int timeout);
ErrorCode::ErrorCodeType getGlobalCruiseSpeed(GlobalCruiseSpeed &cruiseSpeed, int timeout);
ErrorCode::ErrorCodeType setGlobalCruiseSpeed(const GlobalCruiseSpeed &cruiseSpeed, int timeout);
ErrorCode::ErrorCodeType uploadAction(std::vector<DJIWaypointV2Action>&actions, int timeout);
ErrorCode::ErrorCodeType getActionRemainMemory(GetRemainRamAck &remainRamAck, int timeout);
ErrorCode::ErrorCodeType getWaypointIndexInList(GetWay-
```

```
pontStartEndIndexAck &startEndIndexAck, int timeout);
    inline DJIWaypointV2MissionState getCurrentState() { return currentState; }
    inline DJIWaypointV2MissionState getPrevState() { return prevState; }
    void setPrevState(DJIWaypointV2MissionState state) { prevState = state; }
    void setCurrentState(DJIWaypointV2MissionState state) {currentState = state; }
    float32_t getTakeoffAltitude(){return takeoffAltitude;};
    void setTakeoffAltitude(float32_t altitude){ takeoffAltitude = altitude;};
    void RegisterMissionEventCallback(void * userData, PushCallback cb = NULL);
    void RegisterMissionStateCallback(void * userData, PushCallback cb = NULL) ;
    private:
    std::vector<WaypointV2> missionV2;
    DJIWaypointV2MissionState currentState;
    DJIWaypointV2MissionState prevState;
    Vehicle * vehiclePtr;
    float32_t takeoffAltitude;
    void RegisterOSDInfoCallback(Vehicle * vehiclePtr);
    };
```

2.3.3.2.2.3　流程设计

航线导入并启动的功能实施流程如下：

1）在之前已经设计好航线的前提下，需要将该航行的数据点存储模式修改为大疆指定的数据点存储方式；

2）将生成的航线通过 API 调用上传航线任务的整体信息。

3）一个航点任务包含航线飞行任务的 ID、航点任务的航点数、任务重复次数、航点任务结束后的动作、最大飞行速度和巡航速度。

4）上传航点信息基础参数：航点坐标（设置航点的经度、纬度和相对于起飞点的高度）、航点类型、航向类型和飞行速度。可选参数：缓冲距离、航向角度、转向模式、兴趣点、单点最大飞行速度、单点巡航速度。

5）将航线任务上传至标校无人机飞控。

6）控制无人机执行航点任务。上传完成后，标校无人机可通过开发者之前预设的动作，如开始、停止或暂停任务、设置或获取巡航速度等。

2.3.3.2.2.4　程序实现

本章节内容旨在将上一章节内容以代码实现。

1）首先判断是否支持航线任务规划的功能，通过如下代码。

代码 12：判断函数是否支持功能

```
if (! vehiclePtr->isM300()) {
DSTATUS("This sample only supports M300!"); /* 当前 waypoint v2 仅支持 M300 机型 */
    return false;
    }
    int timeout = 1;
    GetRemainRamAck actionMemory = {0};
    ErrorCode::ErrorCodeType ret;
    if(! setUpSubscription(timeout))
    {
    DERROR("Failed to set up subscription!");
    return -1;
    }
    else
    {
    DSTATUS("Set up subscription successfully!");
    }
    /*! wait for subscription data come */
    sleep(timeout);
```

2）而后创建 mission 任务，初始化 mission 任务，用到 initMissionSetting () 函数，该函数实现了任务初始化配置的功能，部分代码见代码 7 内容。

代码 13：上传任务的代码

```
/*! init mission */ /* 初始化 mission 任务 */
ret = initMissionSetting(timeout);
if(ret != ErrorCode::SysCommonErr::Success)
return ret;
sleep(timeout);
```

3）其中代码 8 中为添加航迹点的核心代码。是 initMissionSetting 代码中内容，通过这段代码，添加每个航点信息。

代码 14：上传任务的代码

```
/*! Init waypoint settings */
WayPointV2InitSettings missionInitSettings;
missionInitSettings.missionID = rand();
missionInitSettings.repeatTimes = 1;
missionInitSettings.finishedAction = DJIWaypointV2MissionFinishedGoHome;
missionInitSettings.maxFlightSpeed = 10;
missionInitSettings.autoFlightSpeed = 2;
missionInitSettings.exitMissionOnRCSignalLost = 1;
missionInitSettings.gotoFirstWaypointMode = DJIWaypointV2MissionGotoFirstWaypointModePointToPoint;
missionInitSettings.mission = generatePolygonWaypoints(radius, polygonNum);
missionInitSettings.missTotalLen = missionInitSettings.mission.size();
ErrorCode::ErrorCodeType ret = vehiclePtr->waypointV2Mission->init(&missionInitSettings,timeout);
```

4）代码 9 的内容是在代码 8 中进一步展开，特别将 generatePolygonWaypoints () 函数单独讲解，这部分内容是 generatePolygonWaypoints () 函数的内容。该函数实现了航点数据的添加功能。这一步函数调用，也是实现在飞行轨迹设计完成之后，将设计的航迹进行转换，使得无人机飞控可以使用的关键。

代码 15：上传任务的代码

```
std::vector<WaypointV2> WaypointV2MissionSample::gen-
```

```
eratePolygonWaypoints(float32_t radius, uint16_t polygonNum) {
    // Let's create a vector to store our waypoints in.
    std::vector<WaypointV2> waypointList;
    WaypointV2 startPoint;
    WaypointV2 waypointV2;
    Telemetry::TypeMap<TOPIC_GPS_FUSED>::type subscribeGPosition = vehiclePtr->subscribe->getValue<TOPIC_GPS_FUSED>();
    startPoint.latitude = subscribeGPosition.latitude;
    startPoint.longitude = subscribeGPosition.longitude;
    startPoint.relativeHeight = 15;
    setWaypointV2Defaults(startPoint);
    waypointList.push_back(startPoint);
    // Iterative algorithm
    for (int i = 0; i < polygonNum; i++) {
    float32_t angle = i * 2 * M_PI / polygonNum;
    setWaypointV2Defaults(waypointV2);
    float32_t X = radius * cos(angle);
    float32_t Y = radius * sin(angle);
    waypointV2.latitude = X/EARTH_RADIUS + startPoint.latitude;
    waypointV2.longitude = Y/(EARTH_RADIUS * cos(startPoint.latitude)) + startPoint.longitude;
    waypointV2.relativeHeight = startPoint.relativeHeight ;
    waypointList.push_back(waypointV2);
    waypointList.push_back(startPoint);
    return waypointList;
    }
```

5）而后将自定义的航线任务上传至无人机飞控。见代码 8 内容。

代码 16：上传任务的代码

```
/*! upload mission */
/*! upload mission's timeout need to be longer than 2s*/
int uploadMissionTimeOut = 3;
ret = uploadWaypointMission(uploadMissionTimeOut); /*上传任务,注意注释部分,这里的超时需要设置为 3s,设置为 1 时,可能因为上传任务过多,报 index 错误*/
if(ret != ErrorCode::SysCommonErr::Success)
return ret;
sleep(timeout);
```

6）从飞控中下载航线，这一步可以起到检查航线的作用。

代码 17：下载并检查任务的代码

```
/*! download mission */ /* 下载任务,可以查看对比上传的任务 debug 的时候可以用来对比排查问题 */
std::vector<WaypointV2> mission;
ret = downloadWaypointMission(mission, timeout);
if(ret != ErrorCode::SysCommonErr::Success)
return ret;
sleep(timeout);
```

7）而后上传在航线进行过程中，需要涉及的一些飞行器动作。

代码 18：上传飞行器动作的代码

```
/*! upload actions */ /* 上传 actions: */
/*! check action memory */
ret = getActionRemainMemory(actionMemory, timeout); //上传 action 前先检查 action 容量,避免上传的任务过多导致 action 被挤掉了,执行动作可能不完整。最大貌似为 65535 byte
if (actionMemory.remainMemory <= 0)
{
DSTATUS("action memory is not enough. Can not upload more action!");
return ErrorCode::SysCommonErr::UndefinedError;
}
ret = uploadWapointActions(timeout);
if(ret != ErrorCode::SysCommonErr::Success)
return ret;
ret = getActionRemainMemory(actionMemory, timeout);
sleep(timeout);
```

8）开始执行飞行任务。

代码 19：开始执行飞行任务的代码

```
/*! start mission */
ret = startWaypointMission(timeout); //开始执行
if(ret != ErrorCode::SysCommonErr::Success)
return ret;
sleep(20);
```

9）设置飞行过程中的一些参数，比如最大飞行速度等。

代码 20：设置飞行参数的代码

```
/*! set global cruise speed */
setGlobalCruiseSpeed(1.5, timeout); //设置巡航速度,巡航速度为全程航线的巡航速度,不是通过单点航点任务来设置的。可以在单点航点中更改,间接实现不同的航点以不同的速度运行。
sleep(timeout);
```

10）获取巡航速度，以判断飞行器飞行状态。

代码 21：获取飞行参数的代码

```
/*! get global cruise speed */
getGlobalCruiseSpeed(timeout);
sleep(timeout);
```

11）可以测试暂停飞行。

代码 22：暂停飞行的代码

```
/*! pause the mission */
ret = pauseWaypointMission(timeout);
if(ret != ErrorCode::SysCommonErr::Success)
return ret;
sleep(5);
```

12）测试重启飞行任务的功能。

代码 23：重启飞行的代码

```
printf("kyle test: pauseWaypointMission and resumeWaypointMission/n");
/*! resume the mission */
ret = resumeWaypointMission(timeout); //暂停及回复航线指令
if(ret != ErrorCode::SysCommonErr::Success)
```

```
return ret;
sleep(50);
```

2.3.3.3　远控机载应答机功能

当无人机按照既定设计的航线自动飞行后，就需要考虑标校无人机机载应答机的控制问题。

2.3.3.3.1　功能分析

这部分功能需要根据应答机厂家所提供的 API 接口进行开发，但是无论是哪家厂家，根据使用者的需求对接，软件接口至少包含以下内容。

1）开关机控制；

2）信号频率的选择；

3）应答机信号发送强度的衰减值；

4）在应答和信标模式之间切换；

5）射频信号是否允许输出。

6）网络通信功能。可以将这部分功能设计为网络通信的形式，与近端的应答机控制模块进行通信，使其控制远端的应答机。

2.3.3.3.2　程序设计

根据需求分析，可以将功能分为两部分，一部分是参数设置模块，一部分是网络通信模块。

1）参数设置模块。设计一个结构体，将所有的参数包含在内，以方便使用和查看。

代码 24：远控应答机参数设置模块

```
Typedef struct
{
unsigned char OnOff;//开关机控制
unsigned char FreqChoose;//频率选择
unsigned char Attenuator;//衰减器控制
unsigned char WorkMode;//工作方式
unsigned char SPout;//射频输出允许
}
```

2）网络通信模块。

代码 25：远控应答机网络通信模块

```
QudpSocket * SendudpSocket;
SendudpSocket - > writeDatagram ( buf, len, QhostAddress (FSendIp),FSendPort);
```

3　无人机标校应用场景

在某次车载测控设备转场后，受场地限制，没有标校杆或者标校塔可供其进行标校，此时，需要起降标校无人机，进行标校。

1）利用无人机标校系统生成飞行轨迹功能，输入当地的环境因素，如温湿压，海拔，以及当地坐标位置等参数，利用 OSG 在三维仿真空间中，生成飞行轨迹；

2）利用无人机标校系统导入航线的功能，通过 OSDK 上传飞行轨迹；

3）利用无人机标校系统控制无人机的功能，控制无人机起飞，并飞行至等待点位置；

4）通过无人机标校系统远控机载应答机的功能，设置合适的应答机频点及强度，并开启应答机；

5）标校无人机根据飞行轨迹实施完毕静态及动态标校，并返航。

4　结束语

通过设计从生成飞行轨迹、到将飞行轨迹导入飞控并实施以及远控应答机完整功能的软件，并配套相应的硬件模块，提升了无人机标校的可靠性、为后续无人机标校工作的推动提供了一套可行的技术方案。

参考文献：

[1] 王开乐. 基于 OSG 的结构数据可视化展示研究 [J]. 人民长江，2021，52 (S2)：330 - 334.

[2] 崔晓宇，刘志春. 基于 osgEarth 的三维态势显示系统设计 [J]. 数据，2022 (3)：80 - 82.

[3] 邹进贵，翟若明. OSGEarth 的 3D 场景可视化关键技术实现 [J]. 测绘地理信息，2020，45 (2)：51 - 55.

[4] 施　斌，王　华，等. 基于 OSGEARTH 的国产航天三维仿真软件设计 [J]. 计算机系统应用，2019，28 (12)：105 - 111.

[5] 闫大洲. 基于 osgEarth 的卫星轨道外推数据可视化仿真系统研究 [D]. 太原：中北大学，2018.

[6] 韩　哲，刘玉明，等. osgEarth 在三维 GIS 开发中的研究与应用 [J]. 现代防御技术，2017 (2)：25 - 32.

[7] 张　明. 无人机三维航迹监视软件设计 [J]. 电子技术与软件工程，2018 (23)：41.

[8] 李迎春. 服务飞行器航天员舱内操控辅助可视化系统设计与实现 [D]. 南昌：南昌航空大学，2018.

[9] 尉嘉维 ，等. 基于 Qt 的便携式应答机测试设备软件设计 [D]. 成都：电子科技大学，2020.

[10] 肖　强. 机载应答机数据协议解析与实现 [D]. 成都：电子科技大学，2018.

[11] 徐东东. 微波应答机地面测控系统研制 [D]. 哈尔滨：哈尔滨工业大学，2015.

[12] 周　阳. 小型测控应答机的研制与改进 [D]. 杭州：浙江大学，2011.

[13] 丁栋威. 箭载应答机测试系统及监控软件设计 [D]. 成都：西南交通大学，2008.

[14] 闫复利. 通用化测控应答机设计 [J]. 信息通信，2020 (8)：88 - 91.

[15] 冯李民. 基于二次开发平台的无人机飞行试验航迹规划仿真 [J]. 智能计算机与应用，2018，8 (1)：91 - 94.

[16] 李敏剑，王向伟，刘佳伟多旋翼无人机在某测控设备标校工作中的应用 [J]. 遥测遥控，2021，42 (3)：26 - 33.

[17] 张　磊，郑庆利. 一种基于无人机的雷达角度零值标校方法 [J]. 中国科技信息，2020 (24)：68 - 69.

[18] 吕晓林. 无人机测控系统误差分析 [J]. 宇航计测技术，2019，39 (5)：79 - 83.

[19] 刘爱东，李知宇，王　丰，等. 基于无人机的标校系统软件设计与实现 [J]. 计算机测量与控制，2018，26 (1)：141 - 144.

[20] 耿　炎. 无人机地面站标校方法研究 [J]. 无线电工程，2016，46 (11)：68 - 70.