

任务规划流程及说明

航线飞行 (waypoint) 是Mobile SDK重要的功能模块。在地图上选定飞行起始点、航线点、飞行终点以及设定飞行任务以后，飞机可以按照指定的地图选定点开始执行飞行任务

新航线任务支持用户手动选择目标点，通过不同的航线规划算法，自动生成飞行航线。用户可以自定义航线飞行参数以及航点的相机动作。

本教程指导如何通过SDK提供的接口，实现对航线任务的规划、上传、执行、暂停、恢复以及对航线任务执行状态与航线信息的监听等。

一、任务规划流程



二、规划算法

1. 航点任务算法接口

```
val result = AlgorithmManager.getInstance().getWaypointPath(pathMission)
```

1.1 算法入参

```
public class PathMission {  
    /** 航点个数*/  
    public short WPNum;  
    /** 是否使用默认半径, 0: 不用, 1: 用*/  
    public short default_R_flag;  
    /** Home点坐标,经纬高*/  
    public double[] HomeLLA;//3  
    /** 航点信息 (目前开放500个) */  
    public PathPoint[] WP_Info_strc;  
    public PathMission() {  
        HomeLLA = new double[3];  
        WP_Info_strc = new PathPoint[0];  
    }  
}
```

1.2 算法入参

子参数PathPoint：航点信息

```

public class PathPoint {
    /** 航点类型 1-代表停下的点，2-代表协调转弯*/
    public short WPTTypeUsr;
    /**表示WPTType=3的时候，自动协调拐弯的弧线距离相应航点的最远距离 范围大于0，小于100 值默
    认为2，单位m*/
    public float WP2ArcDistUsr;
    /** 用户设置的航点，经纬高 */
    public double[] WPLLAUsr;//3
    /** 用户设置转弯半径*/
    public double RadUsr;
    /** 速度 */
    public double VelRefUsr;
    /** 航点高度优先级 1表示优先级比前一个点高，-1表示比前一个点低，0表示无优先级 */
    public short AltPrioUsr;
    /** 此航点中 动作朝向模式 1表示协调转弯，2表示手动，3表示自定义 */
    public short Heading_Mode;
    /** 兴趣点是否有效 -1 表示无兴趣点， 1 表示有兴趣点 */
    public short POI_Valid;
    /** 对应兴趣点的坐标，经纬高 */
    public double[] POIUsr;//3
    /** 航点的应急动作类型 0 为无动作，1 为悬停，2 为降落，3 为返航*/
    public short EmgActUsr = 3;
    /** 此航点相机动作总数 */
    public short ActionNum;
    /** 航点动作（10个）实际Action_Num个，最大为10 */
    public CameraActionJNI[] MSN_ActionInfo;//10
}

```

1.3 算法入参

子参数CameraActionJNI: 相机动作

```

public class CameraActionJNI {
    /** 动作类型 */
    public int Action_Type;
    /** 云台Pitch角 */
    public float Gimbal_Pitch;
    /** 云台Roll角 */
    public float Gimbal_Roll;
    /** 偏航角度 */
    public float Action_Yaw_Ref;
    /** 定时拍照间隔(s) */
    public int Shoot_Time_Interval;
    /** 定距拍照距离间隔(mm) */
    public float Shoot_Dis_Interval;
    /** 动作执行时长(s) */
    public int Action_Time;
    /** 变焦倍数 */
    public int Zoom_Rate;
    /** 预留 */
    public int[] reserved;//2
}

```

1.4 算法出参

```

public class PathResultMission {
    /** 总飞行时间 */
    public float T_ttl_fly;
    /** 总飞行路程 */
    public float L_ttl_fly;
    /** 总照片张数 */
    public int Photo_Num;
    /** 面积,航点任务没有 */
    public double area;
    /** 航线个数 */
    public short FPNum;
    /** 绘图点个数 */
    public int Pts4PlotNum;
    /** 无人机机头朝向*/
    public float UAVHeading;
    /** 所有绘图点坐标展开 (Pts4PlotNum个经度, Pts4PlotNum个维度, Pts4PlotNum个高度, 剩下的为0 有效数值为前3*Pts4PlotNum个 */
    public double[] Pts4PlotLLA;//141030
    /** 箭头个数 */
    public int ArrowNum;
    /** 箭头经纬高和方向(以正东为x轴, 正南为y, 向下为z)。所有箭头点坐标展开 (ArrowNum个纬度, ArrowNum个经度, ArrowNum个高度, ArrowNum个箭头方向,有效数值为前4*ArrowNum个**/
    public double[] ArrowPosDirLLA;
    /** 航点处可设置的最大半径 */
    public float[] R_max;
    /** 航线信息 (1000个) 有效点为FPNum个 */
    public PathResultLine[] FP_Info_strc;//1010
    /**
     * 算法调用返回的航线错误码
     * //0-normal,1-相邻航点间距离超过10公里, 2-相邻航点间高度差超过1公里, 3-包含1和2的错误总和
     */
    public short errorCode;
}

```

算法出参 (子参数: PathResultLine*航点任务中的一条航线*)

```

public class PathResultLine {
    /** 航点类型 1代表停下的点, 2代表协调转弯 */
    public short WPTTypeExe;
    /** 起点坐标, 经纬高 */
    public double[] WPPrevLLAExe;//3
    /** 终点坐标, 经纬高 */
    public double[] WPCurrLLAExe;//3
    /** 圆心坐标, 经纬高 */
    public double[] WPCentLLAExe;//3
    /** 航点速度 */
    public float VelRef_FP;
    /** 下个航点速度 */
    public float VelRefNxt_FP;
    /** 航点高度优先级 1 代表当前最高, 0 代表等优先级, -1 代表当前最低*/
    public short AltPrio_FP;
    /** 航线长度 */
    public float FP_length;
    /** 航线预计飞行时间 */
    public float T_curr;
}

```

```

/** 此航点中动作朝向模式 1表示协调转弯, 2表示手动, 3表示自定义 */
public short Heading_Mode_FP;
/** 兴趣点是否有效 -1 表示无兴趣点, 1 表示有兴趣点 */
public short POI_Valid_FP;
/** 对应兴趣点的坐标, 经纬高 */
public double[] POI_FP;//3
/** 此航点动作总数 */
public short ActionNum_FP;
/* 每个航段的首航点的应急动作类型。0 为无动作, 1 为悬停, 2 为降落, 3 为返航*/
public short EmgAct_FP;
/**是否有航段动作*/
public short ActExist;
/** 航点动作（11个）实际ActionNum_FP+1个, 最后一个是一个航点的第一个动作 */
public PathResultCameraAction[] MSN_ActionInfo;//11
}

```

算法出参（子参数：PathResultCameraAction相机动作）

```

public class PathResultCameraAction {
    /** 动作类型 */
    public float Action_Type;
    /** 云台Pitch角 */
    public float Gimbal_Pitch;
    /** 云台Roll角 */
    public float Gimbal_Roll;
    /** 偏航角指令 */
    public float Action_Yaw_Ref;
    /** 定时拍照间隔 */
    public float Shoot_Time_Interval;
    /** 定距拍照距离间隔 */
    public float Shoot_Dis_Interval;
    /** 动作执行时长 */
    public float Action_Time;
    /** 变焦倍数 */
    public float Zoom_Rate;
    /** 预留 */
    public float[] reserved;
}

```

三、任务文件生成

航线任务算法出来的结果最终要转换成任务文件，上传给飞机执行；即 PathResultMission->MissionInfoJNI

任务文件结构（MissionInfoJNI）

```

public class MissionInfoJNI {
    /** 任务Index */
    public int Mission_ID;
    /** 相对高度or海拔高度(m) */
    public int Altitude_type;
    /** 任务类型 */
    public int Mission_type;
    /** 任务完成后执行动作 1悬停 2返航 3降落 */
    public int Finish_Action;
    /** 图传断联后执行动作 */
    public int RC_Lost_Action;
}

```

```

/** 最小绕障距离 单位cm*/
public int Min_OA_Dist;
/** 航点个数 */
public int Waypoint_Num;
/** 避障模式 */
public int Obstacle_Mode;
/** 测绘是否开启双网格 */
public int Gride_Enable_Mapping;
/** 相机视场角 degree */
public float VFOV_Mapping;
/** 测绘航向角度 degree */
public float Yaw_Ref_Mapping;
/** 航向重叠率 */
public int Overlap_Mapping;
/** 测绘云台俯仰角度 degree */
public float Gimbal_Pitch_Mapping;
/** 任务总时长(0.01s) */
public int Mission_Time;
/** 任务总路程(0.01m) */
public int Mission_Length;
/** 对应航点信息 * 500 */
public WaypointInfoJNI[] waypoints;
/** 预留:0:高程优化标志位;1:和航向垂直于航线参数 */
public int[] reserved;//2
/** 航线相机动作 */
public CameraActionJNI Action_Default ;
/** 任务唯一ID: 每次上传任务时生成, 任务启动依赖此Id作为参数 */
public long GUID;
}

```

四、任务上传

初始化任务文件保存目录:

```

import com.autel.drone.sdk.libbase.common.dsp.FileConstants
//初始化Mission文件保存目录: app私有目录
FileConstants.init(this)

```

上传任务:

```

DeviceManager.getDeviceManager().getFirstDroneDevice()?.getWayPointMissionManager().uploadMissionFile(
    missionInfoJNI: MissionInfoJNI,
    callback: CommonCallbacks.CompletionCallbackWithProgressAndParam<Long> )

```

五、启动任务

```

DeviceManager.getDeviceManager().getFirstDroneDevice()?.getWayPointMissionManager().startMission(
    guid: MissionWaypointGUIDBean,
    callback: CommonCallbacks.CompletionCallbackWithParam<Void>)

```

六、暂停任务

```
DeviceManager.getDeviceManager().getFirstDroneDevice()?.getWayPointMissionManager().pauseMission(  
    callback : CommonCallbacks.CompletionCallbackWithParam<Void>)
```

七、恢复任务

```
DeviceManager.getDeviceManager().getFirstDroneDevice()?.getWayPointMissionManager().resumeMission(  
    guid: MissionWaypointGUIDBean,  
    callback: CommonCallbacks.CompletionCallbackWithParam<Void>)
```

八、停止任务

```
DeviceManager.getDeviceManager().getFirstDroneDevice()?.getWayPointMissionManager().exitMission(  
    callback: CommonCallbacks.CompletionCallbackWithParam<Void>)
```

九、监听任务状态

```
DeviceManager.getDeviceManager().getFirstDroneDevice()?.getWayPointMissionManager().addwaypointMissionExecuteStateListener(  
    listener: CommonCallbacks.KeyListener<MissionWaypointStatusReportNtfyBean>)
```

十、取消任务状态监听

```
DeviceManager.getDeviceManager().getFirstDroneDevice()?.getWayPointMissionManager().removewaypointMissionExecuteStateListener(  
    listener: CommonCallbacks.KeyListener<MissionWaypointStatusReportNtfyBean>)
```

十一、查询断点任务信息

```
DeviceManager.getDeviceManager().getFirstDroneDevice()?.getWayPointMissionManager().queryMissionBreakpointInfo(  
    param: MissionWaypointGUIDBean,  
    callback:  
    CommonCallbacks.CompletionCallbackWithParam<MissionWaypointBreakRspBean>)
```