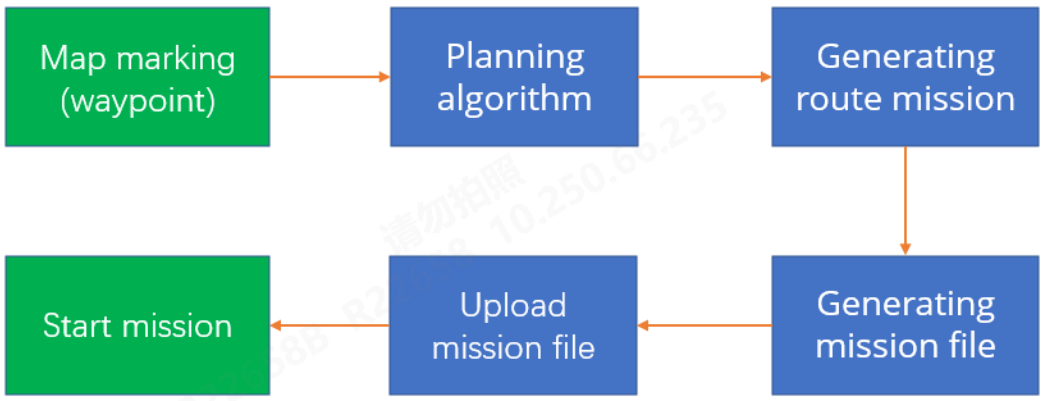
Task Planning Process and Explanation

The waypoint flight is an important functionality module in the Mobile SDK. After selecting the starting point, waypoints, and the destination on the map, the aircraft can navigate to the specified waypoints.

To execute waypoint flight tasks, the system supports users in manually selecting target points and automatically generating flight routes using various route planning algorithms. Users can customize the flight parameters for the route and define camera actions for waypoints.

This tutorial provides guidance on how to use the SDK's interfaces to plan, upload, execute, pause, resume, and monitor the execution status and route information of waypoint flight tasks.

1、Task Planning Process.



2. Planning Algorithms

1. Waypoint Task Algorithm Interface

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

1.1 Algorithm Input Parameters

```
public class PathMission {
    /** Number of waypoints */
    public short WPNum;
    /** Whether to use the default radius, 0: No, 1: Yes */
    public short default_R_flag;
    /** Home point coordinates in longitude, latitude, and altitude */
    public double[] HomeLLA;//3
    /** Waypoint information (currently supporting up to 500 waypoints) */
    public PathPoint[] WP_Info_strc;
    public PathMission() {
        HomeLLA = new double[3];
        WP_Info_strc = new PathPoint[0];
    }
}
```

1.2 Algorithm Input Parameters

Sub parameter PathPoint: Waypoint Information

```
public class PathPoint {
    /** Waypoint type: 1 for stopping point, 2 for coordinated turn */
    public short WPTypeUsr;
    /** When WPType=3, the maximum distance of the automatic coordinated turn
arc from the corresponding waypoint, range greater than 0, less than 100,
default value is 2, unit in meters */
    public float WP2ArcDistUsr;
    /** User-defined waypoint coordinates in longitude, latitude, and altitude
*/
    public double[] WPLLAUsr;//3
    /** User-defined turn radius */
    public double RadUsr;
    /** Speed */
    public double VelRefUsr;
    /** Waypoint altitude priority: 1 for higher priority than the previous
point, -1 for lower priority than the previous point, 0 for no priority */
    public short AltPrioUsr;
    /** Action direction mode in this waypoint: 1 for coordinated turn, 2 for
manual, 3 for custom */
    public short Heading_Mode;
    /** Whether the waypoint has a valid point of interest (POI): -1 for no POI,
1 for a valid POI */
    public short POI_Valid;
    /** Coordinates of the corresponding POI in longitude, latitude, and
altitude */
    public double[] POIUsr;//3
    /** Emergency action type for this waypoint: 0 for no action, 1 for hover, 2
for landing, 3 for return to home */
    public short EmgActUsr = 3;
    /** Total number of camera actions at this waypoint */
    public short ActionNum;
    /** Waypoint actions (up to 10) with actual Action_Num, maximum is 10 */
    public CameraActionJNI[] MSN_ActionInfo;//10
}
```

1.3 Algorithm Input Parameters

Sub parameter CameraActionJNI: Camera Actions

```
public class CameraActionJNI {
    /** Action type */
    public int Action_Type;
    /** Gimbal Pitch angle */
    public float Gimbal_Pitch;
    /** Gimbal Roll angle */
    public float Gimbal_Roll;
    /** Yaw angle */
    public float Action_Yaw_Ref;
    /** Timed photo interval (s) */
    public int Shoot_Time_Interval;
    /** Distance interval for timed photos (mm) */
    public float Shoot_Dis_Interval;
    /** Action execution duration (s) */
```

```
    public int Action_Time;
    /** Zoom ratio */
    public int Zoom_Rate;
    /** Reserved */
    public int[] reserved;//2
}
```

1.4 Algorithm Output Parameters

```
public class PathResultMission {
    /** Total flight time */
    public float T_ttl_fly;
    /** Total flight distance */
    public float L_ttl_fly;
    /** Total number of photos */
    public int Photo_Num;
    /** Area (not applicable for waypoint tasks) */
    public double area;
    /** Number of flight paths */
    public short FPNum;
    /** Number of plotting points */
    public int Pts4PlotNum;
    /** UAV heading */
    public float UAVHeading;
    /** Coordinates of all plotting points (Pts4PlotNum longitude, Pts4PlotNum
latitude, Pts4PlotNum altitude, the rest are zeros, valid values for the first 3
* Pts4PlotNum) */
    public double[] Pts4PlotLLA;//141030
    /** Number of arrows **/
    public int ArrowNum;
    /** Coordinates and directions of arrows (with positive east as the x-axis,
positive south as y, and downward as z). Coordinates of all arrow points
(ArrowNum latitude, ArrowNum longitude, ArrowNum altitude, ArrowNum arrow
direction, valid values for the first 4 * ArrowNum)**/
    public double[] ArrowPosDirLLA;
    /** Maximum radius that can be set at waypoints **/
    public float[] R_max;
    /* Flight path information (up to 1000) with FPNum valid points */
    public PathResultLine[] FP_Info_strc;//1010
    /**
     * Flight path error code returned by the algorithm
     * //0-normal,1-distance between adjacent waypoints exceeds 10 kilometers,
2-height difference between adjacent waypoints exceeds 1 kilometer, 3-sum of
errors 1 and 2
     */
    public short errorCode;
}
```

1.5 Algorithm Output Parameters (Subparameter: PathResultLine a single flight path in waypoint tasks)

```
public class PathResultLine {
    /** Waypoint type: 1 for stopping point, 2 for coordinated turn */
    public short WPTypeExe;
    /** Starting point coordinates in longitude, latitude, and altitude */
    public double[] WPPrevLLAExe;//3
```

```
    /** Ending point coordinates in longitude, latitude, and altitude */
    public double[] WPCurrLLAExe;//3
    /** Center point coordinates in longitude, latitude, and altitude */
    public double[] WPCentLLAExe;//3
    /** Waypoint speed */
    public float VelRef_FP;
    /** Speed at the next waypoint */
    public float VelRefNxt_FP;
    /** Waypoint altitude priority: 1 for highest, 0 for equal, -1 for lowest */
    public short AltPrio_FP;
    /** Flight path length */
    public float FP_length;
    /** Estimated flight time for this flight path */
    public float T_curr;
    /** Action direction mode in this waypoint: 1 for coordinated turn, 2 for
manual, 3 for custom */
    public short Heading_Mode_FP;
    /** Whether the waypoint has a valid point of interest (POI): -1 for no POI,
1 for a valid POI */
    public short POI_Valid_FP;
    /** Coordinates of the corresponding POI in longitude, latitude, and
altitude */
    public double[] POI_FP;//3
    /** Total number of actions at this waypoint */
    public short ActionNum_FP;
    /* Emergency action type for the first waypoint in each flight segment. 0
for no action, 1 for hover, 2 for landing, 3 for return to home */
    public short EmgAct_FP;
    /** Whether there are segment actions */
    public short ActExist;
    /** Waypoint actions (up to 11) with actual ActionNum_FP+1, the last one is
the first action of the next waypoint */
    public PathResultCameraAction[] MSN_ActionInfo;//11
}
```

1.6 Algorithm Output Parameters (Subparameter: PathResultCameraAction camera actions)

```
public class PathResultCameraAction {
    /** Action type */
    public float Action_Type;
    /** Gimbal Pitch angle */
    public float Gimbal_Pitch;
    /** Gimbal Roll angle */
    public float Gimbal_Roll;
    /** Yaw angle command */
    public float Action_Yaw_Ref;
    /** Timed photo interval */
    public float Shoot_Time_Interval;
    /** Distance interval for timed photos */
    public float Shoot_Dis_Interval;
    /** Action execution duration */
    public float Action_Time;
    /** Zoom ratio */
    public float Zoom_Rate;
    /** Reserved */
    public float[] reserved;
}
```

3、Mission File Generation

The results obtained from the waypoint mission algorithm need to be converted into a mission file for upload and execution by the aircraft.

This is achieved by converting PathResultMission into MissionInfoJNI.

Mission file structure (MissionInfoJNI):

```
public class MissionInfoJNI {
    /** Mission Index */
    public int Mission_ID;
    /** Relative altitude or altitude above sea level (meters) */
    public int Altitude_type;
    /** Mission type */
    public int Mission_type;
    /** Action to perform after mission completion: 1 for hover, 2 for return to
home, 3 for landing */
    public int Finish_Action;
    /** Action to perform on RC signal loss */
    public int RC_Lost_Action;
    /** Minimum obstacle avoidance distance (in centimeters) */
    public int Min_OA_Dist;
    /** Number of waypoints */
    public int Waypoint_Num;
    /** Obstacle avoidance mode */
    public int Obstacle_Mode;
    /** Whether double grid is enabled in mapping */
    public int Gride_Enable_Mapping;
    /** Camera field of view (degrees) */
    public float VFOV_Mapping;
    /** Mapping heading angle (degrees) */
    public float Yaw_Ref_Mapping;
    /** Heading overlap rate */
    public int Overlap_Mapping;
    /** Gimbal pitch angle for mapping (degrees) */
    public float Gimbal_Pitch_Mapping;
    /** Total mission duration (0.01 seconds) */
    public int Mission_Time;
    /** Total mission length (0.01 meters) */
    public int Mission_Length;
    /** Waypoint information corresponding to each waypoint (up to 500) */
    public WaypointInfoJNI[] Waypoints;
    /** Reserved: 0 for altitude optimization flag, 1 for parameters
perpendicular to heading */
    public int[] reserved;//2
    /** Default camera action for the mission */
    public CameraActionJNI Action_Default;
    /** Unique ID for the mission: generated each time a mission is uploaded and
used as a parameter for mission startup */
    public long GUID;
}
```

4、Mission Upload

Initialize the mission file storage directory:

```
// Initialize the Mission file storage directory: app private directory
FileConstants.init(this)
```

Upload the mission:

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

5、Start Mission

```
DeviceManager.getDeviceManager().getFirstDroneDevice()?.getWayPointMissionManage
r().startMission(
                    guid:
MissionWaypointGUIDBean,callback:CommonCallbacks.CompletionCallbackWithParam<Voi
d>)
```

6、Pause Mission

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

7、Resume Mission

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

8、Stop Mission

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

9、Mission Status Monitoring

```
DeviceManager.getDeviceManager().getFirstDroneDevice()?.getWayPointMissionManage
r().addWaypointMissionExecuteStateListener(
listener: CommonCallbacks.KeyListener<MissionWaypointStatusReportNtfyBean>)
```

10、Cancel Mission Status Monitoring

```
DeviceManager.getDeviceManager().getFirstDroneDevice()?.getWayPointMissionManage
r().removeWaypointMissionExecuteStateListener(
listener: CommonCallbacks.KeyListener<MissionWaypointStatusReportNtfyBean>)
```

11、Query Breakpoint Mission Information

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