使用通用 MCU 实现无人机飞行任务的快速二次开发

---TIDronePilot 外部控制 offboard 模式介绍

无名小哥 2024 年 1 月 1 日

1. 传统飞控二次开发方法和主要存在的问题简介

通过对前面几讲中《零基础竞赛无人机积木式编程指南》系列开发教程的学习可知，在以往 TI 电赛真题的学习训练方案中飞行任务代码开发主要集中在 Substask_Demo.c 和 Developer_Mode.c 两个程序文件，其中在 Substask_Demo.c 内负责对具体飞行任务中每个阶段的无人机的飞行动作、航点位置、目标追踪、巡航速度、目标姿态、执行机构驱动(如蜂鸣器、激光笔、舵机、电机)等进行流程化的设计，Developer_Mode.c 内用于实现自动起飞、降落、不同的飞行任务切换与执行。

尽管飞控代码内的相关的 API 函数接口已经实现得相当完备，但上述开发过程的对于初次接触整个飞控代码这一系统工程的学习者来讲，想要直接上手去二次开发仍然还有一定的难度，特别是在对飞控系统代码框架（硬件驱动层、传感器驱动、传感器滤波、姿态解算、惯性导航、机器视觉、基本飞行控制实现、API 函数、导航控制、SDK 自主任务等）没有整体的把握的情况下。

仅有 C 语言+单片机常用基础芯片资源使用知识的初学者按照提供的真题案例和二次开发教程“依葫芦画瓢”去实现特定任务往往容易会出现一些常识性错误，有些错误会直接导致整个飞控系统的“崩盘”。下面以实现无人机在前进的过程中搜寻到色块后，对色块进行跟踪这一任务为例子，将新手易错的问题整理如下:

① 不理解单片机通过定时器实现控制器需要周期性执行这一基本要求的重要性，如下图中在 5ms 周期性任务中引入了延时函数的飞行任务中，开发者 OS 是希望延时函数上方的速度控制执行 10S，认为加了延时之后会速度控制函数就会一直执行。实则是程序会运行一次速度控制函数后，程序会在此处空转等待，后续的高度控制、姿态控制、PWM 输出等得不到及时的执行，只有在 10S 等待空窗期后才会继续执行后续的控制，在 10S 等待时间内，飞控不会对自身的位置、速度、姿态进行即时高频（200Hz 相对 0.1Hz 来讲）的调整与修正，无人机处于间歇性“失控”状态，飞行器的控制周期从 5ms 变成了 10S，用此段代码去控制你的无人机，惨烈的炸机也就变得不可避免。上述问题的根源是延时函数破坏了控制器的周期性执行，连最基本的姿态控制都得不到保障，动摇了飞控系统的“国本”。

```
case 1://前进的同时判断色块是否识别
{
  OpticalFlow_Pos_Ctrl_Output.x=0; //x方向速度给0
  OpticalFlow_Pos_Ctrl_Output.y=10;//y方向速度给10cm/s
  OpticalFlow_Vel_Control(OpticalFlow_Pos_Ctrl_Output);//速度控制
  delay_ms(10000);//约束本任务执行时间为10S
  if(Opv_Top_View_Target.trust_flag==1)//如果openmv视野内识别到了色块
  {
    *mode+=1;//自加后，跳出本前进任务,继续执行色块对准任务
  }
  Flight.yaw_ctrl_mode=ROTATE;
  Flight.yaw_outer_control_output  =RC_Data.rc_rpyt[RC_YAW];
  Flight_Alt_Hold_Control(ALTHOLD_MANUAL_CTRL,NUL,NUL);//高度控制
}
break;
```

错误范例

② 对飞控代码的执行流程与代码产生实际作用理解错乱，不会对任务按流程进行拆分，将部分重复作用的代码“杂乱堆砌”在一起，造成逻辑混乱，实际作用不明。如在下方代码中色块对准函数内部实现有光流速度控制函数，当视觉模块识别到了色块时，case 1 内部的速度控制函数和色块控制内部的速度控制函数会顺序执行，即同一个控制周期 5ms 内，速度控制函数执行了两遍，第一次运行的控制器输出会被后续第二次运行的结果覆盖掉，似乎第一次可以视为“无用”代码，看似不影响最终的控制效果。但是事实是我们需要考虑到 PID 控制器的运算过程，同一个控制器一个周期内执行两遍，**相当于积分 I 做了两次运算，微分项不起任何作用。**因为两次计算过程中，当反馈和期望不变的情况下，第一次的运算过程的偏差和第二次运算的偏差相等，并且第一次的偏差赋值给了上次的偏差，即微分项会恒等于 0，即不管微分项参数 D 为多少，最终计算出来的微分项结果恒等于 0。在 PID 三个参数都存在的情况下，某个控制器**重复执行亦会导致灾难性的 BUG**。

```
69    case 1://前进的同时判断色块是否识别
70    {
71      OpticalFlow_Pos_Ctrl_Output.x=0; //x方向速度给0
72      OpticalFlow_Pos_Ctrl_Output.y=10;//y方向速度给10cm/s
73      OpticalFlow_Vel_Control(OpticalFlow_Pos_Ctrl_Output);//速度控制
74      if(Opv_Top_View_Target.trust_flag==1)//如果openmv视野内识别到了色块
75      {
76        Color_Block_Control_Pilot();//俯视OPENMV视觉水平追踪
77      }
78      Flight.yaw_ctrl_mode=ROTATE;
79      Flight.yaw_outer_control_output  =RC_Data.rc_rpyt[RC_YAW];
80      Flight_Alt_Hold_Control(ALTHOLD_MANUAL_CTRL,NUL,NUL);//高度控制
81    }
82    break;
```

错误范例

```
804       Total_Controller.SDK_Pitch_Position_Control.FeedBack=Opv_Top_View_Target.sdk_target.y;
805       PID_Control_SDK_Err_LPF(&Total_Controller.SDK_Pitch_Position_Control,Opv_Top_View_Target.trust_flag,0.1f);
806
807       OpticalFlow_Pos_Ctrl_Output.x=-Total_Controller.SDK_Roll_Position_Control.Control_OutPut;
808       OpticalFlow_Pos_Ctrl_Output.y=-Total_Controller.SDK_Pitch_Position_Control.Control_OutPut;
809     }
810     else//丢失目标
811     {
812       miss_flag=1;
813     }
814   }
815
816
817   if(miss_flag==1)//目标丢失
818   {
819     if(miss_cnt==1)//初始丢失跟踪目标后，锁定当前位置后，进行普通光流控制
820     {
821       miss_cnt=2;
822       OpticalFlow_Control_Pure(1);//20ms
823     }
824     else if(miss_cnt==2)//丢失跟踪目标后，进行普通光流控制
825     {
826       OpticalFlow_Control_Pure(0);//20ms
827     }
828   }
829   else//目标未丢失,10ms
830   {
831     OpticalFlow_Vel_Control(OpticalFlow_Pos_Ctrl_Output);//速度控制周期20ms
832   }
```

视觉偏差控制器输出赋值给速度期望

色块丢失时

有色块时

③ 认为控制代码只需要执行一次就能达到期望的控制效果，如果从 A 飞到 B，AB 两点的距离为 100cm，试想无人机需要在一个控制周期 5ms 内就能实现 A 到 B，假设无人机做匀加速直线运动，零初始状态下无人机的加速度要到多少才能完成这一目标，无人机的实际推重比是否能达到这个要求？如果按照这个加速度进行加速无人机 1S 后是否会脱离地球轨道？

```
69      case 1://前进的同时判断色块是否识别
70      {
71        OpticalFlow_Pos_Ctrl_Output.x=0; //x方向速度给0
72        OpticalFlow_Pos_Ctrl_Output.y=10;//y方向速度给10cm/s
73        OpticalFlow_Vel_Control(OpticalFlow_Pos_Ctrl_Output);//速度控制
74        Flight.yaw_ctrl_mode=ROTATE;
75        Flight.yaw_outer_control_output  =RC_Data.rc_rpyt[RC_YAW];
76        Flight_Alt_Hold_Control(ALTHOLD_MANUAL_CTRL,NUL,NUL);//高度控制
77        *mode+=1;//自加后，跳出本前进任务，执行色块对准任务
78      }
79      break;
80      case 2://俯视OPENMV视觉追踪色块
81      {
82        Color_Block_Control_Pilot();//俯视OPENMV视觉水平追踪
83        Flight.yaw_ctrl_mode=ROTATE;
84        Flight.yaw_outer_control_output  =RC_Data.rc_rpyt[RC_YAW];
85        Flight_Alt_Hold_Control(ALTHOLD_MANUAL_CTRL,NUL,NUL);//高度控制
86      }
87      break;
```

错误范例

case 1只执行一次就跳到case 2

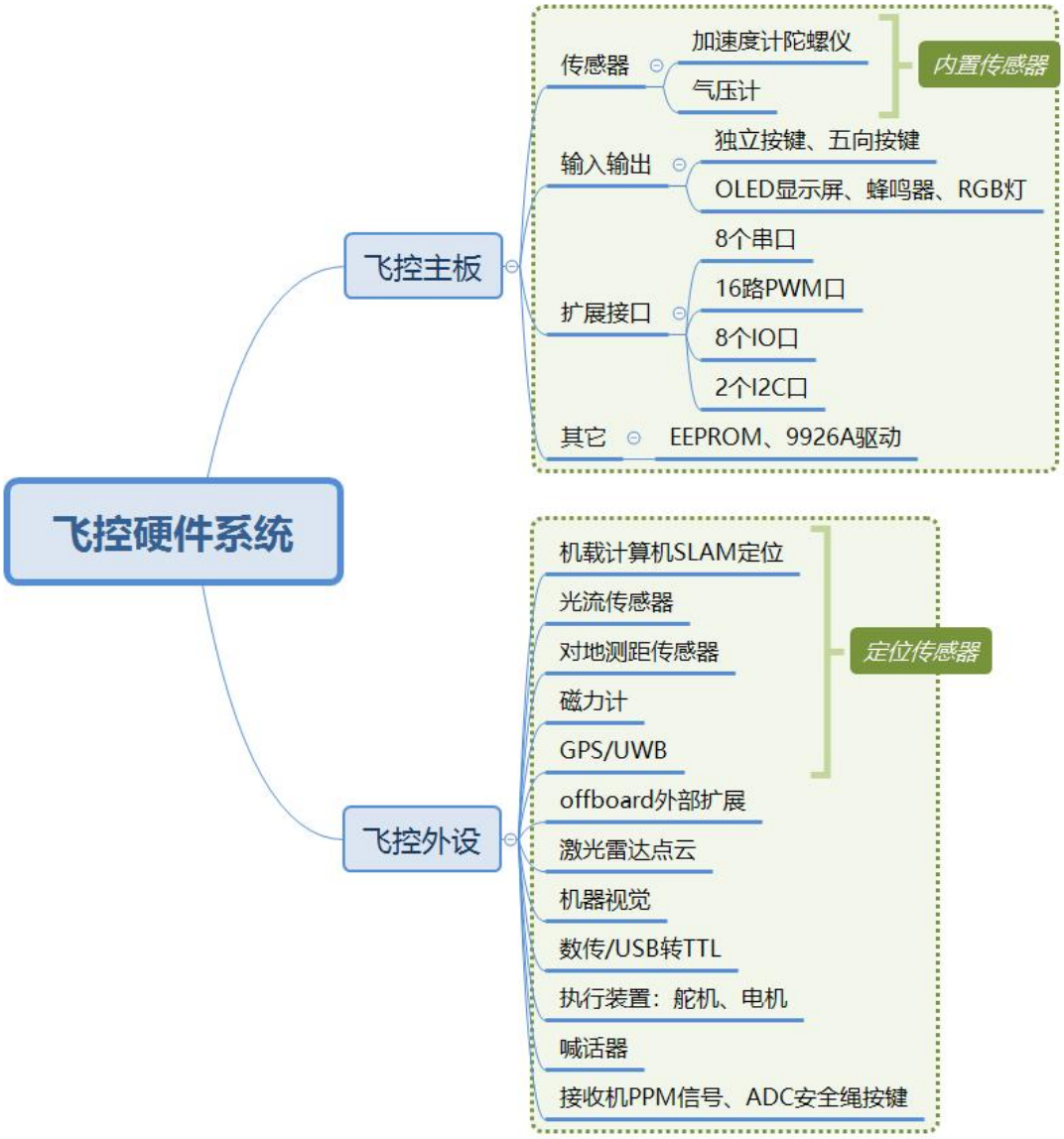
④ 另外由于飞控自身硬件资源有限，飞控自身外界的外设占用比较多，可供用户它用的串口、PWM、IO 资源不再富裕，使用起来使用捉襟见肘，常常需要外部 MCU 进行扩展，既然都引入了外部 MCU 到无人机平台中了，很自然的会想到用自己熟悉的 MCU 飞无人机飞行任务进行开发。

综上所述，针对新手初学者来讲，面对略显庞杂的飞控系统代码，在进行二次开发时，若写出某些天坑级的 BUG 会导致无人机系统的整体崩溃，基本的姿态自稳都得不到保障，就会出现灾难级的炸机事故。因此为了使得二次开发更加简单、开发更加安全，有必要将和用户二次开发相关的 API 函数、导航控制、

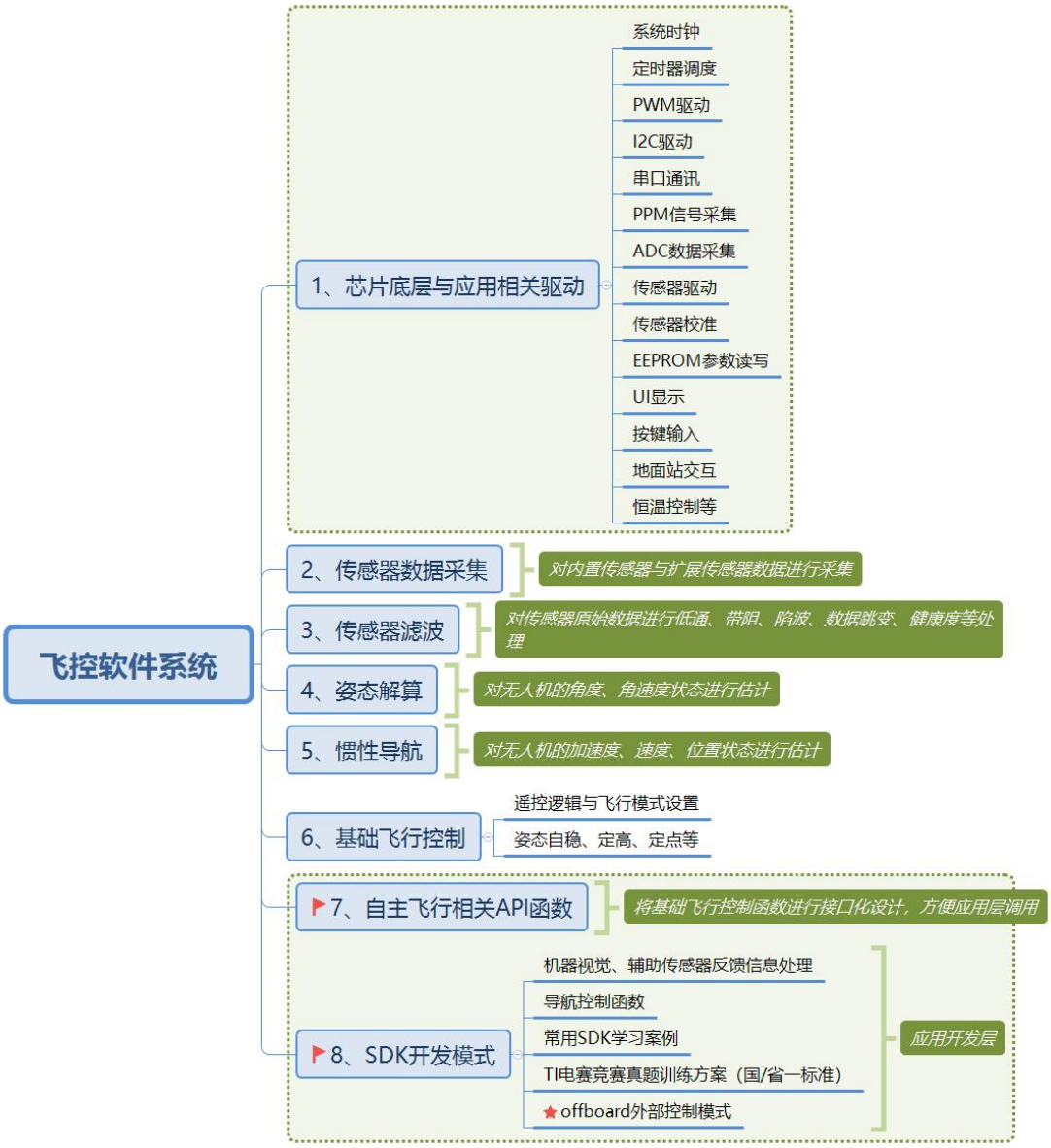
SDK 自主任务控制部分代码的实现，单独用一个通用控制器去实现，我们只需要在飞控端将已有的 API 函数接口进行简单整合，在 SDK 模式中新增加外部控制 offboard 模式就可以，这部分工作量很少，很多都是现成的我们在后面的教程中进行介绍。

2. TIVA 飞控的硬件组成与系统框架

TIVA 飞控硬件系统包括飞控主板与外设两部分组成，其中飞控主板上板载加速度计陀螺仪（带温控电路）、气压计传感器，带独立按键和五向按键，配备 OLED 显示屏进行人机交互，带蜂鸣器和 RGB 灯提示，板载的扩展接口包括 8 路串口、16 路 PWM、8 个预留 IO 口、2 组 I2C 口等。扩展接口用于外接机载端 SLAM 定位、offboard 外部控制、光流、对地激光测距、GPS、机器视觉（如 OPENMV/K210/树莓派 OPENCV）、激光雷达点云数据等。



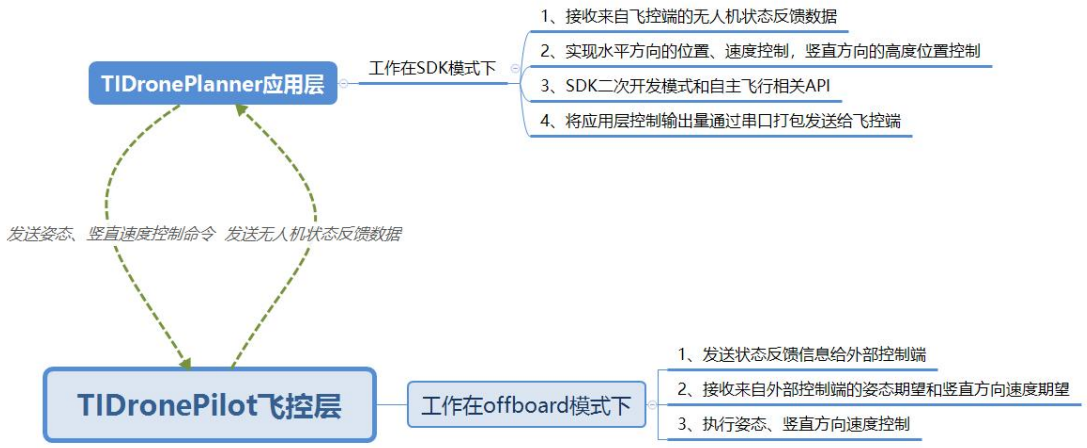
TIVA 飞控软件系统包括芯片底层与应用相关驱动、传感器数据采集、传感器滤波、姿态解算、惯性导航、基础飞行控制、自主飞行 API 函数、SDK 开发者模式等。



3. 运用通用单片机开发板实现对飞控外部控制

在传统的开发模式中主要是在飞控代码内部对 SDK 开发模式中编写飞行任务代码去实现特定任务，这些开发流程在之前按的教程中有详细的阐明。本教程需要解决的问题是如何将飞控软件框架中的自主飞行相关 API 函数和 SDK 开发者模式的应用层开发代码放在外部的通用的 MCU 中去实现，比如盘古 TI MCU 系统板(已完成)、STM32F103 系列核心板(已完成)、STC32G12K128 开发板(完成度 90%)等。由于外部通用 MCU 中需要实现的代码量很少，对单片机处理资源

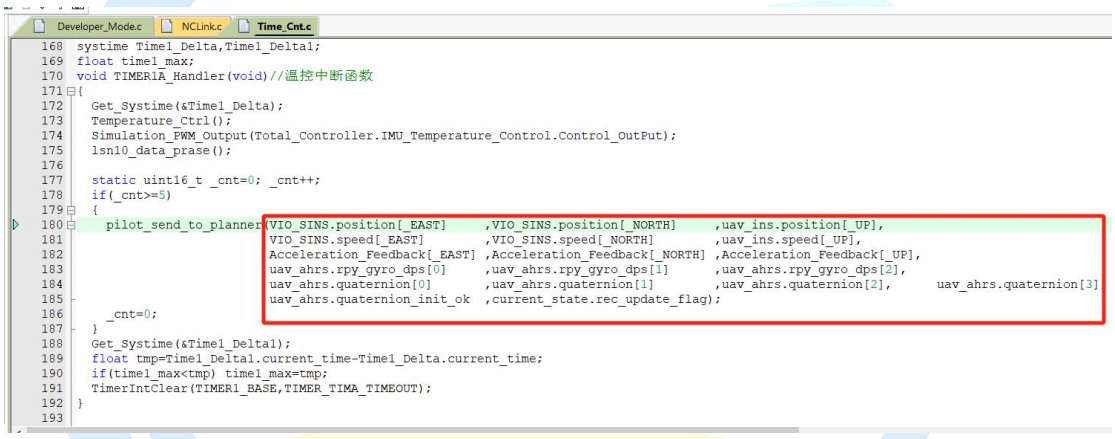
和性能要求并不高，因此用户完全可以用任何一款自己熟悉的单片机，高效率得心应手的去开发，参照上述已实现的三款核心板方案，去自己实现飞控的外部控制这部分设计。为了方便后面表述，我们将带有 offboard 外部控制的飞控称为 TIDronePilot(简称 TPT/下位机/飞控端)，将外部通用 MCU 开发板称为 TIDronePlanner(简称 TPR/上位机/应用端)。



■ TIDronePilot 中 offboard 外部控制相关函数的实现

TPT 的实现工作量很小，就是在保留原来 TIVA 飞控所有功能的前提（仍然可以用传统方案开发）下，主要新增加了一下三点：

① TPT 发送飞控内部估计出的三维位置、速度、加速度、角速度、姿态四元数、融合标志等信息给外部控制端，TPR 端解析到上述反馈数据后用于对无人机实现位置、速度等控制，并将最终的控制量串口打包发送给飞控端。



② TPT 解析来自 TPR 端的姿态和竖直方向速度控制信息，作为 offboard 外部控制模式下对应项的期望值。

```
Developer_Mode.c  NCLink.c
2024  else if(state==6&&data==NCLink_End[0])//帧尾0
2025  {
2026      state = 7;
2027      cmd_buf[4+data_cnt++]=data;
2028  }
2029  else if(state==7&&data==NCLink_End[1])//帧尾1
2030  {
2031    state = 0;
2032    cmd_buf[4+data_cnt]=data;
2033    uint16_t _cnt=data_cnt+5;
2034    uint8_t sum = 0;
2035    for(uint8_t i=0;i<(_cnt-3);i++)  sum ^= *(cmd_buf+i);
2036    if(!(sum==*(cmd_buf+_cnt-3)))                                    return;//判断sum
2037    if(!(*(cmd_buf)==NCLink_Head[0]&&*(cmd_buf+1)==NCLink_Head[1]))  return;//判断帧头
2038    if(!(*(cmd_buf+_cnt-2)==NCLink_End[0]&&*(cmd_buf+_cnt-1)==NCLink_End[1])) return;//帧尾校验
2039    if(*(cmd_buf+2)==NCLINK_SEND_AHRS)//数据帧类型判断
2040    {
2041      planner_cmd._yaw_ctrl_mode=*(cmd_buf+4);
2042      planner_cmd._yaw_ctrl_start=*(cmd_buf+5);
2043
2044      Byte2Float(cmd_buf, 6, &planner_cmd._roll_outer_control_output);
2045      Byte2Float(cmd_buf, 10,&planner_cmd._pitch_outer_control_output);
2046      Byte2Float(cmd_buf, 14,&planner_cmd._yaw_outer_control_output);
2047
2048      Byte2Float(cmd_buf, 18,&planner_cmd._nav_target_vel.x);
2049      Byte2Float(cmd_buf, 22,&planner_cmd._nav_target_vel.y);
2050      Byte2Float(cmd_buf, 26,&planner_cmd._nav_target_vel.z);
2051
2052      planner_cmd._execution_time_ms=(int32_t)(*(cmd_buf+30)<<24 | *(cmd_buf+31)<<16 | *(cmd_buf+32)<<8 | *(cmd_buf+33));
2053    }
2054    Get_Systime(&navcmd_t[1]);
2055    navcmd_dt=navcmd_t[1].current_time-navcmd_t[0].current_time;
2056  }
2057  else state = 0;
2058 }
2059
2060
2061
```

③ 在 SDK 模式中额外新增加了 offboard 模式，在此模式下飞控端的姿态期望、竖直方向速度期望来源于外部控制器的给定（TPT 串口解析来之 TPR 的控制指令），同时也可以通过遥控器手动介入接管无人机的控制权限，可以实现不切换遥控器开关挡位的情况下对无人机的直接控制。需要注意的是飞控端切入 SDK 后 offboard_start_flag 会置 1，飞控端会将此标志为发送给外部控制端用于触发外部控制端执行其内部的 SDK 任务。

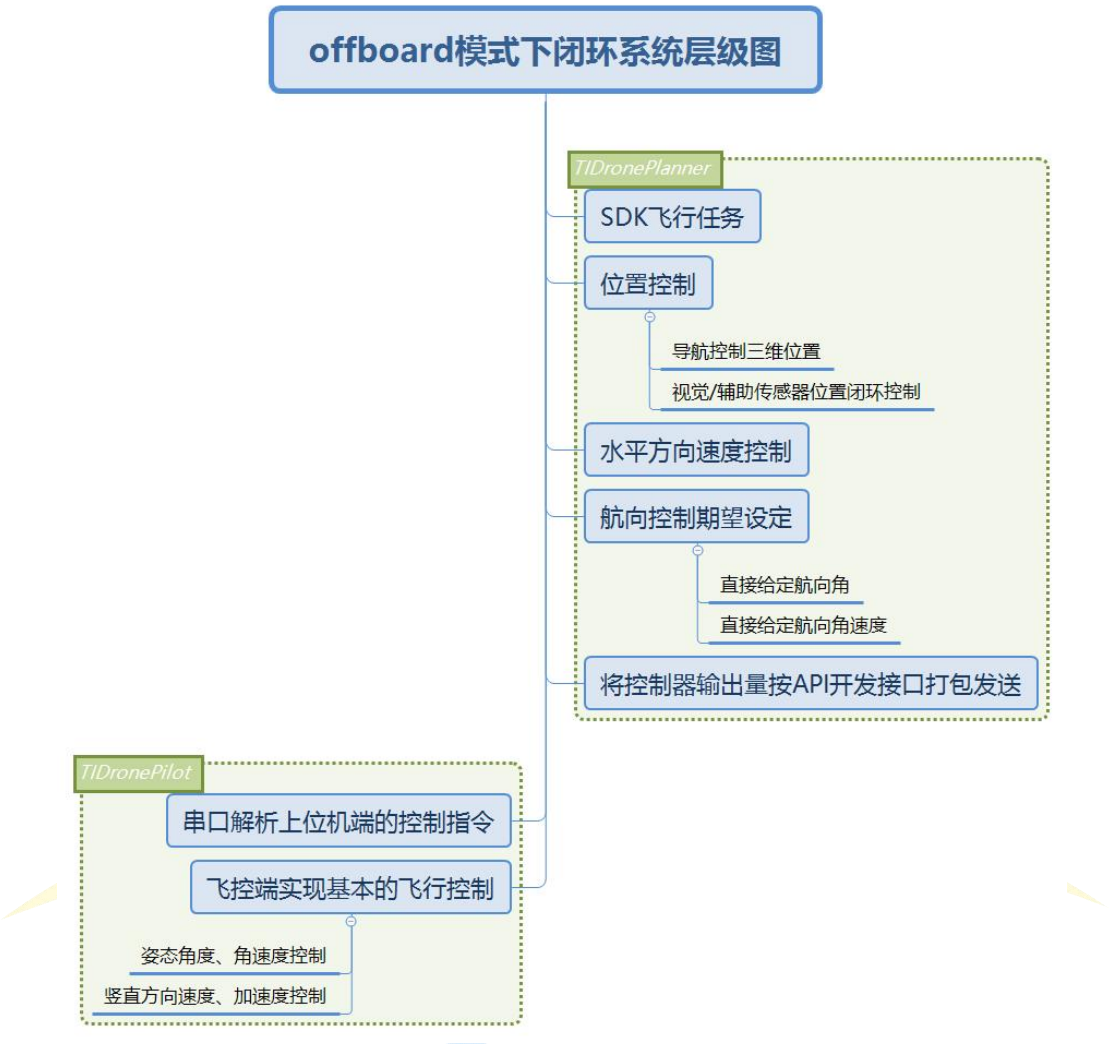
```
Developer_Mode.c
232   case 24://offboard外部控制模式
233   {
234     offboard_start_flag=1;//offboard启动标志位
235     //不加此行代码，当后续全程无油门上下动作后，飞机最后自动降落到地面不会自动上锁
236     Unwanted_Lock_Flag=0;//允许飞机自动上锁，原理和手动推油起飞类似
237     if(Roll_Control==0&&Pitch_Control==0&&RC_Data.rc_rpyt[RC_THR]==0)//遥控量无任何给定
238     {                                    切入sdk后，飞控的姿态、竖直方向的速度期望来源于外部控制器给定
239       //偏航角度/角速度控制期望来源于Planner端
240       Flight.yaw_ctrl_start=planner_cmd._yaw_ctrl_start;
241       Flight.yaw_ctrl_mode=planner_cmd._yaw_ctrl_mode;
242       Flight.execution_time_ms=planner_cmd._execution_time_ms;
243       Flight.yaw_outer_control_output  =planner_cmd._yaw_outer_control_output;
244       //俯仰、横滚角度控制期望来源于Planner端
245       Flight.roll_outer_control_output =planner_cmd._roll_outer_control_output;
246       Flight.pitch_outer_control_output=planner_cmd._pitch_outer_control_output;
247       //高度位置/速度控制期望来源于Planner端
248       Flight_Alt_Hold_Control(ALTHOLD_AUTO_VEL_CTRL,NUL,planner_cmd._nav_target_vel.z);//高度控制
249
250       //复位Pilot端的位置、速度控制
251       OpticalFlow_Ctrl_Reset();//目的是为切出SDK模式后,Pilot直接接管位置/速度控制做准备
252     }
253     else//水平方向遥杆有动作后恢复到定高+姿态自稳定模式
254     {
255       Flight.roll_outer_control_output =RC_Data.rc_rpyt[RC_ROLL];
256       Flight.pitch_outer_control_output=RC_Data.rc_rpyt[RC_PITCH];
257       Flight.yaw_ctrl_mode=ROTATE;                  当遥控介入时控制权限给到遥控器，用于手动控制
258       Flight.yaw_outer_control_output  =RC_Data.rc_rpyt[RC_YAW];
259       Flight_Alt_Hold_Control(ALTHOLD_MANUAL_CTRL,NUL,NUL);//高度控制
260     }
261   }
262   break;
```

至此用于实现 offboard 外部控制飞控端的主要工作就全部实现了，单从控制部分抽象来看 offboard 模式下飞控端**“退化”**成只执行基本的飞行控制部分，主要包括姿态自稳、部分定高(竖直速度、加速度)功能，飞控内部并没有运行对无人机的位置、速度进行直接的控制的代码，相当于飞控在 offboard 外部控制模式下变成了一个“指哪打哪”、“一切行动听指挥”严格响应外部控制端 TPR 命令的被控对象。飞控负责保持自身姿态和竖直方向速度的稳定，其它应用层开发

全部放在外部控制端 TPR 端去实现，新的开发模式下用户可以不用在飞控 TPT 端修改代码，把无人机作为一个整体被控对象去开发，自然也就不会出现第一节中天坑级 BUG 导致灾难性炸机的情况，在 TPR 端开发应用层代码不会影响飞控 TPT 端内部的控制，任何情况下遥控器都可以直接对无人机的控制进行接管。

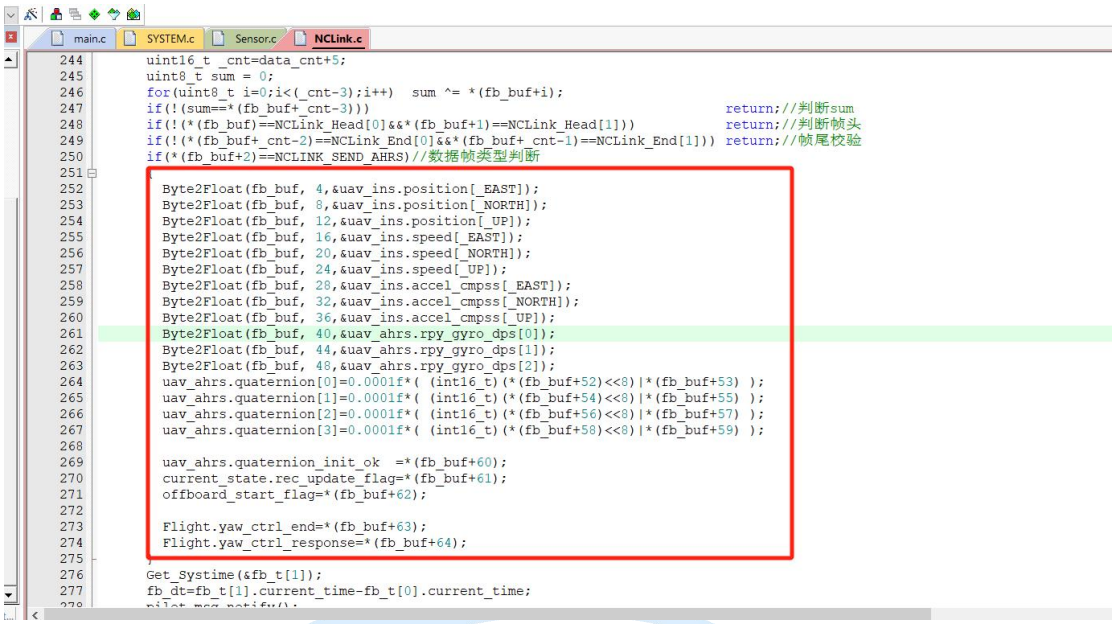
■ TIDronePlanner 中应用开发层相关函数的实现

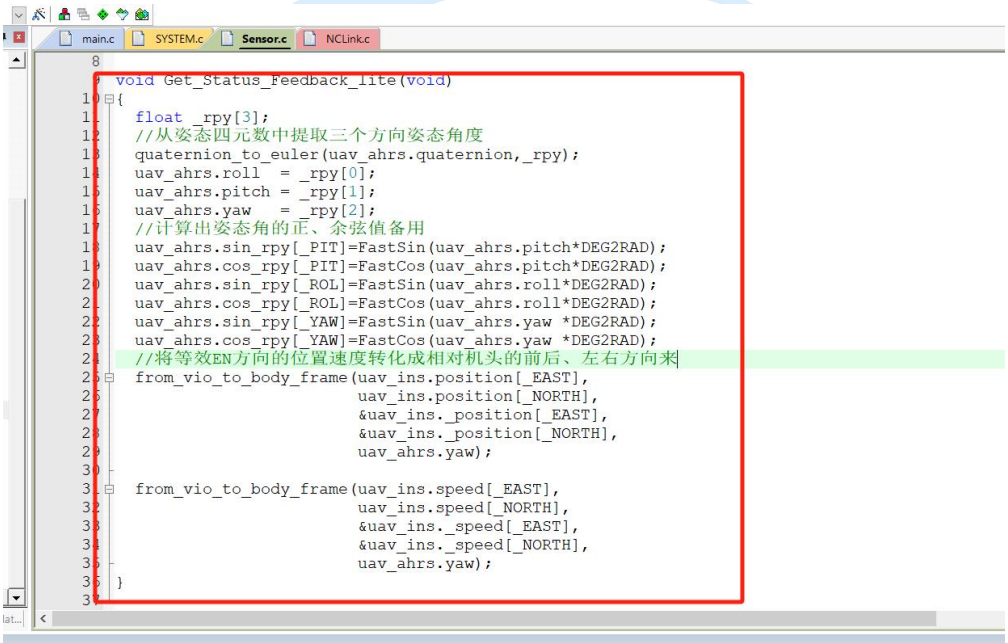
TIDronePlanner 作为飞控 offboard 外部控制模式下的上层控制器，其主要作用是实现原飞控中用于自主飞行相关 API 函数、SDK 开发模式的功能，由于 TPR 应用端内部需要实现无人机水平方向的位置、速度控制，竖直方向的高度位置控制，自然需要飞控端 TPT 给应用端 TPR 发送自身的位置、速度、姿态等无人机飞行状态的反馈信息，TPR 接收到反馈信息后用于无人机的位置、速度闭环控制，最终将 TPR 控制器的输出封装成相关 API 接口变量，通过串口通讯打包发给 TPT 飞控端，从而实现了整个飞控位置、速度、加速度、姿态角度、姿态角速度的系统的完整闭环控制。



在通用 MCU 系统板中实现 TIDronePlanner 的工作量同样很少，除了对应单片机资源如串口通讯、PWM 输出、按键、显示屏、定时器调度驱动实现外，剩下的工作只剩下解析飞控端发送过来的状态反馈信息、移植原飞控端位置控制、速度控制、SDK 开发模式、API 接口函数、串口打包发送 API 接口变量。

① 解析来自飞控端的状态反馈数据，按照飞控发送时对应数据协议帧来解析。



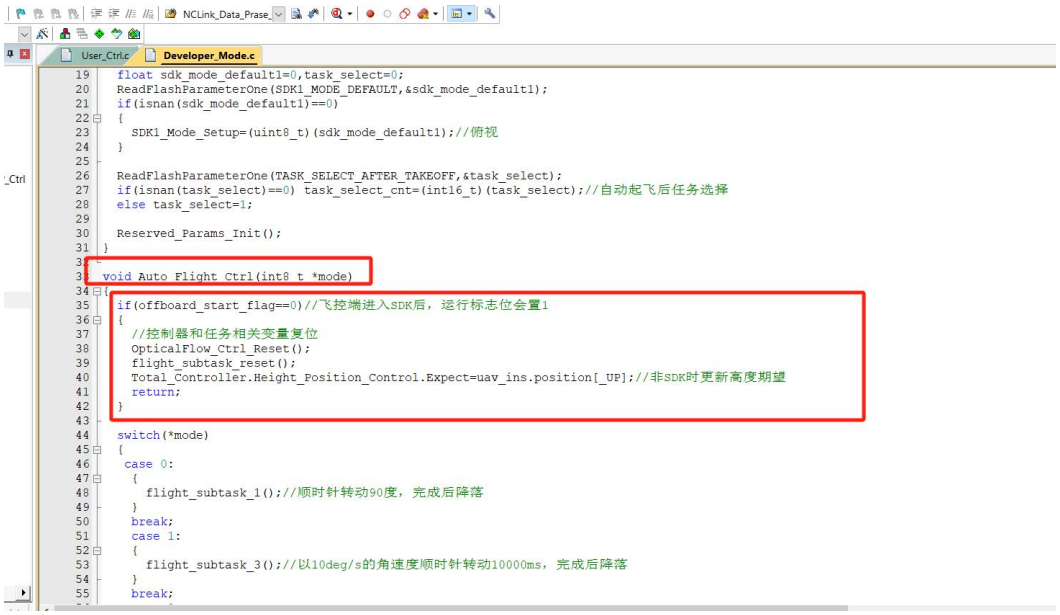


② 实现位置控制、速度控制、SDK 开发模式、API 接口函数，由于这部分在飞控内部均有实现，直接移植过来就可以，此处不作过多介绍，在之前的教程中有对相关函数实现进行讲解。

```c
/*************************************************************
@功能描述：系统调度定时器中断服务函数：主要进行遥控器解析、
传感数据采集、数字滤波、姿态解算、惯性导航、控制等对周期有
严格要求的函数
@作者：无名小哥
@日期：2019年01月27日
*************************************************************/
void TIMER0A_Handler(void)          //系统调度中断函数    4.4ms
{
  Get_Systime(&Time0_Delta);
  Get_Status_Feedback_lite();       //状态反馈信息获取
  Auto_Flight_Ctrl(&SDK1_Mode_Setup);//SDK开发者模式
  planner_send_to_pilot();          //串口数据打包发给飞控
  Read_Button_State_All();          //按键状态读取
  Bling_Working(Bling_Mode);        //状态指示灯运行
  Laser_Light_Work(&laser_light_1); //激光笔/RGB灯/蜂鸣器1驱动
  Laser_Light_Work(&laser_light_2); //激光笔/RGB灯/蜂鸣器2驱动
  Laser_Light_Work(&buzzer);        //电源板蜂鸣器驱动
  Get_Systime(&Time0_Delta1);
  float tmp=Time0_Delta1.current_time-Time0_Delta.current_time;
  if(time0_max<tmp) time0_max=tmp;
  TimerIntClear(TIMER0_BASE,TIMER_TIMA_TIMEOUT);
}
```

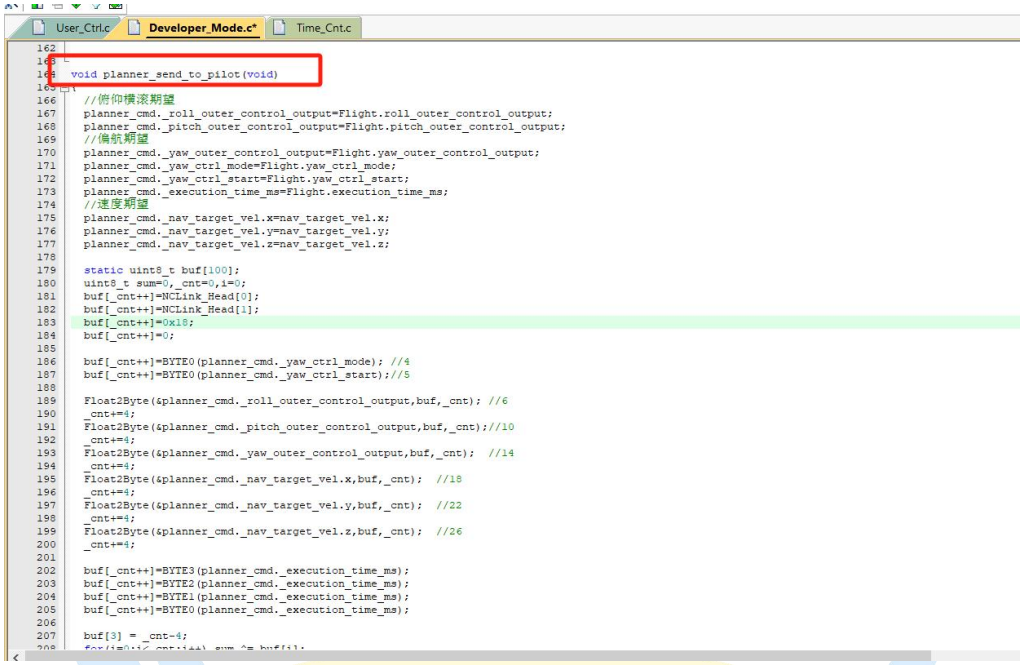
```c
        OpticalFlow_Pos_Ctrl_Expect.x=uav_ins.position[_EAST];
        OpticalFlow_Pos_Ctrl_Expect.y=uav_ins.position[_NORTH];
        Total_Controller.Height_Position_Control.Expect=uav_ins.position[_UP];
      }
    }
  }
}

void Horizontal_Control_VIO(uint8_t force_brake_flag)
{
  /*************************光流位置控制器**********************************/
  if(OpticalFlow_Pos_Ctrl_Recode==1||force_brake_flag==1)
  {
    OpticalFlow_Pos_Ctrl_Expect.x=uav_ins.position[_EAST];
    OpticalFlow_Pos_Ctrl_Expect.y=uav_ins.position[_NORTH];
    OpticalFlow_Pos_Ctrl_Recode=0;
  }
  OpticalFlow_Pos_Control_VIO();
  /**********当只需要速度控制时，开启以下注释，仅限调试时用*************/
//  OpticalFlow_Pos_Ctrl_Output.x=0;
//  OpticalFlow_Pos_Ctrl_Output.y=0;
  /*************************基于模型的加速度-姿态角映射，相比直接给姿态，参数差异大概在20倍左右**********************/
  OpticalFlow_Vel_Control(OpticalFlow_Pos_Ctrl_Output);//速度期望
}


void OpticalFlow_X_Vel_Control(float target_x)//机头左侧为X+
{
  static uint16_t OpticalFlow_Vel_Ctrl_Cnt=0;
  OpticalFlow_Vel_Ctrl_Expect.x= target_x;//速度期望
  OpticalFlow_Vel_Ctrl_Cnt++;
  if(OpticalFlow_Vel_Ctrl_Cnt>=2)//10ms控制一次速度，避免输入频率过大，系统响应不过来
```

```c
//适用于激光雷达SLAM定位条件下，普通光流（LC307、LC302）定位条件下无效
void Horizontal_Navigation(float x,float y,float z,uint8_t nav_mode,uint8_t frame_id)
{
  if(nav_mode==RELATIVE_MODE)//相对模式
  {
    switch(frame_id)
    {
      case BODY_FRAME://机体坐标系下
      {
        float map_x=0,map_y=0;
        from_body_to_nav_frame(x,y,&map_x,&map_y,uav_ahrs.yaw);
        OpticalFlow_Pos_Ctrl_Expect.x=uav_ins.position[_EAST]+map_x;
        OpticalFlow_Pos_Ctrl_Expect.y=uav_ins.position[_NORTH]+map_y;
        Total_Controller.Height_Position_Control.Expect=uav_ins.position[_UP]+z;
      }
      break;
      case MAP_FRAME://导航坐标系下
      {
        OpticalFlow_Pos_Ctrl_Expect.x=uav_ins.position[_EAST]+x;
        OpticalFlow_Pos_Ctrl_Expect.y=uav_ins.position[_NORTH]+y;
        Total_Controller.Height_Position_Control.Expect=uav_ins.position[_UP]+z;
      }
      break;
    }
  }
  else if(nav_mode==GLOBAL_MODE)//全局模式
  {
    switch(frame_id)
    {
      case MAP_FRAME://导航坐标系下
      {
        OpticalFlow_Pos_Ctrl_Expect.x=x;
        OpticalFlow_Pos_Ctrl_Expect.y=y;
        Total_Controller.Height_Position_Control.Expect=z;
```



这里需要注意的是虽然 Auto_Flight_Ctrl 函数在定时器内周期性执行，但 SDK 任务执行与否取决于 offboard_start_flag 变量的值，当用遥控器/ADC 按键操作飞控端切入 offboard 模式后，此标志位会被置 1，Auto_Flight_Ctrl 中的自主飞行任务才会执行。

③ 串口打包发送 API 接口变量



TIDronePlanner 中实现部分还包括外设如舵机/电机、激光笔、蜂鸣器等设备的驱动，辅助传感器如激光雷达点云、机器视觉等数据的接入，具体根据实际飞行任务要求来，由于 TIDronePlanner 的实现仅需要占用一个串口资源就可以实现其应用层，因此通用 MCU 核心板的其它预留出来的资源就可以供二次开发使用，此举能有效解决传统开发方案中，资源不够用的情况。

4. 基于 TIDronePilot 和 TIDronePlanner 分层式开发 Q&A

■ 上位机和下位机通过串口通讯的方式进行数据交互，数据的实时性如何，如何保证二者通讯时数据的可靠性？

✓ 这里我们看下用于 offboard 外部控制的通讯串口波特率为 2250000bps，传输带个字节数据包括 1 个起始位、1 个停止位、8 个数据位共占用 10 个二进制位，因此传输单个字节时间开销为 1/225000 秒，即约为 4.4us，可知传输 100 个字节时，时间开销为约为 0.44ms，实际当前传输带宽为 68 个字节，约为 0.3ms。

✓ 串口通讯时为了确保串口不丢帧，需要考虑优先级高于当前串口中断的任务的最大执行时间+串口中断函数执行时间，务必小于串口接收单个字节所需的时间，才能确保不丢帧。

1、多个串口通讯时，串口通讯波特率可以降低一点。

2、合计设计优先级，当存在不同波特率通讯时，通讯波特率高的串口中断优先级要高于波特率低的。

3、存在优先级高于串口中断的其它中断任务时，其它中断任务的总的最大时间开销也要考虑。

■ 在上位机端开发应用层代码的情况下，是不是可以完全不用管飞控端、机载计算端的代码实现？

✓ 新增 offboard 外部控制模式的主要目的是简化开发过程，让新手能更容易上手在飞控应用层做二次开发，无需管飞控整个飞控系统代码。因此用户可以完全不应管飞控各系统模块的内部具体的实现过程，新手用户把飞控当作“盲盒”也能实现二次开发，就像之前的教程中把机载端当作一个能提供室内高精度位姿数据的“GPS 传感器”一样，只需要知道如何安装、接线、设置、操控就可以。

■ 文档中是以室内激光雷达 SLAM 定位为例的，如果我需要在室外实现相关应用层的二次开发，还能按照上述方式来开发吗？

✓ 能，只需要修改飞控端发送的位置、速度数据融合来源就可以，比如在室外 GPS 定位下，飞控在给外部应用层发反馈状态数据时，将内部 GPS 融合得到的位置、速度数据替换掉原来 SLAM 定位融合的数据，原来外部控制器的开发过程不需要做任何调整，同理 UWB 定位模式下也也是这个处理思路。

- 外部通用 MCU 系统板能用树莓派等机载端去实现吗，用机载端去控制无人机有什么优势？

✓ 对于通用 MCU 的要求比较小，用树莓派 ROS 端去做开发当然可以，使用机载端去控制飞控端对于有 ROS 基础的同学更为直接，同样是将相关驱动和应用层、数据交互等移植到机载端去跑就可以实现。

✓ 在熟悉平台的情况下用 ROS 端去开发，能做更多深入工作，比如利用 ROS 开发环境下成熟的资源包去实现自主导航/避障、路径规划、多机协同等复杂任务。

✓ 是否使用机载计算机去控制飞控会比用单片机去控制飞控更为高级？单从电赛这些年的要求来看，**用单片机完全足以胜任飞行任务的设计**，没有必要形成比设备成本、拼学校/家庭经济实力、乱搞“军备竞赛”、极限碾压式内卷的“不正之风”，这样的鄙视链可以休矣，风清气正的竞赛环境要靠大家来共同来营造，不管用什么平台只要是能高效率的解决实际问题就行。

✓ 同时对于某些主频偏低的单片机平台，其串口最大波特率到不了 2250000 bps，可以适当调小通讯波特率，但建议不要低于 921600 bps。

TIDronePilot 和 TIDronePlanner 配合使用时的具体操作参见配套的视频教程...