# 中国机器人大赛自动分拣－自动分拣 编程报告

**本次报告主要分为6大部分**

1. [*调试组件*](#调试)
   * [优点](#优点)
   * [源码](#相关源码)
   * [效果](#效果图片)
2. [*底层电机驱动*](#电机)
   * [采用的新方案](#电机的区别)
   * [问题及解决方案](#方案及测试)
   * [运行效果](#编码器效果展示)
   * [优化方向](#编码器优化方向)
3. [*陀螺仪*](#陀螺仪)
   * [概括](#陀螺仪信息)
   * [数据获取](#陀螺仪数据获取)
   * [数据应用](#陀螺仪数据应用)
4. [*循迹版部分*](#循迹板部分)
   * [信息](#循迹板概述)
   * [拓展板](#循迹拓展板)
   * [主控板](#循迹主控板)
5. [*上层机械臂*](#机械臂)
6. [*系统整体*](#整体框架)
   * [运行流程图](#整体流程图)
   * [FreeRTOS](#RTOS)
   * [对外通讯协议](#模块通讯协议)

### [联系作者](#联系方式)

## 调试组件

#### 提升

今年的新一代程序在原有的基础上加入了USMART调试组件, 其在诸如PID调参时起到了重大的作用，相比于以往的重复烧录的繁琐以及对内存寿命的影响，提升了工作效率。在平时进行各项测试时更加灵活多变。

#### 串口接收函数（基于正点原子通讯协议进行修改）

因为部分移动端串口调试设备不支持自动末尾追加0X0D 0X0A换行符，所以增加了以!为结尾信号的通讯模式。   
**根据运行情况的需要，通过修改宏定义来实现不同种类设备的支持！**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* @Name USART1\_IRQHandler  
 \* @功能说明 usrat1 handler  
 \* @param None  
 \* @返回值  
 \* @author peach99CPP  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
void U1\_IRQHandler(void)  
{  
 uint8\_t rec;  
 if (\_\_HAL\_UART\_GET\_FLAG(&huart1, UART\_FLAG\_RXNE))  
 {  
 \_\_HAL\_UART\_ENABLE\_IT(&huart1, UART\_IT\_RXNE);  
 rec = huart1.Instance->RDR;  
 if(!(USART\_RX\_STA & 0x8000))//接收未完成  
 {  
#ifdef USE\_BLE  
 if(rec == 0x0a || rec == 0x21)//兼容非电脑设备的串口发送 以！为结尾  
  
 {  
 USART\_RX\_STA |= 0x8000;  
 return ;  
 }  
 if( rec == 0x0d ) return ;  
 else  
 {  
 USART\_RX\_BUF[USART\_RX\_STA & 0x3fff] = rec;  
 USART\_RX\_STA++;  
 if(USART\_RX\_STA & USART\_RX\_STA & 0x3fff == MAX\_SIZE) USART\_RX\_STA = 0;  
 }  
#endif  
#ifdef USE\_ATK  
 if(rec == 0x0d)  
 {  
 USART\_RX\_STA |= 0x4000;  
 return;  
 }  
 if( USART\_RX\_STA & 0x4000 && rec == 0x0a)  
 {  
 USART\_RX\_STA |= 0x8000;  
 return;  
 }  
 else  
 {  
 USART\_RX\_BUF[USART\_RX\_STA & 0x3fff] = rec;  
 USART\_RX\_STA++;  
 if(USART\_RX\_STA & USART\_RX\_STA & 0x3fff == MAX\_SIZE) USART\_RX\_STA = 0;  
 }  
  
#endif  
 }  
 }  
}

#### 串口发送函数

基于队列进行管理，需要对printf函数进行重定向，将要打印的字符放入队列。

//重定义fputc函数,使用队列来进行printf,此时要另外开一个task循环刷  
int fputc(int ch, FILE \*f)  
{  
 if(tx\_queue != NULL)  
 {  
 if(vPortGetIPSR())  
 {  
 BaseType\_t if\_higher\_woken = pdFALSE;  
 while( xQueueSendFromISR(tx\_queue, &ch, &if\_higher\_woken) != pdTRUE); //阻塞式。确保此时已经成功把消息放入队列  
 portYIELD\_FROM\_ISR(if\_higher\_woken);//判断是否需要进行任务调度  
 }  
 else  
 {  
 //此时并不是在中断中被调用，可以直接写入数据  
 xQueueSend(tx\_queue, &ch, 1);  
 }  
 }  
 \_\_HAL\_UART\_ENABLE\_IT(&huart1, UART\_IT\_TXE);  
 return ch;  
}  
//串口中断处理函数中的部分  
if(\_\_HAL\_UART\_GET\_FLAG(&huart1, UART\_FLAG\_TXE))  
{  
\_\_HAL\_UART\_CLEAR\_FLAG(&huart1, UART\_FLAG\_TXE);  
BaseType\_t xTaskWokenByReceive = pdFALSE;  
//发送队列中有数据需要发送  
if (xQueueReceiveFromISR(tx\_queue, (void\*)&rec, &xTaskWokenByReceive) == pdPASS)  
huart1.Instance->TDR = rec;  
else  
//无数据发送就关闭发送中断  
\_\_HAL\_UART\_DISABLE\_IT(&huart1, UART\_IT\_TXE);  
}  
//增强健壮性，在中断函数中加入对ORE中断，避免卡死在该中断  
if(\_\_HAL\_UART\_GET\_FLAG(&huart1, UART\_FLAG\_ORE))//处理ORE错误导致卡死在中断里  
{  
uint8\_t tmp;  
tmp = USART1->ISR;  
tmp = USART1->RDR;  
\_\_HAL\_UART\_CLEAR\_FLAG(&huart1, UART\_FLAG\_ORE);  
}

#### ***用户配置USMART的源码***

#include "usmart.h"  
#include "usmart\_str.h"  
////////////////////////////用户配置区///////////////////////////////////////////////  
//这下面要包含所用到的函数所申明的头文件(用户自己添加)  
#include "delay.h"  
#include "motor.h"  
#include "chassis.h"  
#include "imu\_pid.h"  
#include "track\_bar\_receive.h"  
#include "chassis\_control.h"  
#include "openmv.h"  
#include "servo.h"  
#include "atk\_imu.h"  
#include "read\_status.h"  
  
//函数名列表初始化(用户自己添加)  
//用户直接在这里输入要执行的函数名及其查找串  
struct \_m\_usmart\_nametab usmart\_nametab[] =  
{  
#if USMART\_USE\_WRFUNS==1 //如果使能了读写操作  
 (void\*)read\_addr, "u32 read\_addr(u32 addr)",  
 (void\*)write\_addr, "void write\_addr(u32 addr,u32 val)",  
#endif  
 (void\*)set\_speed, "void set\_speed(int x, int y, int w)",  
 (void\*)set\_debug\_motor, "void set\_debug\_motor(int status, int motor\_id)",  
 (void\*)move\_by\_encoder, "void move\_by\_encoder(int direct, int val)",  
 (void\*)direct\_move, "void direct\_move(int direct, int line\_num,int edge\_if)",  
 (void\*)set\_track\_pid, "void set\_track\_pid(int kp, int ki, int kd)",  
 (void\*)track\_status, "void track\_status(int id, int status)",  
 (void\*)set\_imu\_param, "void set\_imu\_param(int p,int i,int d)",  
 (void\*)set\_imu\_status, "void set\_imu\_status(int status)",  
 (void\*)Set\_InitYaw, "void Set\_InitYaw(int target)",  
 (void\*)turn\_angle, "void turn\_angle(int mode ,int angle)",  
 (void\*)Wait\_Switches, "void Wait\_Switches(int dir)",  
};  
///////////////////////////////////END///////////////////////////////////////////////  
/////////////////////////////////////////////////////////////////////////////////////  
//函数控制管理器初始化  
//得到各个受控函数的名字  
//得到函数总数量  
struct \_m\_usmart\_dev usmart\_dev =  
{  
 usmart\_nametab,  
 usmart\_init,  
 usmart\_cmd\_rec,  
 usmart\_exe,  
 usmart\_scan,  
 sizeof(usmart\_nametab) / sizeof(struct \_m\_usmart\_nametab), //函数数量  
 0, //参数数量  
 0, //函数ID  
 1, //参数显示类型,0,10进制;1,16进制  
 0, //参数类型.bitx:,0,数字;1,字符串  
 0, //每个参数的长度暂存表,需要MAX\_PARM个0初始化  
 0, //函数的参数,需要PARM\_LEN个0初始化  
};

#### 串口助手显示效果

此处应有一张图片

## 底层电机驱动部分（核心）

#### 区别及优点

* 由于今年采用了新一代的主控板且在设计阶段就决定不采用以往的定时器集成式编码器方案来计算电机的实时转速，而是对编码器方波周期进行检测并反比例反映出转速情况，采用时间作为转速的计算标准，相对以往的计算方波边沿的计数方式并周期性查询，新方法不会有因为查询频率的变化而数值变化，提高了数据的准确度。

#### 实现过程中发现的**问题**及其**解决方案**

* 1. 关于捕获极性的选择  
     关于捕获极性的选择，一开始选择的是双边沿检测方法，但因未尝试单边沿方法的相对优缺点。后来在驱动电机时，发现在低转速情况下，采用单边沿检测将会因为周期过长而无法检测，导致**卡死在等待下一个周期边沿到来的过程中无法退出**。虽然单边沿的检测逻辑与实现较为简单，但无法解决低转速的问题。故最后采用的是双边沿检测方法。
  2. 关于数据处理问题  
     由于方波信号毛刺的存在，电机的转速值原始值存在不小的波动，如果直接采用原始数据，将会降低稳定性，特别是因为毛刺干扰了对转动反向的判断，将会瞬间得到一个与真实值相反的异常值，对于此问题，在经过相关的测试之后，决定进行滤波处理以及加入反向检测，避免对转动方向  
     的误判。  
     **以下是代码示例，以1号电机为例。**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* @Name HAL\_TIM\_IC\_CaptureCallback  
 \* @declaration : handle tim ic event for encoder  
 \* @param htim: [输入/出] tim structure ptr  
 \* @retval : void  
 \* @author peach99CPP  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
void HAL\_TIM\_IC\_CaptureCallback(TIM\_HandleTypeDef \*htim)  
{  
 double temp\_val = 0;  
 if(htim == motor1.IC.Tim && htim->Channel == motor1.IC.Active\_Channel)  
 {  
 if(!status\_flag[1])//第一次捕获是捕获到上升沿  
 {  
 status\_flag[1] = 1;//状态标志位置1，下次进中断是在下一步  
 rising\_val[1] = HAL\_TIM\_ReadCapturedValue(motor1.IC.Tim, motor1.IC.Channel);//读取此时上升沿的值  
 update\_count[1] = 0;//更新事件计数器 置0  
 //判断方向，分辨是正转还是反转  
 if(HAL\_GPIO\_ReadPin(motor1.Encoder\_IO.Port, motor1.Encoder\_IO.Pin) == GPIO\_PIN\_RESET)  
 {  
 direct\_[1] = FORWARD;  
 }  
 else  
 {  
 direct\_[1] = BACKWARD;  
 }  
 \_\_HAL\_TIM\_SET\_CAPTUREPOLARITY(motor1.IC.Tim, motor1.IC.Channel, TIM\_ICPOLARITY\_FALLING);//下一次是捕获下降沿  
 }  
 else//捕获到下降沿  
 {  
 status\_flag[1] = 0 ;//状态位清除，一个捕获循环完成。下一次就是捕获上升沿  
 falling\_val[1] = HAL\_TIM\_ReadCapturedValue(motor1.IC.Tim, motor1.IC.Channel);//读取下降沿的值  
 cap\_temp\_val[1] += (SPEED\_PARAM / (falling\_val[1] - rising\_val[1] + TIM\_COUNT\_VAL \* update\_count[1])) \* direct\_[1];//计算本次得到的脉宽。反映出转速的快慢，并累加  
 cap\_cnt[1]++;//采样次数累加，根据采样次数和滤波次数求均值  
  
 \_\_HAL\_TIM\_SET\_CAPTUREPOLARITY(motor1.IC.Tim, motor1.IC.Channel, TIM\_ICPOLARITY\_RISING);//本采样循环完成，回到初始状态，准备对上升沿进行采样  
  
 if(cap\_cnt[1] == FILTER)//采样次数到达了  
 {  
 if(!first\_flag[1])//第一次的时候，因为没有上一次的值，需要进行特殊处理  
 {  
 first\_flag[1] = 1;  
 encoder\_val[1] = cap\_temp\_val[1] / FILTER;  
 }  
 else  
 {  
 //普遍的情况  
 temp\_val = cap\_temp\_val[1] / FILTER;//获取本采样周期内的平均值  
 if(!(fabs(temp\_val + encoder\_val[1]) < THRESHOLD\_)) //没有因为毛刺发生方向跳变，有的话直接舍弃本次获得的值  
 {  
 //均值滤波  
 temp\_val += encoder\_val[1];  
 encoder\_val[1] = temp\_val / (2.0);  
 /\*舍弃下面方法原因在于\  
 有概率在第一第二句执行间隙时间内，编码器值被读取  
 此时编码器的值加上了临时值会异常的大  
 容易引起异常  
 故舍弃。  
 encoder\_val[1] += temp\_val;  
 encoder\_val[1] /= 2.0;//均值滤波  
  
 \*/  
 }  
 }  
 //相关变量清0 ！记得清0！  
 temp\_val = 0;  
 cap\_cnt[1] = 0;  
 cap\_temp\_val[1] = 0 ;  
 }  
  
 }  
  
  
 }

1. 转速值的回零问题  
    在[编码器实现逻辑](#实现)中已经说过，通过检测方波边沿来进入计算的函数。该方法存在一个显著的问题在于，当**电机转速为0时，因为方波不再产生，将不再进入计算转速的函数中**，也就是说，`转速没有得到更新 停留在上一个值==,对于此问题，先后有两个解决方案 ：

* 1. 在每一次读取完转速值后将其清0  
   此方案借鉴了之前读取编码器的操作，读取之后便将其清0，但是因为滤波的操作，此方法将会降低数值的真实性以及导致速值的实时反应速度低。  
     
   2. 检测 是否超时  
   因为在编码器计算过程中会不断存在定时器中断的产生，在某些特殊时刻，编码器值的计算还与编码器的更新频率有关。因此在定时器中断中检测上一次进入边沿中断的时间，如果时长超过一定阈值(说明此时停转），则将电机的转速清0，并且将其他相关变量重新初始化一遍。经过测试，此方法有较高的可行性，能够较为及时准确地对停转的情况做出反应。
* **代码示例**
* /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  \*@name:HAL\_TIM\_PeriodElapsedCallback  
  \*@function:利用定时器来刷新任务,计算时长，只有不修改IRQHandler才能触发此函数  
  \*@param:定时器结构体  
  \*@return:无  
  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  void MY\_PeriodElapsedCallback(TIM\_HandleTypeDef \*htim)  
  {  
   if(htim->Instance == TIM3)  
   {  
   //用于计算脉宽，处理捕获中途发生定时器更新事件的情况  
   if(++update\_count[1] >= 3) //当更新中断事件发生太多，说明此时的电机处于不转的状态，故电机转速置0  
   {  
   cap\_cnt[1] = 0;//重新启动 滤波器  
   cap\_temp\_val[1] = 0 ;//重置临时转速存储值  
   status\_flag[1] = 0;//回到对上升沿的捕获  
   update\_count[1] = 0;//清空时间计数器  
   encoder\_val[1] = 0;//转速清0  
   }  
   if(++update\_count[2] >= 3)  
   {  
   cap\_cnt[2] = 0;  
   cap\_temp\_val[2] = 0 ;  
   status\_flag[2] = 0;  
   update\_count[2] = 0;  
   encoder\_val[2] = 0;  
   }  
   }  
   else if(htim->Instance == TIM5)  
   {  
   if(++update\_count[3] >= 3)  
   {  
   cap\_cnt[3] = 0;  
   cap\_temp\_val[3] = 0 ;  
   status\_flag[3] = 0;  
   update\_count[3] = 0;  
   encoder\_val[3] = 0;  
   }  
   if(++update\_count[4] >= 3)  
   {  
   cap\_cnt[4] = 0;  
   cap\_temp\_val[4] = 0 ;  
   status\_flag[4] = 0;  
   update\_count[4] = 0;  
   encoder\_val[4] = 0;  
   }  
   }  
    
  }

#### 效果

* 此处应有一张图

#### 升级目标

* 当前的编码器的计算值都为什10^3数量级（为了增大对低转速情况的敏感程度），因为数量级的大，导致误差对数据的影响也被放大，当前方案下，稳定转动时，计算的得到的值的波动范围在+- 15范围，
* 所以接下来的目标在于如何优化数据处理的部分，进一步降低数据的波动程度同时确保数据的真实性
  + -

## 陀螺仪

**陀螺仪逻辑流程图**

### 选型

本程序使用的陀螺仪型号为**正点原子ATK\_IMU601**，因其官方例程中提供了解析数据的函数，所以该部分主要是基于陀螺仪数据实现功能的讲解

#### 结构体介绍:

当运行环境发生改变时，只需要修改结构体成员变量的值即可

ATK\_IMU\_t imu =  
{  
 /\*移植时只需要修改以下结构体变量即可\*/  
  
 .imu\_uart = &huart6, //串口号  
 .yaw\_ptr = &(attitude.yaw), //解析出来的原始数据的指针  
 .target\_angle = 0, //pid的目标角度  
 .init\_angle = 0, //初始化角度，补偿上电时的初始角度  
 .switch\_ = 1, //使能开关  
 .get\_angle = Get\_Yaw //函数指针，返回经过限幅和相对0的角度  
};

#### 2.1 **陀螺仪数据接口**

1. 解析函数本质上调用的是正点原子的解析函数。因为在**实际运行环境**下，会发现原有的单字节接收处理方法很容易触发ORE中断而卡死在中断（与循迹板的接收函数同样问题）  
   因为此时系统内有其他多个中断正在运行，串口收到的字节极易未被CPU处理就直接被下一个字节覆盖而触发ORE中断。为避免出现上述问题，采用DMA进行传输。  
   **以下为自行编写的接收函数,通过DMA接收数据，然后再对一整帧的数据进行解析**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* @Name IMU\_IRQ  
 \* @declaration :陀螺仪的中断处理函数，处理DMA收到的数据,在DMA 开启的情况下，放在HAL\_UART\_RxCpltCallback中  
 \* @param None  
 \* @retval : 无  
 \* @author peach99CPP  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
void IMU\_IRQ(void)  
{  
  
 for(uint8\_t i = 0; i < BUFFER\_SIZE; ++i)//开始遍历DMA接收到的数据  
 {  
 if(imu901\_unpack(imu\_cmd[i]))//接收完成  
 atkpParsing(&rxPacket);//开始解码，得到姿态角，此函数是正点原子例程中的函数  
 }  
  
 HAL\_UART\_Receive\_DMA(imu.imu\_uart, imu\_cmd, BUFFER\_SIZE);//再次开启DMA  
}

2.角度限幅函数，用于将角度转换到+-180范围之内，实现传入参数的规范化

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* @Name angle\_limit  
 \* @declaration :  
 \* @param angle: [输入/出]  
 \* @retval :  
 \* @author peach99CPP  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
float angle\_limit(float angle)  
{  
 //把传进来的角度限制在正负180范围  
limit\_label:  
 //采用while的原因在于相比IF可以实现更大的处理范围  
 while(angle > 180) angle -= 360;  
 while(angle <= -180) angle += 360;  
 if(ABS(angle) > 180) goto limit\_label;//意义不大，但是避免出错  
 return angle;  
}

3.角度初始化函数（参数为想设定当前角度为多少度）  
 因为陀螺仪上电时默认为0度，且不随着软件复位而复位
故编写此函数，实现将当前角度值设定为指定角度(软件层面)而无需修改IMU硬件  
***核心思想在于设置补偿角，用于补偿上电时的误差角度***

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* @Name Set\_InitYaw  
 \* @declaration : 设置当前角度为xx度  
 \* @param target: [输入/出] 想设置的角度值  
 \* @retval : 无  
 \* @author peach99CPP  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
void Set\_InitYaw(int target)  
{  
 imu.switch\_ = 0;//先把陀螺仪关掉，否则容易在修改过程中异常  
  
 imu.target\_angle = angle\_limit(target);//同步修改  
  
 float last\_yaw, current\_yaw;  
 int init\_times = 0;  
  
 current\_yaw = last\_yaw = \* imu.yaw\_ptr;//获取当前原生数据  
 while(init\_times < INIT\_TIMES)//陀螺仪未达到稳定  
 {  
 if(fabs( current\_yaw - last\_yaw) < ERROR\_THRESHILD)//偏移很小  
 init\_times ++;  
 else init\_times = 0;//飘了，清0，重置  
 //更新数值  
 last\_yaw = current\_yaw;  
 current\_yaw = \* imu.yaw\_ptr;  
 //10ms一次查询,阻塞式  
 HAL\_Delay(10);  
 }  
 //陀螺仪稳定，开始获取数据  
 imu.init\_angle = angle\_limit(- angle\_limit(current\_yaw) + angle\_limit(target));  
 //恢复陀螺仪的使能状态  
 imu.switch\_ = 1;  
}

1. 基于上一个函数实现了软件层面的**指定初始角度的修改**，因此使用的不再是传感器返回的数据，而是经过处理的函数  
   此函数用于返回**经过处理的角度**，结构体中的函数指针就是指向此函数

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* @Name Get\_Yaw  
 \* @declaration :获取经过限幅的相对于上电位置的Yaw角  
 \* @param None  
 \* @retval : 无  
 \* @author peach99CPP  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
float Get\_Yaw(void)  
{  
 float angle = \*(imu.yaw\_ptr) + imu.init\_angle ;//获取当前原生数据加上补偿角  
 return angle\_limit(angle);  
}

***上述函数在移植时可以直接使用，只需要根据实际修改结构体的成员变量***

#### 2.2 **基于陀螺仪数据的应用**

全部源码在imu\_pid.c  
 *核心思想在于设置想转到的角度为PID的目标值，而后利用实时系统来周期性刷新PID函数*   
 *并将其控制值加在速度上*

1. 转弯的入口函数

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* @Name turn\_angle  
 \* @declaration : 转弯的函数实现，可实现绝对角度的转弯和相对角度的转弯  
 \* @param mode: [输入/出] 转弯的类型  
 relative(1): 相对角度，相对当前位置转动  
 absolute(2): 绝对角度，转动到绝对角度（软件设定的0°）  
\*\* angle: [输入/出] 角度数值  
 \* @retval : 无  
 \* @author peach99CPP  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
void turn\_angle(int mode, int angle)  
{  
 if(imu.switch\_)//检查使能状态  
 {  
 //限幅  
 angle = angle\_limit(angle);  
 //相对角度模式  
 if(mode == relative)  
 imu.target\_angle = angle\_limit(imu.get\_angle() + angle);  
 //绝对角度模式  
 else if( mode == absolute )  
 imu.target\_angle = angle;  
 while(!get\_turn\_status()) osDelay(10);//转动结束再退出该函数  
 //开三秒循迹后关闭，防止转弯后车身偏离线太多导致循迹无法修正  
 x\_leftbar.if\_switch = true;  
 x\_rightbar.if\_switch = true;  
 y\_bar.if\_switch = true;  
 osDelay(3000);  
 x\_leftbar.if\_switch = false;  
 x\_rightbar.if\_switch = false;  
 y\_bar.if\_switch = false;  
 }  
}

1. 陀螺仪修正函数(底层实现逻辑)  
   依赖此函数的周期调用来实现车身角度的修正

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* @Name imu\_correct\_val  
 \* @declaration : imu pid实现的核心函数  
 \* @param None  
 \* @retval :  
 \* @author peach99CPP  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
float imu\_correct\_val(void)  
{  
 static float now\_angle;//过程变量，为避免重复声明，使用静态变量  
 //判断此时转弯的状态  
 if(fabs(imu.get\_angle() - imu.target\_angle) < 2.0) if\_completed = 1;  
 else if\_completed = 1 ;  
  
 if(! imu.switch\_ ) return 0; //未使能则直接返回0，不做修改  
 else  
 {  
 imu\_data.expect = imu.target\_angle;//设置好pid的目标  
 now\_angle = imu.get\_angle();//获取角度数值  
 //取最优路径  
 if(now\_angle - imu.target\_angle > 180 ) now\_angle -= 360;  
 if(now\_angle - imu.target\_angle < -180 ) now\_angle += 360;  
 //pid传参  
 imu\_data.feedback = now\_angle;  
 //获取PID值  
 delta = pos\_pid\_cal(&imu\_data, &imu\_para);  
 return delta;//返回计算值  
  
 }  
}

## 循迹板（巡线）

1. 循迹板信息  
   本程序使用了宏佳电子的八路激光循迹版，以下是其外观:
2. 必要的信息：  
   **实现逻辑**   
   [循迹板流程图直链](https://m.liuchengtu.com/lct//#R26b8d6fa19f4b990bac03dac30e8a959)  
     
     
   此款循迹板支持多种输出口，包括**串口**、**IO口**、**PWM**、**ADC**等输出方式  
   但因为其串口输出最高频率为20Hz，不满足要求，所以只能使用最基础的直接读取IO口电平的方法来实现数据的采集  
   此外，由于该款循迹板扫到白线后为低电平输出，但在代码逻辑中读到白线是高，所以最后在输出结果时需要按位进行取反操作或者在进行判断时手动取反。
   1. 两种控制模式：  
         
       **以下为代码实现，重点在**获取陀螺仪使能状态之后的两种处理模式  
      **将循迹版的PID用于不同用途**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*@name:chassis\_synthetic\_control  
\*@function:底盘的综合控制函数，包含多种控制  
\*@param:无  
\*@return:无  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
void chassis\_synthetic\_control(void)  
{  
 static float x, y, w, factor;  
 static double max\_val;  
 if (chassis.\_switch == false ) return; //如果底盘不被使能，则没有后续操作  
  
 if (++time\_count == TIME\_PARAM)  
 {  
 time\_count = 0;  
 y\_error = track\_pid\_cal(&y\_bar);  
 x\_error = (track\_pid\_cal(&x\_rightbar) - track\_pid\_cal(&x\_leftbar)) / 2.0;  
 }  
  
 max\_val = 0;//对最大值数据进行初始化  
 factor = 1;//倍率因子初始化  
 min\_val = chassis.x\_speed;  
 if(min\_val > y) min\_val = chassis.y\_speed;  
 if(min\_val > w) min\_val = chassis.w\_speed;  
 if(min\_val > 100)  
 {  
 speed\_factor = min\_val / 100.0;  
 }  
 else speed\_factor = 1;  
  
 if(Get\_IMUStatus())  
 {  
 //陀螺仪开启时的巡线模式，已测试通过  
 w\_error = imu\_correct\_val();  
 x = chassis.x\_speed - y\_error \* speed\_factor ;  
 y = chassis.y\_speed - x\_error \* speed\_factor;  
 w = chassis.w\_speed + w\_error \* speed\_factor;  
  
 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 1\*\*\*\*\*\*\*\*\*\*\*\*\*2  
 \*\*\*\*\*\*\*\*\*\*\*\*\*  
 \*\*\*\*\*\*\*\*\*\*\*\*\*  
 \*\*\*\*\*\*\*\*\*\*\*\*\*  
 \*\*\*\*\*\*\*\*\*\*\*\*\*  
 3\*\*\*\*\*\*\*\*\*\*\*\*\*4  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 motor\_target[1] = 0.707 \* y + 0.707 \* x - Radius\_[1] \* w;  
 motor\_target[2] = -0.707 \* y + 0.707 \* x - Radius\_[2] \* w;  
 motor\_target[3] = 0.707 \* y - 0.707 \* x - Radius\_[3] \* w;  
 motor\_target[4] = -0.707 \* y - 0.707 \* x - Radius\_[4] \* w ;  
 }  
 else  
 {  
 //陀螺仪关闭状态下的巡线，待测试  
 x = chassis.x\_speed;  
 y = chassis.y\_speed;  
 w = chassis.w\_speed;  
 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 1\*\*\*\*\*\*\*\*\*\*\*\*\*2  
 \*\*\*\*\*\*\*\*\*\*\*\*\*  
 \*\*\*\*\*\*\*\*\*\*\*\*\*  
 \*\*\*\*\*\*\*\*\*\*\*\*\*  
 \*\*\*\*\*\*\*\*\*\*\*\*\*  
 3\*\*\*\*\*\*\*\*\*\*\*\*\*4  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 //明日调试内容  
 motor\_target[1] = 0.707 \* y + 0.707 \* x - Radius\_[1] \* w + speed\_factor \* (y\_error + x\_error);  
 motor\_target[2] = -0.707 \* y + 0.707 \* x - Radius\_[2] \* w + speed\_factor \* (y\_error + x\_error);  
 motor\_target[3] = 0.707 \* y - 0.707 \* x - Radius\_[3] \* w + speed\_factor \* (y\_error + x\_error);  
 motor\_target[4] = -0.707 \* y - 0.707 \* x - Radius\_[4] \* w + speed\_factor \* (y\_error + x\_error);  
 }  
  
 //再来一个限幅操作，避免单边速度过高导致控制效果不理想  
 //  
  
  
  
 for (i = 1; i <= 4; ++i) //找出最大值  
 {  
 if (motor\_target[i] > max\_val)  
 max\_val = motor\_target[i];  
 }  
 factor = (max\_val > MAX\_SPEED) ? MAX\_SPEED / max\_val : 1;  
 if (max\_val > MAX\_SPEED)//最大值是否超限制，进行操作，确保最大值仍在范围内且转速比例 不变  
 {  
 factor = MAX\_SPEED / max\_val;  
 for (i = 1; i < 4; ++i)  
 {  
 motor\_target[i] \*= factor;  
 }  
  
 }  
  
 for (i = 1; i <= 4; ++i)  
 {  
 /\*  
 \*对电机进行遍历  
 \*首先获取转速期待值  
 \*读取编码器参数  
 \*传入PID计算函数得到计算值,以返回值形式传参  
 \*由计算值控制电机  
 \*/  
 motor\_data[i].expect = motor\_target[i];  
 motor\_data[i].feedback = read\_encoder(i);  
 control\_val[i] = pid\_control(&motor\_data[i], &motor\_param);  
 if(control\_val[i] > MAX\_CONTROL\_VAL ) control\_val[i] = MAX\_CONTROL\_VAL;  
 if(control\_val[i] < - MAX\_CONTROL\_VAL ) control\_val[i] = - MAX\_CONTROL\_VAL;  
 set\_motor(i, control\_val[i]);  
  
  
 }  
// printf("%.2f %.2f",motor\_data[debug\_motor\_id].feedback, motor\_data[debug\_motor\_id].expect);  
// printf("\r\n");  
  
}

##### 重要信息：

**由于CubeMX的Bug，默认生成的代码中关于DMA的初始化函数在USART的初始化函数之后**  
**这将导致IDLE和DMA被失能，所以每一次使用CubeMX生成代码之后需要手动将DMA初始化移动到USART初始化之前！！**

1. 拓展板程序：
   1. 对IO口电平进行采集,并将采集到的信息进行编码到一个字节中

void track\_encode(void)//在定时器中刷新该任务  
{  
 //预先准备用宏定义参数解析来完成，但有玄学问题，放弃  
 /\*\*\*\*\*\*\*\*\*\*\*\*\*1号寻迹板\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 if(HAL\_GPIO\_ReadPin(T1\_1\_GPIO\_Port, T1\_1\_Pin) == GPIO\_PIN\_RESET)  
 track\_value[1] |= 1 << 0;  
 else  
 track\_value[1] &= ~(1 << 0);  
  
 if(HAL\_GPIO\_ReadPin(T1\_2\_GPIO\_Port, T1\_2\_Pin) == GPIO\_PIN\_RESET)  
 track\_value[1] |= 1 << 1;  
 else  
 track\_value[1] &= ~(1 << 1);  
  
 if(HAL\_GPIO\_ReadPin(T1\_3\_GPIO\_Port, T1\_3\_Pin) == GPIO\_PIN\_RESET)  
 track\_value[1] |= 1 << 2;  
 else  
 track\_value[1] &= ~(1 << 2);  
  
 if(HAL\_GPIO\_ReadPin(T1\_4\_GPIO\_Port, T1\_4\_Pin) == GPIO\_PIN\_RESET)  
 track\_value[1] |= 1 << 3;  
 else  
 track\_value[1] &= ~(1 << 3);  
  
 if(HAL\_GPIO\_ReadPin(T1\_5\_GPIO\_Port, T1\_5\_Pin) == GPIO\_PIN\_RESET)  
 track\_value[1] |= 1 << 4;  
 else  
 track\_value[1] &= ~(1 << 4);  
  
 if(HAL\_GPIO\_ReadPin(T1\_6\_GPIO\_Port, T1\_6\_Pin) == GPIO\_PIN\_RESET)  
 track\_value[1] |= 1 << 5;  
 else  
 track\_value[1] &= ~(1 << 5);  
  
 if(HAL\_GPIO\_ReadPin(T1\_7\_GPIO\_Port, T1\_7\_Pin) == GPIO\_PIN\_RESET)  
 track\_value[1] |= 1 << 6;  
 else  
 track\_value[1] &= ~(1 << 6);  
  
  
 if(HAL\_GPIO\_ReadPin(T1\_8\_GPIO\_Port, T1\_8\_Pin) == GPIO\_PIN\_RESET)  
 track\_value[1] |= 1 << 7;  
 else  
 track\_value[1] &= ~(1 << 7);  
  
  
  
 /\*\*\*\*\*\*\*\*\*\*\*\*\*2号寻迹板\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 if(HAL\_GPIO\_ReadPin(T2\_1\_GPIO\_Port, T2\_1\_Pin) == GPIO\_PIN\_RESET)  
 track\_value[2] |= 1 << 0;  
 else track\_value[2] &= ~(1 << 0);  
  
 if(HAL\_GPIO\_ReadPin(T2\_2\_GPIO\_Port, T2\_2\_Pin) == GPIO\_PIN\_RESET)  
 track\_value[2] |= 1 << 1;  
 else track\_value[2] &= ~(1 << 1);  
  
 if(HAL\_GPIO\_ReadPin(T2\_3\_GPIO\_Port, T2\_3\_Pin) == GPIO\_PIN\_RESET)  
 track\_value[2] |= 1 << 2;  
 else track\_value[2] &= ~(1 << 2);  
  
 if(HAL\_GPIO\_ReadPin(T2\_4\_GPIO\_Port, T2\_4\_Pin) == GPIO\_PIN\_RESET)  
 track\_value[2] |= 1 << 3;  
 else track\_value[2] &= ~(1 << 3);  
  
 if(HAL\_GPIO\_ReadPin(T2\_5\_GPIO\_Port, T2\_5\_Pin) == GPIO\_PIN\_RESET)  
 track\_value[2] |= 1 << 4;  
 else track\_value[2] &= ~(1 << 4);  
  
 if(HAL\_GPIO\_ReadPin(T2\_6\_GPIO\_Port, T2\_6\_Pin) == GPIO\_PIN\_RESET)  
 track\_value[2] |= 1 << 5;  
 else track\_value[2] &= ~(1 << 5);  
  
 if(HAL\_GPIO\_ReadPin(T2\_7\_GPIO\_Port, T2\_7\_Pin) == GPIO\_PIN\_RESET)  
 track\_value[2] |= 1 << 6;  
 else track\_value[2] &= ~(1 << 6);  
  
 if(HAL\_GPIO\_ReadPin(T2\_8\_GPIO\_Port, T2\_8\_Pin) == GPIO\_PIN\_RESET)  
 track\_value[2] |= 1 << 7;  
 else track\_value[2] &= ~(1 << 7);  
  
  
  
  
 /\*\*\*\*\*\*\*\*\*\*\*\*\*3号寻迹板\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 if(HAL\_GPIO\_ReadPin(T3\_1\_GPIO\_Port, T3\_1\_Pin) == GPIO\_PIN\_RESET)  
 track\_value[3] |= 1 << 0;  
 else track\_value[3] &= ~(1 << 0);  
  
 if(HAL\_GPIO\_ReadPin(T3\_2\_GPIO\_Port, T3\_2\_Pin) == GPIO\_PIN\_RESET)  
 track\_value[3] |= 1 << 1;  
 else track\_value[3] &= ~(1 << 1);  
  
 if(HAL\_GPIO\_ReadPin(T3\_3\_GPIO\_Port, T3\_3\_Pin) == GPIO\_PIN\_RESET)  
 track\_value[3] |= 1 << 2;  
 else track\_value[3] &= ~(1 << 2);  
  
 if(HAL\_GPIO\_ReadPin(T3\_4\_GPIO\_Port, T3\_4\_Pin) == GPIO\_PIN\_RESET)  
 track\_value[3] |= 1 << 3;  
 else track\_value[3] &= ~(1 << 3);  
  
 if(HAL\_GPIO\_ReadPin(T3\_5\_GPIO\_Port, T3\_5\_Pin) == GPIO\_PIN\_RESET)  
 track\_value[3] |= 1 << 4;  
 else track\_value[3] &= ~(1 << 4);  
  
 if(HAL\_GPIO\_ReadPin(T3\_6\_GPIO\_Port, T3\_6\_Pin) == GPIO\_PIN\_RESET)  
 track\_value[3] |= 1 << 5;  
 else track\_value[3] &= ~(1 << 5);  
  
 if(HAL\_GPIO\_ReadPin(T3\_7\_GPIO\_Port, T3\_8\_Pin) == GPIO\_PIN\_RESET)  
 track\_value[3] |= 1 << 6;  
 else track\_value[3] &= ~(1 << 6);  
  
  
 if(HAL\_GPIO\_ReadPin(T3\_8\_GPIO\_Port, T3\_8\_Pin) == GPIO\_PIN\_RESET)  
 track\_value[3] |= 1 << 7;  
 else   
 track\_value[3] &= ~(1 << 7);  
}

1. 将状态信息通过约定的协议进行传输

//编码成一个帧  
void cmd\_encode(uint8\_t id, uint16\_t sum)  
{  
 uart\_cmd[0] = HEAD\_BYTE;  
 uart\_cmd[1] = id;  
 uart\_cmd[2] = track\_value[id];  
 uart\_cmd[3] = (sum&0xFF);  
 uart\_cmd[4] = TAIL\_BYTE;  
}  
//通过串口发送给主控板，为提高实时性，发送频率为200Hz  
void send\_value(void)  
{  
 for(uint8\_t i=1;i<=3;++i)  
 {  
 cmd\_encode(i,(i+track\_value[i]));  
 HAL\_UART\_Transmit(track\_uart.uart,uart\_cmd,BUFFER\_SIZE,0xff);  
 delay\_ms(5);  
 }  
}

1. 主控板程序
   * 数据接收程序  
     主控板对于寻迹板接收程序的处理，如果采用单字节结接收处理，很可能因为处理不及时而产生与陀螺仪数据相似的ORE中断  
     因此，其也需要使用DMA进行数据的传输工作。  
      以下是循迹板的初始化和DMA的中断处理函数

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\* @Name track\_bar\_init  
\* @declaration : 对寻迹板需要的相关变量进行初始化的设置  
\* @param None  
\* @retval : 无  
\* @author peach99CPP  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 void track\_bar\_init(void)//相关的初始化函数  
 {  
 dma\_count = 0;  
 dma\_trans\_pos = 0;  
 //变量参数的初始化  
 y\_bar.id = forward\_bar;//标注身份  
 y\_bar.line\_num = 0;  
 y\_bar.num = 0;  
 y\_bar.line\_num = 0;  
 y\_bar.data.expect = 0;//循迹的目标值恒为0  
 y\_bar.if\_switch = true;//使能  
  
 x\_leftbar.id = left\_bar;  
 x\_leftbar.line\_num = 0;  
 x\_leftbar.num = 0;  
 x\_leftbar.line\_num = 0;  
 x\_leftbar.data.expect = 0;  
 x\_leftbar.if\_switch = true;  
  
 x\_rightbar.id = right\_bar;  
 x\_rightbar.line\_num = 0;  
 x\_rightbar.num = 0;  
 x\_rightbar.line\_num = 0;  
 x\_rightbar.data.expect = 0;  
 x\_rightbar.if\_switch = true;  
 //开启DMA接收，此处应确保main中DMA初始化函数在串口初始化前运行  
 HAL\_UART\_Receive\_DMA(&TRACK\_UART, (uint8\_t\*)track\_dma, BUFF\_SIZE);  
 }  
 /\*DMA中断处理，采用的是多个缓冲区，避免数据未处理便被直接覆盖，但也因此需要处理缓冲区位置的选择问题\*/  
 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* @Name track\_IT\_handle  
 \* @declaration : 接收完成中断  
 \* @param None  
 \* @retval : 无  
 \* @author peach99CPP  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 void track\_IT\_handle(void)  
 {  
 uint8\_t pos = get\_avaiable\_pos();//通过获取这个检查是否有被填充进去  
 if(pos != 0xff)  
 {  
 dma\_count++; //大于0代表有未处理的DMA数据  
 dma\_trans\_pos = ((dma\_trans\_pos + 1) >= MAX\_LINE) ? 0 : dma\_trans\_pos + 1; //此接收器接收完成，。换到空闲接收区进行接收，防止数组下标越界  
 HAL\_UART\_Receive\_DMA(&TRACK\_UART, (uint8\_t\*)track\_dma[dma\_trans\_pos], BUFF\_SIZE);//开启DMA接收  
 }  
 //否则就在原缓冲区继续接收  
 else  
 {  
 memset(memset(track\_dma, 0, sizeof(track\_dma)), 0, sizeof(memset(track\_dma, 0, sizeof(track\_dma)))); //清空掉，作用不大，习惯为止  
 HAL\_UART\_Receive\_DMA(&TRACK\_UART, (uint8\_t\*)track\_dma[dma\_trans\_pos], BUFF\_SIZE);  
 }  
 }

* 数据处理程序

此部分是对循迹版信息的解码处理环节。功能实现的核心依赖于此函数。

通过对循迹版信息的处理获取当前循迹板的状态，并以此实现数线、巡线等功能。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* @Name track\_decode  
 \* @declaration :对DMA收到的数据进行解码计算，实现循迹各项功能的核心代码  
 \* @param None  
 \* @retval : 无  
 \* @author peach99CPP  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
void track\_decode(void)  
{  
 /\*\*\*相关宏定义\*\*\*\*/  
#define EDGE\_THRESHOLD 3 //在边缘数线模式下，几颗灯亮起时为有效计数  
#define NUM\_THRESHOLD 6 //非边缘线计算下，判断到达线的数量  
#define MIN\_NUM 3 //在压线之后，过线了才会计算一根线，根据灯的数量进行计数  
#define EDGE\_VAL 7 //边缘数线状态下的循迹读回来的值  
   
 times\_counts++;//总的处理次数，查看此 数据可以判断是否卡DMA  
 dma\_count--;//待处理数-1  
 uint8\_t pos = get\_avaiable\_pos();//获取可用index  
 uint8\_t led\_num = 0;  
 float track\_value = 0, temp\_val; //相关的变量声明  
   
 //下面的和检验看情况开启  
 if((uint8\_t)( track\_dma[pos][1] + track\_dma[pos][2] ) == track\_dma[pos][3]) //和校验  
 {  
 for(uint8\_t i = 0; i < 8; ++i)  
 {  
 temp\_val = (bool)(((track\_dma[pos][2] << i) & 0x80)) \* track\_weight[i];//根据灯亮与否及其权重得到反馈值  
 if( temp\_val != 0) led\_num++;//计算亮的灯数量  
 track\_value += temp\_val;  
 }  
 switch (track\_dma[pos][1])//判断寻迹板ID  
 {  
 case 1:  
 y\_bar.data.feedback = track\_value;//赋值  
 y\_bar.num = led\_num;//得到灯的数量  
 if(y\_bar.num >= NUM\_THRESHOLD || (edge\_status[0] && y\_bar.num >= EDGE\_THRESHOLD && ABS(y\_bar.data.feedback) >= EDGE\_VAL))  
 {  
 /\*有两种情况，  
 \*一是当跑在非边缘时，此时灯的数量比较多  
 \*二是在边缘跑时，此时灯数量较少且需要加多重判断  
 \*/  
 y\_bar.line\_flag = 1;//此时到线上  
 }  
 if(edge\_status[0] && y\_bar.num >= EDGE\_THRESHOLD && ABS(y\_bar.data.feedback) >= EDGE\_VAL ) //边缘数线的情况下，特殊处理  
 y\_bar.data.feedback = 0;//要放在对线的判断之后；置0是为了防止此时发生偏移  
 if(y\_bar.line\_flag && y\_bar.num <= MIN\_NUM )  
 {  
 //使用此机制为了避免因停留在线上而导致线的数量一直重复计数  
 y\_bar.line\_flag = 0;//数线完成  
 y\_bar.line\_num ++;//线数目加一  
 }  
 break;  
 case 2:  
 x\_leftbar.data.feedback = track\_value;  
 x\_leftbar.num = led\_num;  
  
 if(x\_leftbar.num >= NUM\_THRESHOLD || (edge\_status[1] && x\_leftbar.num >= EDGE\_THRESHOLD && ABS(x\_leftbar.data.feedback) >= EDGE\_VAL))  
 {  
 x\_leftbar.line\_flag = 1; //标记到了线上  
 }  
 if(edge\_status[1] && x\_leftbar.num >= EDGE\_THRESHOLD && ABS(x\_leftbar.data.feedback) >= EDGE\_VAL)  
 x\_leftbar.data.feedback = 0;  
 if(x\_leftbar.line\_flag && x\_leftbar.num <= MIN\_NUM ) //避免因为在线上停留而导致的重复计数问题  
 {  
 x\_leftbar.line\_flag = 0;  
 x\_leftbar.line\_num ++;  
 }  
 break;  
 case 3:  
 x\_rightbar.data.feedback = track\_value;  
 x\_rightbar.num = led\_num;  
 if( x\_rightbar.num >= NUM\_THRESHOLD || (edge\_status[2] && x\_rightbar.num >= EDGE\_THRESHOLD && ABS(x\_rightbar.data.feedback) >= EDGE\_VAL ))  
 {  
 x\_rightbar.line\_flag = 1;  
 }  
 if(edge\_status[2] && x\_rightbar.num >= EDGE\_THRESHOLD && ABS(x\_rightbar.data.feedback) >= EDGE\_VAL)  
 x\_leftbar.data.feedback = 0;  
 if(x\_rightbar.line\_flag && x\_rightbar.num <= MIN\_NUM )  
 {  
 x\_rightbar.line\_flag = 0;  
 x\_rightbar.line\_num ++;  
 }  
 break;  
 default://啥也不干  
 ;  
 }  
 memset(track\_dma[pos], 0, sizeof(track\_dma[pos])); //把用到的清0，等待下次被填充  
 }  
  
}

* 数据应用  
  **计算PID值，并将返回值输出到底盘的速度上**

/\*  
 \* 此为循迹版的输出函数  
 \* PID的目标值已经设置成0  
\*/  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* @Name track\_pid\_cal  
 \* @declaration : 寻迹板pid计算函数  
 \* @param bar: [输入/出] 哪一个寻迹板  
 \* @retval :  
 \* @author peach99CPP  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
float track\_pid\_cal(trackbar\_t \* bar)  
{  
 if(bar->if\_switch == true)//使能，计算pid值并进行返回  
 {  
 return pos\_pid\_cal(&bar->data, &track\_pid\_param);  
 }  
 return 0;//未使能，不做改变  
}  
  
/\*  
 \*下列代码展示了循迹版PID如何参与底盘速度分解  
 \*/  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*@name:chassis\_synthetic\_control  
\*@function:底盘的综合控制函数，包含多种控制  
\*@param:无  
\*@return:无  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
void chassis\_synthetic\_control(void)  
{  
static float x, y, w, factor;  
static double max\_val;  
if (chassis.\_switch == false ) return; //如果底盘不被使能，则没有后续操作  
  
if (++time\_count == TIME\_PARAM)  
{  
time\_count = 0;  
y\_error = track\_pid\_cal(&y\_bar);  
x\_error = (track\_pid\_cal(&x\_rightbar) - track\_pid\_cal(&x\_leftbar)) / 2.0;  
}  
  
max\_val = 0;//对最大值数据进行初始化  
factor = 1;//倍率因子初始化  
  
w\_error = imu\_correct\_val();  
x = chassis.x\_speed - y\_error ;  
y = chassis.y\_speed - x\_error;  
w = chassis.w\_speed + w\_error;  
min\_val = x;  
if(min\_val > y) min\_val = y;  
if(min\_val > w) min\_val = w;

## 上层机械臂初步方案

#### 整体描述

* 对于机械臂，机械臂的整体驱动依靠于外接的24路舵机板。通过串口发送信息来进行控制。接下来将针对各个任务的具体要求展开描述。

#### 圆盘机和条形平台

* 对于圆盘机和普通平台，为节约时间，在结构设计上我们共设计了拨球和夹取两种方案。在此基础上，针对这两个任务，仅需在巡线过程中，当车头的两个轻触开关闭合后，底盘停止运动，调节机械臂到固定位置后，利用Openmv进行颜色识别和形状识别，然后根据二维码识别得到的颜色，将该颜色的圆球推下至铲子即可，并分流到对应仓库中。  
  图一。

#### 阶梯型平台、码垛台

* 初赛：  
  在初赛中所需要夹取的物块为圆球和积木块，此外由于条形平台中还含有小球并且由于同时夹取圆球和积木块可能会导致积木块进入圆球仓库，因此我们路线规划为先去拨下条形平台上的小球，然后再去夹取阶梯型平台的圆球，关闭舱门后，最后夹取阶梯型平台的积木块。由于积木块和圆球逗得夹取顺序不同，Openmv需要识别好矩形和圆球的区别并且此处机械臂共需前后识别两次。  
  图二。
* 复赛和决赛：  
  复赛和决赛中所需要夹取的物块为积木块和塑料环，这两次比赛中取消了初赛中的条形平台，阶梯型平台变为两个，并增加了码垛台和圆盘机。因此我们路线规划为先去拨下圆盘机中的小球，然后再去夹取阶梯型平台的积木块和塑料环，由于积木块和塑料环摆放位置的不确定性，因此对于每个位置都需要设置两套动作组。  
  图三。

#### 立桩

* 直接调节机械臂进行夹取。稍后完善。  
  图四。

#### 将圆环放置到码垛台上

* 直接调节机械臂。稍后完善。

#### 排下圆球和倒出积木块

* 看最终调试效果后再完善。

## 系统整体运行框架

#### 运行框架

* [整体流程图直链](https://m.liuchengtu.com/lct//#R1e79982406d487b28368954fb46414e7)

#### FreeRTOS

* **在本程序中集成了FreeRTOS，使用的是STM32CubeMX中的CMSIS V1版本，经过了一定程度的封装**  
  相对优点有：
  + 提高其移植性，
  + 同时简化任务创建的操作过程  
    以下是在本程序中用到的任务列表   
      
    每个任务的作用，查阅[运行流程图](#运行流程图)中任务部分的介绍。

#### 全局通信协议：

* 该协议主要适用于主控板与拓展板、openmv间通讯。  
   查看具体应用：[完整源码](../Module)  
  1.实现逻辑：
  + 发送时的编码  
    以openmv的解码函数举例
    - 帧头
    - 事件ID
    - 正负标志位
    - 参数低8位
    - 参数高8位
    - 校验和
    - 帧尾

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* @Name cmd\_encode  
 \* @declaration : 根据协议编码发送的内容  
 \* @param event\_id: [输入/出] 时间的类型  
\*\* param: [输入/出] 参数  
 \* @retval :  
 \* @author peach99CPP  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
void cmd\_encode(const uint8\_t event\_id, int param)  
{  
 static uint8\_t pos\_flag;  
 if(param > 0 ) pos\_flag = 1;  
 else  
 {  
 pos\_flag = 2;  
 param \*= -1;  
 }  
  
 uint8\_t h\_byte, l\_byte;//获取参数的高8位和低8位  
 h\_byte = (param >> 8);  
 l\_byte = (param & 0xff);  
 //定义通讯协议  
 MV.mv\_cmd[0] = START\_BYTE;//帧头  
 MV.mv\_cmd[1] = event\_id;//触发的事件id  
 MV.mv\_cmd[2] = pos\_flag;  
 MV.mv\_cmd[3] = l\_byte;//参数高8位  
 MV.mv\_cmd[4] = h\_byte;//参数低8位  
 MV.mv\_cmd[5] = (uint8\_t)(event\_id + pos\_flag + h\_byte + l\_byte);//和校验  
 MV.mv\_cmd[6] = END\_BYTE;//帧尾  
}

* 接收时解码:   
  以openmv的解码函数举例

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* @Name MV\_rec\_decode  
 \* @declaration : 判断接收完成后，对接收的内容进行解码  
 \* @param None  
 \* @retval : 无  
 \* @author peach99CPP  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
void MV\_rec\_decode(void)  
{  
 static int pn = 1; //正负标志符  
 if(MV.rec\_buffer[0] + MV.rec\_buffer[1] + MV.rec\_buffer[2] + MV.rec\_buffer[3] == MV.rec\_buffer[4])  
 {  
 //根据参数内容对参数进行处理  
 if( MV.rec\_buffer[1] == 1 ) pn = 1;  
 else pn = -1;  
  
 mv\_param = (MV.rec\_buffer[2] + (MV.rec\_buffer[3] << 8)) \* pn;  
 }  
 MV.rec\_len = 0;  
 MV.RX\_Status = 0;  
 //处理完之后记得重新初始化结构体中的rec\_len和RX\_status变量，避免出错  
 ;  
}

2.待改进的地方：

* ~~目前的协议在当前的应用范围下未出现问题~~(死要面子 嘴硬)，但对于其他特殊数据，其存在没有解决透明传输的问题，当因传输错误或数据内容中出现与帧头帧尾相同的字节时，未有有效的方法来裁决其属于正文信号或是帧尾信号可能因此出现错误。   
  目前正计划通过字符填充来解决上述问题。**提高鲁棒性**，在参数传输过程中无需考虑其值可能使接收机制产生误判的可能。
* --

# 有问题联系：

* ***13311889904(TEL)***
* ***1831427532(QQ)***
* [***3120000903@mail2.gdut.edu.cn***](mailto:3120000903@mail2.gdut.edu.cn)   
  [***1831427532@qq.com***](mailto:1831427532@qq.com)
* ***Wechat***
* --