**让M3508转起来**

要让M3508转起来的话，只要发信息告诉电调，我需要你输出这么大的电流来使电机转起来即可，电流可正可负，对应不同的转动方向。

电调ID为1~4的电机都会监听CAN报文ID为0x200的信息，并获取自己ID对应数据域的信息进行输出。

电调ID为5~8的电机都会监听CAN报文ID为0x1FF的信息，并获取自己ID对应数据域的信息进行输出。

由于DJI的电机报文基本都使用这些ID（包括M2006，云台电机6623，GM3510等，详见各电机使用说明书），所以一个CAN上最多挂8个电机，F427有2个CAN则最多挂16个。

图（截自C620电调使用说明P31）

机器人底盘使用的电调ID通常为1-4，则发送ID为0x200

#define M3508\_SENDID 0x200

给ID1-4的电调发送电流数据的函数参考如下

/\*\*

\* @brief 设置M3508电机电流值（id号为1~4）

\* @param iqx (x:1~4) 对应id号电机的电流值，范围-16384~0~16384

\* @retval None

\*/

void M3508\_setCurrent(int16\_t iq1, int16\_t iq2, int16\_t iq3, int16\_t iq4){

uint8\_t data[8];

//数据格式详见说明书P32

data[0] = iq1 >> 8;

data[1] = iq1;

data[2] = iq2 >> 8;

data[3] = iq2;

data[4] = iq3 >> 8;

data[5] = iq3;

data[6] = iq4 >> 8;

data[7] = iq4;

CAN\_SendData(CAN1, CAN\_ID\_STD, M3508\_SENDID, data);

}

如果想使用ID5-8把发送报文ID改成0x1FF即可

**获取M3508的反馈数据**

如果要控制电机的速度，肯定需要利用电调反馈的速度数据进行闭环，因为电调自带的电流环虽然可以比较稳定的输出你设置的电流使电机转动，但由于输出恒定的转矩电流会使电机越转越快，而不是保持恒定的速度转动。（F = ma，v = v0+at）



图（截自C620电调使用说明P33）

根据说明书，我们知道了我们可以得到的信息，由此，我定义了一个电机信息的结构体来存储这些信息。

结构体内容如下

typedef struct{

uint16\_t realAngle; //读回来的机械角度

int16\_t realSpeed; //读回来的速度

int16\_t realCurrent; //读回来的实际电流

uint8\_t temperture; //读回来的电机温度

int16\_t targetSpeed; //目标速度

uint16\_t targetAngle; //目标角度

int16\_t outCurrent; //输出电流

incrementalpid\_t pid; //电机pid

uint8\_t infoUpdateFlag; //信息读取更新标志

uint16\_t infoUpdateFrame; //帧率

uint8\_t offLineFlag; //设备离线标志

}M3508s\_t;

底盘使用四个3508驱动

生成四个3508的信息结构体，索引号0-3

M3508s\_t M3508s[4];

底盘电机信息返回的CAN报文id对应0x201~0x204，

#define M3508\_READID\_START 0x201

#define M3508\_READID\_END 0x204

接下来就可以把CAN报文中底盘4个3508的信息存在对应索引的结构体中

报文ID与索引号的关系：

当我们四个M3508的id连续时

索引号 = 报文ID- M3508\_READID\_START

M3508信息获取函数参考如下，放在CAN接收中断里即可

/\*\*

\* @brief 从CAN报文中获取M3508电机信息

\* @param RxMessage CAN报文接收结构体

\* @retval None

\*/

void M3508\_getInfo(CanRxMsg RxMessage){

//报文id确认

if((RxMessage.StdId < M3508\_READID\_START) || (RxMessage.StdId > M3508\_READID\_END))

return;

uint32\_t IndexId;

IndexId = RxMessage.StdId - M3508\_READID\_START;

//解包数据，数据格式详见C620电调说明书P33

M3508s[IndexId].realAngle = (uint16\_t)(RxMessage.Data[0]<<8 | RxMessage.Data[1]);

M3508s[IndexId].realSpeed = (int16\_t)(RxMessage.Data[2]<<8 | RxMessage.Data[3]);

M3508s[IndexId].realCurrent = (int16\_t)(RxMessage.Data[4]<<8 | RxMessage.Data[5]);

M3508s[IndexId].temperture = RxMessage.Data[6];

//帧率统计，数据更新标志位

M3508s[IndexId].infoUpdateFrame++;

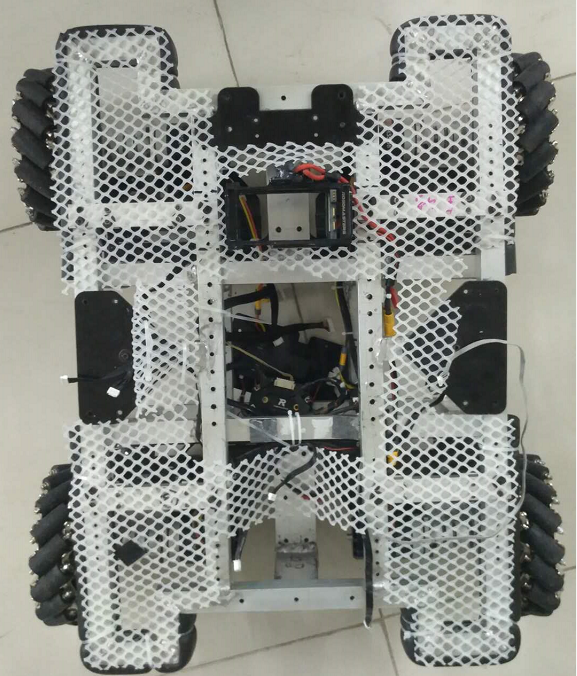
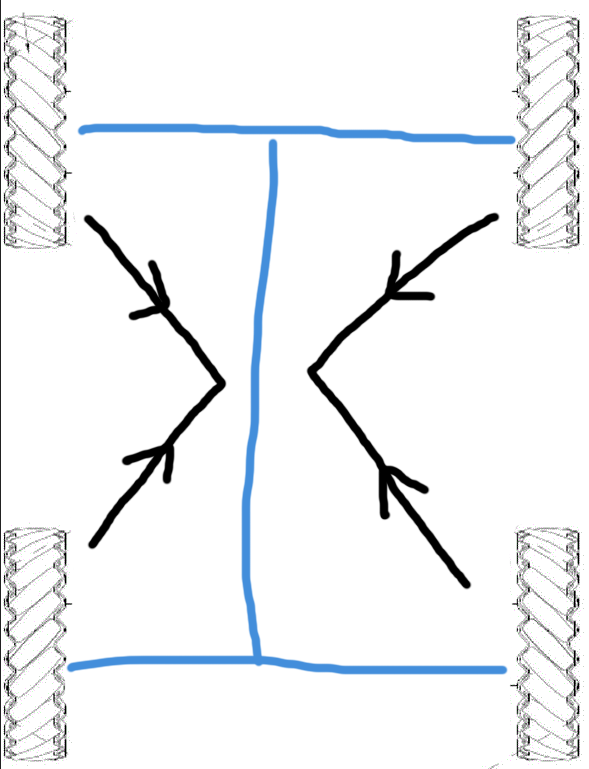
M3508s[IndexId].infoUpdateFlag = 1;

}

**麦克纳姆轮运动模型**

**麦轮的安装方式**

想要麦轮实现全向移动，麦轮必须以这个方式安装(比较常用的安装方式)，**注意小胶轮的朝向**

当然也有其他的安装方式。（旧的6轮工程车使用麦轮+全向轮的底盘，中间全向轮，四边麦轮。全向轮用于自旋和前进，麦轮用于前进和平移。这时麦轮就要以另外一种方式安装，使底盘在Omega方向有个自由度）

**麦轮速度分解**

全向移动底盘是一个纯线性系统，而刚体运动又可以线性分解为三个分量。那么只需要给出麦轮底盘在沿X轴平移(Vx)、沿Y轴平移(Vy)、绕几何中心自转(VOmega)的速度，就可以通过简单的加法，就可以计算出这三种简单运动所合成的「平动+旋转」运动时所需要的四个轮子的转速。

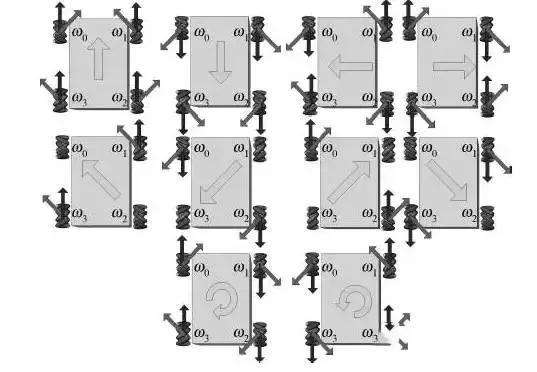


图 麦轮速度分解

上图为底盘运动时每个轮子的转向和受力情况，你也可以尝试把你的底盘按在地上手动实现上图的运动，轮子转向应该和上图相同。

下面我们开始实现底盘的运动

此处设底盘ID如下，当然你也可以和我不一样，只不过正负号会不同

\*\*\*\* 底盘ID \*\*\*\*

车头

2 1

3 4

\*\*\*\* \*\*\*\*

由于电机是对向安装，所以左右两个电机给同样电流输出时转向应该相反。

上面说了每个轮的速度只是底盘3个方向运动速度的加减法

（即Speed【N】 = ± Vx ± Vy ± VOmega，正负号取决于电机实际转向）

首先是Vy（前进和后退），根据上图的1,2可以得出四个轮的速度方向和Vy的关系

这时，只要给Vy值，底盘应该可以实现前后移动。

现在把Vx也加上，根据上图的3,4可以得出四个轮的速度方向和Vx的关系，把他加到轮子速度上

这时，只要给Vx值，底盘应该可以实现左右平移。如果Vy也有速度，也可实现图5~8的移动。

现在把VOmega也加上，根据上图的9,10可以得出四个轮的速度方向和VOmega的关系，把他加到轮子速度上

这时，只要给VOmega值，底盘应该可以实现左右自旋。现在底盘已经可以实现全向移动了。

速度分解代码如下

//四轮速度分解

tempSpeed[0] = Vx - Vy + VOmega;

tempSpeed[1] = Vx + Vy + VOmega;

tempSpeed[2] = -Vx + Vy + VOmega;

tempSpeed[3] = -Vx - Vy + VOmega;

**速度分配**

当你把Vx，Vy，VOmega给到一定程度时，轮子的速度一定会产生一个饱和的现象，就是电机已经达到最大转速，没办法加速来实现全向移动了。但是在电机满速时你还是需要实现底盘的全速全向移动，这时，就需要进行速度分配，每个电机的速度将按照轮子的最大速度和现在的速度的比值进行等比减速。这样得出四个轮子的速度都不会大于你设定的最大速度。

我把单轮最大速度限制在6600

#define WheelMaxSpeed 6600.0f //单轮最大速度

速度分配比= 单轮最大速度 / 四个轮子的最大速度;

速度分配参考代码如下

float MaxSpeed = 0.0f;

float Param = 1.0f;

//寻找最大速度

for(uint8\_t i = 0; i < 4; i++){

if(abs(tempSpeed[i]) > MaxSpeed){

MaxSpeed = abs(tempSpeed[i]);

}

}

//计算速度分配比

if(MaxSpeed > WheelMaxSpeed){

Param = (float)WheelMaxSpeed / MaxSpeed;

}

Speed[0] = tempSpeed[0] \* Param;

Speed[1] = tempSpeed[1] \* Param;

Speed[2] = tempSpeed[2] \* Param;

Speed[3] = tempSpeed[3] \* Param;

经过速度分配后，当某个轮子达到最大速度时，其他轮子的速度会按照比例下降，以满足全向移动的速度比例。

**电流分配**

RM比赛中，对底盘功率有限制，如果不对底盘功率进行一个限制的话很容易就会因超功率而扣血。

对于步兵来说，在60J的底盘功率缓冲用完之前，要对电机的输出电流进行限制。即把80W的功率按照比例分配到四个电机上。这和速度分配有点不一样，电流分配将会把可用的电流按照目前的需求比例分配到每个电机上，而不是仅把其他的电机电流下降。

电流分配函数参考如下

/\*\*

\* @brief 底盘功率限制

\* @param p\_M3508s 4个电机结构体指针

\* CurrentLimit 可输出的总电流

\* @retval None

\*/

void Chassis\_powerLimit(M3508s\_t \*p\_M3508s, float CurrentLimit){

float Current\_Sum = 0.0f;

//计算电流和

Current\_Sum = abs(p\_M3508s[0].outCurrent) + abs(p\_M3508s[1].outCurrent) \

+ abs(p\_M3508s[2].outCurrent) + abs(p\_M3508s[3].outCurrent);

//电流分配

if(Current\_Sum > CurrentLimit){

p\_M3508s[0].outCurrent = CurrentLimit \* p\_M3508s[0].outCurrent / Current\_Sum;

p\_M3508s[1].outCurrent = CurrentLimit \* p\_M3508s[1].outCurrent / Current\_Sum;

p\_M3508s[2].outCurrent = CurrentLimit \* p\_M3508s[2].outCurrent / Current\_Sum;

p\_M3508s[3].outCurrent = CurrentLimit \* p\_M3508s[3].outCurrent / Current\_Sum;

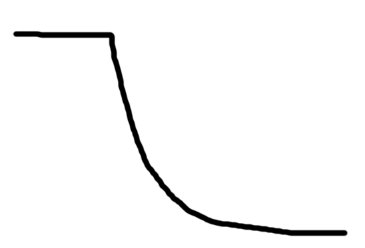
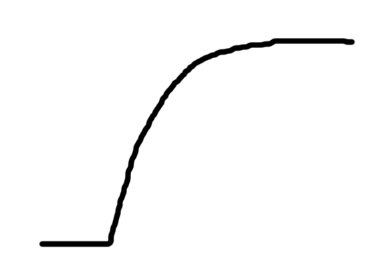
}

}

**速度缓冲**

对底盘速度进行闭环控制之后，轮子速度可以瞬间达到目标值，效果还是不错的。但是车在地上跑得时候，如果电机转速一下子从0到一个比较大的值，轮子肯定会打滑，如果在比较粗糙的地面上跑，对轮子的伤害是很大的。毕竟车有惯性，也无法一下子到达那么快的速度。所以有必要对速度进行一个缓慢上升的处理。

目前，我使用的是 **out = out + Ratio\*(in - out)** 这样一条算式，他可以使输入突变时以下图曲线那样上升、下降。Ratio的大小决定了曲线的斜率。(Ratio的范围：0~1)。



图，上升曲线，下降曲线

这条算式大概意思就是按照输入输出相差的大小，按照比例加到输出上。当相差得比较大的时候会快速上升（下降），相差比较小的时候会缓慢上升（下降）。因为要进行输入输出的运算，所以in和out不能是同一个变量。

函数示例如下

void Filter\_IIRLPF(float \*in,float \*out, float Ratio)

{

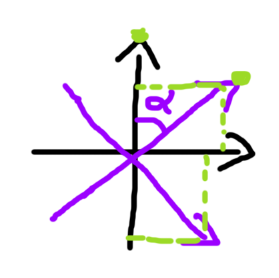
\*out = \*out + Ratio \*(\*in - \*out);

}

虽然这种方法和上述想实现的缓慢加速意思不一样，但当系数比较小的时候，还是有一点缓冲的效果。当然也可采用其他方法，如匀加速法等。

**底盘跟随云台视角移动**

比赛时操作手看的是第一视角，我们底盘的运动方向应与视角方向相同（如图传朝着正左方然后按前进底盘左平移，而图传看着是在前进）虽然这样可以使底盘跟随视角移动，但也要及时把底盘旋正，不然通过性会变差，而且底盘移动的能量损耗也会加大。



这个只要通过类似上图的速度分解就可实现

参考代码如下

//获取现在云台yaw轴相对车头的角度值并转化为弧度值

float YawRad = Cloud\_getYawAngleWithCenter()\*DEG\_TO\_RAD;

//先保存一下Vx的值，因为算Vy的时候还要用到

float temp\_Vx = Chassis\_Vx;

//按照现在角度计算在相对视角Vx的实际速度

Chassis\_Vx = Chassis\_Vx \* cos(YawRad) - Chassis\_Vy \* sin(YawRad);

//按照现在角度计算在相对视角Vy的实际速度

Chassis\_Vy = Chassis\_Vy \* cos(YawRad) + temp\_Vx \* sin(YawRad);

**底盘回中（车头方向与视角方向同向）**

在底盘静止时，如果云台Yaw相对于车头有一定角度可以不用让底盘自旋回中，这样可以减少底盘移动对瞄准射击的影响。当然，如果Yaw超过一定角度就可以让底盘稍微旋转一丢丢，不至于视角被限死。

在底盘运动时，通过Yaw轴电机的机械角度可使底盘自旋回中。（云台有IMU跟随的情况下）。

用个PID使底盘自旋至云台Yaw机械角度等于Cloud\_Yaw\_Center即可

if ((abs(M6623s[0].totalAngle - Cloud\_Yaw\_Center) > 1024 ||

(abs(Vy) + abs(Vx))> 20.0f) && !Chassis.swingFlag)

{

Chassis.FollowtargetYawRaw =

Position\_PID(&Chassis.FollowYawAttitude\_pid,

Cloud\_Yaw\_Center, M6623s[0].totalAngle);

}

**扭腰模式**

扭腰模式的实现主要是云台有IMU跟随的情况下，底盘以设定的速度（Chassis.swingSpeed）转到云台到车头某个角度就反向转，这样不断循环即可。（扭腰时可暂时取消底盘回中功能。）

参考代码如下，把函数返回的值加到底盘的VOmega即可

float Chassis\_swing(void) {

static int dir = 1;

float VOmega = 0.0f;

//如果在扭腰的时候底盘在平动则减慢扭腰速度

if ((abs(Chassis.targetXLPF) + abs(Chassis.targetYLPF))> 200.0f) {

Chassis.swingSpeed = Chassis\_SwingSpeed\_Slow;

}

else {

Chassis.swingSpeed = Chassis\_SwingSpeed\_Normal;

}

//扭到相对于车头（±1000/22.75）度，换向

//左右扭的角度也可设不同的角度

if (M6623s[0].realAngle - Cloud\_Yaw\_Center > 1000) {

dir = -1;

}

else if (M6623s[0].realAngle - Cloud\_Yaw\_Center < -1000) {

dir = 1;

}

VOmega = Chassis.swingSpeed \* dir;

return VOmega;

}

关于底盘应该就这么多了吧，底盘主要还是要灵活和减少能量损耗。