技术报告

夏令营-高中一组

目 录

1. 机械开发

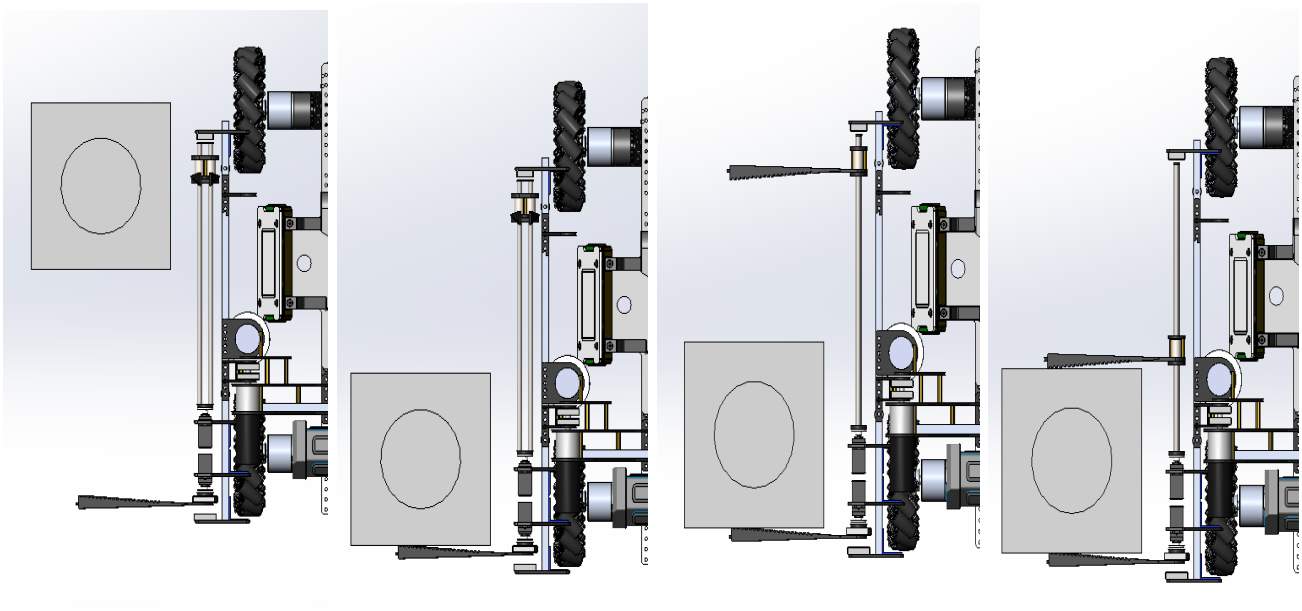
1.1 设计需求

根据比赛任务我们需要将放置在场地上的障碍块中得到子弹区到机器人弹舱中。步兵车本身的宽度为 430 左右，机器人宽度的限制为 600，场地上障碍块之间的宽度也是 600，我们想做两个同样的夹子放在机器人两侧来同时夹取两个障碍块。

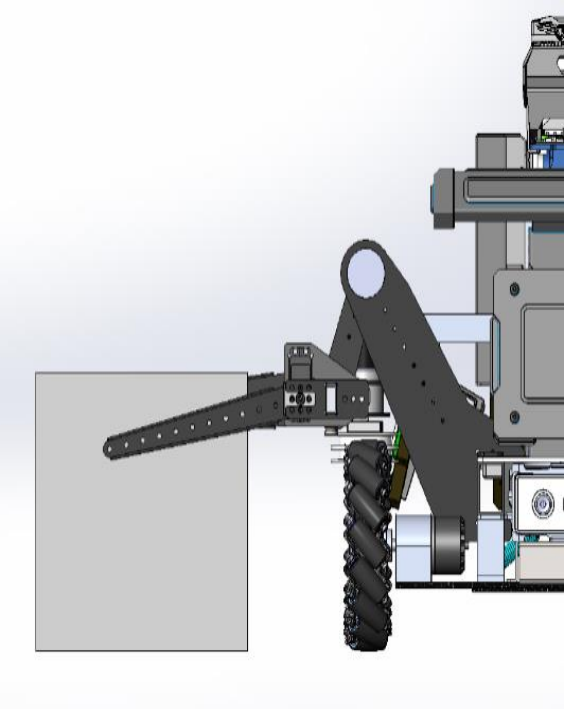
1.2　总体方案

根据设计需求，我们的夹子需要做成可以伸缩的，否则会超出尺寸限制。

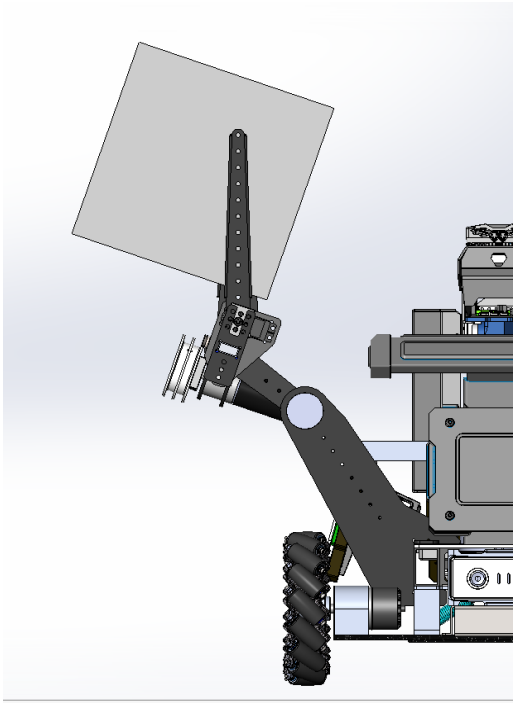
为了方便识别两侧的障碍块我们决定将夹子做成前后可以分别转动的，这样可以先放下后面的夹子，碰到障碍块之后将前半部分夹子放下，夹住障碍块。由于只靠夹子无法将地面上的障碍块中的子弹倒入弹舱中，还需要用一个臂将夹子整个抬起。



Step1　　Step2　　Step3　　Step4



在第一轮比赛时



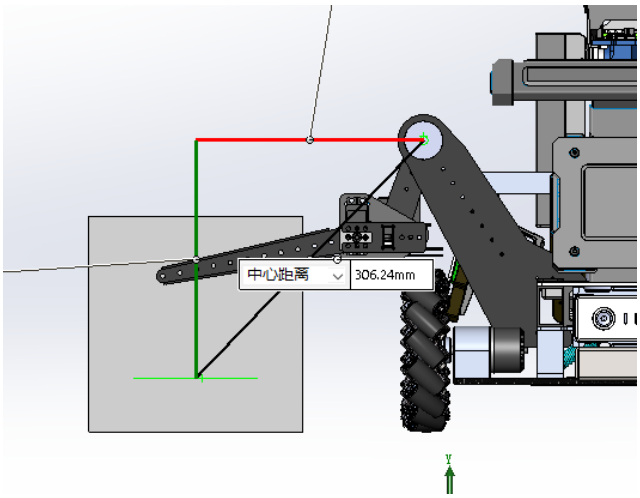
在总决赛比赛时

由于电机的程序没有调通以及钢丝绳很容易绷断，我们将前半部分夹子换成了用伺服驱动的可以在水平面上转动的夹子。

最初方案

最终方案

1.3 动力选型：



根据最初的方案，两侧的臂上的伺服只负责转动夹子，保证在抬起方块时不与大臂发生相对转动即可。比赛开始时下发给每个队的两种伺服在 4.8V 电压下扭矩都大于 10kg • cm，可以达到要求。

拉动夹子前面部分的电机需要对钢丝绳提供 20~30N 的拉力，用 0.3s 完成从光轴最前端移动到最后端的动作，也就是说速度要达到 1m/s，功率需要在 30W 以上。抬起大臂的电机需要在 0.5s 内将障碍块抬离地面 0.5m，将障碍块的质量和我们估计的大臂重量带入 计算后电机需要的功率在 35W 左右，所需扭矩在 40kg • cm 左右。

最后我们在淘宝上选择了一种 24V40W 的无刷电机，配了两种不同的齿轮箱，空载转速分别是 295rpm 和 80rpm，额定最大扭矩分别是 16kg • cm 和 60kg • cm。

在改动方案后由于去掉了转动线轴的电机，需要用伺服提供夹子对障碍块的压力，我们选用了 PDI-HV5932MG-300 数字舵机，这种舵机在 4.8V 电压下扭矩为 17.2kg •• cm。经试验后的确可以夹起障碍快。

2. 嵌入式开发

本组的嵌入式开发建立在清楚明确源代码的各个部分作用，程序结构与逻辑上，以丁致远为核心用三天时间确定了对源程序结构初步的理解，对 stm32f427 开发板板级资源的初步理解以及与初步的机械结构相对应的硬件控制方案，传感器设置，确定了分工。

\MDK-ARM\app\

sys.h

```
#define CHASSIS_CAN hcan1
#define ZGYRO_CAN hcan2
#define CHASSIS_ZGYRO_CAN hcan1
#define GIMBAL_CAN hcan1

#define DBUS_HUART huart1
//for dji remote controler reciever
#define JUDGE_HUART huart3
//connected to judge system
#define CV_HUART huart6
//connected to manifold/TXone
```

\bsp\

mytype.h

```
typedef uint8_t u8;
typedef uint16_t u16;
typedef uint32_t u32;

typedef int8_t s8;
typedef int16_t s16;
typedef int32_t s32;

typedef volatile uint8_t vu8;
typedef volatile uint16_t vu16;
typedef volatile uint32_t vu32;

typedef volatile int8_t vs8;
typedef volatile int16_t vs16;
typedef volatile int32_t vs32;

typedef unsigned char u8;
typedef unsigned short u16;
typedef unsigned long u32;

typedef signed char s8;
typedef signed short s16;
typedef signed long s32;

typedef volatile unsigned char vu8;
typedef volatile unsigned short vu16;
typedef volatile unsigned long vu32;

typedef volatile signed char vs8;
typedef volatile signed short vs16;
typedef volatile signed long vs32;
```

bsp_can.c

```
CAN_TxGimbal_ID = 0x1FF,
CAN_YAW_FEEDBACK_ID = 0x205
CAN_PIT_FEEDBACK_ID = 0x206,
CAN_trigger_FEEDBACK_ID = 0x207,
CAN_ZGYRO_RST_ID = 0x404,
CAN_ZGYRO_FEEDBACK_MSG_ID = 0x401,
CAN_ZGYRO_CHASSIS_MSG_ID = 0x402,

CAN_MotorLF_ID = 0x041,
CAN_MotorRF_ID = 0x042,
CAN_MotorLB_ID = 0x043,
CAN_MotorRB_ID = 0x044,
CAN_4Moto_Target_Speed_ID = 0x046,
CAN_GyroRecev_ID = 0x011,
CAN_GyroReset_ID = 0x012,

CAN_3510MotoAll_ID = 0x200,
CAN_3510Moto1_ID = 0x201,
CAN_3510Moto2_ID = 0x202,
CAN_3510Moto3_ID = 0x203,
CAN_3510Moto4_ID = 0x204,
CAN_DriverPower_ID = 0x80,

CAN_HeartBeat_ID = 0x156,
```

main.c

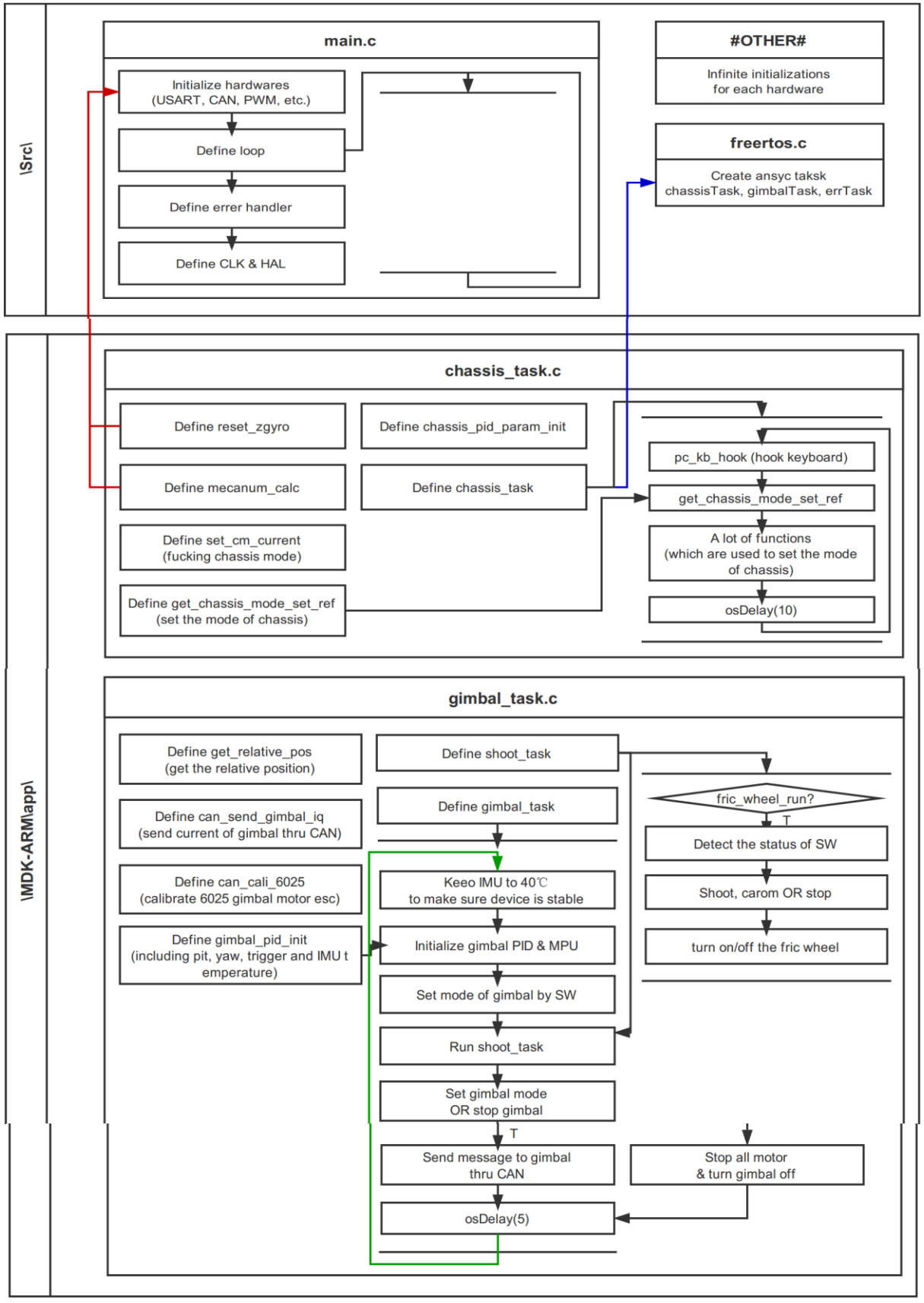
Receive data thru UART from DBus, Bluetooth, judging system, Manifold, etc.

calibrate.c

calibrate gimbal_offset, imu_data, and save these calibration data in flash

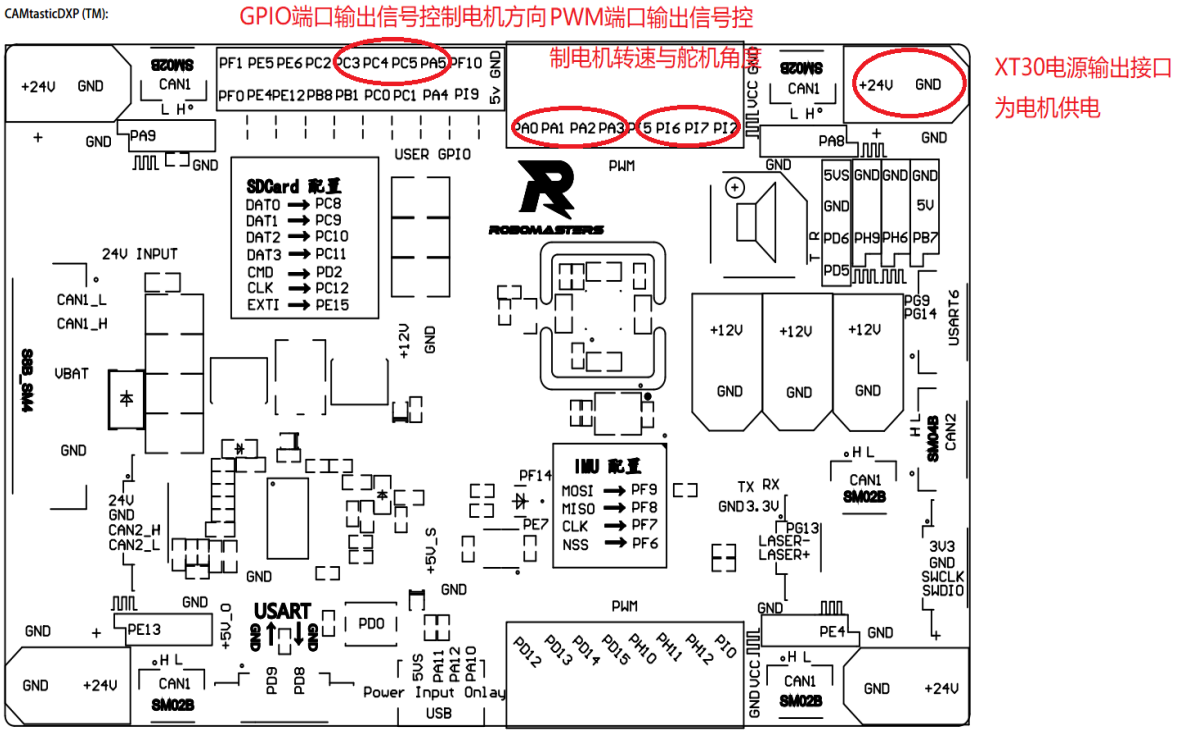
mpu.c

Config MPU6500 and read the data from acce lerator& gyrometer using SPI interface


\Src\
main.c
Initialize hardwares
(USART, CAN, PWM, etc.)
Define loop
Define errer handler
Define CLK & HAL
#OTHER#
Infinite initializations
for each hardware
freertos.c
Create ansyc taksk
chassisTask, gimbalTask, errTask
\MDK-ARM\app\
chassis_task.c
Define reset_zgyro
Define chassis_pid_param_init
Define mecanum_calc
Define chassis_task
Define set_cm_current
(fucking chassis mode)
Define get_chassis_mode_set_ref
(set the mode of chassis)
pc_kb_hook (hook keyboard)
get_chassis_mode_set_ref
A lot of functions
(which are used to set the mode
of chassis)
osDelay(10)
gimbal_task.c
Define get_relative_pos
(get the relative position)
Define can_send_gimbal_iq
(send current of gimbal thru CAN)
Define can_cali_6025
(calibrate 6025 gimbal motor esc)
Define gimbal_pid_init
(including pit, yaw, trigger and IMU t
emperature)
Define shoot_task
Define gimbal_task
Keeo IMU to 40℃
to make sure device is stable
Initialize gimbal PID & MPU
Set mode of gimbal by SW
Run shoot_task
Set gimbal mode
OR stop gimbal
T
Send message to gimbal
thru CAN
osDelay(5)
fric_wheel_run?
T
Detect the status of SW
Shoot, carom OR stop
turn on/off the fric wheel
Stop all motor
& turn gimbal off

2.1 pwm 通信控制

本组的初始电控方案确定使用四个以 pwm 信号与 gpio 信号同时控制功率与方向的电机（motor），四个以 pwm 信号控制的舵机。stm32f427 开发板板载的 pwm 信号与 gpio 信号通信接口能完全提供电控的需要。我组一号方案对板级资源的利用如下图 1.1 所示。



接口引脚标注图

图 1.1 电控方案一端口使用

查阅 《RM 开发板用户手册》*2 《STM32F427xx STM32F429xx 数据手册》*3 后，我理解了 PWM 信号在 stm32 开发板上的输出方式，是以定时器输出数字控制的高低电平的模拟信号，请教了罗一骏导师后，我学会了在源程序中调整定时器输出 PWM 信号的的参数以及如何调用这些参数进行信号输出。下图 1.2 与 1.3 为电控方案一测试时的 PWM 信号参数调整与调用程序，及 PWM 信号在 stm32 上的输出与调用逻辑。

```
//定义变量
int         rotate_motor;//use shift to reverse and use ctrl to brake
int   arm_motor=1;
int   rotate_motor_stop = 1;
int   arm_motor_stop = 1;
                                    int    arm_lift = 0; //arm, both left and right
int         elbow_lift = 0;//elbow, both left and right; edit by Tiger
```

```
//Lizheng Liu editing begins, turn off/on arm

        if (rc.kb.bit.X && !rc.kb.bit.SHIFT)
                {
                        arm_lift = 1;
                }
        if (rc.kb.bit.X && rc.kb.bit.SHIFT)
                {
                        arm_lift = 0;
                }
        if (rc.kb.bit.C && !rc.kb.bit.SHIFT)
                {
                elbow_lift = 1;
                }
        if (rc.kb.bit.C && rc.kb.bit.SHIFT)
                {
                        elbow_lift = 0;
                }

        if (arm_lift == 1)//(rc.sw1 == 1)
                {

                        s1 = 1400;//1400
                        s3 = 1800;//1800


                }
        else
                {


                        s1 = 2300;//2300  RF
                        s3 = 900;//900  LF


                }
```

```
if (elbow_lift == 1)//(rc.sw1 == 3)
                {

                        s2 = 1700;//1700 RB
                        s4 = 1400;//1400 LB


                }
        else
                {

                        s2 = 500;//500
                        s4 = 2300;//2300


                }

        if (rc.kb.bit.V && !rc.kb.bit.SHIFT
&& !rc.kb.bit.CTRL && !rc.kb.bit.Q)
                {
                        arm_motor_stop=0;
                        osDelay(500);
                        arm_motor = 1;
                }
        if (rc.kb.bit.V && rc.kb.bit.SHIFT
&& !rc.kb.bit.CTRL)
                {
                        arm_motor = 0;
                }
        if (rc.kb.bit.B && !rc.kb.bit.SHIFT
&& !rc.kb.bit.CTRL && !rc.kb.bit.Q)
                {
                        rotate_motor_stop=0;
                        osDelay(500);
                        rotate_motor = 1;
                }
        if (rc.kb.bit.B && rc.kb.bit.SHIFT
&& !rc.kb.bit.CTRL)
                {
```

```
if (arm_motor == 1)
		{


	HAL_GPIO_WritePin(GPIOA,
GPIO_PIN_5, GPIO_PIN_SET);
			s5 = 400;

	HAL_GPIO_WritePin(GPIOC,
GPIO_PIN_4, GPIO_PIN_RESET);
			s7 = 400;

		}
	else {


	HAL_GPIO_WritePin(GPIOA,
GPIO_PIN_5, GPIO_PIN_RESET);
			s5 = 400;

	HAL_GPIO_WritePin(GPIOC,
GPIO_PIN_4, GPIO_PIN_SET);
			s7 = 400;

		}
	if (arm_motor_stop == 1){
			//maybe need direction
function

	HAL_GPIO_WritePin(GPIOA,
GPIO_PIN_5, GPIO_PIN_SET);
			s5 = 100;

	HAL_GPIO_WritePin(GPIOC,
GPIO_PIN_4, GPIO_PIN_SET);
			s7 = 100;

		}

	if (rotate_motor == 1)
		{


	HAL_GPIO_WritePin(GPIOC,
GPIO_PIN_5, GPIO_PIN_SET);
			s6 = 200;

	HAL_GPIO_WritePin(GPIOC,
GPIO_PIN_3, GPIO_PIN_RESET);
			s9 = 200;
```

```
TIM2->CCR1 = s1;//LF
	TIM2->CCR2 = s2;//LB
	TIM2->CCR3 = s3;//RF
	TIM2->CCR4 = s4;//RB

	TIM8->CCR1 = s5;
	TIM8->CCR2 = s6;
	TIM8->CCR3 = s7;
	TIM8->CCR4 = s9;

	//涵道

	s_duct = rc.ch2 * 150/66 + 1000;
	TIM5->CCR1 = s_duct;



	//Lizheng Liu editing ends
```

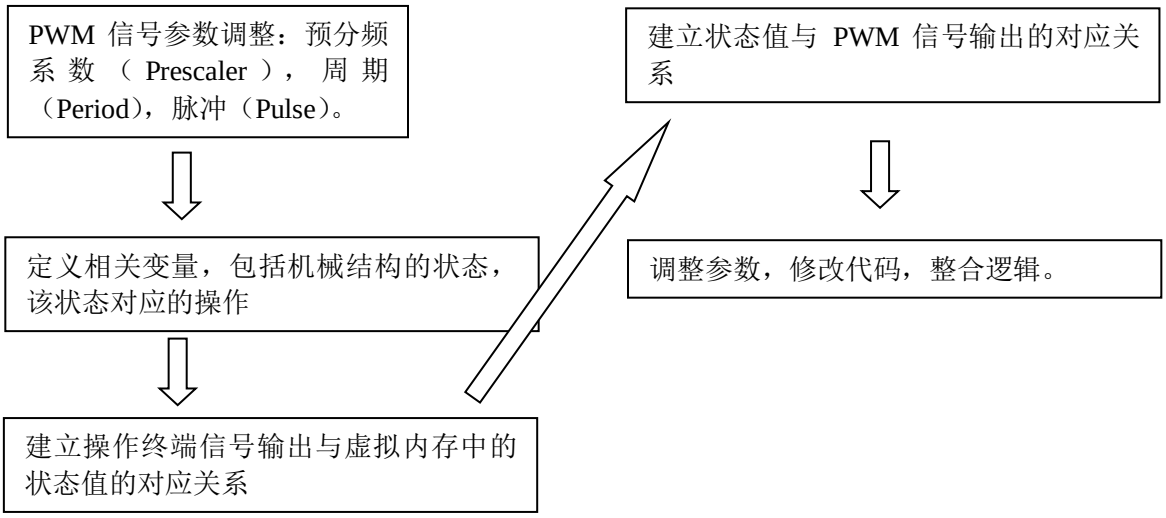
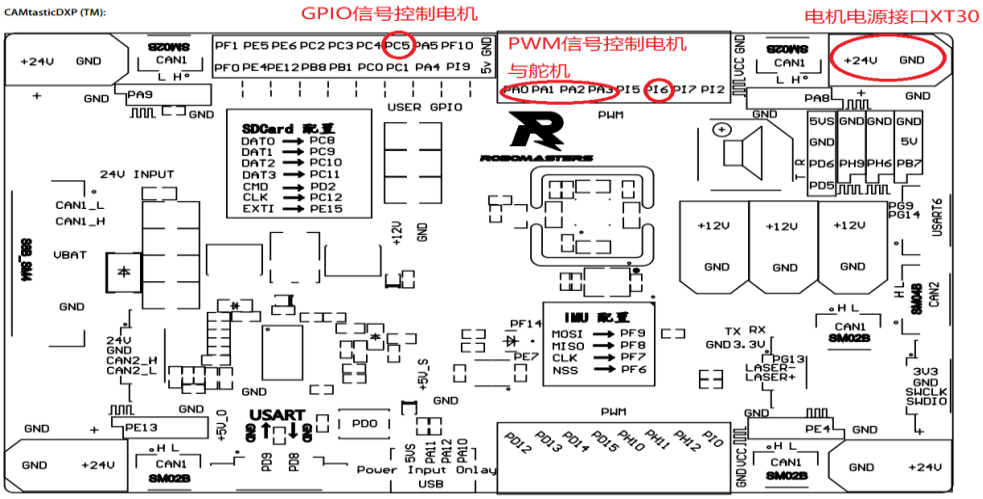


图 1.3 电控方案一程序操作逻辑

嵌入式小组确定了控制舵机与电机的软件与硬件方案后，开始了耗时两天的调参与 debug。在配合机械组进行机械臂运动的测试时，主要出现的问题有以下几种：PWM 接口反接导致无信号输出，信号线损坏导致信号丢失，程序逻辑混乱导致操作不灵敏。在调试过程中，我们使用万用表，示波器来检测 PWM 信号是否正确输出，使用 KEIL 开发软件的控制面板进行调参，在出现预期值与实际值不符合的情况时，我们采用一步步排查的方式，以控制面板输出测试 ，通信线测试，通信接口测试，舵机测试的顺序排除软件与硬件的错误

虽然电控方案一由于设置了太多需要用 PWM 信号控制的电机，最终导致调参的难度以及 debug 的复杂性。嵌入式组根据简化的新机械结构设置了电控方案二使用更少的电机，完成同样的任务，给开发板虚拟资源与电路的压力相应减小，调试的难度也降低不少。电控方案二的板级资源利用图如下。

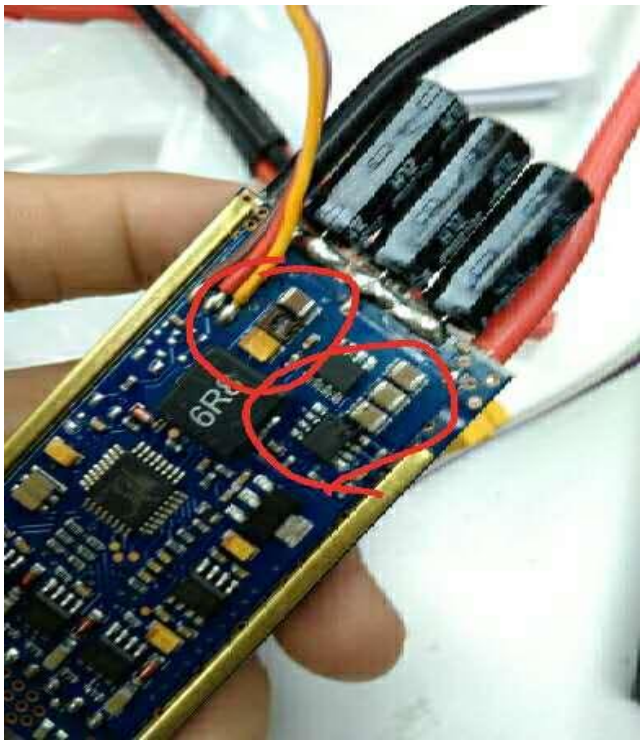


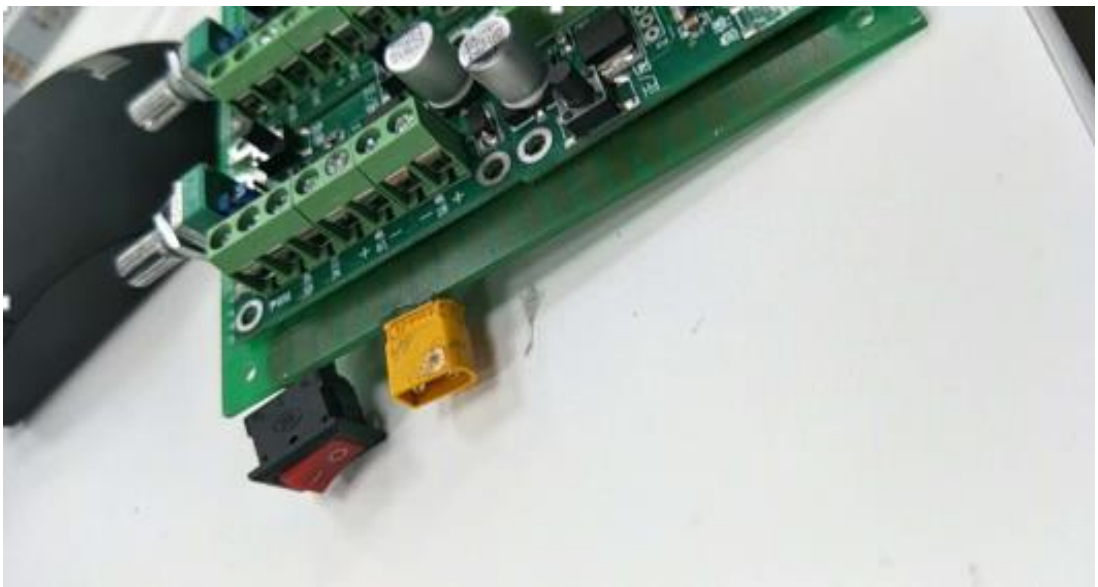
接口引脚标注图

图 1.3 电控方案二板级资源利用图

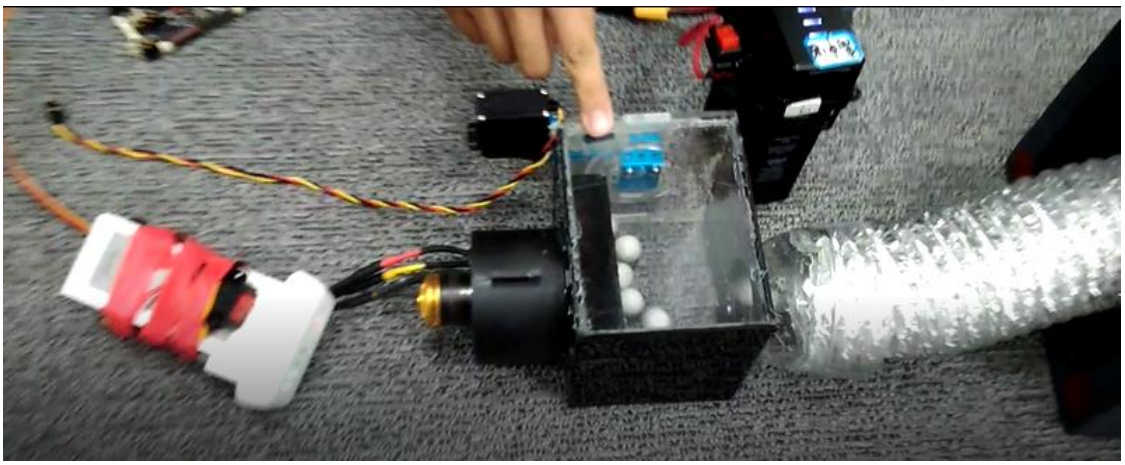
2.2 硬件改进

考虑到机械臂的动力部分较多，负载较大。起初为了方便控制及减轻步兵车体底盘电池的压力，我制作了一个由4个主动力减速电机的电调，一个降压模块，一个涵道电调组成的电机涵道控制板，独立DJI电源供电，由STM32主控共地后通过PWM信号控制。





在第一次测试赛中，机械部分的方案一机械臂设计出现了一些故障，其中也有程序上尚未解决的漏洞，于是我将之前的未调试好的涵道解决了，作为备选方案随时等待替换可能出现故障的机械臂，考虑到其负载过大的问题，我们也用了独立电源供电，其电调由主控发出 PWM 控制。



2.3 夹障碍块自动部分

（1）行程开关

行程开关遇到障碍块开始做任务，如果有意外，可以通过按键退出这个模式，或单独控制机械臂上的电机做某个动作。

```
gYaw.last_mode = gYaw.ctrl_mode;
switch (rc.sw2)
{
  case (RC_UP):
  {
    if (gYaw.ctrl_mode != GIMBAL_INIT)
      gYaw.ctrl_mode = GIMBAL_CLOSE_LOOP_ZGYRO;

    /* gimbal back to center, can modify pid parameter to make gimbal slow in this model */
    if (gYaw.last_mode == GIMBAL_RELAX && gYaw.ctrl_mode != GIMBAL_RELAX)
      gYaw.ctrl_mode = GIMBAL_INIT;
  }
  break;
  default:
    gYaw.ctrl_mode = GIMBAL_RELAX;
  break;
}

if (rc.kb.bit.Q)
{
  ChuPengMode=1;
}

if (rc.kb.bit.Q && rc.kb.bit.SHIFT)
{
  ChuPengMode=0;
  bChassisStop=0;
}

if  (ChuPengMode==1)
{
  bL=arduino_buff[6];
  bR=arduino_buff[7];
}

if  (anjian)
{
  taskfen;
}

if  (ChuPengMode==1 && (bL || bR) )
{
  bChassisStop=1;
  if  (1)
  {
    TASK;
    if (rc.kb.bit.Q && rc.kb.bit.SHIFT)
    {
      ChuPengMode=0;
      bChassisStop=0;
      break;
    }
    break;
  }
```

```
if (rc.kb.bit.Q && rc.kb.bit.SHIFT)
{
  ChuPengMode=0;
  bChassisStop=0;
}

if  (ChuPengMode==1)
{
  bL=arduino_buff[6];
  bR=arduino_buff[7];
}

if  (anjian)
{
  taskfen;
}

if  (ChuPengMode==1 && (bL || bR) )
{
  bChassisStop=1;
  if  (1)
  {
    TASK;
    if (rc.kb.bit.Q && rc.kb.bit.SHIFT)
    {
      ChuPengMode=0;
      bChassisStop=0;
      break;
    }
    break;
  }
  bChassisStop =0;

}
shoot task();
```

（2）pid 循迹的程序。利用地上的白线，车上装有两个返回模拟量的灰度传感器。用灰度传感器循迹，车稳定的走一根直线，结合上面行程开关的功能。功能：通过按键开启这个模式后，车还是用遥控器控制，遇到白线自动开始循迹（这时不能通过遥控器控制车）。

//循迹的伪代码。程序在打不开的 3 号电脑上

```
LightValue = read light sensor;
error = LightValue - offset;
integral = integral + error;
derivative = error - lastError;
Turn = Kp*error + Ki*integral +  Kd*derivative;
```

利用白线和车上的两个灰度传感器，不方便用 pid 循迹的情况下使车完全垂直于白线。因为白线和场地（灰）之间，同一个灰度值会有一个比较固定的相对位置，所以可以利用这个，利用车左右两个灰度传感器，将车垂直于白线，停下抓取障碍块。

```
errorLeft = -1 * （sensorLeft - offest）;
errorRight = -1 * （sensorRight - offest）;
TurnLeft = Kp*errorLeft;
TurnRight = Kp*errorRight;
```

2.4 单独控制程序

云台单独控制，底盘会每 8 秒换一个随机数作为底盘需要转到的角度，效果就是底盘在原地不停转动，云台可以单独控制瞄准对方的步兵。底盘不停转动可以减少被击中的次数。

云台单独控制，底盘角度单独控制，底盘前后左右单独控制。目的同上，改进版。

```
void get_chassis_mode_set_ref(RC_Type *rc)
{
  chassis.last_mode = chassis.mode;

  switch (rc->sw1)
  {
    case RC_UP:
      chassis.mode = CHASSIS_AUTO;  //senior students no use
    break;

    case RC_MI:
      //chassis.mode = CHASSIS_CLOSE_GYRO_LOOP;
      chassis.mode = CHASSIS_FOLLOW_GIMBAL; //
    break;
  //gai-Webb begin
    case RC_DN:
      chassis.mode = CHASSIS_OPEN_LOOP;  //senior students no use
    break;
  //gai-Webb end
    default:
    break;
  }
```

```
while (1)
{
  pc_kb_hook();

  get_chassis_mode_set_ref(&rc);


  switch (chassis.mode)
  {
    case CHASSIS_FOLLOW_GIMBAL:
    {
      chassis.vy = rc.ch2 * CHASSIS_RC_MOVE_RATIO_Y + km.vy * CHASSIS_PC_MOVE_RATIO_Y;
      chassis.vx = rc.ch1 * CHASSIS_RC_MOVE_RATIO_X + km.vx * CHASSIS_PC_MOVE_RATIO_X;
    }break;
//gai-Webb begin
    case CHASSIS_OPEN_LOOP:
    {
      chassis.vy = rc.ch2 * CHASSIS_RC_MOVE_RATIO_Y + km.vy * CHASSIS_PC_MOVE_RATIO_Y;
      chassis.vx = rc.ch1 * CHASSIS_RC_MOVE_RATIO_X + km.vx * CHASSIS_PC_MOVE_RATIO_X;
      chassis.vw = rc->ch3;
    }
//gai-Webb end
    default:
    {
      chassis.vy = 0;
      chassis.vx = 0;
    }break;
  }

  if (gYaw.ctrl_mode == GIMBAL_CLOSE_LOOP_ZGYRO)
  {
    chassis.vw = pid_calc(&pid_chassis_angle, yaw_relative_pos, 0);
  }
  else
  {
    chassis.vw = 0;
  }
```

```
42              /* yaw arrive and switch gimbal state */
43              gYaw.ctrl_mode = GIMBAL_CLOSE_LOOP_ZGYRO;
44
45              gYaw.zgyro_offset = yaw_zgyro_angle;
46              pit_angle_ref = 0;
47              yaw_angle_ref = 0;
48            }
49          }
50        }break;
51        //gai-Webb begin
52        case GIMBAL_CLOSE_LOOP_ZGYRO:
53        {
54          self_ref_w = yaw_zgyro_angle - gYaw.zgyro_offset - yaw_relative_angle;
55
56          switch (chassis.mode)
57          {
58            case CHASSIS_OPEN_LOOP:
59            {
60              pit_angle_fdb = pit_relative_angle;
61              pit_angle_ref += rc.mouse.y * -0.01f;
62
63              yaw_angle_fdb = yaw_zgyro_angle - gYaw.zgyro_offset;
64              yaw_angle_ctr += -rc.mouse.x * 0.01f;
65              yaw_angle_ref = (rc.kb.bit.SHIFT ? self_ref_w : yaw_angle_ctr);
66
67            }break;
68            //begin case GIMBAL_CLOSE_LOOP_ZGYRO:
69            case (CHASSIS_FOLLOW_GIMBAL):
70            {
71              pit_angle_fdb = pit_relative_angle;
72              pit_angle_ref += rc.ch4 * 0.003 - rc.mouse.y * 0.01f;
73
74              yaw_angle_fdb = yaw_zgyro_angle - gYaw.zgyro_offset;
75              yaw_angle_ctr += -rc.ch3 * 0.0015f - rc.mouse.x * 0.01f;
76              yaw_angle_ref = yaw_angle_ctr;
77            }break;
78            //end
79            default:
80            {
81              pit_angle_fdb = pit_relative_angle;
82
83              yaw_angle_fdb = yaw_zgyro_angle - gYaw.zgyro_offset;
84            }
85            break;
86          }
87        }
88        break;
89        //gai-Webb end
90        default:
91        break;
92      }
93
```

在不用自动的情况下，操作手怎样知道步兵的机械臂有没有到达可以夹取障碍块的位置。

摄像头

行程开关，LED

在相机图传模块旁边装一个 LED，形成开关被按下时，LED 亮，操作手就可以通过屏幕知道行程开关是否被按下，也就是步兵有没有到达夹取障碍块的位置。

调云台双环 PID 的参数，射频，射速

```
/**
  * @brief     initialize gimbal pid parameter, such as pitch/yaw/trigger motor,
  *            imu temperature
  * @attention before gimbal control loop in gimbal_task() function
  */
void gimbal_pid_init(void)
{
  PID_struct_init(&pid_pit, POSITION_PID, 4000, 1000,
                  240.0f, 0, 0); //200.0f, 0.5, 0
  PID_struct_init(&pid_pit_speed, POSITION_PID, 4000, 1000,
                  53.0f, 0, 0); //20.0f, 0.2, 0.8f

  PID_struct_init(&pid_yaw, POSITION_PID, 9000, 400,
                  280, 0, 0); //180, 0.4, 150
  PID_struct_init(&pid_yaw_speed, POSITION_PID, 8000, 800,
                  22, 0, 0);  //20, d0.3, 20

  PID_struct_init(&pid_trigger, POSITION_PID, 10000, 2000, //position
                  15.0f, 0, 10);
  PID_struct_init(&pid_trigger_speed, POSITION_PID, 7000, 4000, //vw
                  1.5f, 0.3f, 5);

  PID_struct_init(&pid_imu_tmp, POSITION_PID, 1000, 300,
                  180, 0.1f, 0);
}


void shoot_task(void)
{
  */
void chassis_pid_param_init(void)
{
  for (int k = 0; k < 4; k++)
  {
    //max current = 20000
    PID_struct_init(&pid_spd[k], POSITION_PID, 10000, 1000, 4, 0.05f, 5.0f);
  }
  PID_struct_init(&pid_chassis_angle, POSITION_PID, 600, 80, 1.0f, 0.0f, 0.0f);
  pid_chassis_angle.max_err  = 60 * 22.75f; // err angle > 60 cut the output
  pid_chassis_angle.deadband = 35;
}
```