# 电子设计比赛“魑魅魍魉”组报告

2020 年 12 月 5 日

## 目录

# 1 队伍简介

**队伍名称：**魑魅魍魉

**队伍成绩：**三十二强（通过初试）

**小车图片：**

# 2 小车及技术资料

## 2.1 所用模块和器件

**小车所用器件:**

- 四驱车底板
- 稳压模块
- 1100mah电池
- 红外避障模块
- 面包板
- LED
- 电机驱动模块L298N
- 杜邦线
- STM32F103RCT6
- Zigbee无线串口收发模块
- 陀螺仪JY62

**调试所用器件:**

- TTL转串口
- HC-05蓝牙模块
- ST-LINK仿真器

## 2.2 整体设计思路

### 2.2.1 小车组成（硬件）设计

为了平衡和易于控制，我们采用了四轮车的设计，将两对电机固定于四驱车底板上，引出各个电机的驱动电压输入端以及编码器，单片机生成特定占空比的PWM波输入电机驱动模块，电机驱动模块进而根据接收到的信号，输出相应占空比的电压幅值为12V的PWM波，用来驱动电机运转。由于四个电机是相对独立的，通过设置各个电机对应PWM波的性质，就可以完成最基本的四轮驱动。

为了让小车根据周围环境和上位机提供的信息，做出下一步行进的判断，所以还需要加入传感器，我们添加了红外传感器、Zigbee以及陀螺仪，其中，Zigbee直接与上位机交流，通过调用zigbee库中的函数，获取病人、医院、物资的位置信息，陀螺仪可以获得小车转角的信息，用于转弯和控制直行，而红外传感用于获得场内黑线的信息，用于辅助陀螺仪进行控制，另外上面提到利用霍尔效应的编码器，每次车轮转过一个确定的角度，会返回一个脉冲，所以可以反映小车车轮的转速和行进距离，也是一个重要的传感器，用于形成闭环，从而可以采用PID算法，不断调控小车自身，防止出现较大偏差。

以上是硬件方面，我们的全部设计思路，考虑到公平性，我们没有采用赛事方提供物资之外的材料。

### 2.2.2 程序设计

首先，我们实现了四个轮子的单个驱动的函数，单个轮子的函数接受的是0-1000的整数，对应0-100%的占空比，如果是正数，则单个轮子前进，如是负数，那么单个轮子后退，下面展示的是右前轮的驱动函数，其他三个轮子与之相仿：

```
void RightF(int value)
{
        if(value>0)
        {
        __HAL_TIM_SetCompare(&htim2,TIM_CHANNEL_1,value);
        __HAL_TIM_SetCompare(&htim2,TIM_CHANNEL_2,0);
        }
        else
        {
        __HAL_TIM_SetCompare(&htim2,TIM_CHANNEL_1,0);
        __HAL_TIM_SetCompare(&htim2,TIM_CHANNEL_2,-value);
        }
}
```

将四个轮子单独的运行函数进一步打包，可以得到行进和转弯函数，如下：

```
float Forward(float i1,float i2)
{
        RightB((int)(i1*800+i2*200));
        RightF((int)(i1*800+i2*200));
        LeftB((int)(i1*800-i2*200));
        LeftF((int)(i1*800-i2*200));
}

void Turn(float m)
{
```

```
        RightB(m*800);
        RightF(m*800);
        LeftB(-m*800);
        LeftF(-m*800);
}
```

其中输入的参数由三个PID函数给出，其中i1控制直行速度，i2用于行进过程中调节保持直行，m用于调控转弯速度，相应地，三个PID函数分别用于行进给定距离，转弯，保持直行，具体代码如下：

```
float PIDturn(float error_sum1,float now1,float aim1)//转弯
{
    float error,error_last,output1;
    error=now1-aim1;
    output1=error+error_sum1+(error-error_last);
    error_sum1=++error;
    if(error>45)
       error=45;
    if(error<-45)
       error=-45;
    error_last=error;
    return output1;
}

float PIDforward(float error_sum2,float now2,float aim2)//保持直行
{
    int error,error_last;
    float output2;
    error=now2-aim2;
    output2=1.0*error+1.0*error_sum2+1.0*(error-error_last);
    error_sum2=++error;
    error_last=error;
    return output2;
}

float PIDstop(int now3,int aim3)//行进给定距离
{
    int error,error_last;
    float output3;
    error_sum3=0;
    now3=Bip;
    error=aim3-now3;
```

```
    output3=2.0*error/500+error_sum3*0.01/500+0.1*(error-error_last)
       /500;
    if(error>400)
       error=400;
    if(error_sum3>5000)
       error_sum3=5000;
    return output3;
}
```

而三个PID算法需要形成闭环，也就是需要传感器的信息输入，那么就需要处理传感器信号的函数，及记录并处理编码器脉冲数，处理陀螺仪信号的函数，而脉冲数与行进距离，轮胎转速成正比，所以可以直接使用，只需要一个记录的函数，并且需要时刻记录，所以在STM32CubeMX中设置了两个并行进程，其中一个专门用来记录轮胎编码器的脉冲数，另外一个则用于执行最主要的决策和行进函数，记录脉冲数的函数如下：

```
Bip=0;
int a=0,b=0;

for(;;)
{
   b=a;
   a=HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_11);
   if(a!=b)
   {
     Bip+=1;
   }
}
```

而陀螺仪输出信号需要经过一定处理才能转化为可用的角度信息（该函数输出为相对于某一特定方向的绝对角度），处理函数如下：

```
float angle()
{
   static unsigned char ucRxBuffer[250];
   float a;
   int i=0;
   HAL_UART_Receive(&huart1,(uint8_t *)ucRxBuffer,20,0xFFFF);
   for(i=0;i<250;i++)
   {
      if(ucRxBuffer[i]==0x55&&ucRxBuffer[i+1]==0x53)
      {
         a=((short)(ucRxBuffer[i+7]<<8|ucRxBuffer[i+6]))/32768.0*180;
```

```
      }
    }
    return a;
}
```

至此，上面的行进和转弯函数都已经完整可调用，进一步需要打包一个接收决策函数输出并执行该输出对应的指令的函数，经过商讨，我们认为决策函数与执行函数之间通过字符串沟通比较方便，字符串由“字母+数字”的序列对组成，字母有四个，为F、B、L、R，分别表示相对于比赛场地中的绝对坐标的前后左右，而后面紧跟的数字表示直行几格，由此，得到执行函数如下：

```
char direction[30];
void Run()
{
    for(i=0; direction[i]!='\0'; i=i+2)
    {
        if(direction[i]=='L')
        {
            int o,num=0;
            o=(int)direction[i+1]-(int)'0';
            aim1=90+A;
            aim3=o*B;
            float error_sum1=0.0,error_sum2=0.0,now1;
            int error_sum3=0;
            now1=angle();
            for(; now1-aim1>0.5||now1-aim1<-0.5;)
            {
                now1=angle();
                Turn(PIDturn(error_sum1,now1,aim1));
            }
            Bip=0;
            aim2=90+A;
            for(;;)
            {
                now2=angle();
                now3=Bip;
                RunForward(PIDstop(now3,aim3),PIDforward(error_sum2,
                    now2,aim2));
            }
        }
        if(direction[i]=='R')
        {
```

```
            int o,num=0;
            o=(int)direction[i+1]-(int)'0';
            aim1=-90+A;
            aim3=o*B;
            float error_sum1=0.0,error_sum2=0.0,now1;
            int error_sum3=0;
            for(; now1-aim1>0.5||now1-aim1<-0.5;)
            {
                now1=angle();
                Turn(PIDturn(error_sum1,now1,aim1));
            }
            bbip=Bip;
            aim2=-90+A;
            for(; num;)
            {
                now2=angle();
                now3=bbip-Bip;
                RunForward(PIDstop(error_sum3,now3,aim3),PIDforward(
                    error_sum2,now2,aim2));
            }
        }
        if(direction[i]=='F')
        {
            int o,num=0;
            o=(int)direction[i+1]-(int)'0';
            aim1=A;
            aim3=o*B;
            float error_sum1=0.0,error_sum2=0.0,now1;
            int error_sum3=0;
            for(; now1-aim1>0.5||now1-aim1<-0.5;)
            {
                now1=angle();
                Turn(PIDturn(error_sum1,now1,aim1));
            }
            bbip=Bip;
            aim2=A;
            for(; num;)
            {
                now2=angle();
```

```
                now3=bbip-Bip;
                RunForward(PIDstop(error_sum3,now3,aim3),PIDforward(
                    error_sum2,now2,aim2));
            }
        }
        if(direction[i]=='B')
        {
            int o,num=0;
            o=(int)direction[i+1]-(int)'0';
            aim1=-A;
            aim3=o*B;
            float error_sum1=0.0,error_sum2=0.0,now1;
            int error_sum3=0;
            for(; now1-aim1>0.5||now1-aim1<-0.5;)
            {
                now1=angle();
                Turn(PIDturn(error_sum1,now1,aim1));
            }
            bbip=Bip;
            aim2=-A;
            for(; num;)
            {
                now2=angle();
                now3=bbip-Bip;
                RunForward(PIDstop(error_sum3,now3,aim3),PIDforward(
                    error_sum2,now2,aim2));
            }
        }
        if(direction[i]=='\0')
        {
            StopRightNow();
        }
    }
}
```

其中direction[]即为传递信息的字符串。那么，最后剩下的就是根据上位机提供的物资、病人、医院位置，以及目前的小车位置，生成行进命令字符串的决策函数HEART()以及其他相关的打包函数，如下：

```
int map[6][6][4] = {0}; //为无障碍物，00,,, U1D3L4R
int Road[6][6] = {0};    //代表没走过，代表走过，因为走过的说明之前更短就能到达，就
```

不必继续了01

```
int Value[6][6] = {0};
char operationTotal[30];
char direction[30];
int ui = 0;
int u = 1;
typedef struct node
{
    struct node *last;
    char dir; //从上一节点到此节点的方向
    int x;
    int y;
    struct node *next;
} Node;
Node *NodeSon[4]; //0U,1D,2L,3R
typedef struct queue
{
    Node *front;
    Node *rear;
} Queue;

Queue *RoadFinder;
Node *hospital;

int getValue(int x, int y, int V)
{
    Value[x][y] = V;
}

void InitializeQueue(Queue *pq, Node *start)
{
    pq->front = start;
    pq->rear = start;
    pq->front->next = pq->rear;
}

void Roadzero()
{
    for (int i = 0; i < 6; i++)
        for (int j = 0; j < 6; j++)
```

```
            Road[i][j] = 0;
}

bool EnQueue(Node *pnew, Queue *pq)
{
    pnew->next = NULL;
    pq->rear->next = pnew;
    pq->rear = pnew;
    return true;
}

Node *DeQueue(Queue *pq)
{
    Node *pt;
    pt = pq->front;
    if (pq->front != pq->rear)
        pq->front = pq->front->next;
    return pt;
}

void EmptyQueue(Node *start)
{
    Node *node2;
    while (start->next != NULL)
    {
        node2 = start->next;
        if (start != hospital)
            free(start);
        start = node2;
    }
    free(RoadFinder);
}
void Emptymiddle(Node *start, Node *end)
{
    Node *node2 = start;
    node2 = node2->next;
    while (node2 != NULL)
    {
        start = node2;
```

```
        node2 = start->next;
        if (start != hospital)
            free(start);
    }
    free(RoadFinder);
}
Node **CNB(Node *temp) // Create Node Branch
{
    if (temp->x > 0 && map[temp->x][temp->y][0] == 0 && Road[(temp->x)
        - 1][temp->y] != 1) //向上能走且没走
       过
    {
        NodeSon[0] = (Node *)malloc(sizeof(Node));
        NodeSon[0]->last = temp;
        NodeSon[0]->dir = 'U';
        NodeSon[0]->x = (temp->x) - 1;
        NodeSon[0]->y = (temp->y);
        Road[temp->x - 1][temp->y] = 1;
    }
    else
    {
        NodeSon[0] = NULL;
    }

    if (temp->x < 5 && map[temp->x][temp->y][1] == 0 && Road[(temp->x)
        + 1][temp->y] != 1) //向下能走且没走
       过
    {
        NodeSon[1] = (Node *)malloc(sizeof(Node));
        NodeSon[1]->last = temp;
        NodeSon[1]->dir = 'D';
        NodeSon[1]->x = (temp->x) + 1;
        NodeSon[1]->y = (temp->y);
        Road[temp->x + 1][temp->y] = 1;
    }
    else
    {
        NodeSon[1] = NULL;
    }
```

```
    if (temp->y > 0 && map[temp->x][temp->y][2] == 0 && Road[temp->x][
       temp->y - 1] != 1) //向下能走且没走
       过
    {
        NodeSon[2] = (Node *)malloc(sizeof(Node));
        NodeSon[2]->last = temp;
        NodeSon[2]->dir = 'L';
        NodeSon[2]->x = temp->x;
        NodeSon[2]->y = temp->y - 1;
        Road[temp->x][temp->y - 1] = 1;
    }
    else
    {
        NodeSon[2] = NULL;
    }

    if (temp->y < 5 && map[temp->x][temp->y][3] == 0 && Road[temp->x][
       temp->y + 1] != 1) //向下能走且没走
       过
    {
        NodeSon[3] = (Node *)malloc(sizeof(Node));
        NodeSon[3]->last = temp;
        NodeSon[3]->dir = 'R';
        NodeSon[3]->x = temp->x;
        NodeSon[3]->y = temp->y + 1;
        Road[temp->x][temp->y + 1] = 1;
    }
    else
    {
        NodeSon[3] = NULL;
    }
    return NodeSon;
}

Node *Check(Node **next4mb, Node *end)
{
    int i;
    for (i = 0; i < 4; i++)
    {
        if (next4mb[i] == NULL)
```

```
            continue;
        else
        {
            if (next4mb[i]->x == end->x && next4mb[i]->y == end->y)
                break;
        }
    }
    if (i == 4)
        return NULL;
    else
        return next4mb[i];
}

int En4mbQueue(Node **next4mb, Queue *RoadFinder)
{
    for (int i = 0; i < 4; i++)
    {
        if (next4mb[i] != NULL)
            EnQueue(next4mb[i], RoadFinder);
    }
    return 1;
}
int PrintRoad(Node *EndInQueue, int Printb) //Printb is 1 the Print
{
    char operation[100];
    Node *nod1 = EndInQueue, *nod2;
    int i;
    for (i = 0; i < 100; i++)
    {
        nod2 = nod1->last;
        if (nod2 == NULL)
            break;
        if (Value[nod2->x][nod2->y] == 10 && Printb == 1)
            Value[nod2->x][nod2->y] = 0;
        operation[i] = nod1->dir;
        nod1 = nod2;
    }
    if (Printb == 1)
    {
```

```
        for (int m = i - 1; m >= 0; m--)
        {
            operationTotal[ui] = operation[m];
            ui++;
        }
    }
    return i; //共走了步i
}
int FindRoad(Node *start, Node *end)
{
    Roadzero();
    Node *temp; //出来的节点FIFO
    Node *EndInQueue = NULL;
    RoadFinder = (Queue *)malloc(sizeof(Queue));
    InitializeQueue(RoadFinder, start);
    RoadFinder->front = start;
    while (EndInQueue == NULL)
    {
        temp = DeQueue(RoadFinder);
        Node **next4mb = CNB(temp); //找到个4的下一部分节点temp
        EndInQueue = Check(next4mb, end);
        En4mbQueue(next4mb, RoadFinder);
    }
    if (PrintRoad(EndInQueue, 1))
        return 1;
    else
        return 0;
}
int CalDistance(Node *start, Node *end)
{
    Roadzero();
    RoadFinder = (Queue *)malloc(sizeof(Queue));
    Node *temp; //出来的节点FIFO
    Node *EndInQueue = NULL;

    InitializeQueue(RoadFinder, start);
    RoadFinder->front = start;
    while (EndInQueue == NULL)
    {
```

```
        temp = DeQueue(RoadFinder);
        Node **next4mb = CNB(temp); //找到个4的下一部分节点temp
        EndInQueue = Check(next4mb, end);
        En4mbQueue(next4mb, RoadFinder);
    }
    return PrintRoad(EndInQueue, 0);
}
Node *findtheclose(Node *start)
{

    int l = 0;
    int distance[10];
    Node *head = (Node *)malloc(sizeof(Node));
    Node *temp1 = head;
    Node *temp2;
    for (int i = 0; i < 6; i++)
        for (int j = 0; j < 6; j++)
        {
            if (Value[i][j] > 0 && Value[i][j] < 15)
            {
                temp1->x = i;
                temp1->y = j;
                if (Value[i][j] == 10)
                {
                    distance[l] = CalDistance(start, temp1);
                    Emptymiddle(start, temp1);
                }
                else
                {
                    distance[l] = CalDistance(start, temp1);
                    Emptymiddle(start, temp1);
                    temp1->last = NULL;
                    for (int m = 0; m < 6; m++)
                        for (int n = 0; n < 6; n++)
                            if (Value[m][n] == 25)
                            {
                                hospital->x = m;
                                hospital->y = n;
                            }
```

```
                distance[l] += CalDistance(temp1, hospital);
                distance[l] = distance[l] / 3;
                Emptymiddle(start, temp1);
            }
            temp2 = temp1;
            temp1 = (Node *)malloc(sizeof(Node));
            temp2->next = temp1;
            l++;
        } //共有0-l个有用节点（实质上有——-10个）和节点lhospital
    }
int i;
int j;
for (i = 0; i < l; i++)
{
    for (j = 0; j < l; j++)
    {
        if (distance[i] <= distance[j])
            continue;
        else
            break;
    }
    if (j == l)
        break;
}
Node *good = head;
for (j = 0; j < i; j++)
{
    good = good->next;
}
Node *temp = head;
Node *ml;
for (int m = 0; m < l + 1; m++)
{
    ml = temp->next;
    if (m != i)
        free(temp);
    temp = ml;
}
return good;
```

```
}

Node *generateRoad(Node *start)
{
    start->last = NULL;
    Node *end = findtheclose(start);
    FindRoad(start, end);
    end->last = NULL;
    if (Value[end->x][end->y] == 5)
    {
        FindRoad(end, hospital);
        Value[hospital->x][hospital->y] = 0;
        Value[end->x][end->y] = 0;
        EmptyQueue(start);
        return hospital;
    }
    Value[end->x][end->y] = 0;
    EmptyQueue(start);
    return end;
}
void getmap()
{
    map[0][0][3] = 1;
    map[0][1][2] = 1;
    map[1][3][3] = 1;
    map[1][4][2] = 1;
    map[1][1][1] = 1;
    map[2][1][0] = 1;
    map[3][4][3] = 1;
    map[3][5][2] = 1;
    map[3][3][1] = 1;
    map[4][3][0] = 1;
    map[4][1][3] = 1;
    map[4][2][2] = 1;
    map[4][4][1] = 1;
    map[5][4][0] = 1;
    map[5][4][3] = 1;
    map[5][5][2] = 1;
    map[3][1][3] = 1;
```

```
    map[3][2][2] = 1;
}
void ValueSet()
{
    getValue(0, 2, 10); //到时候可能需要定时中断，定过程中断
    getValue(2, 4, 10);
    getValue(3, 0, 10);
    getValue(3, 4, 10);
    getValue(4, 5, 10);
    getValue(5, 2, 10);
    getValue(2, 2, 5);
    getValue(5, 0, 25);
}

void Roadtrans()
{
    direction[0] = operationTotal[0];
    char reference = operationTotal[0];
    int l = 0;
    for (int i = 0; i < ui; i++)
    {
        if (operationTotal[i] == reference)
        {
            l++;
        }
        else
        {
            direction[u] = 48 + l;
            u++;
            direction[u] = operationTotal[i];
            reference = operationTotal[i];
            u++;
            l = 1;
        }
        if(i == ui-1)
        {
            direction[u] = l + 48;
            u++;
        }
```

```
    }
}
int HEART()
{
    hospital = (Node *)malloc(sizeof(Node));
    getmap();
    ValueSet();
    Node *start;
    start = (Node *)malloc(sizeof(Node));
    scanf("%d%d", &start->x, &start->y);
    for (int i = 0; i < 3; i++)
        start = generateRoad(start);
    Roadtrans();
    free(start);
    free(hospital);
}
```

# 3 存在的问题及改进思路

# 4 个人贡献以及参赛感想与建议

## 4.1 个人贡献

## 4.2 参赛感想

## 4.3 建议

# 5 附录

# 参考文献

[1] 无