关于RM夏令营机器人的技术报告大纲 (14组)

组员及分工：

- 韩宇轩（程序调试）
- 唐浩云（主体结构设计）
- 王凯博（结构设计）
- 唐志尧（结构设计和电路规划）
- 卢晟安（结构搭建与后勤）
- 刘伊芃（前端设计与视觉算法）
- 钟思哲（算法编写）

一、需求分析

机械部分

1. 底盘：能轻松通过方块间空隙，移动平稳。
2. 得分装置：能够夹取得分物，并带有升降功能。
3. 储存装置：能够存储多个得分物，减少跑路时间。

嵌入部分

根据负责机械制图同学的 SolidWorks 3D 建模，我对于机器的嵌入式开发部分进行了需求分析，整台机器可划分为四个部分：

模块	执行器	传感器
升降摇臂	2006 电机*2	编码器
底盘	3508 电机*4	编码器
夹取装置	2006 电机*1	编码器
存储传送带	3508 电机*1	编码器、光电传感器

算法部分

第一阶段

第一阶段的满铺方案。

第二阶段

提供根据博弈论的策略指导。

视觉部分

实现自动对位，增加遥控效率。

前端部分

比赛场上使用手机 App 作为算法 UI 可减少对准备时间的占用，以最快速度完成算法需求。

二、所需技术点、关键词

机械部分

- 承载式车身
- 摇臂平行四边形防死点
- 皮筋平衡重力
- 独立悬挂

嵌入部分

- PID 闭环控制
- 开环堵转
- 低通滤波

算法部分

第一阶段

Version 1.X

- 暴力搜索
- 枚举
- 深度优先搜索

Version 2.X

- 动态规划
- 轮廓线
- 状态压缩
- 深度优先搜索 (Version 2.1)
- 并查集 (Version 2.1)
- 连通性 (Version 2.2)
- 记忆化广度优先搜索 (Version 2.2)
- 队列 (Version 2.2)
- 动态开点 (Version 2.2)
- 平衡二叉搜索树 (Version 2.2)

视觉部分

- OpenCV
- Kotlin
- 形态学操作
- 均值滤波
- 绘制矩形

前端部分

- Kotlin
- Anko Layout
- Vert.x
- Kotlin Coroutine
- ViewPager + Fragment
- Material Design

三、总体方案

夹子夹取→摇臂上旋转→储存仓存储→底盘运送→储存仓释放→摇臂下旋→夹子防开

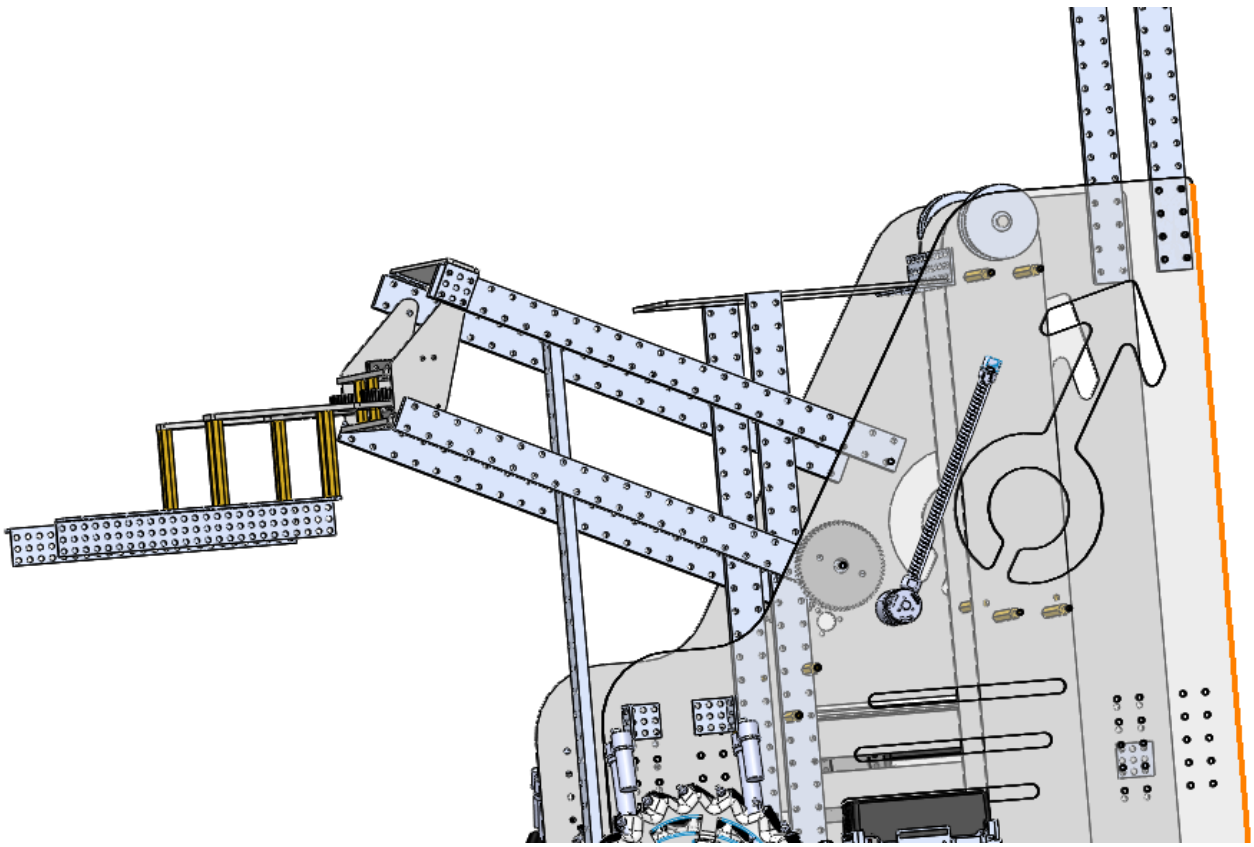
四、各模块方案

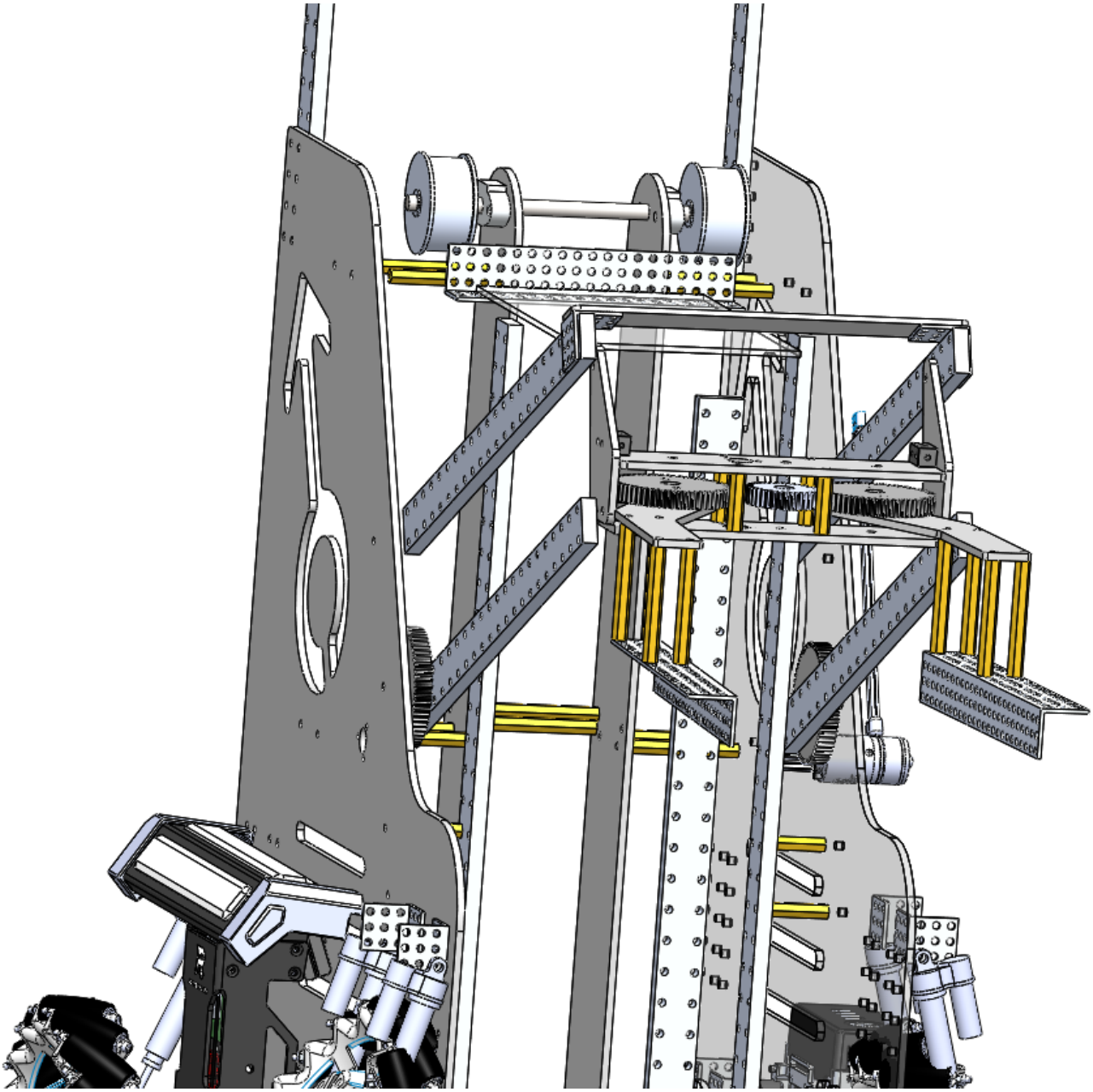
机械部分

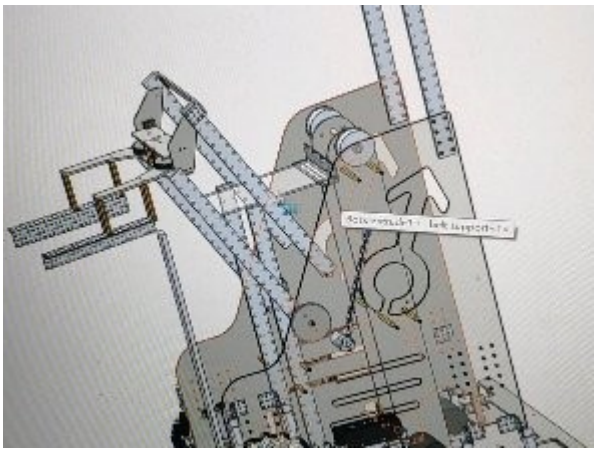
整体布局

- 夹子
- 摇臂
- 储存仓
- 图传位
- 悬挂
- 承载式车架

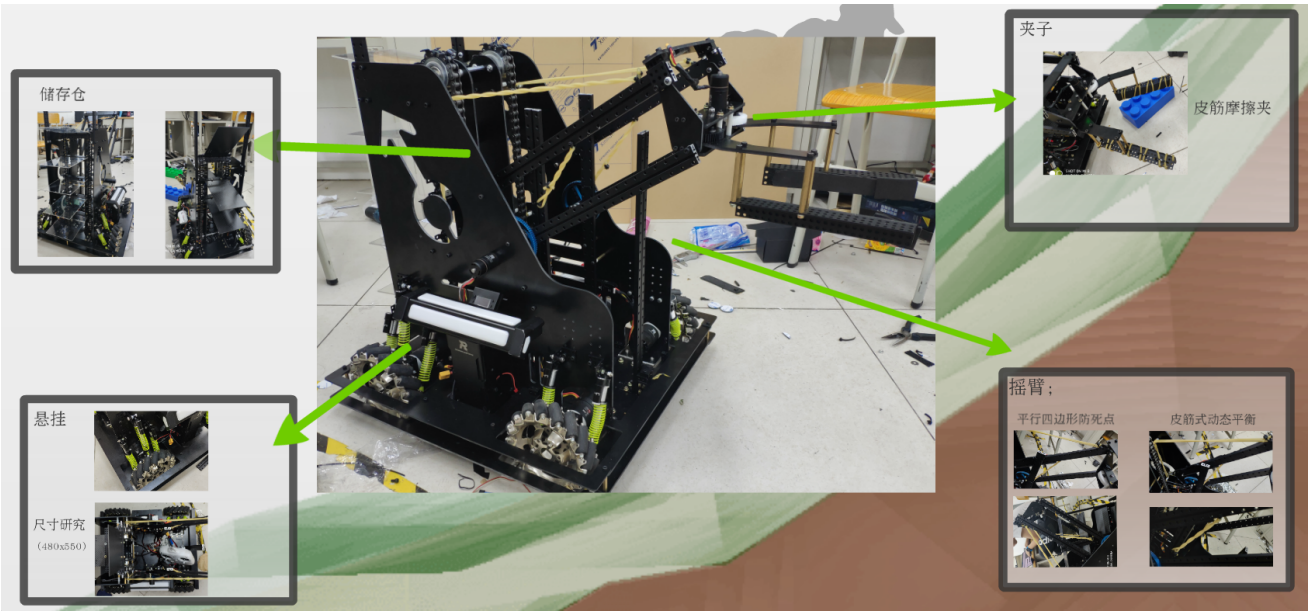
机械结构草图







各模块排布



嵌入部分

升降摇臂

升降摇臂实质上是平行四连杆结构，负责将地上的块夹取到传送带上。同时，在第二阶段中拆，以及建造城堡都要用到升降摇臂。在这种情况下，对其进行 PID 闭环显得非常重要。这种做法的好处在于机械臂可以更好的停留在遥控手希望机器停下来的位置（实质为锁死）。

底盘

底盘部分采用了麦克纳姆轮，形成了二维平面全自由度的运动。为了使遥控手获得更加线性的遥控体验（如果直接对摇杆读值和电机输出电流进行映射，遥控手对于底盘的控制可能会因为转速增量不均匀而导致操作失误），我认为对底盘进行 PID 闭速度环控制是有必要的。其实现方法是以摇杆读值作为输入，对通过编码器角度值微分后所得到的相对速度进行控制。

夹取装置

夹取装置看似简单，但实际上可以比作是木桶效应中的短板。没有一个高效可靠的夹取设计很难成就一台优秀的机器。因此，简单的实现控制夹取装置开合是远远不够的。对于嵌入式开发而言，能让夹子夹紧块并且不掉下来才是首要任务。根据我的经验，其实现方法可分为三种：

开环堵转

开环堵转的优点是实现起来非常容易，缺点也显而易见：没有反馈，对于任何情况都只能采取固定的对应措施。

PID+低通滤波器（遗忘积分）

```
long LowPassFilter_Run(long value,LowPassFilter &c){
    c.Memory = ((100-c.Forget)*c.Memory + c.Forget*value)/100;
    return c.Memory;
}
```

PID+低通滤波器听起来可能晦涩难懂，其实际原理可理解为加了惯性的PID控制，通过调整积分的遗忘率（占比）来调整惯性的大小。e.g.假设夹取一个块并关闭夹子的目标角度为1000度。如果仅仅将电机闭环至1000度，夹子是必然不能把块夹紧的。这这时，我们如果给控制系统的target增加惯性，如目标为继续转100度，那么在夹子旋转达到1000度时虽然不能继续旋转，但是剩余的误差所产生的反馈可以将块夹紧，从而达到稳定运输块的效果。

PID控制并加大target

PID控制且加大目标值其实原理与上文相同，只需手动增加固定的目标值即可解决问题。

分析比较

上述的三种方法实现复杂度：

$$\text{PID+低通滤波器} > \text{PID控制并加大target} > \text{开环堵转}$$

再次对实际情况进行分析后两种方法其实在对于夹取物型变量较大（软）时有着很好的表现。但对于非常坚硬难以形变的积木块来说，其实际使用效果和开环小电流堵转相比区别不大。 因此，我选择使用开环堵转的方法实现夹紧积木块。

存储传送带

存储传送带的使用逻辑非常简单：保证每个升降平台相对于摇臂的位置一致，且能自动运行。所以机器需要知道当前传送带储存层数，以及传送带的储存上限。我们可以使用编码器对于传送链条进行位置控制，保证每次旋转周期相等。而通过编码器角度，我们可以知道当前使用的是第几个储存平台。激光传感器负责向机器反馈积木块的储存状况。若接收器可以收到激光，则证明目前的绝对层中没有物块（平台碳板自身厚度足够薄，产生的数据可忽略）。若不能接收到激光（被遮挡），则说明此绝对层中存有物块。至此，机器可以获取到实现传送带所需的所有数据和控制逻辑。同时，在PID控制过程中，由于传送带滑槽的摩擦力较大，导致控制系统中阻尼过大，提前趋近于稳定状态。虽然可以同积分项调节最终达到target目标值，但是需要约5s才能矫正回零位。所以，我们将估计加大积分项参数，使得系统超调，从而达到最快消除稳态误差的效果。

算法部分

算法思路

Version 1.0

首先考虑将同种颜色的四个点分为两对，将问题转化为八对点构建路径覆盖全图问题。然后再用最朴素的暴力搜索思想，使用深度优先搜索遍历所有路径，找到一组合法解后立即输出并退出。(注意：对于每一种颜色的四个点A、B、C、D，我们都有三种分割方法：AB/CD、AC/BD、AD/BC。因此总分割方案数为 81 种。)

Version 2.0

由于搜索算法的时间受场面影响较大，因此我们尝试寻找一个时间上更稳定的算法。因此我们采用了动态规划算法，设计了全新的算法。通过轮廓线思想，我们迅速完成了第一版动态规划算法。

不难发现，在整个图上的空白区域上，每个格子中的路径只有六种不同形态，即 ┌、┐、└、┘、─、│，因此我们可以在每填一个格子时讨论这六种插头的可能性。同时我们发现，未填格与已填格的交界线（即轮廓线）是唯一影响接下来填色的因素。因此可以通过对轮廓线上的格边（即插头）进行动态规划求解。

由于动态规划合并了状态，因此只统计了方案数。我们通过保留每一状态的最后一个合法前驱状态，然后从结束状态想回走即可得到转移路径，画出路径方案。

Version 2.1

在实现 Version 2.0 后，我们发现，给出的方案可能有回路。因此我们对算法进行了改进。我们去除了记录每一状态的最后一个合法前驱状态的步骤，改为在规划结束后从末状态向初始状态沿有效状态进行深度优先搜索。每找到一条规划路径后便用并查集判断该方案中是否合法（没有回路），并输出第一组合法的解。

虽然程序运行时间受到了搜索的影响，但是其与 Version 2.2 相对较易实现、运行时间比 Version 1.0 稳定得多的优点使得它最终成为了我们移植的版本。

Version 2.2

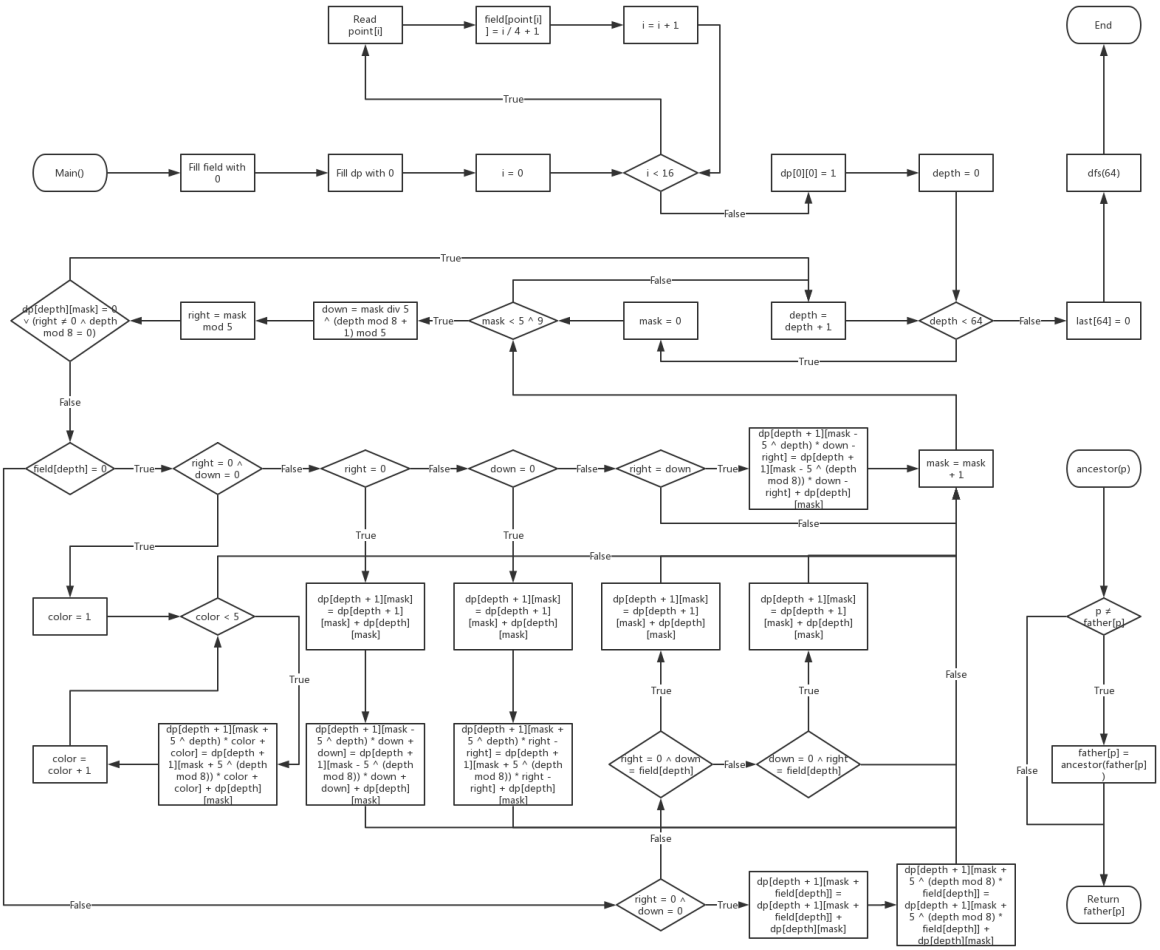
在该版本中，我们将连通性融入轮廓线状态中。但连通性的加入大大增加了状态数目，同时复杂化了状态的转移。内存空间也因此不够用了。因此，我们用平衡二叉搜索树以达到动态开点的目的。同时采取了基于队列的记忆化广度优先搜索（本质上还是动态规划）来避免访问不合法状态。

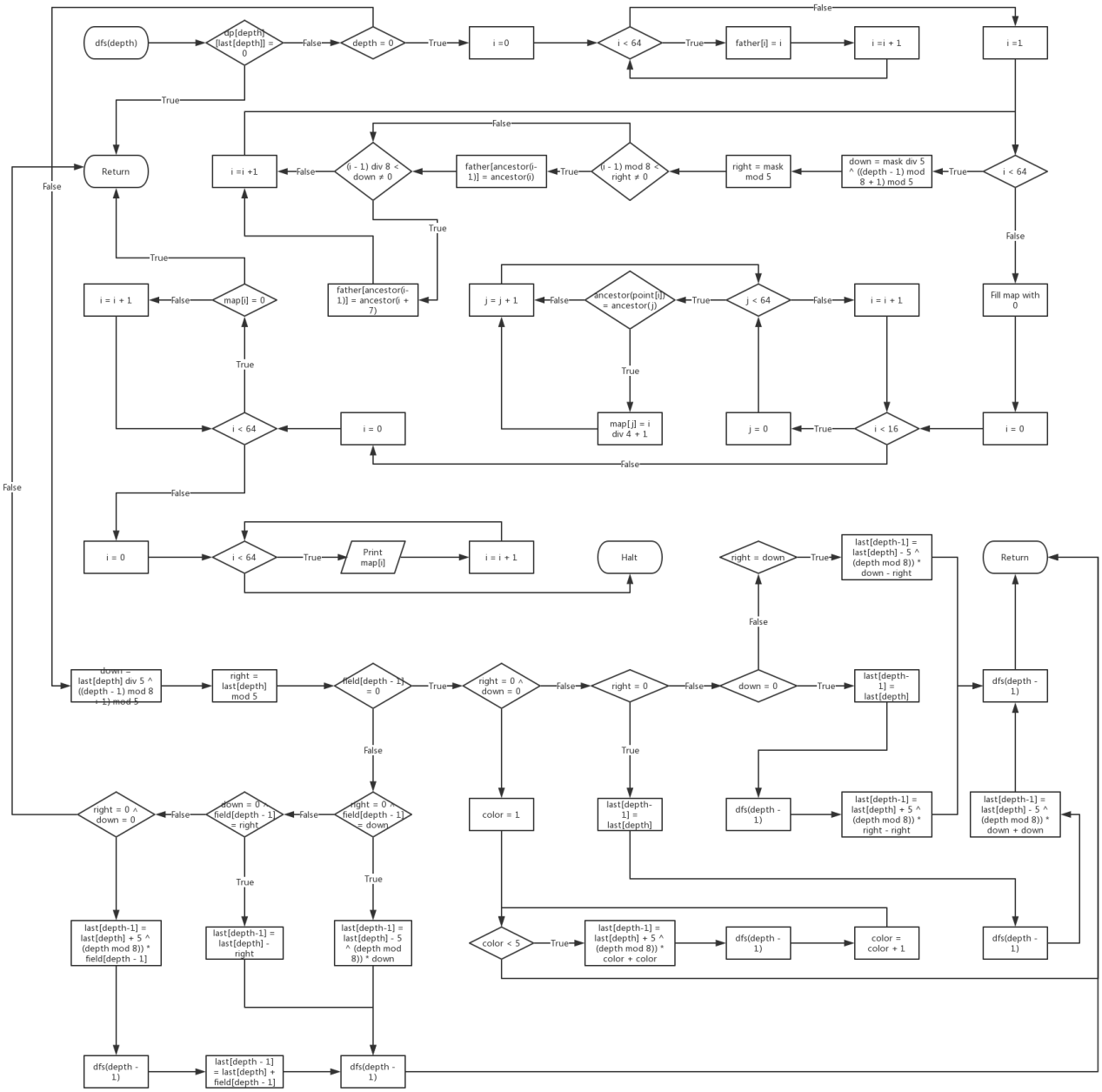
由于此版本中用了较多数据结构，大大复杂化了程序，因此我们仅完成了统计方案数的部分，未采用记录最后一个合法前驱状态或向回深度优先搜索的方法实现输出方案数。

值得一提的是，这一版本中没有不合法的方案，因此可以在深度优先搜索时遍历所有方案，选取较优或最优方案。

算法流程图

最终采用版本：Version 2.1





视觉部分

与 18 组相同，通过颜色识别找出块后通过计算块的长宽高比例，绘制相对坐标。通过 Uart 通信从妙算传输至主控后，通过底盘的 PID闭速度环校准位置。

前端部分

布局

采用单 `Activity` 多 `Fragment`，`ViewPager` 负责切换。算法UI `Fragment` 中以垂直 `LinearLayout` 作为根布局，每个色块为 `ImageButton` 存放在 `GridLayout` 中；下部 4 个取色器采用 `Button`，构成一个水平 `LinearLayout`；最下面的确认按钮和删除按钮为 `design` 中的 `FloatingActionButton`。

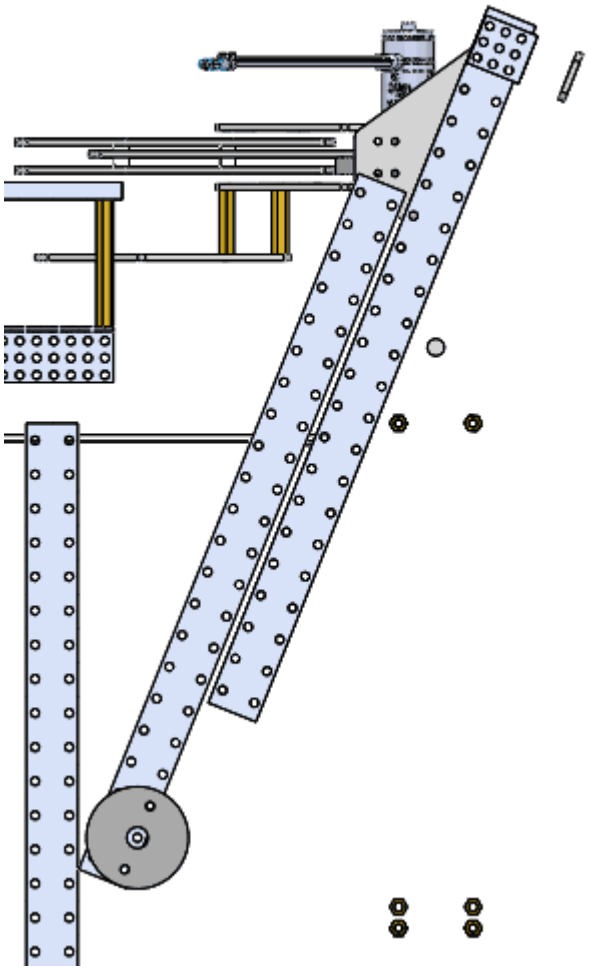
逻辑

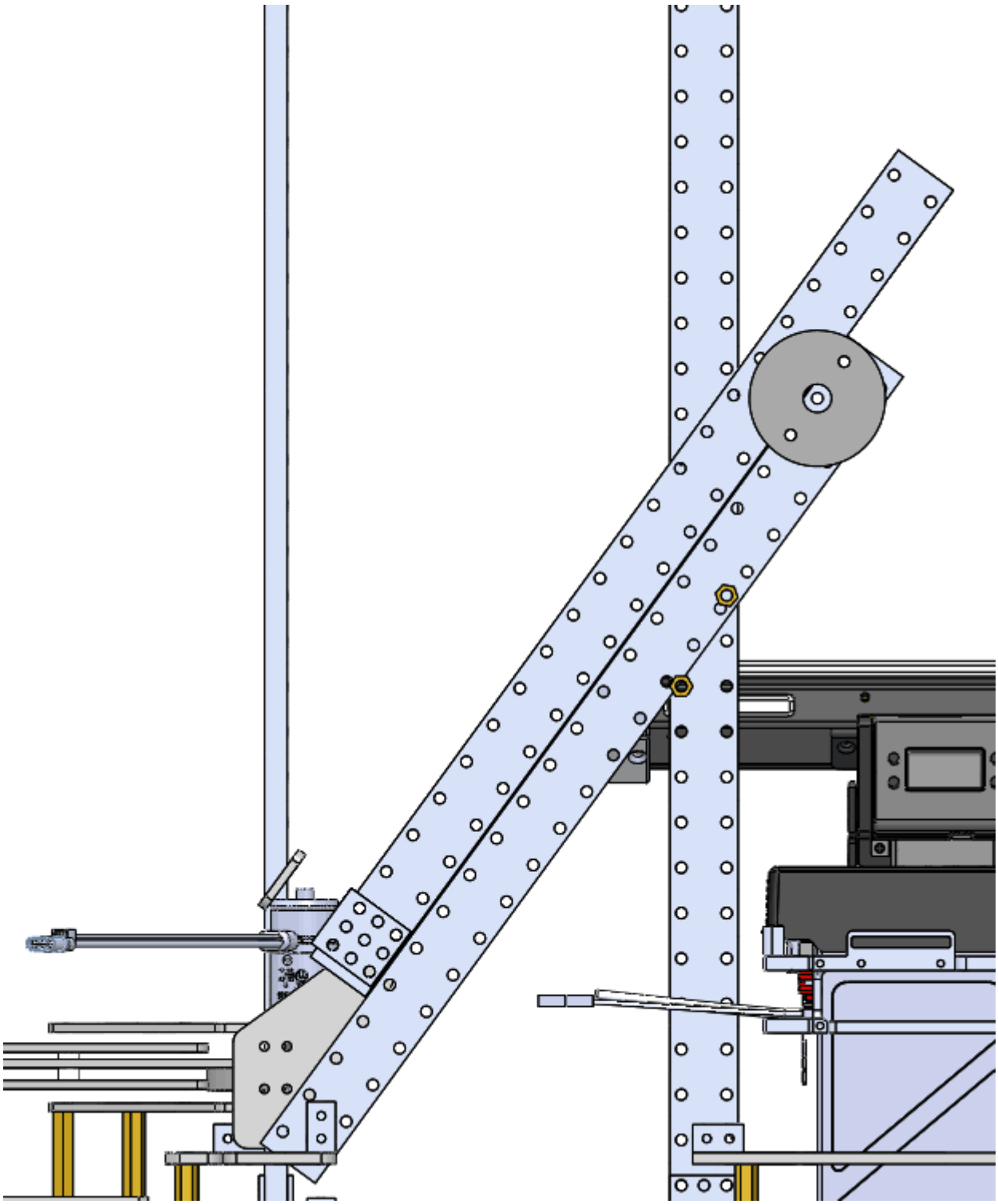
在取色器中选一种颜色后，在网格中可绘制城堡，用小山的图标标记。如有绘制错误情况，快速双击色块会清除为空白。提交结果前，程序会初步校验每种颜色城堡数量，避免进行无意义计算。在解算完毕后，程序将进入标记模式。此时电机色块会将其涂为奇怪的颜色，用来记录场上色块摆放情况。

五、理论分析 (Analysis)

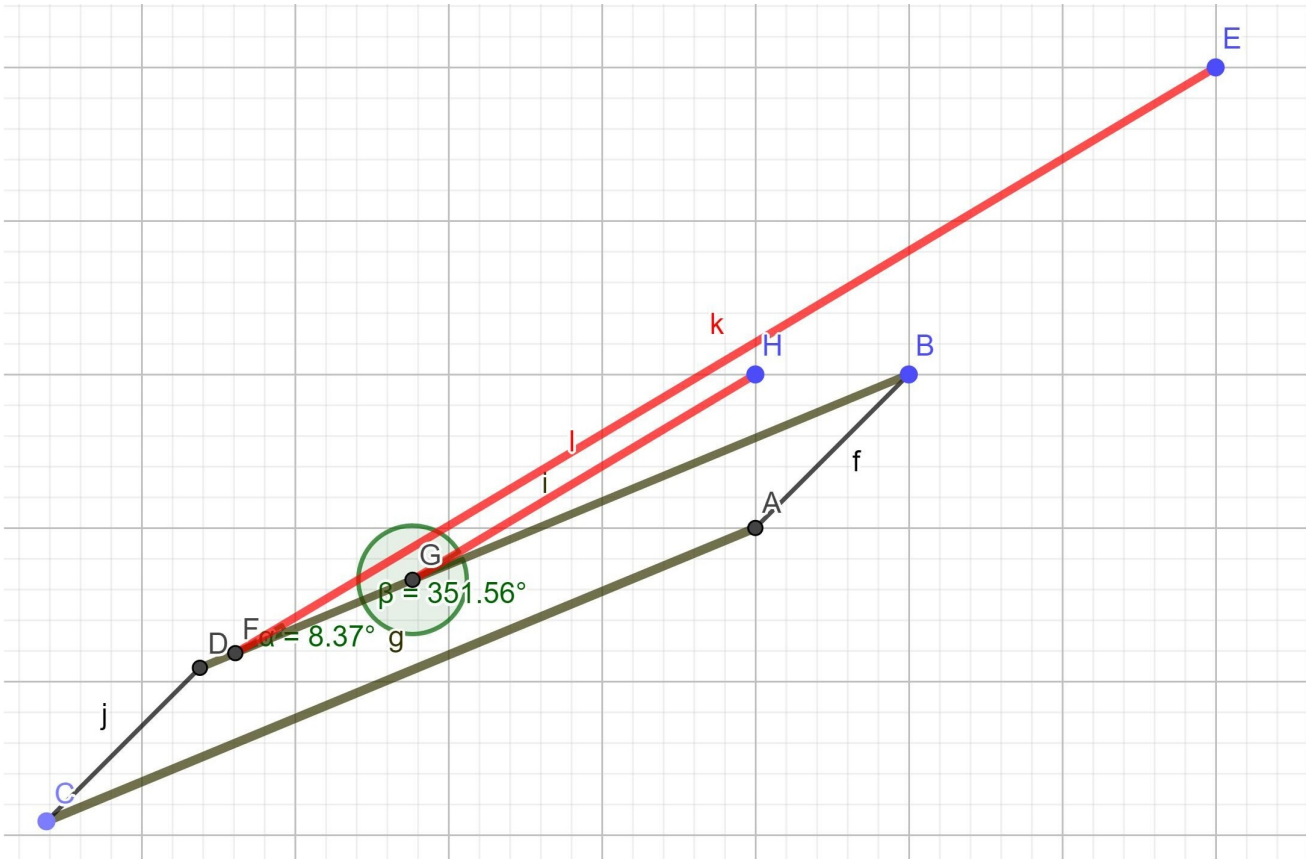
机械部分

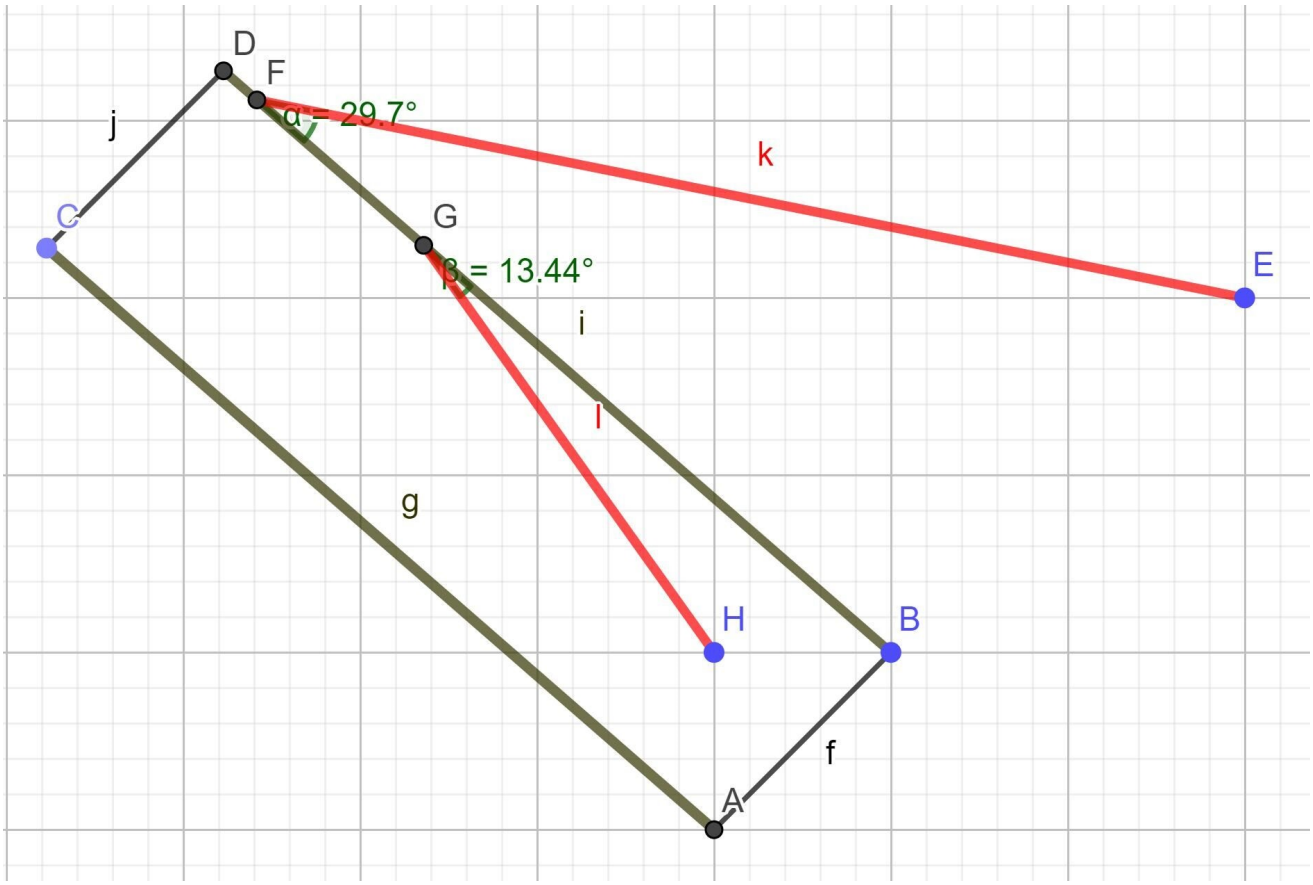
- 承载式车身（重量轻，空间利用率大）
- 底盘尺寸：480mm×500mm（保证通过性）
- 储存仓轴距：550mm（保证放块数量）
- 平行四边形升降防死点

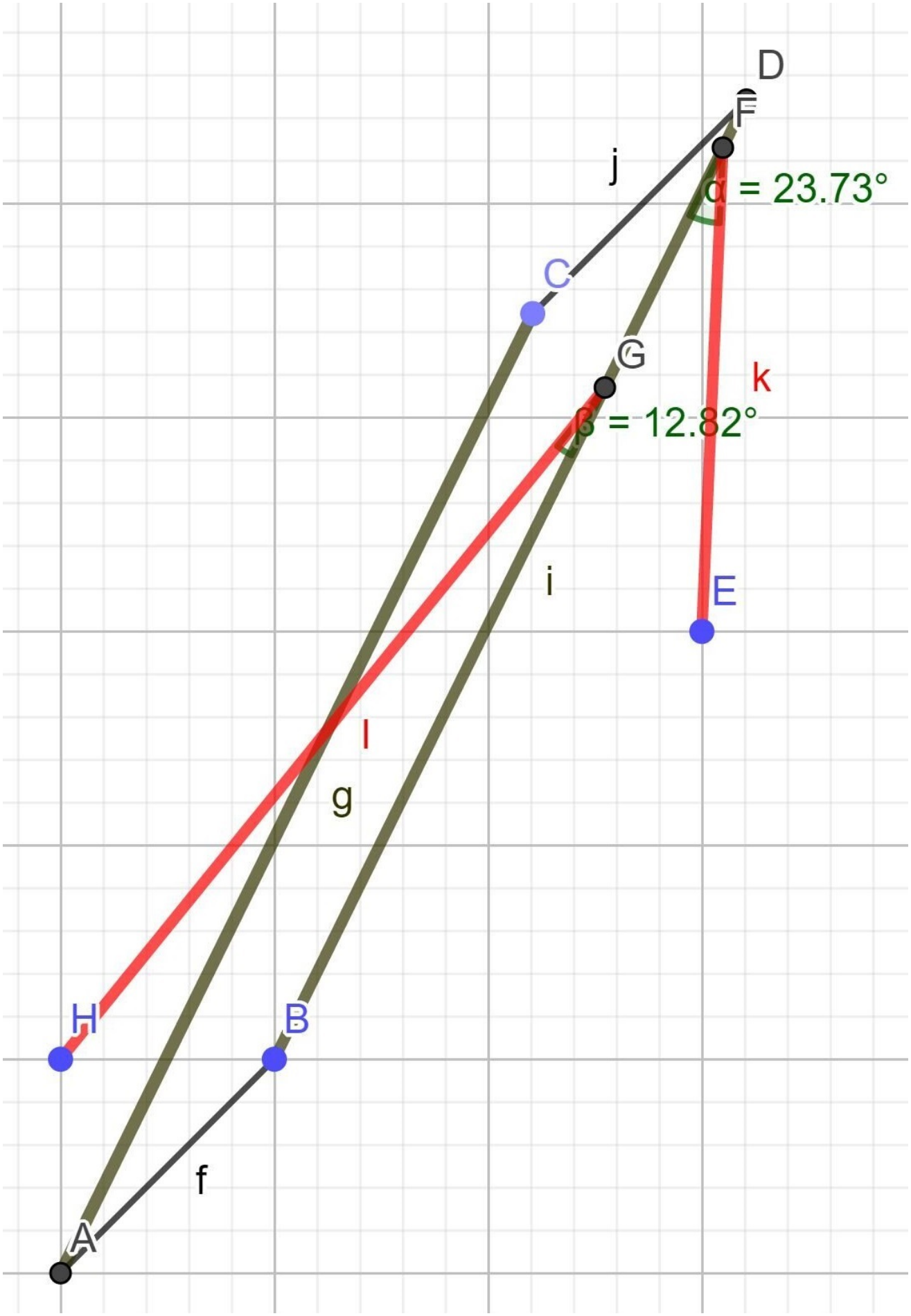




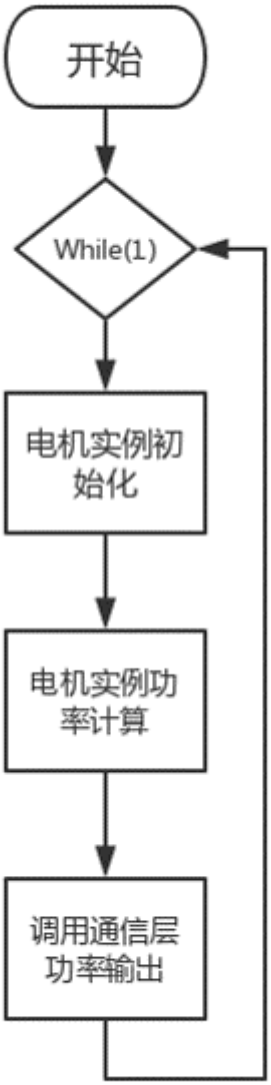
- 皮筋平衡升降结构重力

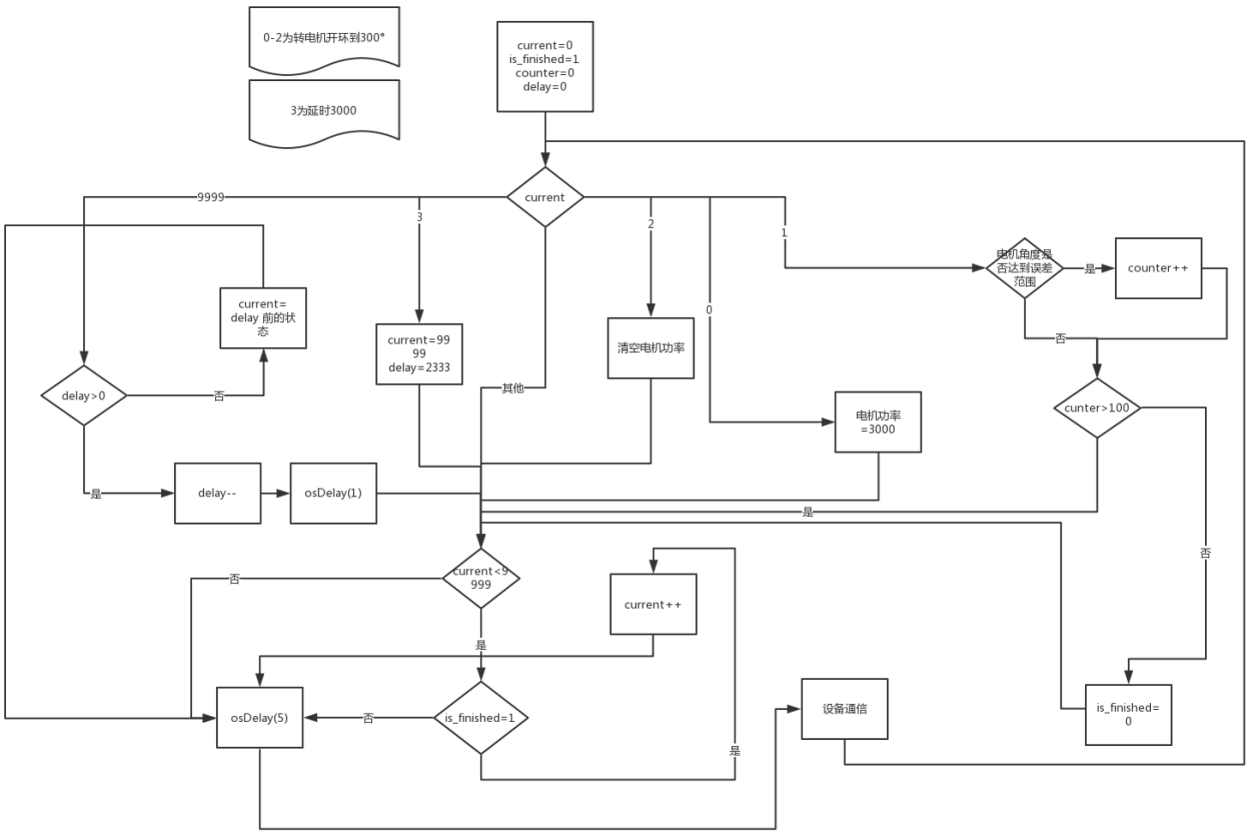

E
k
H
B
l
i
f
A
G
β = 351.56°
F
α = 8.37°
g
D
j
C


D
F
α = 29.7°
j
k
C
G
E
β = 13.44°
i
l
g
H
B
f
A


D
F
j
α = 23.73°
C
G
k
β = 12.82°
i
E
l
g
H
B
f
A

嵌入部分





算法部分

运行时间测试

测试数据

测试点	数据	测试点	数据	测试点	数据
#1	44 50 24 27 1 4 20 14 0 60 43 38 40 56 16 19	#6	7 23 21 10 9 32 5 51 60 63 20 18 22 52 59 40	#11	1 4 28 39 29 23 54 40 11 7 20 25 26 32 49 47
#2	24 25 19 4 49 56 28 30 50 43 36 7 11 22 57 58	#7	6 20 38 63 0 17 32 50 3 19 58 62 40 57 35 21	#12	0 4 56 60 1 5 57 61 2 6 58 62 3 7 59 63
#3	9 34 40 56 43 37 28 31 0 7 14 17 20 15 44 63	#8	35 41 54 63 0 24 34 37 25 48 53 55 4 14 56 58	#13	0 1 2 3 4 5 6 7 56 57 58 59 60 61 62 63
#4	51 42 27 10 63 47 38 23 19 33 16 57 53 36 20 7	#9	51 54 29 19 2 18 42 36 11 7 30 58 3 13 26 32	#14	0 7 32 39 8 15 40 47 16 23 48 55 24 31 56 63
#5	49 34 12 15 51 32 0 7 56 17 44 22 59 54 29 23	#10	10 17 34 59 60 54 36 30 41 58 7 22 61 46 56 0	-	-

测试结果

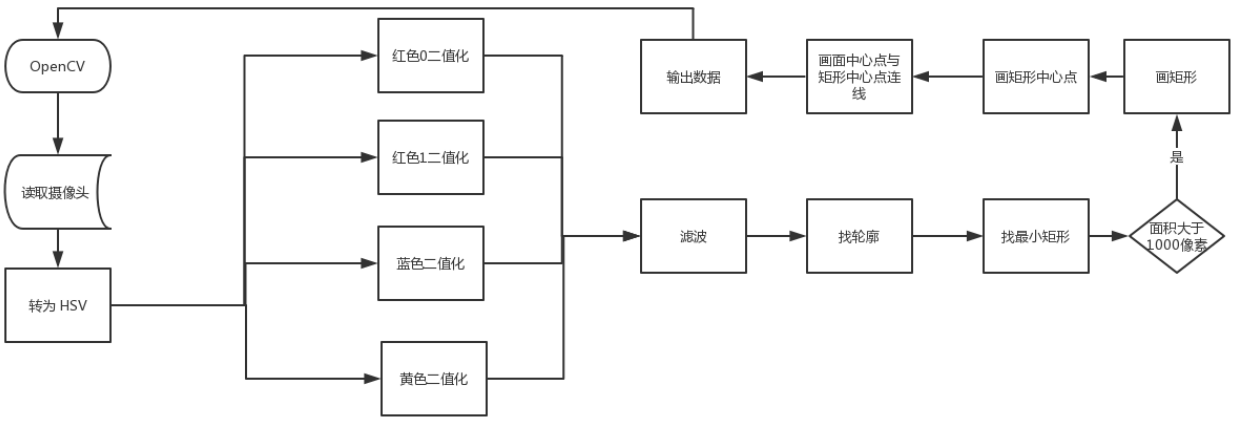
测试点	Version 1.0	Version 2.0	Version 2.1
#1	N/A	4082 ms	4221 ms
#2	118.1 ms	3607 ms	18100 ms
#3	6610 ms	3647 ms	3744 ms
#4	16320 ms	3755 ms	3664 ms
#5	4928 ms	3608 ms	3669 ms
#6	7396 ms	3726 ms	3997 ms
#7	N/A	3537 ms	3833 ms
#8	N/A	3609 ms	3751 ms
#9	211.5 ms	3650 ms	3756 ms
#10	23200 ms	3645 ms	3793 ms
#11	777.9 ms	3668 ms	3687 ms
#12	N/A	3607 ms	3767 ms
#13	93.72 ms	3771 ms	3762 ms
#14	N/A	3782 ms	3751 ms

运行时间因机器而异，测得时间保留四位有效数字。最大允许运行时间为 60000 ms，若仍未结束则记为 N/A。

算法正确性证明

见`各模块方案 -> 算法部分`。

视觉部分



前端部分

算法使用 C++ 语言编写，在 UI 完成后我们将算法重构为 Kotlin 语言，使其与 Android App 对接。然而在运行测试时，程序初始化时总会抛出 `OutOfMemory` 异常，无法正常运行。经过内存分析软件调试后，我们发现，在算法中变量初始化时要消耗约 1GB 内存。手机厂商一般会将 `Android Dalvik` 虚拟机的 heap 限制为 512MB ，致使算法无法在手机上正常运行。为了解决该问题，我们设计了一个 Http 服务器，用来代理手机进行计算。手机将颜色数据以 `GET` 请求方式通过 `PathUrl` 向电脑的服务端发送请求。服务器计算完后会将颜色数据发回手机。

六、制作与测试流程

问题

储存仓传送链条摩擦较大，承载板无法保持水平。

原因

由于 3508 电机在系统中不能实现低转速高扭矩，导致控制系统提前趋向于稳定（发现者：韩宇轩）。

解决方案

重新设计储存仓，加入滑槽设计，以解决问题。

七、结果与评价

机械部分

优点

- 重量相对轻
- 车内空间利用率高
- 车身通过性高
- 完成任务效率高

创新点

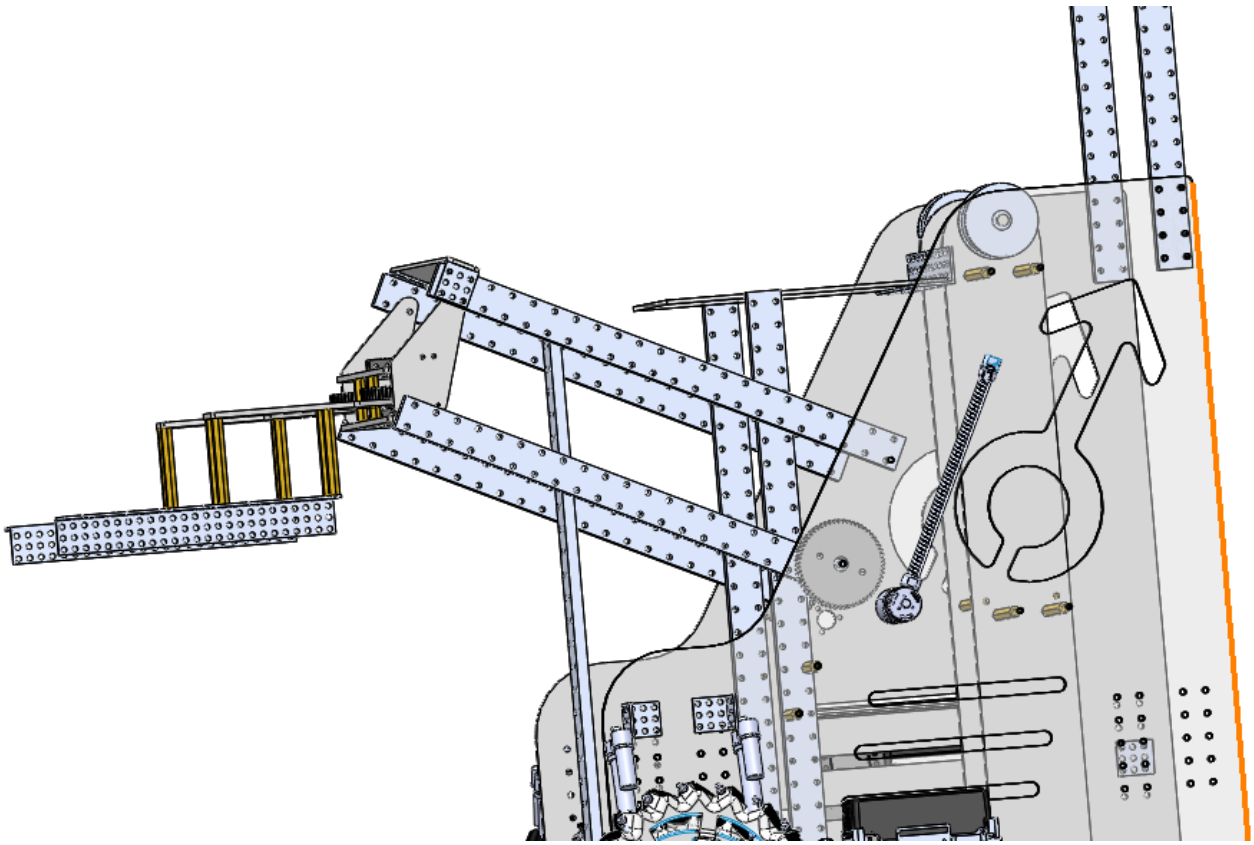
三块板材完成车身（重量相对轻、重心也相对低一些、车内空间利用率高）

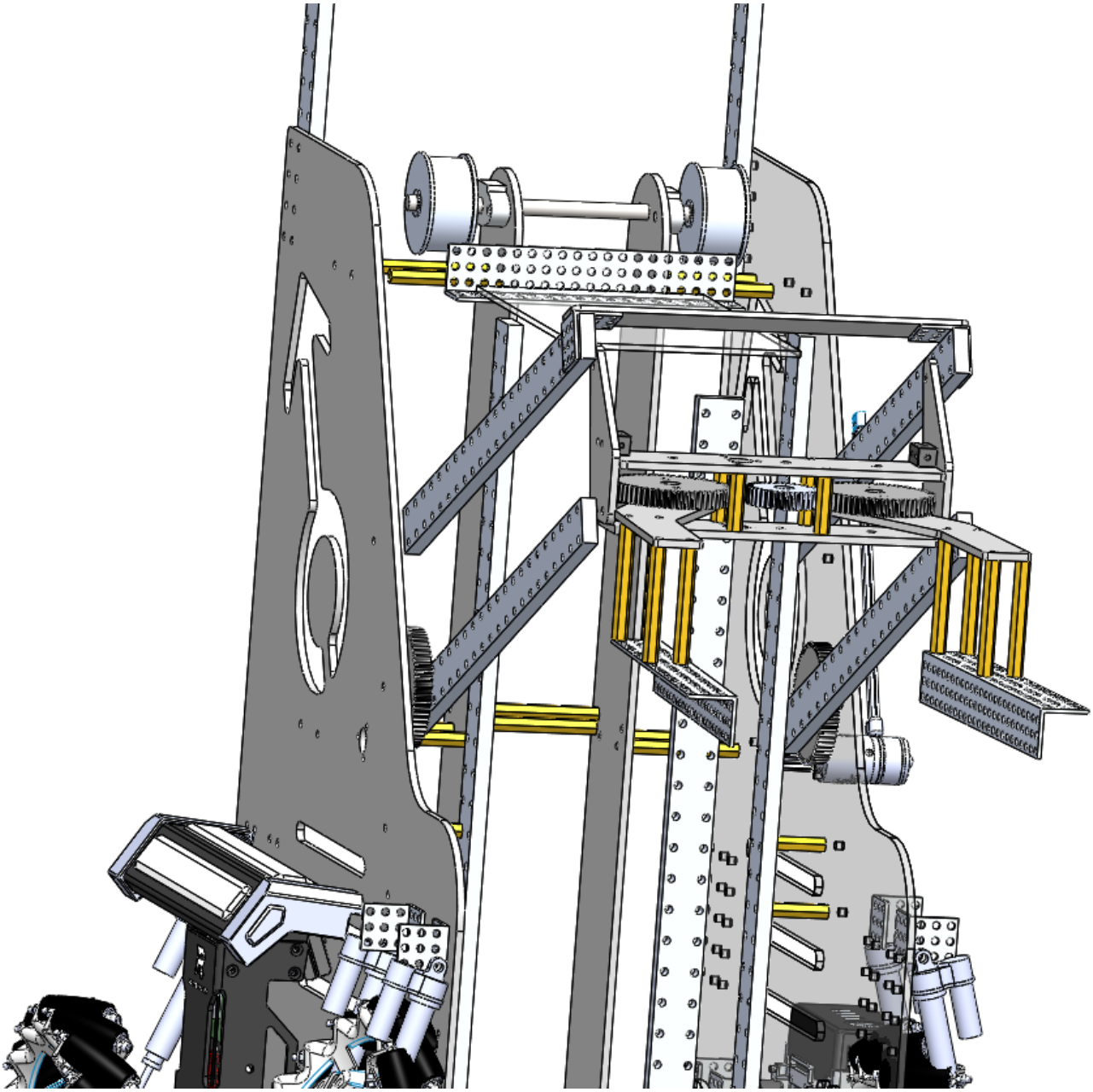
算法部分

在采用并查集滤去不合法方案后， Version 2.1B 的表现不逊色于复杂度更高的 Version 2.2 。这也是最终我们选择移植它的最重要原因。与搜索相比，它在运行时间上的微小劣势完全无法掩盖其在稳定性方面的巨大优势。

八、附录 (Appendix)

机械部分





嵌入部分

Source Code

RM-Final.zip

The "FRAMEWORK" (ALPHA)

求大神帮忙 debug `qwq`

https://github.com/tony3641/RM_Rewrite

算法部分

Version 1.0

```
#include<cstdio>
```

```
#include<cstring>
struct cood {
    int x,y;
};
int field[10][10];
cood point[4][4],pair[8][2];
int solve(int clr,int pos_x,int pos_y)
{
    if((pos_x==pair[clr][1].x)&&(pos_y==pair[clr][1].y))
    {
        if(clr==7)
        {
            for(int i=1;i<=8;i++)
                for(int j=1;j<=8;j++)
                    if(field[i][j]==0)
                        return 0;
            return 1;
        }
        else
            return solve(clr+1,pair[clr+1][0].x,pair[clr+1][0].y);
    }
    if((field[pos_x-1][pos_y]==0)||(field[pos_x-1][pos_y]==11+clr))
    {
        field[pos_x-1][pos_y]=clr+1;
        if(solve(clr,pos_x-1,pos_y)==1)
            return 1;
        field[pos_x-1][pos_y]=0;
    }
    if((field[pos_x][pos_y-1]==0)||(field[pos_x][pos_y-1]==11+clr))
    {
        field[pos_x][pos_y-1]=clr+1;
        if(solve(clr,pos_x,pos_y-1)==1)
            return 1;
        field[pos_x][pos_y-1]=0;
    }
    if((field[pos_x+1][pos_y]==0)||(field[pos_x+1][pos_y]==11+clr))
    {
        field[pos_x+1][pos_y]=clr+1;
        if(solve(clr,pos_x+1,pos_y)==1)
            return 1;
        field[pos_x+1][pos_y]=0;
    }
    if((field[pos_x][pos_y+1]==0)||(field[pos_x][pos_y+1]==11+clr))
    {
        field[pos_x][pos_y+1]=clr+1;
        if(solve(clr,pos_x,pos_y+1)==1)
            return 1;
        field[pos_x][pos_y+1]=0;
    }
    return 0;
}
int main()
{
```

```
    memset(field,-1,sizeof(field));
    for(int i=0;i<4;i++)
        for(int j=0,tmp;j<4;j++)
        {
            scanf("%d",&tmp);
            point[i][j].x=tmp/8+1;
            point[i][j].y=tmp%8+1;
        }
    for(int mask=0;mask<81;mask++)
    {
        for(int i=1;i<=8;i++)
            for(int j=1;j<=8;j++)
                field[i][j]=0;
        for(int i=0,tmp=mask;i<4;i++)
        {
            pair[2*i][0]=point[i][0];
            pair[2*i][1]=point[i][tmp%3+1];
            switch(tmp%3+1)
            {
                case 1:
                    pair[2*i+1][0]=point[i][2];
                    pair[2*i+1][1]=point[i][3];
                    break;
                case 2:
                    pair[2*i+1][0]=point[i][1];
                    pair[2*i+1][1]=point[i][3];
                    break;
                case 3:
                    pair[2*i+1][0]=point[i][2];
                    pair[2*i+1][1]=point[i][1];
                    break;
            }
            tmp/=3;
        }
        for(int i=0;i<8;i++)
            for(int j=0;j<2;j++)
                field[pair[i][j].x][pair[i][j].y]=10*j+i+1;
        if(solve(0,pair[0][0].x,pair[0][0].y)==1)
        {
            for(int i=0;i<8;i++)
                for(int j=0;j<8;j++)
                    printf("%d",(field[i+1][j+1]%10+1)/2);
            return 0;
        }
    }
    return 0;
}
```

Version 2.0

```
#include<cstdio>
#include<cstring>
```

```
#include<cstdlib>
long long dp[65][1953125];
int lst[65][1953125];
int field[64];
int power(int bit)
{
    switch(bit%8)
    {
        case 0:
            return 390625;
        case 1:
            return 78125;
        case 2:
            return 15625;
        case 3:
            return 3125;
        case 4:
            return 625;
        case 5:
            return 125;
        case 6:
            return 25;
        case 7:
            return 5;
    }
    return 1;
}
int plug(int mask,int bit)
{
    return (mask/power(bit))%5;
}
void dfs(int depth,int mask)
{
    printf("%d ",depth);
    for(int i=0,tmp=mask;i<9;tmp/=5,i++)
        printf("%d",tmp%5);
    putchar('\n');
    if(depth==0)
    {
        if(mask!=0)
            return;
        for(int i=0;i<64;i++)
            if(field[i]==0)
                return;
        for(int i=0;i<64;i++)
            printf("%d",field[i]);
        exit(0);
    }
    if(field[depth-1]==0)
    {
        field[depth-1]=(mask%5!=0)?mask%5:field[depth-1];
        field[depth-1]=(plug(mask,depth-1)!=0)?plug(mask,depth-1):field[depth-1];
        field[depth-1]=(lst[depth][mask]%5!=0)?lst[depth][mask]%5:field[depth-1];
```

```
        field[depth-1]=(plug(lst[depth][mask],depth-1)!=0)?plug(lst[depth][mask],depth-
1):field[depth-1];
    }
    dfs(depth-1,lst[depth][mask]);
}
int main()
{
    memset(field,0,sizeof(field));
    memset(dp,0,sizeof(dp));
    dp[0][0]=1;
    for(int i=0,tmp;i<16;i++)
    {
        scanf("%d",&tmp);
        field[tmp]=i/4+1;
    }
    for(int i=0;i<64;i++)
        for(int mask=0;mask<1953125;mask++)
        {
            int down=plug(mask,i),right=mask%5;
            if((dp[i][mask]==0)||((i%8==0)&&(right!=0)))
                continue;
            if(field[i]==0)
            {
                if((down==0)&&(right==0))
                {
                    for(int color=1;color<5;color++)
                    {
                        dp[i+1][mask+power(i)*color+color]+=dp[i][mask];
                        lst[i+1][mask+power(i)*color+color]=mask;
                    }
                    continue;
                }
                if(down==right)
                {
                    dp[i+1][mask-power(i)*down-right]+=dp[i][mask];
                    lst[i+1][mask-power(i)*down-right]=mask;
                    continue;
                }
                if(down==0)
                {
                    dp[i+1][mask]+=dp[i][mask];
                    lst[i+1][mask]=mask;
                    dp[i+1][mask+power(i)*right-right]+=dp[i][mask];
                    lst[i+1][mask+power(i)*right-right]=mask;
                    continue;
                }
                if(right==0)
                {
                    dp[i+1][mask]+=dp[i][mask];
                    lst[i+1][mask]=mask;
                    dp[i+1][mask-power(i)*down+down]+=dp[i][mask];
                    lst[i+1][mask-power(i)*down+down]=mask;
                    continue;
```

```
                }
            }
            else
            {
                if((down==0)&&(right==field[i]))
                {
                    dp[i+1][mask-right]+=dp[i][mask];
                    lst[i+1][mask-right]=mask;
                }
                if((right==0)&&(down==field[i]))
                {
                    dp[i+1][mask-power(i)*down]+=dp[i][mask];
                    lst[i+1][mask-power(i)*down]=mask;
                }
                if((down==0)&&(right==0))
                {
                    dp[i+1][mask+field[i]]+=dp[i][mask];
                    lst[i+1][mask+field[i]]=mask;
                    dp[i+1][mask+power(i)*field[i]]+=dp[i][mask];
                    lst[i+1][mask+power(i)*field[i]]=mask;
                }
            }
        }
    printf("%lld\n",dp[64][0]);
    dfs(64,0);
    return 0;
}
```

Version 2.1

```
#include<cstdio>
#include<cstring>
#include<cstdlib>
long long dp[65][1953125];
int point[16];
int field[64],map[64],uf_set[64],lst[65];
int power(int bit)
{
    switch(bit%8)
    {
        case 0:
            return 390625;
        case 1:
            return 78125;
        case 2:
            return 15625;
        case 3:
            return 3125;
        case 4:
            return 625;
        case 5:
            return 125;
```

```
        case 6:
            return 25;
        case 7:
            return 5;
    }
    return 1;
}
int plug(int mask,int bit)
{
    return (mask/power(bit))%5;
}
int uf_find(int i)
{
    if(uf_set[i]==i)
        return i;
    return uf_set[i]=uf_find(uf_set[i]);
}
void dfs(int depth)
{
    if(dp[depth][lst[depth]]==0)
        return;
    if(depth==0)
    {
        for(int i=0;i<64;i++)
            uf_set[i]=i;
        for(int i=1;i<64;i++)
        {
            int down=plug(lst[i],i-1),right=lst[i]%5;
            if(((i-1)%8<7)&&(right!=0))
                uf_set[uf_find(i-1)]=uf_find(i);
            if(((i-1)/8<7)&&(down!=0))
                uf_set[uf_find(i-1)]=uf_find(i+7);
        }
        memset(map,0,sizeof(map));
        for(int i=0;i<16;i++)
            for(int j=0;j<64;j++)
                if(uf_find(point[i])==uf_find(j))
                    map[j]=i/4+1;
        for(int i=0;i<64;i++)
            if(map[i]==0)
                return;
        for(int i=0;i<64;i++)
            printf("%d",map[i]);
        exit(0);
    }
    int down=plug(lst[depth],depth-1),right=lst[depth]%5;
    if(field[depth-1]==0)
    {
        if((down==0)&&(right==0))
        {
            for(int color=1;color<5;color++)
            {
                lst[depth-1]=lst[depth]+power(depth-1)*color+color;
```

```
                dfs(depth-1);
            }
            return;
        }
        if(down==right)
        {
            lst[depth-1]=lst[depth]-power(depth-1)*down-right;
            dfs(depth-1);
        }
        if(down==0)
        {
            lst[depth-1]=lst[depth];
            dfs(depth-1);
            lst[depth-1]=lst[depth]+power(depth-1)*right-right;
            dfs(depth-1);
        }
        if(right==0)
        {
            lst[depth-1]=lst[depth];
            dfs(depth-1);
            lst[depth-1]=lst[depth]-power(depth-1)*down+down;
            dfs(depth-1);
        }
    }
    else
    {
        if((down==0)&&(right==field[depth-1]))
        {
            lst[depth-1]=lst[depth]-right;
            dfs(depth-1);
        }
        if((right==0)&&(down==field[depth-1]))
        {
            lst[depth-1]=lst[depth]-power(depth-1)*down;
            dfs(depth-1);
        }
        if((down==0)&&(right==0))
        {
            lst[depth-1]=lst[depth]+power(depth-1)*field[depth-1];
            dfs(depth-1);
            lst[depth-1]=lst[depth]+field[depth-1];
            dfs(depth-1);
        }
    }
}
int main()
{
    memset(field,0,sizeof(field));
    memset(dp,0,sizeof(dp));
    dp[0][0]=1;
    for(int i=0;i<16;i++)
    {
        scanf("%d",&point[i]);
```

```
        field[point[i]]=i/4+1;
    }
    for(int i=0;i<64;i++)
        for(int mask=0;mask<1953125;mask++)
        {
            int down=plug(mask,i),right=mask%5;
            if((dp[i][mask]==0)||((i%8==0)&&(right!=0)))
                continue;
            if(field[i]==0)
            {
                if((down==0)&&(right==0))
                {
                    for(int color=1;color<5;color++)
                        dp[i+1][mask+power(i)*color+color]+=dp[i][mask];
                    continue;
                }
                if(down==right)
                    dp[i+1][mask-power(i)*down-right]+=dp[i][mask];
                if(down==0)
                {
                    dp[i+1][mask]+=dp[i][mask];
                    dp[i+1][mask+power(i)*right-right]+=dp[i][mask];
                }
                if(right==0)
                {
                    dp[i+1][mask]+=dp[i][mask];
                    dp[i+1][mask-power(i)*down+down]+=dp[i][mask];
                }
            }
            else
            {
                if((down==0)&&(right==field[i]))
                    dp[i+1][mask-right]+=dp[i][mask];
                if((right==0)&&(down==field[i]))
                    dp[i+1][mask-power(i)*down]+=dp[i][mask];
                if((down==0)&&(right==0))
                {
                    dp[i+1][mask+field[i]]+=dp[i][mask];
                    dp[i+1][mask+power(i)*field[i]]+=dp[i][mask];
                }
            }
        }
    printf("%lld\n",dp[64][0]);
    lst[64]=0;
    dfs(64);
    return 0;
}
```

Version 2.2

```
#include<bits/stdc++.h>
using namespace std;
```

```
struct plug
{
    int connectivity,colour;
    int encrypt()
    {
        return connectivity*4+colour;
    }
    void decrypt(int plug_mask)
    {
        if(plug_mask==0)
        {
            connectivity=0;
            colour=0;
            return;
        }
        connectivity=(plug_mask-1)/4;
        colour=(plug_mask-1)%4+1;
    }
};
map<int,long long> dp[65];
queue<pair<int,long long> > q;
int field[64],point[16],search[64];
plug contour[9];
long long compress()
{
    long long mask=0;
    for(int i=8;i>=0;i--)
    {
        mask*=37;
        mask+=contour[i].encrypt();
    }
    return mask;
}
void decompress(long long mask)
{
    for(int i=0;i<9;i++)
    {
        contour[i].decrypt(mask%37);
        mask/=37;
    }
}
bool is_null(int depth,long long mask)
{
    return dp[depth].find(mask)==dp[depth].end();
}
void add(int depth,long long mask)
{
    long long tmp=compress();
    if((is_null(depth,mask))&&(depth<63))
        q.push(make_pair(depth+1,tmp));
    dp[depth+1][tmp]+=dp[depth][mask];
}
/*void dfs(int depth,long long mask)
```

```
{
    if(depth==0)
    {
        for(int i=0;i<64;i++)
            printf("%d",search[i]);
        exit(0);
    }
    decompress(mask);
    if(field[depth-1]==0)
    {

    }
    else
    {

    }
}//*/
int main()
{
    memset(field,0,sizeof(field));
    for(int i=0;i<16;i++)
    {
        scanf("%d",&point[i]);
        field[point[i]]=i/4+1;
    }
    dp[0][0]=1;
    q.push(make_pair(0,0));
    while(!q.empty())
    {
        int depth=(q.front()).first;
        long long mask=(q.front()).second;
        q.pop();
        decompress(mask);
        if(field[depth]==0)
        {
            if((contour[8].encrypt()==0)&&(contour[depth%8].encrypt()==0))
            {
                contour[depth%8].connectivity=8;
                contour[8].connectivity=depth%8;
                for(int c=1;c<5;c++)
                {
                    contour[8].colour=contour[depth%8].colour=c;
                    add(depth,mask);
                }
                continue;
            }
            if((contour[8].encrypt()==0)&&(contour[depth%8].encrypt()!=0))
            {
                if(depth/8<7)
                    add(depth,mask);
                if(depth%8<7)
                {
                    if(contour[depth%8].connectivity==depth%8)
```

```
                        contour[8].connectivity=8;
                    else
                        contour[8].connectivity=contour[depth%8].connectivity;
                    contour[8].colour=contour[depth%8].colour;
                    contour[contour[8].connectivity].connectivity=8;
                    contour[depth%8].decrypt(0);
                    add(depth,mask);
                }
                continue;
            }
            if((contour[8].encrypt()!=0)&&(contour[depth%8].encrypt()==0))
            {
                if(depth%8<7)
                    add(depth,mask);
                if(depth/8<7)
                {
                    if(contour[8].connectivity==8)
                        contour[depth%8].connectivity=depth%8;
                    else
                        contour[depth%8].connectivity=contour[8].connectivity;
                    contour[depth%8].colour=contour[8].colour;
                    contour[contour[depth%8].connectivity].connectivity=depth%8;
                    contour[8].decrypt(0);
                    add(depth,mask);
                }
                continue;
            }
            if((contour[8].connectivity!=depth%8)&&(contour[depth%8].connectivity!=8)&&
(contour[8].colour==contour[depth%8].colour))
            {
                if((contour[8].connectivity==8)||(contour[depth%8].connectivity==depth%8))
                {
                    contour[contour[8].connectivity].connectivity=contour[8].connectivity;

contour[contour[depth%8].connectivity].connectivity=contour[depth%8].connectivity;
                }
                else
                {
                    contour[contour[8].connectivity].connectivity=contour[depth%8].connectivity;
                    contour[contour[depth%8].connectivity].connectivity=contour[8].connectivity;
                }
                contour[8].decrypt(0);
                contour[depth%8].decrypt(0);
                add(depth,mask);
                continue;
            }
        }
        else
        {
            if((contour[depth%8].encrypt()==0)&&(contour[8].encrypt()==0))
            {
                if(depth%8<7)
                {
```

```
                    contour[8].connectivity=8;
                    contour[8].colour=field[depth];
                    add(depth,mask);
                }
                contour[8].decrypt(0);
                if(depth/8<7)
                {
                    contour[depth%8].connectivity=depth%8;
                    contour[depth%8].colour=field[depth];
                    add(depth,mask);
                }
                continue;
            }
            if((contour[depth%8].encrypt()==0)&&(contour[8].colour==field[depth]))
            {
                contour[contour[8].connectivity].connectivity=contour[8].connectivity;
                contour[8].decrypt(0);
                add(depth,mask);
                continue;
            }
            if((contour[8].encrypt()==0)&&(contour[depth%8].colour==field[depth]))
            {

contour[contour[depth%8].connectivity].connectivity=contour[depth%8].connectivity;
                contour[depth%8].decrypt(0);
                add(depth,mask);
                continue;
            }
        }
    }
    printf("%lld\n",dp[64][0]);
    memcpy(search,field,sizeof(search));
    // dfs(64,0);
}
```

用户界面（调试用）

```
<html>
    <head>
        <title>RMTest</title>
        <style type="text/css">
            input {height: 50px; width: 50px; margin: 20px;}
            .field td {border-color: thin solid black; height: 50px; width: 50px;}
            .blank {background-color: white; color: white;}
            .blue {background-color: blue; color: blue;}
            .green {background-color: green; color: green;}
            .red {background-color: red; color: red;}
            .yellow {background-color: yellow; color: yellow;}
            #datain, #dataout {width: 40%; height: 100px; padding: 0;margin: 0;}
        </style>
    </head>
    <body>
```

```
        <table align="center" class="field">
            <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
            <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
            <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
            <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
            <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
            <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
            <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
            <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
        </table>
        <div style="text-align: center;">
            <p>
                <input type="button" class="blue" value="Blue" onclick="status=0;" />
                <input type="button" class="green" value="Green" onclick="status=1;" />
                <input type="button" class="red" value="Red" onclick="status=2;" />
                <input type="button" class="yellow" value="Yellow" onclick="status=3;" />
                <input type="button" value="Clear" onclick="clear_field()" />
            </p>
            <textarea id="datain"></textarea>
            <textarea id="dataout" onchange="read()"></textarea>
        </div>
        <script type="text/javascript">
            var point=Array(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
            var size=Array(0,0,0,0);
            var status=-1;
            var td=document.getElementsByTagName('td');
            for(var i=0;i<td.length;i++) {
                td[i].innerHTML=i;
                td[i].className='blank';
                td[i].addEventListener('click',addpoint);
            }
            function clear_field() {
                status=-1;
                for(var i=0;i<td.length;i++)
                    td[i].className='blank';
                size[0]=size[1]=size[2]=size[3]=0;
            }
            function read() {
                for(var i=0;i<td.length;i++)
                    switch(document.getElementById('dataout').value[i]) {
                        case '1':td[i].className='blue';break;
                        case '2':td[i].className='green';break;
                        case '3':td[i].className='red';break;
                        case '4':td[i].className='yellow';break;
                    }
            }
            function addpoint(e) {
                if(status==0) e.target.className='blue';
                if(status==1) e.target.className='green';
                if(status==2) e.target.className='red';
                if(status==3) e.target.className='yellow';
                if(status!=-1) {
                    point[status*4+size[status]%4]=Number(e.target.innerHTML);
```

```
                    size[status]++;
                    document.getElementById('datain').value='';
                    for(var i=0;i<16;i++) {
                        document.getElementById('datain').value+=String(point[i]);
                        if(i%4==3)
                            document.getElementById('datain').value+='\n';
                        else
                            document.getElementById('datain').value+=' ';
                    }
                }
            }
        </script>
    </body>
</html>
```

视觉部分

deepdark-cv.zip

前端部分

服务器端

breathless-server.zip

用户端

https://github.com/tony3641/RM_Block_Assistant

九、感想与感悟

韩宇轩

经过本次夏令营，我对嵌入式的理解更加深刻。在深入讨论了机械提出的需求后，我发现并不是所有地方只需要 一种控制方式。有时与组员讨论后，换取另外一种思路也可以取得相当不错的效果。

唐浩云

我觉得我的造车技术和开车技术得到了明显的提高。帮助团队提出总体方案，也学会了怎么与团队合作，帮助我队未来机械工程专业的学习有所提高。

王凯博

我学到了如何和组内不同分工的组员们协同工作，与组内不同分工的组员们协同工作提升了我的知识储备锻炼了我的机械设计能力。

卢晟安

我的造车技术和理论分析得到了锻炼，不再是盲目的搭车而是根据需求去简化方案，学习到了怎么才能给团队带来效益，帮助团队解决材料危机，做好了组内经费的管理。

唐志尧

更加好好学习理论知识，锻炼了自己以后的专业知识。也认识了不同领域的朋友，用自己所学的电子工程的知识帮助大家在机器搭建时的问题，学到了如何去帮助队伍。

刘伊芃

与其说前端开发是算法的一部分，倒不如说是用应用在手机上做嵌入。你需要根据手机环境的特性决定交互方式、通讯方式、是否需要云计算等问题。如何给操作手提供一个直观的界面，对我也是一种特别的锻炼。

钟思哲

本次参加夏令营，我的团队意识得到了提升。不同于算法竞赛的孤军奋战，在这里，你需要与通过与机械与嵌入式的交流确定算法的目的；通过与导师的交流确定比赛的规则，制定比赛时的策略与算法的设计。