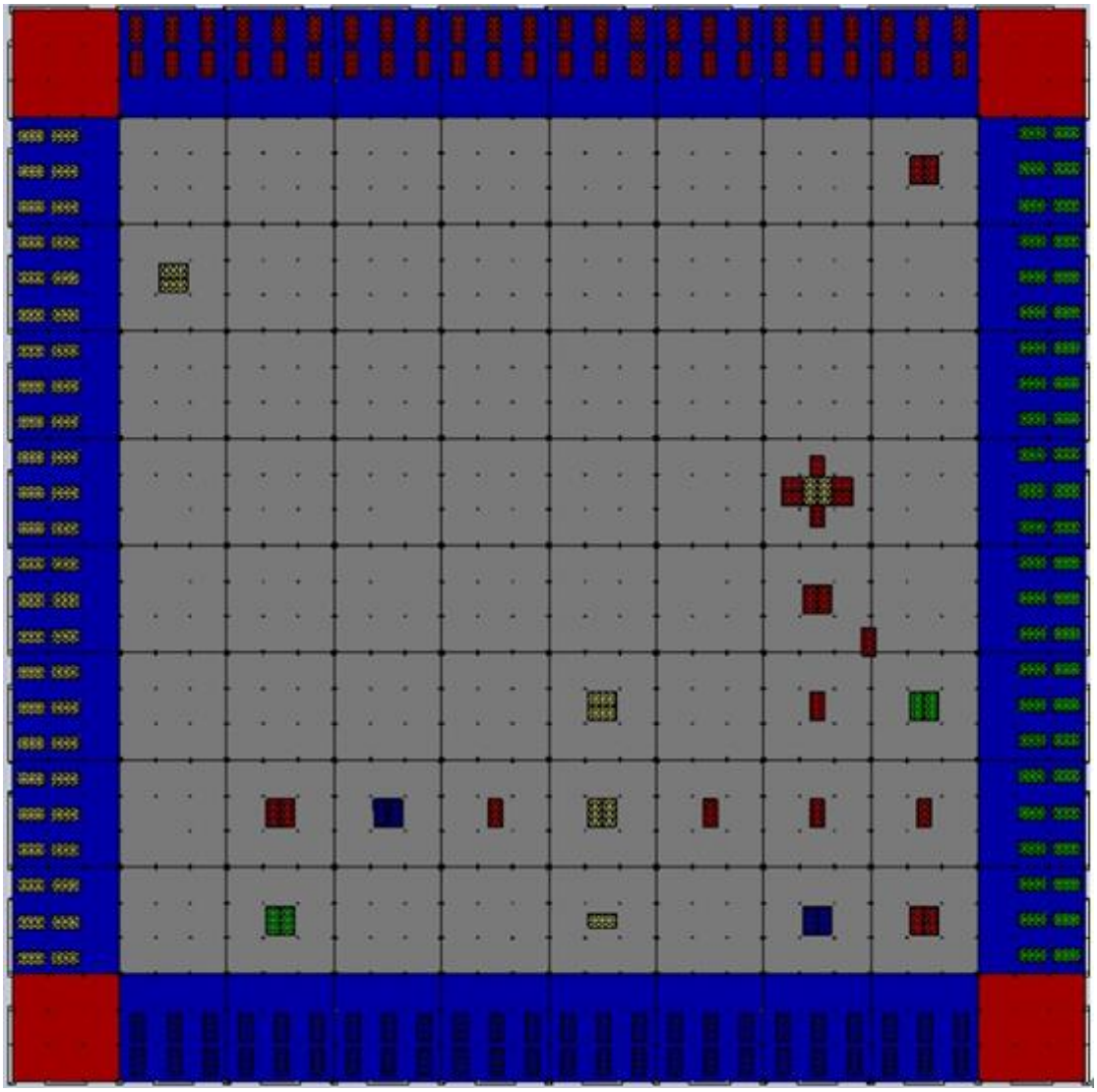
17 组技术报告(机械)

一、技术分析

在机械组内进行了商讨，决定了从启动区出去并且竖向抓取积木块。因为根据最初的场地图纸，纵向砖块的间距较小，车子难以从空隙中通过，所以我们选择从正面竖向抓取积木块。



算法上，第一算法就是使用递归，DP 方面来找出所有解。第二阶段是使用 bfs 方面来找出曼哈顿距离，算分。在 GUI 里，我们添加了巡线，城堡，颜色，状态，城堡高度，等等。

嵌入式上，场地较为狭小，如果被方块铺满的话，机器人想要自动在其中导航，将会变的困难。操作手也很可能因为延迟或视野等原因撞到场地方块上。因此添加巡线系统进行辅助导航，很可能是必要的。

将方块放入存储系统中，需要抬升系统极高的精度。

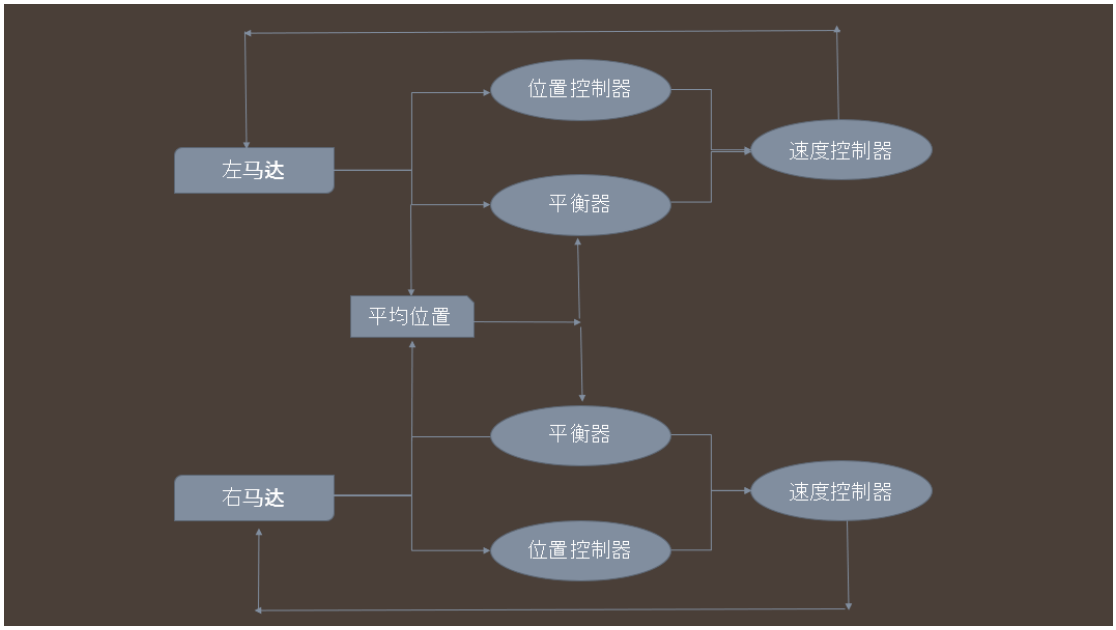
二、所需技术点

算上第二阶段的防守，机械爪必须具备横向和纵向两种夹取方法，最简单的方法就是做一个可以从四个方向夹取积木块的夹子，并且可以旋转。（横向夹块、竖向夹块、储存系统、颜色分类）

最后的储物柜可以单独存方块，每次可以取所需要的颜色。
在算法里，我们使用了递归方法，拿同一个函数不断地处理不同位置的方块。函数里面也是用了 DP 的元素，因为每个方块会根据以前放方块的位置来决定他当前的状态。

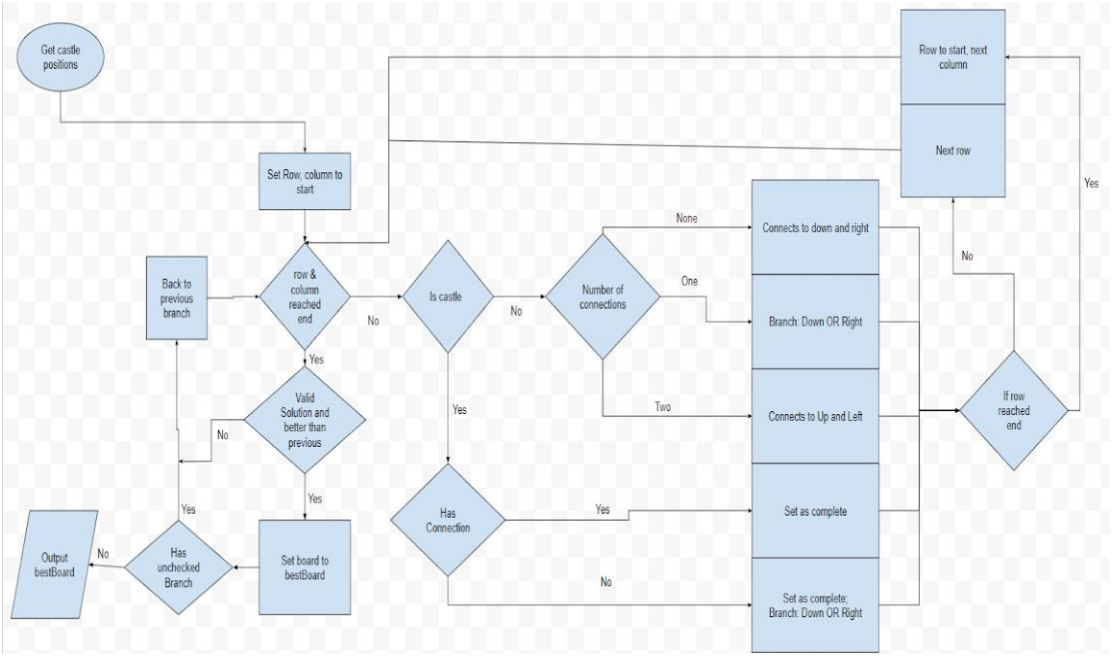
三、总体方案
由夹子夹取后（如果有必要可以由小夹子先夹住，再换到大夹子并送到柜子里），对颜色进行记录，并存入柜中。结构渲染请参照附录。

嵌入式：



用六个 PID 控制器，实现左右升降马达的平衡，和升降机位置的控制。

算法：



在阶段 1，有很多搜索方法。上图是基于插头思想的搜索方法，成为方法 1。同种走法绝不会搜第二次 ，比前期的按路深搜好。

方法 1 大体思路是一个一个格子按顺序填插头，如果当前格子有多种合法填法就分别进入 dfs，进入下层递归后从下一个格子继续填。

下面具体叙述：

初始化各个标记格子信息的数组后从第一个格子进入 dfs。

在 dfs 中，如果当前决策格子是个城堡，那么它只能有一个插头与相邻格子连接，如果它已经有一个插头，那就不用管这个格子了，继续决策下一个格子。如果它还没有插头，那么就看它右方下方两个格子能不能与自己连接。如果有一个能，就更新两个格子的相关信息 ,然后进入下一个格子决策。如果两个都能连 ,就先连一个，更新信息进入下层递归。递归完了把信息还原（即回溯），再连另一种连法，进入递归，事成之后还是要还原信息。

如果当前格子不是城堡，那它决策后必须有两个插头。如果目前它已有两个插头，那就像刚才一样不管他，接着下个格子决策。如果已经有一个插头，那就像上面一

样找能连的格子，一个能连我快乐，两个能连我只好苦逼地递归去。如果这个倒霉格子还没有插头，那就皆大欢喜，因为它只能有一种填插头方式，那就是  型插头。

如果我们惊奇地发现 64 个格子都被幸运地填上了格子，那就顺着插头从城堡开始走一发，如果能走通，就说明颜色设计也稳得一匹，找到一个合法解。

如果要找最优解，不退出，接着搜索即可，碰到一个解看他是不是比当前最优解综合评分高。

解的综合评分，与每个路基格子与此格子计划颜色的资源区距离有关

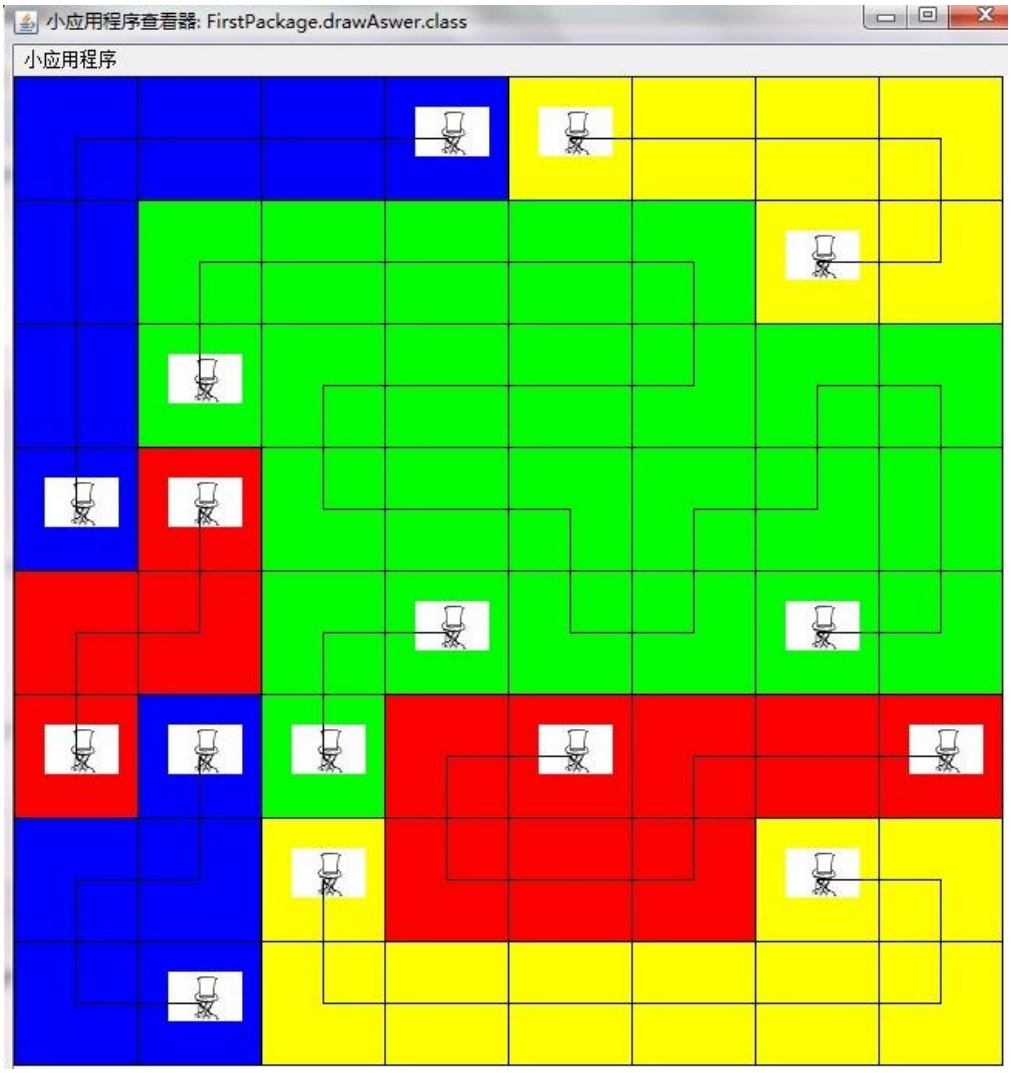
着是目前我组能写出的第一阶段最快的搜索方式，只有很少的格子会有两种选择，绝大部分格子只有一种插头填法，不用 dfs。可是究竟最多有多少格子有两个选择以及怎么证明我们也不会（时间复杂度在理论分析有提）。一般实际情况下我们的底层代码（c++版本且没 GUI）只要零点几秒便能找到所有不同的路线，Java 含 GUI 代码不到五秒就能画出一个最优解。

还有方法 2 是纯搜索加些剪枝。每次进入一层 dfs，就把所有有唯一走法的走掉，判断走法合不合法，除最 naive 的出界转圈颜色匹配外，还有死胡同，孤独空白，孤独颜色等知识水平高些的判断。剩下的都是多种走法的，分别进入 dfs。这种方法不稳定，有的图跑得比 xx journalist 还要快，有的图却要比西方 xx 还要慢。

第二阶段的算法主要是 Alex 写的，算法可以算出每个格子变成城堡能得到多少获益，或者是拔掉路基或城堡能砍多少分

两个搞算法的人讨论出了上述思路，分别用 c++和 Java 实现了代码。框图是 Alex 做的。GUI 两人分别有一套，连士熙是在已经写完 GUI 的 Alex 的帮助下写出的，比较简陋，请到了怪盗基德来客串，GUI 画出了计划的颜色和路线。

效果如下：



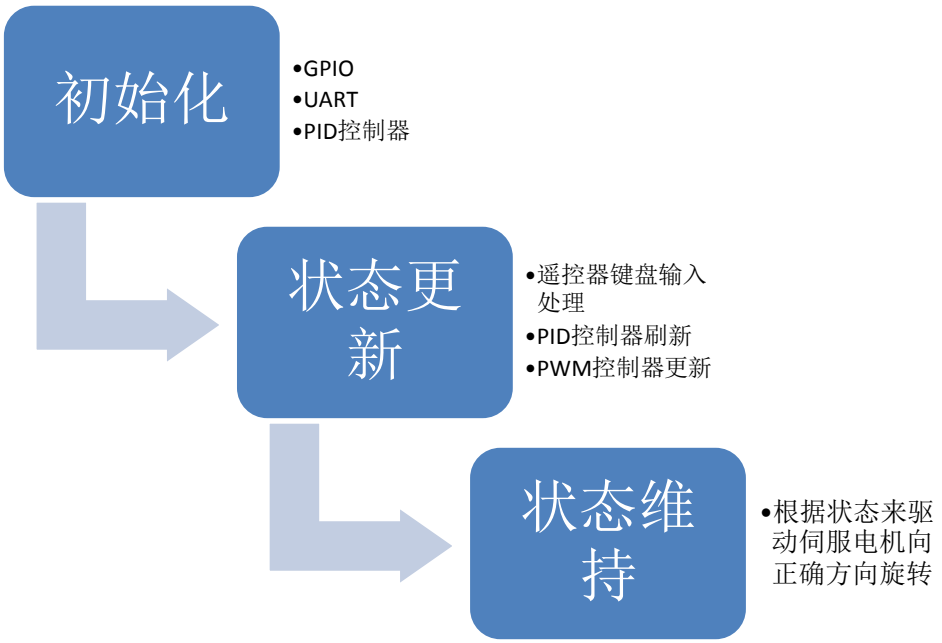
Alex 的 GUI 便比较厉害，功能齐全，还自带一个诡异的谜之 BGM，而且还有一位可敬可爱的，善良的，神勇无敌的，英明神武的，伟岸的，帅气的神秘嘉宾在他的 GUI 中客串，然而助教说不太行，Alex 就让嘉宾领便当了。据说联盟 6 因为成功在 GUI 客串这位嘉宾多次而变成第一了。截图不足以说明 Alex 版 GUI 的强大之处，如果可以的话必定要秀一发。

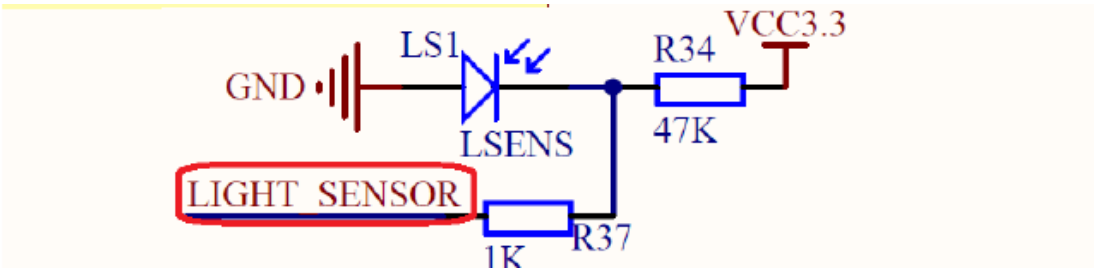
嵌入式上，PID 闭环控制，光敏二极管，ADC，I2C，陀螺仪

四、各模块方案

爪子与升降机：3508 拉线滑轨，爪子由六个 mg995 舵机驱动，四个驱动大夹子，两个驱动小夹子。

嵌入式：





底盘：Chassis design was very debatable at first. The team had very different opinions about the priority of chassis design. We initially believed that we should design the chassis first because of its supportive nature. However, we soon realized that settling the chassis design early in the process could potentially kill many design ideas of storage and elevation systems. Thus, we held off the design of chassis and turned our attention to the making of the elevator system, which proved to be more intriguing to create. When the design and the assembly was done, I returned my attention to finish designing the chassis.

I chose to go with a “sandwich plate” design, where two plates, an upper and a lower, sandwich the motors in between. Although an elevated wheelbase design was also proposed, it was rejected in the end, since we do not see the necessity of crossing over road blocks. The Sandwich plates proved to be structurally solid and it provides easy wiring access. During the matches, the wheelbase held up well and it did its job perfectly.

五、理论分析
两块之间的距离约为 185mm， 而块的宽度为 122，所以轮距设为了 580， 两个前轮可以卡入边上两块的缝隙中

丝杆使用 GM3510 的电机，是因为 GM3510 没有减速箱，所以转速快，大概能够达到 1200rpm 左右（来源于说明书），可以更高效地操作丝杆。

嵌入式：
PID 控制器参数：

elevator_balance: 3, 0.1, 0
elevator _position: 1.5, 0.03, 0.5
elevators _speed: 1, 0.05, 0.5
claw_pos[0]: 30, 0.2, 1
claw_pos[1]: 30, 0.2, 1
flywheel_speed: 1, 0.1, 0.5);

claw_rotate_pos: 2, 0.1, 0.5
claw_rotate_speed: 1.5, 0, 0.2

在 GUI 的结构当中，颜色匹配使算法展示出的结果非常明显，容易被操作手分析及应用。第一阶段里，使用的巡线让操作手和容易看出那个城堡连去那里，显示出的灰格子很快的让操作手知道需要搭那些方块。到了第二阶段，操作手需要对 GUI 多作一些输入，但是用户端始终会给操作手关键的信息。 防守时，会显示得分最多的新城堡位置。进攻时，会选得分最关键的城堡或路基来删除。第二阶段的当场分数计算虽然用处不大，但是能展示目前比赛的局势。

在许多情况下，这个算法可以在二秒内找出一个最佳结果。如果碰到非常简单的问题（八个直线连接）这个程序可以跑 O(n)的时间复杂度。如果碰到非常多解的问题 （4 个同样颜色在每个角落）复杂度可到达 O(2^(n/2))，因为每分支一次，就有很大的几率迫使另一方块获得非分支的状态.因此，算法里使用了一个截断的特征，不让算法超过 5 秒运行。

算法原本使用的是递归与逻辑推理来减去分支的数量。后来发现，这个算法的最差时间复杂度（O(3^n/2)）支持不了所有解中找出最优，从任何一个随机场地都需要 10-20 秒的运算时

间。为了改良这一点，Alex 把算法里放入 DP 的元素，保存以前的结果，也缩小了分支的数量使它变到 O(2^n/2)。虽然这并不能保持“秒杀”结果，再放一个时间监控使这算法能够 5 秒内找出好解。

在制造第二版算法的过程中，发现 DP 算法非常难找出一个填满的板子是不是一个正确的解。Alex 后来发现这是因为位掩码只让每一个方格指向一个方向。发现后，我们给所有路基两个路向。这个不仅让我们的跟线式检测方法变得非常简单，还让我们轻松地往 GUI 上画巡线。之后，连士熙想到了更好的改进：在每个城堡连接后将其标为 true。到了场地填满时，只需看所有城堡有没设为 true，来检验整体解的正确性。
嵌入式，给出控制参数。
写出遇到的困难和问题，重点写出走过的弯路，例如没有进行负载的估计，导致动力选型不合理等（介绍各部分任务的完成者）

六、制作与测试流程
Solidworks 建模及测量，整个车已经在电脑上进行模拟装配以及运动算例，在确认无误之后开始加工组装和调试。

升降机实际测试的精度不理想。原因：1、Limit Switch 因为各种奇怪的 bug，最终没有被启用。2、升降机自重太高而且没有 counterweight， 导致 I 控制器很难短时间内消除所有静态误差。

七、结果与评价
机械上，爪子非常的稳，基本上没有半路掉块的情况。但是爪子到储存柜的衔接仍需要一些调整，存取不是太稳定，只能保证 80%的正确率

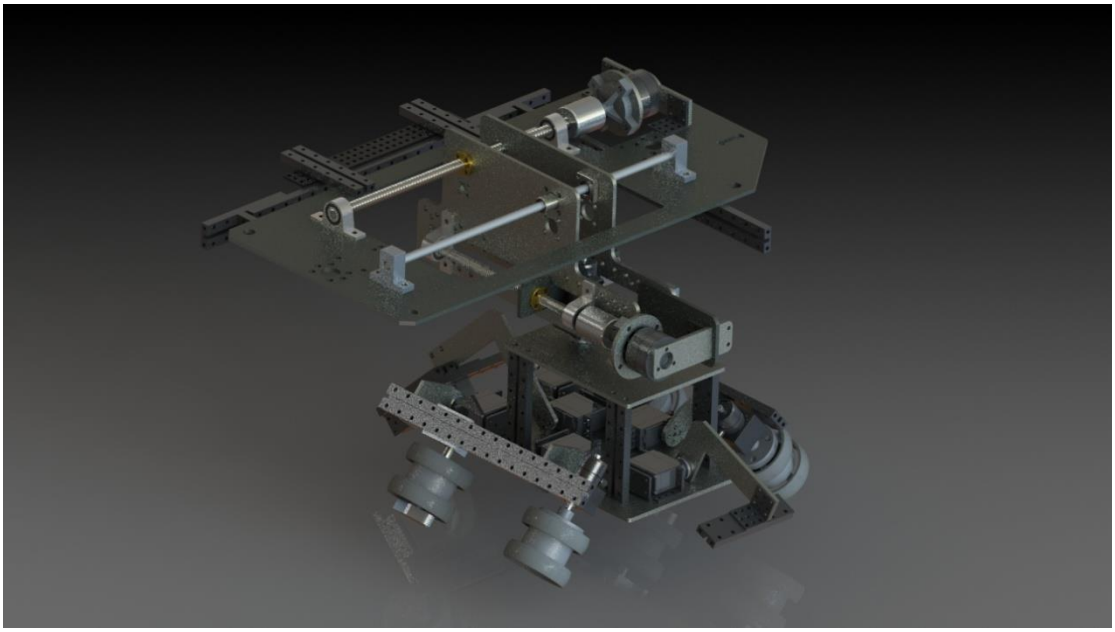
在算法的区域里，因为对插头 dp 的算法少了解，并且对当时的 dfs 暴力算法满足，Alex 和连士熙没有仔细研究插头 DP 的算法，认为这种方法不能实现。一直等到闻知另外一组实施了插头 DP,才开始制作。如果插头找点实现，改进的时间及质量会提高。

八、附录

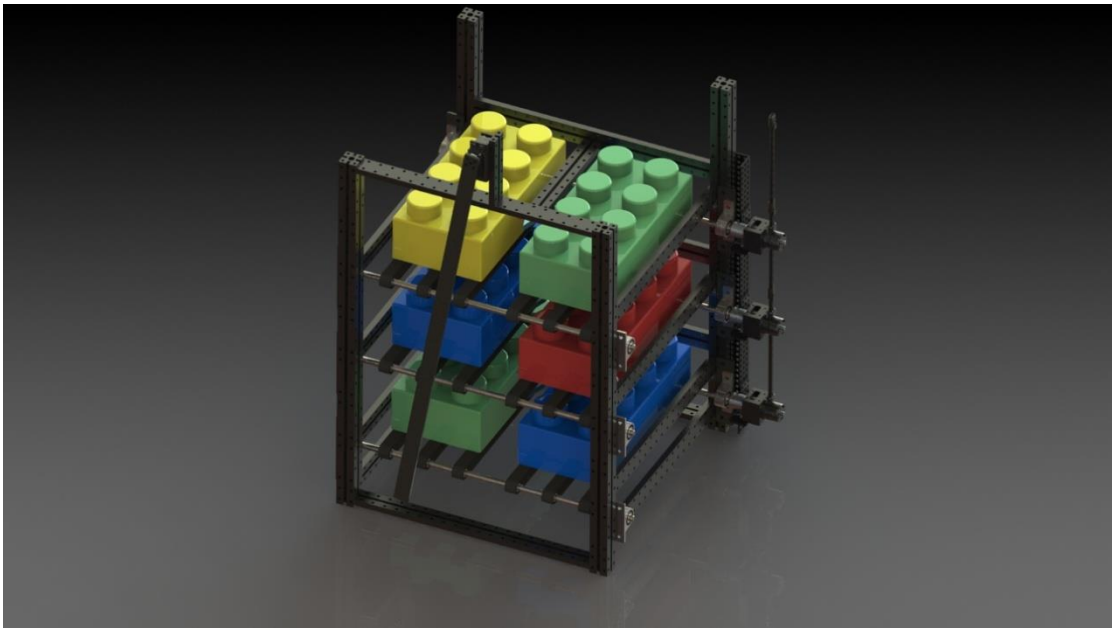
嵌入式代码：https://github.com/Neotoxin4365/dji-hs-summer-camp-2018

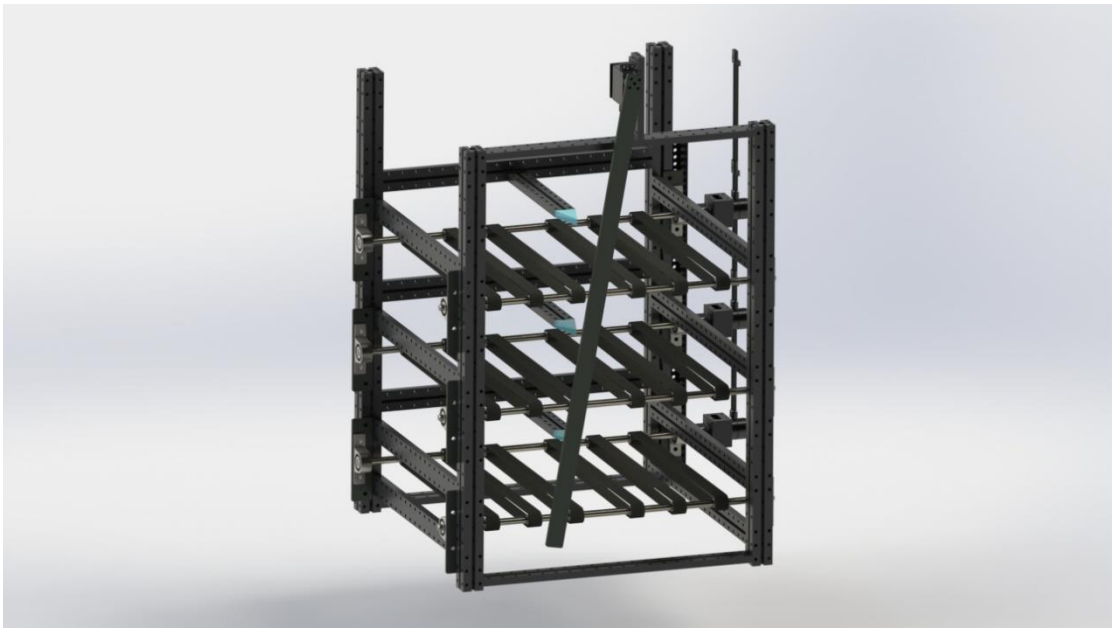
建模渲染：

多轴机械爪

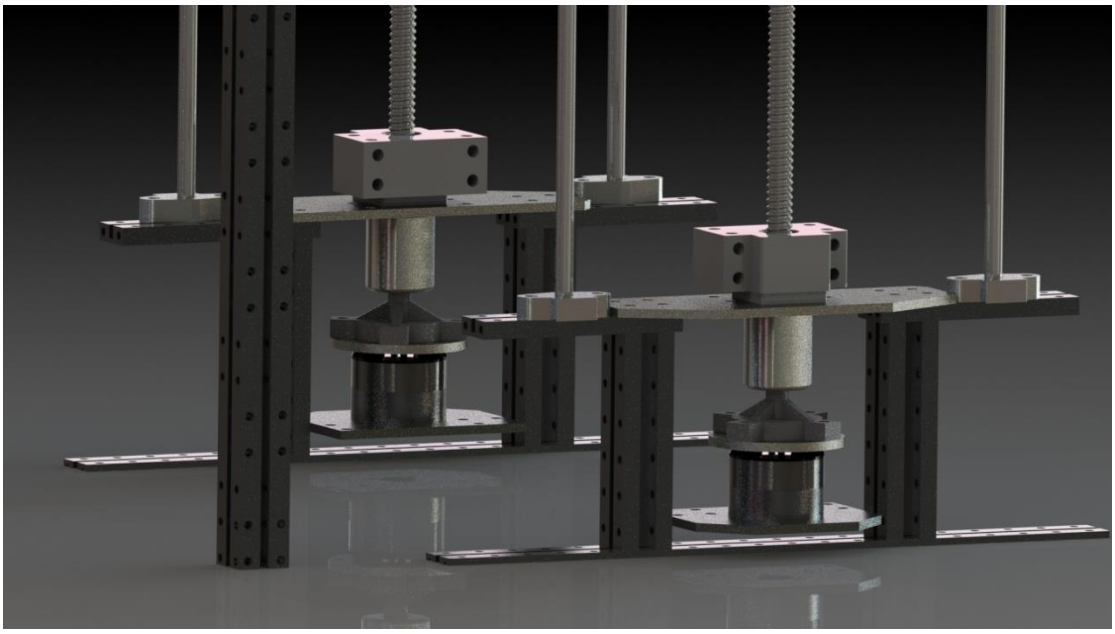


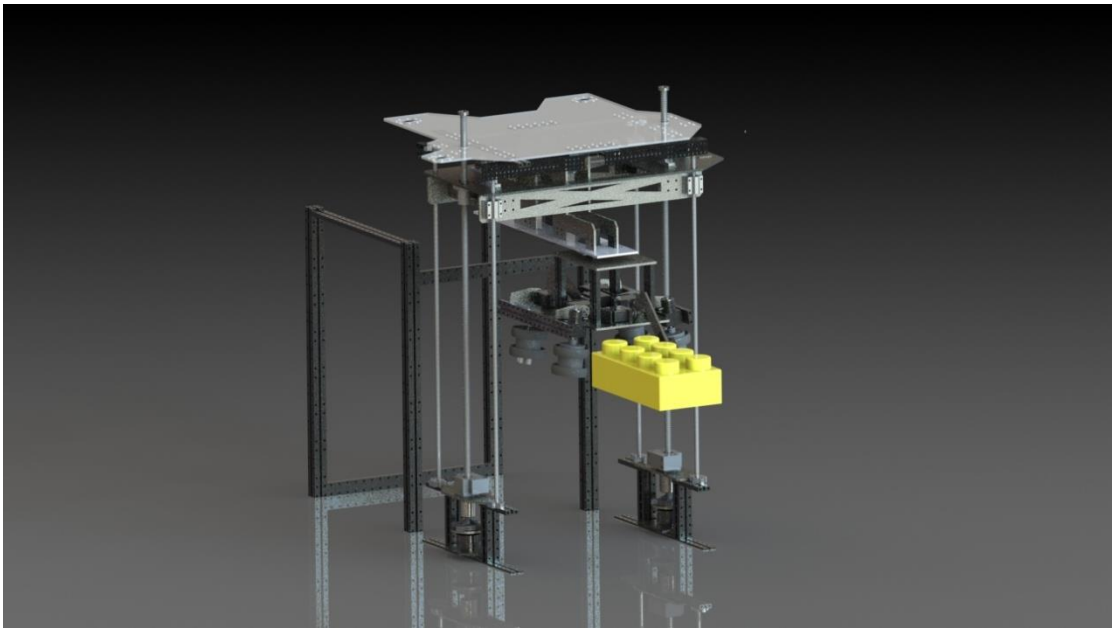
分类存储系统





第一代抬升电梯





第二代机器人整体

九、感想与感悟

这次积累了很多技术上的经验和知识，也交到了很多的朋友。

世上的知识永无止境，我们一定要广泛涉猎不同的技术方向，再从中找出自己感兴趣的部分深入学习。