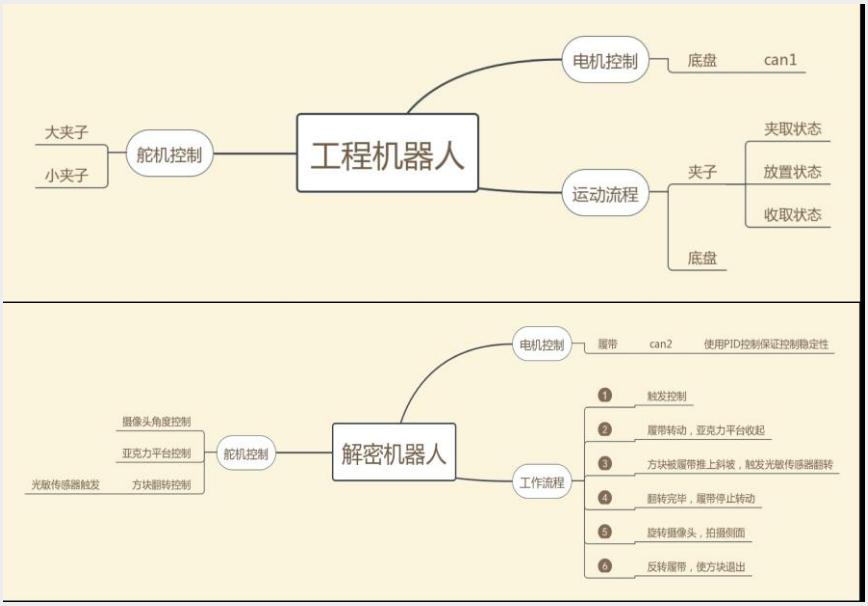
最终项目方案

组别：7组

项目类别	*具体内容*
一、技术分析	***机械*** 让小车能完成指定任务并同时保持结构稳定性。 ***嵌入*** ***巡线：***沿着一条固定的黑线行走,保持车头正对行走方向 ***PID：***保证机器运行稳定性 ***视觉*** 用妙算 1 和 OpenCV 库配合摄像头实现在迷宫中自动识别红方箱子,蓝方箱子 ***算法*** 指导工兵自动移动并以最高效率运行.
二、所需技术点	***机械*** 自定义底盘、双夹、正方形底盘、导轮、自动翻转结构、同步带 ***嵌入*** PID、巡线、串口通信 ***视觉*** OpenCV ***算法*** Floyd、SPFA、A*、BFS、DFS
三、总体方案	我们计划打造一辆自动化运行的双夹工兵车，以及一辆小巧的解密机器人。通过算法智能计算工兵下一个要前往的点以及前往该点的路径，找到密钥和所有密文，并运送给解密机器人识别解密。
四、各模块方案	***机械*** ***解密机器人*** 传送带使用 3 条同步带制作传送带、 使用亚克力板制作翻板 翻板制作接受区提高容错率 使用同步带制作传送带将方块传送至解密区 翻转方块使 4 个面被扫描到 转动摄像头扫描到余下的两个面 反转传送带将方块抛出 用 3508 电机带动传送带 用舵机实现翻转方块及旋转摄像头 ***嵌入*** ***巡线：***7 个灰度传感器，如下图排列: [|=+1 234 5+=|] | - - - - - - |

```
|6  - - -  7|
| - - - - - |
[|=+  ---  +=|]
```

选用多个灰度传感器的好处就是可以降低脱线的几率，减少需要人工干预的次数



视觉

通过观察比赛场地,我们发现:每个有箱子的区域都会存在一个二维码颜色为蓝色或红色的箱子箱子会被放在方形场地块的正中间在机器人顶部安装向前下方看的摄像头,每次路过可能存在方块的角落时先用巡线算法辅助对位,然后进行图片捕捉,最后通过算法进行颜色判断

算法

导航 V1:基于 Floyd 算法预处理出的任意两点间最短路来进行导航。
导航 V2.0:基于 SPFA 算法，实时更新最短路径。
导航 V2.1: 改用 A*算法求最短路。
导航 V3: 预处理出全图最短路树并使用 LCT 维护。
导航 V4.0: 每次计算路径时只运行一次 SPFA 算法。
导航 V4.1: 将 SPFA 算法换为 A*算法。
路径规划: 对于路径规划，我们提出了四种思路，详见理论分析。
对于会车时的处理，我们选择在会车时临时单向删去会车边，然后继续前往原定目的地。这样做可以“强迫”寻找一条新的路径，而不是退一格后又调头回来。

机械

三条同步带可以承受方块且基本不下陷
亚克力板不遮挡环境光使 4 个面可直接被扫描到无需补光
翻板大大提升了操作手的容错率
3508 电机转速足以反转同步带将解密后方块怕抛出
舵机精准控制摄像头旋转角度

五、理论分析

嵌入

巡线:

* 3 号灰度用于被动定位
* 2，4 负责矫正旋转姿态，即 2&4 号被触发后左&右转直到 3 号被触发
* 1，5 负责在机器人离黑线比较远的时候拉回来

一开始本来想直接用旋转，即 2&4 号用的方案。后来在实际测试中，麦轮旋

转有几率发生侧向漂移导致旋转中心改变导致脱线所以 1, 5 触发时机器人会左右平移直到中间 3 号灰度碰到黑线, 表示机器人已经回到线上

＊6, 7 号光感负责检测十字路口, 当 12345 中有两个或以上的灰度碰到黑线时, 机器人会直走一段, 以越过黑色的 RIFD 触发块后继续巡线,这时当 6, 7 中有一个碰到了, 就说明已经到了十字路口, 可以执行下一步转弯, 直行或掉头操作了

巡线需要的下一个场地块位置从算法接口传过来, 在自动模式下直接控制底盘

PID：工程机器人底盘的稳定是比赛的基础。我们通过 PID 控制减轻底盘在急刹车时的打滑，限制总输出防止超速。满杆转弯时可以无碰撞传过“发夹弯。

解密机器人履带速度稳定奠定翻转方块的基础。我们通过在翻转时履带停止，大大增加了翻转的成功率，而在这个过程中，PID 控制是不可或缺的。

视觉

箱子区域的识别:

＊ 通过摄像头获取图片

＊ 因为每次识别时箱子与摄像头的相对位置都是固定的,
可以从图片中裁剪掉大量不含有箱子的多余部分,以节省提高运算效率

＊ 将图片转换至 HSV 空间(更适合颜色识别)

＊ 设置蓝色和红色的 HSV 阈值

＊ 遍历裁剪后的图片,通过比较每个像素的HSV值和阈值,分别统计红色,蓝色像素点的个数

＊ 如果蓝色像素点数目大于一个预设值,方块被认为是蓝方箱子,反之亦然
如果两种像素点个数都小于各自的预设值,说明场地块中没有放置箱子

算法

导航 V1：优点为运算量少（时间复杂度为 O(n^3+t)），速度快。缺点为该算法无法考虑旋转门带来的影响。

导航 V2.0：该算法优点在于易于实现，且出现任何问题时都可以立刻刷新，缺点在于它占用了较多运算资源（时间复杂度为 O(t*n^2)），且在会车时会出现问题（退一格后掉头走老路）。

导航 V2.1：由于 A*是搜索算法，它可以通过在某些地方绕小圈等待旋转门状态切换。而 SPFA 被设计用于求静态图最短路，可以以局部最优作全局最优，故无法通过绕小圈或者停车等待旋转门。(时间复杂度为 O(t*n^2*log n))

导航 V3：将旋转门状态变化视为删边和加边操作。随后发现此算法有误（最短路树删边后变化不仅是重新连一条新边)。(时间复杂度为 O((n+t)log n))

导航 V4.0：通过留出一些误差时间，消除因车速不均带来的影响（在误差时间内始终可以通过的旋转门才可以通过)。同时实现会车时重新导航。(时间复杂度为 O(n^2))

导航 V4.1：原因同 Version 2.1。(时间复杂度为 O(n^2*log n))

路径规划：第一种思路是先捡密钥，然后每次选择路程最近的可能存在方块的位置，每次取满方块后就把它放回解密区。这种方法非常简单，但在后期剩下的点可能非常散乱，路径很长。

第二种思路是使用启发式搜索或者模拟退火，找遍历所有点的最短路径。但是估价函数比较难找，且无法证明这个方案是否最优（我们并不知道密文方块的位置)。

第三种思路是人工预设路径。这种方式基本保证路径优秀，但是这样非常耗费时间。

第四种思路是手动设定一部分遍历序列作为，其余的让机器自己找。这种做法糅合了第一种思路和第三种思路，然而实现起来相当繁琐。

隐藏思路：用无监督型机器学习，实现真・智能路径规划。但是机器学习数据量极大（粗估 10^10)，且算力不足，未被采纳。

六、制作与测试流程	***机械*** 放置方块进行多次测试 ***嵌入*** 巡线方案在实际测试中效果很好，即使是大角度也可以轻松完成修正，给操作手留了很大的容错率. 后期可以加入 PID 控制以实现更顺化的修正. 我们计划在妙算和 STM32 中搭建了 UART 通信，将妙算 1 搭载在工程机器人上，并运行需求性能高的导航算法。我们在妙算 1 运行导航算法，将路径通过 UART 输出给 STM32 上。以此实现自动运行。我们还准备了冗余方案，与妙算 1 组成网络，操作手可以指定目标或路径控制机器人运行。 ***视觉*** 该方案在环境光线稳定的情况下有良好的识别率,但由于迷宫板子的遮挡,场地亮度并不均匀,会对 V(亮度分量)有较大的影响,摄像头也可能存在过曝或欠曝的问题,在不同位置对亮度参数的设定也不太一样. 可能还需要加装一个光线传感器来检测不同位置环境光线的强度，这样就可以实时调亮度阈值以增强适应性,提高识别的准确性. ***算法*** 我们最开始的思路简单粗暴，我们不考虑旋转门将来的状态，只通过当前的系统时间计算出目前旋转门的状态，然后将整个迷宫当成一个静态的图计算出从给定起点到终点的路径。这样做显然可能出现机器人沿着路径走到一半需要经过的旋转门关闭的情况（因为路径是不考虑未来旋转门的状态变化的）。于是我们每个 1 秒重新根据当前的时间与机器人位置计算路径，这样子一定程度上保证了路径能够根据实时情况做出变化，但有较大的可能中途频繁更换路径从而绕远路。考虑到机器人的位置不断变动，每次计算时的起点和终点不断变化，我们使用多源最短路算法 floyd 预处理出迷宫间任意两点在旋转门两种状态下的最短路径。于是这便有了我们的 version1 算法。 后来我们尝试使用单源最短路算法 SPFA。我们首先给出了一种新的处理旋转门的办法，我们假定机器人移动的速度固定，根据时间=路程/速度我们可以由两点之间的最短路程求出两点之间移动的最小耗费时间，于是我们可以依据当前位置移动到某一旋转门的最小时间与当前时间计算出机器人能否通过旋转门。当然这样的计算并不是绝对正确的，这是由于我们总是花费最小的时间来到旋转门前，而这时旋转门可能处于关闭状态，若花费稍长一些时间再来到旋转门，旋转门可能便处于打开状态，这时花费的总时间甚至少于花费最短时间到达旋转门的方案，有的情况下，该算法甚至无法找出可行路径，换句话说“局部最优不等于全局最优”。考虑到机器人速度不匀等实际因素,本方案与 version1 同样每隔一小段时间重新计算一遍路径，这样便导致该方案占用时空资源较多。此为我们的 version2 算法。 为了避免占用过多的时空资源，我们又引入了容错时间的概念。容错时间是一个常量，比如说可以是 2 秒，在机器人速度不是完美均匀的情况下，机器人可能比计算出的预定时间早或者晚一些到达旋转门，我们分别考虑机器人在预定时间减去容错时间或预定时间加上容错时间时到达旋转门，假如这两种情况下机器人都能够通过旋转门，那么我们认为当前方案可行，否则不可行。这样我们不必每个一小段时间如此频繁地调用导航算法，节省了时间与空间，但考虑到容错时间不可能设得很大，机器人的速度必须没有大波动。这个方案是们的 version4 算法，它仍然没有解决 version2 中提到的“局部最优不等于全局最优”的问题。 Version3 算法放在 version4 算法后面讲，是考虑到 version3 算法与之前提到的思路较为不同，属于奇思妙想。这个算法存在一定的错误，且实现非常复杂，我不做赘述。 为了解决“局部最优不等于全局最优”的问题，我们想到了搜索，但是单纯的搜索效率较低，因此我想到了 A*。这里的 A*算法与传统的基于广度优先搜索与二叉

	堆的 A*不同，我采用深度优先搜索以便保留方案，同时采用了 A*的估价函数，这里我简单地将估价函数设为两点之间的曼哈顿距离，在实际应用中已经取得较好的效果，在与他人的交流中有人提出可以使用 version1 算法中求出的多源最短路作为估价函数的返回值，我认为很有道理，但是迫于时间限制没有去实现。在执行深搜 A*前，我先进行一遍广度优先搜索求出一条不经过的旋转门的路径，再以这条路径的长度作为深搜的深度限制，配合估价函数可以起到很好的剪枝效果。由于搜索自身穷举方案的思路，该算法不会只考虑花费最短时间到达旋转门的方案，还会考虑到达旋转门花费时间稍长但总时间更短的方案，因此用该种算法求出的路径往往优于，至少不会劣于 version2 算法的结果。此为我们的 version2.1 算法。 在 version2.1 的基础上我们再引入了容错时间,进而形成了我们的 version4.1 算法。 Version4.1 算法原本将会作为最终的方案与嵌入式进行对接，但由于本组计划搭载在工兵机器人上的妙算 1 出现故障，我们的算法只能再 STM32 上执行。考虑到 STM32 较小的内存空间与较低的算力，version4.1 由于会出现较多的递归调用而被放弃，最终采用 version2.1 算法。 值得一提的是，由于迷宫间相邻点的移动花费相同，version2.1 的 SPFA 算法在本题特定环境下与 BFS 没有差异，因此最终实现时我直接采用了广度优先搜索进行实现。 之前听闻其他组存在计算出导航路线后对沿途经过的密文方块不闻不问的情况，针对这个问题，我们组的专门增加了转向操作，即机器人再沿着导航路径行进的过程中，会在死胡同前停留并转向，以检查死胡同中是否有密文方块，假如存在我方的方块且机械手处于空闲状态，我方操作手将夹取该方块，若机械手已经抓住一个方块，则将新发现的方块坐标记录下来，以后机器人将返回此处取走方块。
七、结果与评价	***机械*** 可以很轻松的将方块丢在翻板上 解秘机构可以成功识别六个面并将方块推出 解密机构基本完成 ***嵌入*** 巡线运行相当平稳，PID 控制也相当不错，美中不足的是妙算 1 固件刷新失败，导致无法启动，最终导致自动化失败。 ***视觉*** 该方案在环境光线稳定的情况下有良好的识别率,但由于迷宫板子的遮挡,场地亮度并不均匀,会对 V(亮度分量)有较大的影响,摄像头也可能存在过曝或欠曝的问题,在不同位置对亮度参数的设定也不太一样. 可能还需要加装一个光线传感器来检测不同位置环境光线的强度, 这样就可以实时调亮度阈值以增强适应性,提高识别的准确性. ***算法*** ***导航：***由于算力的原因，我们最终采用了 Version 4.0。 ***路径规划：***我们原计划使用第四种思路。但是由于我们的机械爪需要先抓取密钥，再抓取密文。经过一番讨论与尝试，我们最终选择了第一种思路。
八、附录	***无***
九、感悟	参加本次冬令营，通过大疆的工程师向我们分享的技术，我们学到了很多东西。我们头一次认识到，原来许多看似高大上的技术，其实只是我们已经学到知识的延伸拓展与整合。通过冬令营的比赛，我更加明白了团队合作的重要性。大家就像是机械结构里的每一个部件、嵌入和算法文件里的每一行代码，只有大家各尽其职、共同合作，才能成功。