2018 RoboMaster 夏令营

技术报告

第 2 组

2018.8.1

目　录

1. 机械部分

1.1 整体设计说明

整体设计上沿用官方 ICRA 车的设计，并结合夏令营的取弹任务，对车进行了三个方面的改动。

首先是在车的后方添加了夹取机构。夏令营的取弹任务主要包括三方面的要点，一个是需要同时取上下高度差有$400mm$的弹药箱，一个是需要将弹药箱的弹药倒入车身上的弹舱中，最后是能够夹住并夹稳弹药箱。三个要点分别对应了夹取机构设计对应的三个自由度，因此所设计的夹取机构包括升降、翻转、夹取三个自由度。

其次是根据夹取机构的位置以及车身的尺寸限制对车的底盘进行改造。车的原始尺寸限制是$600 \times 600 \times 600$，而原车的长度就接近$600$，加上夹取机构必然超尺寸，因此缩短长度尺寸是必需的。除此之外，由于灯柱的原始位置位于车的后方，与夹取机构的位置相冲突，因此需要在底盘的上寻找安放灯柱的位置。

最后是在进行取弹倒弹的动作时，原始的弹舱并不适配倒弹动作，而且结合在取用 400 高度的弹药箱时，为了提高效率造成的倒弹位置较高的情况，需要对弹药箱进行改动。

经过改动后，车体如图所示：

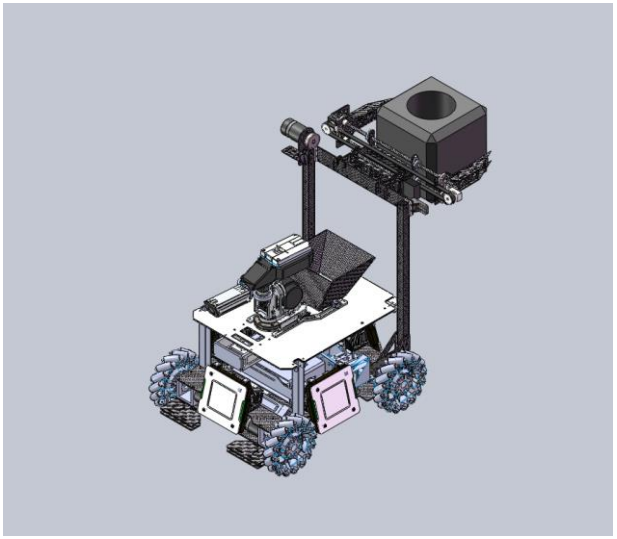


图 1 整体设计

在车体的前方是防撞设施，后方则置有夹取机构。整个车身在长度以及宽度的尺寸上都有所缩减。

1.2 设计方案（可详解重点或亮点部分）

如前文所述，对三个方面分别进行较为详细的描述。

1.2.1 夹取机构

夹取机构整体的设计如下图所示。

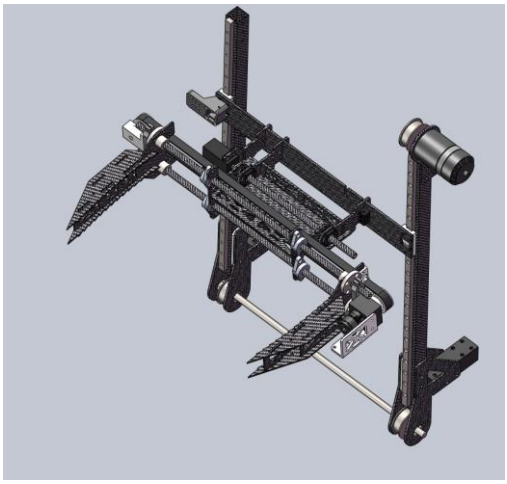


图 2 夹取机构总体设计

该夹取机构由三个自由度组成，分别是夹取、翻转以及升降三个自由度。

夹取自由度采用 2006 电机带动同步带的形式进行传动，利用同步带两边的运动方向相反的原理，将夹子固定在不同的两边上，使得夹子可以实现相对运动，从而实现夹紧放松的动作。夹取部分的机构设计如下图所示。

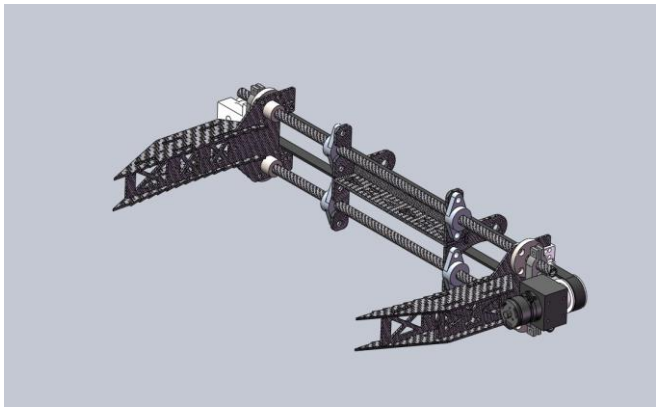


图 3 夹取机构整体

爪子部分如下图所示，采用双层板的结构，提高与弹药箱的接触面积，同时，由于弹药箱是泡沫材料，具有一定的柔性，因此在爪板上采用一定的锯齿，使得在夹紧时齿顶的压强较大可以陷进弹箱，提高摩擦力。同时爪板上的锯齿的分度线具有一定的曲线，使得在夹取时爪子由于直线轴承与碳杆之间存在的间隙造成爪板有一定程度的变形时能有更多的齿顶接触到弹药箱，以提高接触的摩擦力。

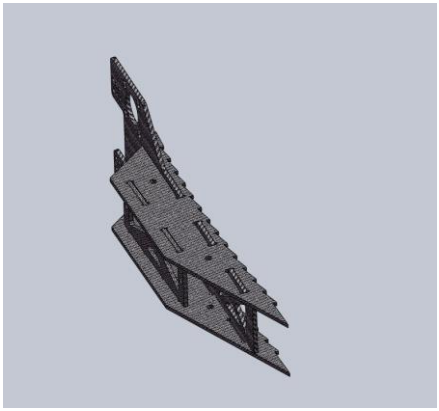


图 4 夹取爪

在直线运动方面，采用碳杆加直线轴承的形式，相比于传统的直线滑轨或者光杆的形式其重量得到大大减轻，同时在运动的流畅度以及结构的强度方面完全足够，但由于碳杆的刚度较低，结构在同步带轮张紧的情况下有少许变形，但影响较小。

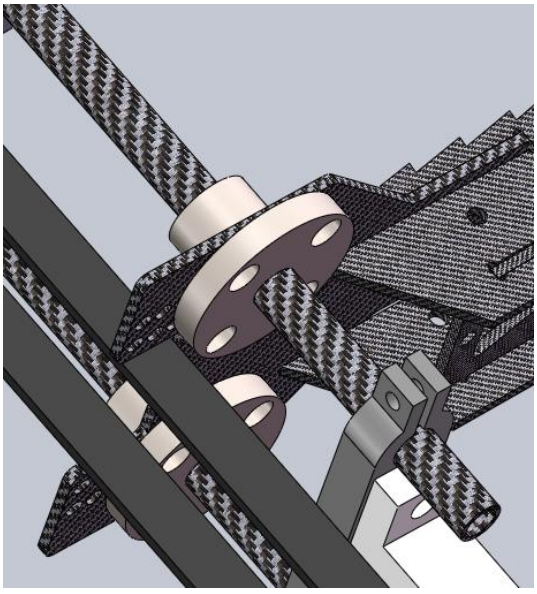


图 5 直线传动方式

整个夹取机构，爪子张开之间的间距为$336mm$，夹紧限位是$185mm$，而弹箱的宽度是$200mm$，这样在夹取方面留有足够多的裕量，能够降低视觉对准的难度，同时在弹药箱具有一定的角度时也能够顺利夹起。

翻转自由度采用 GSW DS945MG 舵机进行驱动，该舵机具有$50kg/cm$的扭矩，留有充分的扭矩裕度，能够保证将弹药箱顺利翻转。同时，由于采用舵机，对于电机的控制难度大大降低，而且由于扭矩足够，能够保证舵机能够执行到位。

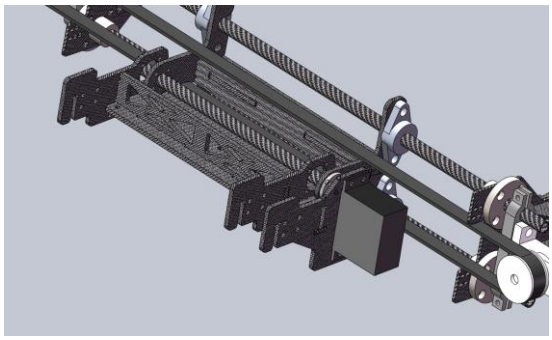


图 6 翻转自由度细节

升降采用同步带加直线导轨的形式进行传动，同时，在底部增加光轴并在另一侧布置同步带，使得在升降的过程中同步性更好，避免爪子因为单边支撑受力不均导致倾斜。

整体结构多采用碳板通过榫卯结果进行连接，在提高结构强度、刚度的同时降低整体的重量，相比于采用角件加螺栓等形式，其重量大大降低，缺点是在配合较紧的情况下会较难拆装，而且多次拆装可能造成零件之间的配合松动。

1.2.2 底盘与弹舱

底盘方面，在官方车的基础上没有较大改动，由于长度有限制，在增加了夹取机构的前提下，需要减小车的长度，首先需要将车体的前后轴距减小，其次，在第二版车中，为了减小车身的尺寸，以提高车身的灵活度，使之能够通过更为狭小的地形，还进一步降低了车身的宽度。同时，由于原版车的裁判系统位于车身的后方也即是位于夹取机构的位置，因此需要移动裁判系统的位置，考虑了车身侧方等位置后，决定将裁判系统置于车身的下层板与中层板之间，原因是这两层板之间的空间比较充裕，原本无其它东西放置。同时考虑到裁判系统需要设置，将裁判系统的设置面板下置，当需要进行设置时，便可以翻起车身进行设置。

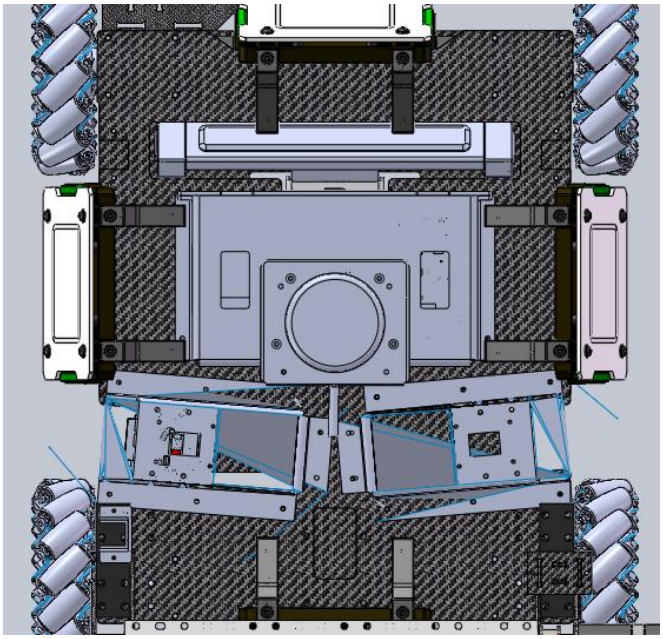


图 7 底盘改装

弹舱方面，考虑到官方车原本的弹舱的高度较低，且位置较为靠前，若沿用此设计，对于夹取倒弹机构的设计难度较大，且在夹取位于墙上的弹药箱时，需

要将弹药箱下移才能够倒入，效率较低。并且，由于弹药箱的前板高度较低，倒弹时子弹由于惯性可能会飞出弹舱范围。基于以上三点原因，重新设计了弹舱。

所设计的弹舱采用底座加挡板组成，在形成如图所示的基座之后，在周围围上一圈塑料网提高弹舱的高度，使倒弹更加顺利。

图 8 弹舱基座

1.3 最新的装配图（附零件清单）

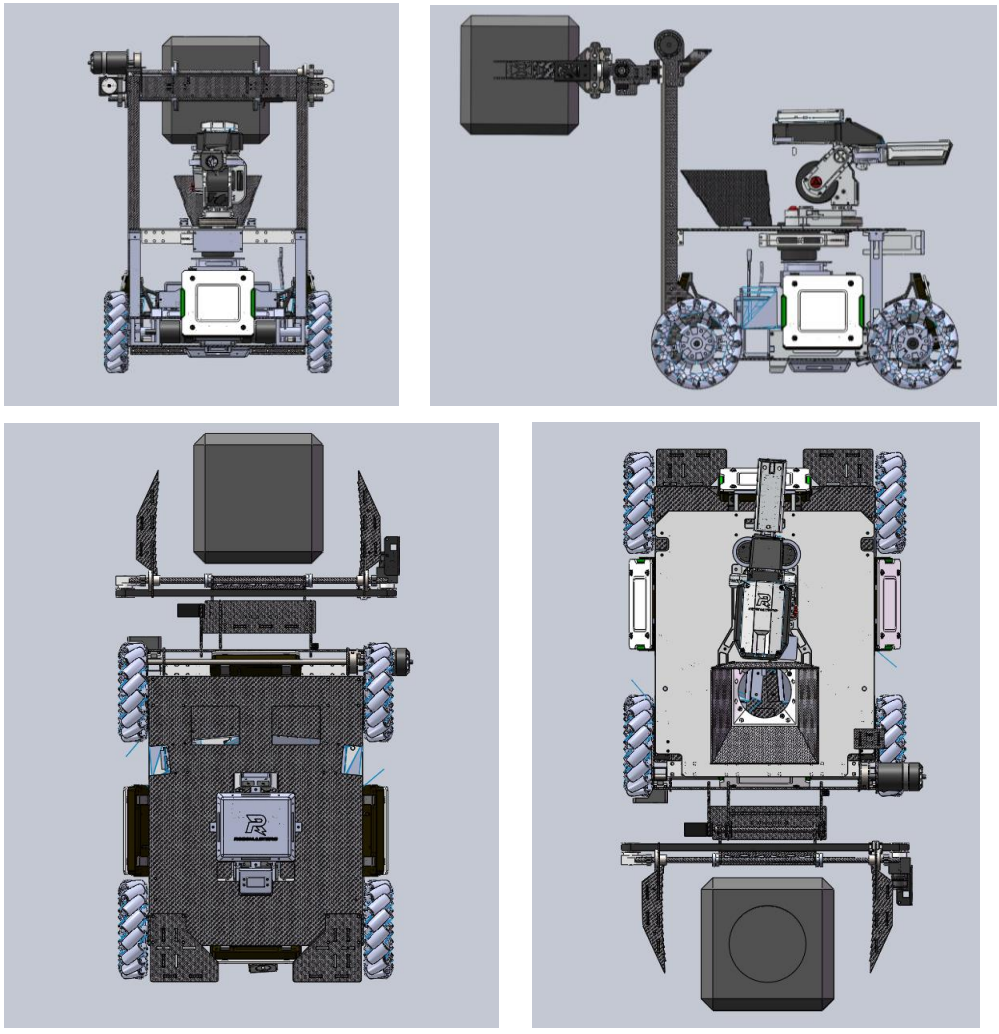


图 9 整体装配图（从左到右、从上到下依次：主视图、侧视图、仰视图、俯视图）

附零件清单：

项目号	零件号	说明	数量
1	底盘下面板		1
2	3508 电机架		4
3	M3508 减速电机		5
4	内径 17 外径 30 厚度 7 轴承		4
5	轮毂联轴器		4
6	麦轮镂空（右）		4
7	底盘上面板		1
8	Rm 小装甲（带原版支架）		4
9	装甲板限位检测		2
10	裁判系统主控灯柱		2
11	新版电池架（外售版）		1
12	底盘下面板(新)		1
13	新 RFID 模块 01		1
14	RFID 底板		1
15	2020 角铝		1
16	型材外部卡键 01		2
17	ICRA 云台碳底板 02		2
18	TX2 保护架		1
19	TX2		1
20	TX2 拓展板		1
21	雷达顶盖 02		1
22	雷达座		1
23	41mm		4
24	激光雷达底座		1
25	激光雷达上盖		1
26	设备盖		1
27	绿联 usb3.0HUB		1
28	ORICOUSB3.0HUB		1
29	ICRA6623 电机 V2-hdl		2
30	ICRARU44 回转平台		1
31	ICRA 侧板 A		1
32	ICRA 供弹弯管		1
33	ICRA 内 20 外 27 厚 4 轴承		2

34	ICRA 发射机构末端弯管		1
35	ICRA 枪管		1
36	ICRA 侧板 B		1
37	ICRAPITCH 盖		1
38	ICRA 云台后盖		1
39	ICRAPITCH 电机轴 MXL32 齿同步轮		1
40	ICRApitch 电机抱轴		1
41	ICRApitch 电机抱轴端盖		1
42	ICRA 摩擦轮安装板		1
43	ICRA 主控支板		1
44	ICRADJI Snail 2305motor		2
45	ICRA 摩擦轮内衬		2
46	ICRA 摩擦轮上盖 1		2
47	ICRA 摩擦轮包胶		2
48	ICRA 压轮架		1
49	ICRA 内 2 外 5 厚 2.5 轴 承		3
50	ICRA3M 同步带 pitch 轴		1
51	ICRA 触碰开关绿色小号		1
52	ICRA 绿色小号行程开关 盒 子		1
53	ICRAyaw 转轴同步轮		1
54	ICRAyaw 电机轴套		1
55	ICRAyaw 同步轮垫片		1
56	ICRA 底板 n0.2		1
57	ICRAyaw 壳		1
58	ICRAyaw 电机 XXL36 齿 同 步轮		1
59	ICRAYAW 限位		2
60	ICRA 新相机图传		1
61	ICRA 摩擦枪头短版_3		1
62	ICRA 弹仓 n05		1
63	ICRADR16 接收机		1
64	ICRA 主控板支撑柱		7
65	ICRAyaw 同步带 2		1

66	ICRAPITCH 轴 MXL32 齿同 步轮		1
67	ICRApitch 限位		1
68	ICRAyaw 口弯管 1A		1
69	ICRAyaw 口弯管 1B		1
70	ICRA 云台底板		1
71	17mm 测速模块基体 n0.5		1
72	17mm 测速模块枪标 n0.5		1
73	17mm 测速模块上壳 n0.5		1
74	DJIM2.5- 6 扁头内六角螺丝		11
75	M3 滚花铜螺母		1
76	测速模块主控板 2018n0.5		1
77	枪管（旧）		1
78	17mm 测速模块侧灯板 An 0.5		1
79	17mm 测速模块侧灯板 Bn 0.5		1
80	M2.5 滚花铜螺母		13
81	DJIM3-8 扁头内六角螺丝		1
82	RM2017_DZ_SP0005_V1 _1207C		1
83	M8 公直头外模		2
84	开发板 DB8548		1
85	单轴陀螺仪		2
86	扎线板		1
87	ICRA 弹仓快拆底座 n20		2
88	ICRA 弹仓快拆套筒 n20		2
89	ICRA 弹仓快拆顶盖 n20		2
90	2mm 直径钢球 n20		8
91	ICRA 弹仓快拆外套筒 n21		2
92	弹簧		2
93	ICRA 云台前盖		1
94	ICRA 下供弹管		1
95	ICRAyaw 口弯管 2		1
96	ICRA 主控开发板		1

97	弹仓内钣金		1
98	ICRAyaw 张紧装置		2
99	ICRAM4 内六角螺丝		2
100	ICRA 内 4 外 12 厚 4 轴承		4
101	ICRA 云台转接板底座		1
102	ICRA 云台转接板		1
103	垂直供弹管 A		1
104	垂直供弹管 B		1
105	WIFI 天线		2
106	镜头保护片		1
107	摄像头安装架		2
108	安装基座		1
109	直线导轨		2
110	滑块		2
111	连接板		1
112	400 碳纤维管		2
113	碳管连接		2
114	2006 电机座		1
115	M2006 P36 直流无刷减速电机		1
116	内径 6 立式轴承座		1
117	6mm 光轴		1
118	同步带		1
119	法兰直线轴承		4
120	40 齿同步带轮		2
121	爪子架板		2
122	爪板		4
123	支撑板整体		2
124	弹药箱		1
125	8mm 卧式轴座		4
126	旋转支架		2
127	舵机		1
128	8mm 碳纤维 200		1
129	25T 小舵盘		1
130	同步带固定		1
131	同步带固定下面板		1
132	支撑臂		2
133	S688ZZ 深沟球轴承		2
134	轴承挡板		4

135	主架板 2		1
136	舵机架板		1
137	主架板架梁		1
138	旋转支架连接竖		1
139	旋转支架连接横		1
140	铝方管		2
141	3508 电机支架		1
142	50 齿同步轮		3
143	立柱架子		4
144	MakeBlock 铝型材		2
145	4mm		2
146	矩形连接件		1
147	8mm 光轴		1
148	拖链支架		1
149	摄像头保护套		1
150	摄像头固定板		1
151	光电传感器托		1
152	MakeBlock 铝型材 290mm		1
153	弹仓座(2)		1
154	弹舱侧板		2
155	弹舱前板		1
156	弹舱后板		1
157	电池保护壳		2
158	电池		1
159	uwb 底座		1
160	uwb 基座		2
161	uwb 支撑肋		1
162	摄像头固定板(大)		1
163	摄像头保护套(大)		1
164	护栏基座		1
165	护栏挡板		1

1.4 还可以优化的方向

针对于比赛中出现的几点不足，我认为可能有以下几点可供改进。

首先是夹取机构驱动方式。尽管使用电机加同步带的方式有例如速度较快，夹取力足够，容易调节行程等优点，但是在比赛以及调试过程中出现了当障碍块是斜置时，由于电机的位置控制一直不到位，导致电机电流过大产生堵转，堵转之后 2006 电机重启导致归位不正常，影响后面的夹取。因此个人认为采用气缸会是一个更加稳定的选择。由于气缸的行程本身就是固定的，并且只有伸出以及缩入两个状态，其控制更为简单，并且对于本次赛题所要求的负载而言，只需要很小的气缸便可以提供足够的夹紧力。尽管气缸需要额外携带气瓶、气阀、气管等，可能对于小步兵车来讲有点冗重，但如果布置得好的话其实车身上仍然有充足的位置。

其次是底盘应该加适当的悬挂。原版官方车带有前桥悬挂，使得前轮有一段较小的活动空间。在去掉前桥后，步兵车的移动明显有些飘。尽管比赛场地相对来讲是比较平整的，但是仍然有些许坑洼，并且由于加工以及装配的原因，并不能够保证四个车轮充分与地面进行接触。因此还是需要在底盘上加上适当的悬挂保证车轮的充分接触而使之不会打滑。当然打滑与车身重量减轻也有一定的关系，所以在给车身减重的同时也应当考虑一些稳定性的问题。

最后是车身的尺寸应当可以进一步缩小。整个车身的布置来讲并不是很紧凑，可以对空间进行进一步的压缩。对于本次比赛的场地来讲，由于随机障碍物的布置，自场地的某些位置会出现一些比较狭小却能够供车通行的通道，这些通道由于路径规划算法的原因，对于原版车来讲通过性是比较差的。但是将车身尺寸改小之后，通过修改一些膨胀系数之类的参数，可以使机器人更加有效地利用这些通道，从而提高在场上的作战优势。

2. 嵌入式部分

- **整体方案**

步兵车部分以 stm32f427IIH 为主控，基于 freertos 操作系统的多任务控制。主要包括了底盘控制任务，云台控制任务，错误检测任务，串口发送任务。

主要方案，底盘的四个轮子接受 TX2 根据地图和导航信息解算出来的 x,y 方向和 w 轴的速度根据运动学解算得到每个麦轮的独立转速，再根据速度闭环的 PID 通过 can 线发送给相应的电调；云台根据 TX2 发送的敌人相对于视频图像中心的 x，y 方向和距离信息，结合云台两个电机码盘返回的当前角度信息计算出敌人位置的相对角度值，利用云台的两个双环 PID 做出角度值的位置闭环控制。

抓取竖直和水平采用电机控制位置，由舵机控制夹取之后的倒弹位置，夹取的电机添加在 can2，增加光电传感判断是否夹取的指令。

算法部分，本次任务使用到的算法主要包括控制算法和滤波算法。控制算法双环 PID 控制算法，结合实际效果配置的分段 PID 控制算法，用于实现敌人实时跟踪的前馈控制算法。

滤波算法主要包括处理摄像头信息的 kalman 滤波，使云台处理相机信息更加具有实时性。ramp 斜坡响应使云台控制变的平滑。

内部结构

- 多任务环境，相比传统步兵控制框架，可以直接实现多线程逻辑，以及阻塞任务；
- 较为完备的底层上层通信协议，实现上层对步兵各个机构模块的反馈信息获取和控制；
- 内部程序模式切换、数据、控制各个任务隔离处理，方便添加和裁剪功能；
- 底盘、云台、发射等机构，内部的控制模式去耦合，便于不同需求控制时的模式切换；

程序框架

- 基于 HAL 库的 BSP 层，主要提供 can、uart、spi、flash、io 等板级的配置和通信接口；
- 数据交互层，这里是程序中会调用 BSP 层函数唯一的地方，主要为应用程序和硬件设备的数据交互；
- 通信层，负责数据和 uart 硬件间发送和接收，以及这些数据的打包、解析，包含协议部分；
- 信息获取，交互层可以直接使用的数据、通信层解析好的数据，在这个任务中获取后转化为反馈和控制信息；

- 模式切换，在不修改控制架构的情况下，实现现有机构功能模式的自定义组合唯一需要修改的地方；
- 控制任务，云台、底盘、射击这三个机构的控制，包含这些机构的示例模式。

模式选择梳理：

云台

```
typedef enum
{
  GIMBAL_RELAX              = 0,        云台断电
  GIMBAL_INIT               = 1,        云台由断电状态回中
  GIMBAL_NO_ARTI_INPUT = 2,  无手动控制信息输入模式
  GIMBAL_FOLLOW_ZGYRO   = 3,云台正常的手动控制
  GIMBAL_TRACK_ARMOR    = 4,      追踪装甲，保留
  GIMBAL_PATROL_MODE    = 5,巡逻模式，云台 yaw 保持周期运动
  GIMBAL_SHOOT_BUFF     = 6, 打大符模式，保留
  GIMBAL_POSITION_MODE = 7, 相机在底盘上时使用
  GIMBAL_RELATIVE_MODE = 8,  相机在云台 yaw 轴上时使用
} gimbal_mode_e;
```

底盘

```
typedef enum
{
  CHASSIS_RELAX                 = 0,      底盘断电
  CHASSIS_STOP                  = 1,      底盘停止/刹车
  MANUAL_SEPARATE_GIMBAL        = 2,      手动模式底盘云台分离
  MANUAL_FOLLOW_GIMBAL          = 3,          手动模式底盘跟随云台
    CATCH                       = 4,          抓取模式
  AUTO_SEPARATE_GIMBAL          = 5, 底盘和云台分离模式，旋转、平移受上层控制
  AUTO_FOLLOW_GIMBAL            = 6,    底盘跟随云台，平移受上层控制
DODGE_MODE                      = 7,  底盘躲避模式，底盘固定旋转，
平移不受控制

}
```

射击结构

```
typedef enum
{
  SHOT_DISABLE          = 0,          发射机构断电
  REMOTE_CTRL_SHOT      = 1,    遥控器控制发射机构
```

```
    KEYBOARD_CTRL_SHOT = 2,      键盘控制发射机构
    SEMIAUTO_CTRL_SHOT = 3,      单发、连发上层控制
    AUTO_CTRL_SHOT       = 4,          摩擦轮开关、速度、单发、连发全部上层控制
} shoot_mode_e;
```

- **夹取模块**

第一阶段：抓取弹药箱运用舵机实现垂直方向的移动和水平的夹取，由 stm32f427 板上输出的 PWM 波控制舵机的转动位置。我们在底盘模式里添加了抓取模式，当导航到达目的地后，等待视觉校准后，将信号传至 PC，PC 收到信号后底层接受上层数据控制底盘，实施抓取。

底盘模式：增加一个逻辑变量（光电触发），当上层给底盘发送抓取模式后，底层控制底盘向相对于车子的后方运动而且爪子放下并打开后等待光电触发，光电触发后，添加了 0.1 的计数延时后进行抓取确保弹药箱到达理想夹取位置，然后实施夹取。夹取采用舵机，需要通过 stm32 输出 PWM 波控制舵机转动位置。了解上下层通信协议，在通讯方面在上下层实现了信息互通。

```
case CATCH:
{
  if(lightsensor_state == 1)
  {
    turn_down_servo();
    static int cnt = 0;
    chassis.vy = 0;
    chassis.vx = GO_BACK_SPEED;
    chassis.vw = 0;
     if(get_leftlightsensor_state() == 0 || get_rightlightsensor_state() == 0)
     {
       if(cnt > 10)
       {
       lightsensor_state = 0;
       cnt = 0;
       }
       cnt++;
     }
  }
  else
  {
    chassis.vx = 0;
    chassis.vy = 0;
    chassis.vw = 0;
    if(putdown == 1)
    {
      delay_state = 0;
    }
    catch_bulletbox();
  }
}break;
```

如果只加有读取光电信息的话，要执行这套动作的时候一直需要触发，否则就完成不了动作，为避免这种情况的，所以要增加光电的逻辑变量，是为了让光电触发一次之后可以完整的执行完一整套动作。

第二阶段：更改机械结构后夹取弹药箱的模式发生变化，水平夹取和垂直移动运用电机驱动同步带连接，由于 can1 已经被占用满，所以电机添加到 can2 通讯接口并要设置 ID 来接受和发送信号，can2 都电机采用了双环 PID 控制：

外环为速度环，设定为内环输出量，反馈值为电机获取角度。

内环为位置环，设定值电机期望角度，反馈值为电机实际转动角度，最终输出电流作用于电调。

在调试时先调内环位置环，再调节外环速度环。通过 P 项即可实现电机的控制， P 越大， 电机的响应越快， 但 P 过大时会产生超调， 这时电机会抖动， 减小 P 项即可消除。积分项 I 主要用于消除静态误差， 对于精度有要求的地方需要加上 I 项，比如自动射击。微分项 D 项相当于一个阻尼的作用，防止过调。

在调节垂直方向的电机时需要加一个前馈，用来抵消重力带来的影响，从而更稳定的调节 PID。

实现抓取流程：

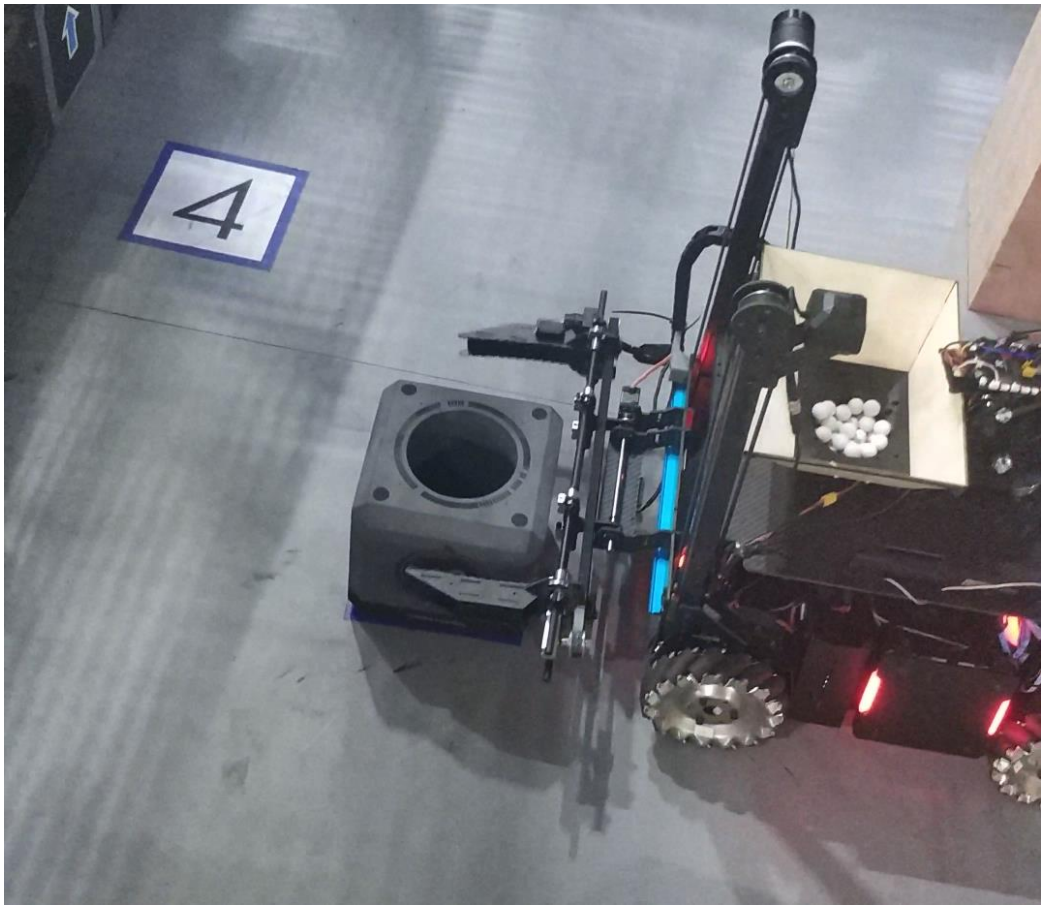
导航到达目标位置发送——>底盘执行对应高低操作等待视觉识别——>视觉开始校准——>校准完成后发送信息到 PC——>PC 处理信息，下发指令到底层（选择是否抓取）——>执行对应操作——>电机复位

在该过程运用了较多的逻辑变量，例：逻辑变量 clawflag，该变量控制着模式选择接受一次 PC 的信号后发生变化，执行完对应的一整套程序后才对逻辑变量进行重置操作，确保指令运行的稳定性。

```
void complete_grabbing(void)
{
      if((claw.mode == 1 || claw.mode == 2) && (clawflag == 0))
      {
        clawState = claw.mode;
        clawflag  = 1;
      }

    switch(clawState)    //选择高低抓取模式
    {

      static int cnt = 0;
      case 1:   //高
      {
        if(claw.decide == 0 && claw.mode == 1)     //判断是否抓取
        {
        chassis.vx = (float)pc_recv_mesg.chassis_control_data.x_spd;
        chassis.vy = (float)pc_recv_mesg.chassis_control_data.y_spd;
        chassis.vw =  pc_recv_mesg.chassis_control_data.w_info.w_spd;
        claw_target(OPEN, UP, BW);
        claw.claw_succeed = 0;
        }
        else if(claw.decide == 1 && claw.mode == 1 )
        {
          claw_target(OPEN, UP, FW);

          if(get_lightsensor_state() == 1 && clawstop == 0)
          {
            chassis.vx = -300;
            chassis.vy = 0;
```

抓取上方弹药箱

```
case 2:   //低
{

  if( claw.decide == 0 && claw.mode == 2  )
  {
  claw_target(OPEN, 750, BW);
  chassis.vx = (float)pc_recv_mesg.chassis_control_data.x_spd;
  chassis.vy = (float)pc_recv_mesg.chassis_control_data.y_spd;
  chassis.vw =  pc_recv_mesg.chassis_control_data.w_info.w_spd;
  claw.claw_succeed = 0;
  }
  else if( claw.decide == 1 && claw.mode == 2 )
  {
    claw_target(OPEN, DOWN, FW);

    if(get_lightsensor_state() == 1 && clawstop == 0)
    {
      chassis.vx = -300;
      chassis.vy = 0;
      chassis.vw = 0;
    }
    else
    {
      lightsensor_state = 1;
      chassis_stop_handler();
        clawstop = 1;
    }
    if(lightsensor_state == 1)
      {
      if(cnt < 100)
      {
      claw_target(CLOSE, DOWN, FW);
        cnt++;
      }
```



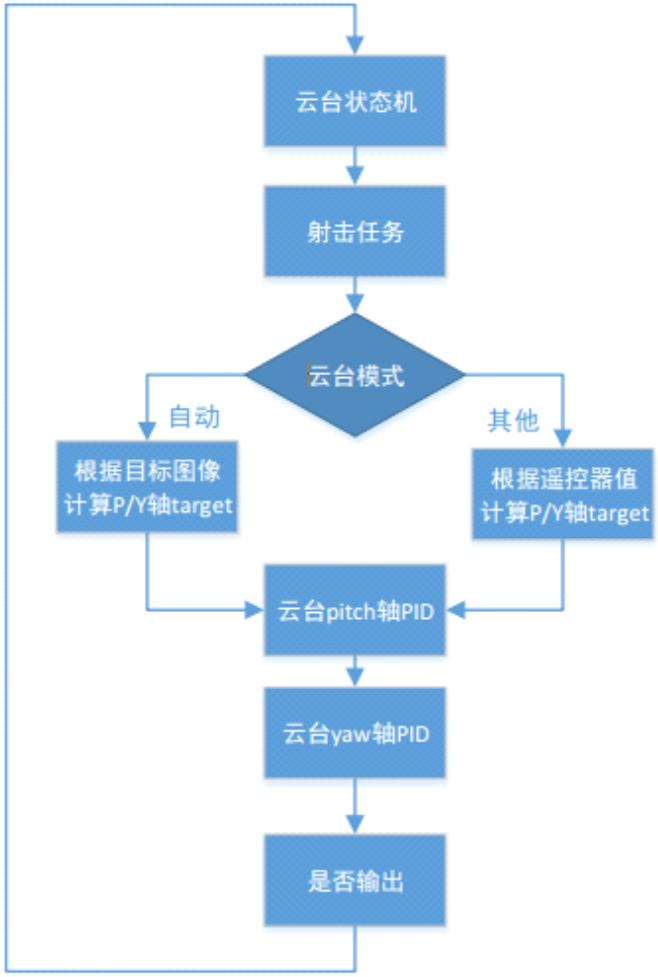
抓取下方弹药箱

- **射击模块**

GIMBAL_PATROL_MODE　　= 5,巡逻模式，云台 yaw 保持周期运动

GIMBAL_RELATIVE_MODE = 8, 相机在云台 yaw 轴上时使用

射击模式的实现先是建立在云台的模式下，第二阶段的射击是建立在全自动控制下，所以云台的模式多用为上面两种情况下。



在云台处理信息算法中运用了卡尔曼滤波对移动的目标有更加准确的预测（卡尔曼滤波官方代码中已经拥有只需要调用就可以），对云台的移动采用了斜坡响应处理使云台移动更加稳定。

射击射速和射频的选择：为保障射击热量值在限定范围之内不会超热量而出现扣血，调节射频和射速参数，实现伤害最大化。

- **难点与不足**

难点：

1.对子弹进行抓取和释放，所以可能用到直流电机、无刷电机、舵机等，用妙算进行控制其调试过程较复杂。

2.在初期调整 PID 的时候，只要稍微增强一点控制量，电机抖动就很大，无论怎么调整参数都无法满足要求。要么响应速度非常慢，要么就是超调发散。通过观察 PID 结构体里面的数据发现其 P 值抖动幅度非常大，我们想要的要求是当目标偏差很大的时候能够迅速到达设定角度，又要保证电机超调量很小，迅速收敛。于是经过一番讨论，决定采用分段 PID 控制。当偏差很大的时候给响应的一个很大的 P 参数，当误差小于一定范围内时削弱 Kp，增强 Ki。经过实验，发现分段 PID 的效果比之前要好很多，既能够在电机偏差角度较大的时候迅速响应，又能在接近目标值的时候通过加强的积分作用迅速消除静差。由于此时 Kp 被削弱吗所以超调量很小。利用分段 PID 取得了非常显著的控制效果。

底盘问题出现：

1.水平夹取弹药箱时，因为运用的是电机位置过多会出现堵转现象，位置不到的话出现无法加紧的状况，所以以后横向夹取采用气动会比较好，可以解决这种现象。

2.复位的时候应该控制底盘向小车前方移动，小车的摄像头是装在爪子下面，如果没有退出的动作的话，复位的动作会破坏摄像头。

3.成功夹取弹药箱的时候，需要加一个向前缓慢的运动，避免导航定位会误判步兵车卡入墙里而无法进行后续操作。

4.底盘运动打滑现象解决方案：在官方的代码中底盘控制使用的是速度闭环控制方案。在速度环的作用下，轮子能够很快达到设定转速。但是此时存在一个问题，由于轮子响应速度过快，会导致轮子在地面上打滑的现象。并且启动时会有较大的震动。所以我们假如了一个斜坡作用使得速度环的输出能够由阶跃响应编程斜坡响应。具体实现方法是在速度环有阶跃响应时，给偏差 err 乘上一个 0～1.0 递增的系数，这个系数会随时间慢慢从 0 增加到 1，从而使偏差有一个阶梯性的上升过程，从而使轮子输出有一个从 0 到设定转速的过渡过程，使底盘的控制变得更加平滑。

3. 算法部分

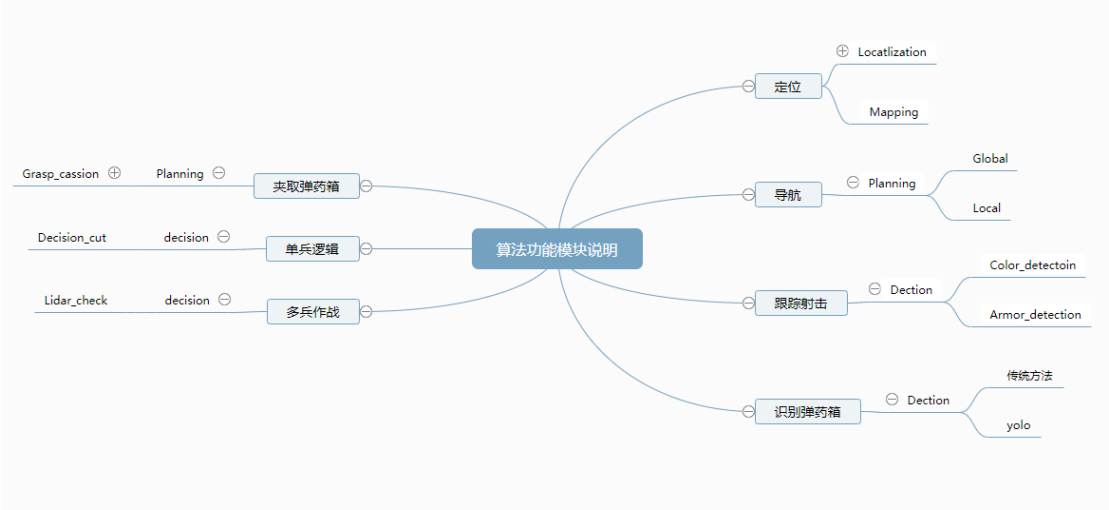
3.1 开发环境介绍

硬件环境：TX2
软件环境：基于 Ubuntu16 的 ROS 框架

3.2 整体技术方案概述

整体任务主要分为捉取弹药箱与识别射击敌人装甲板。为了完成这两项任务，需要使用定位算法和导航算法使机器人到达指定点进行弹药箱捉取和到部分区域进行进攻防守及获取 buff。其中，在弹药箱捉取中，运用视觉算法，用 yolo 进行弹药箱的识别及像素坐标的校准，进行精确捉取弹药箱；在识别敌人装甲板过程中，利用敌方装甲板颜色及装甲板特征，定位装甲板灯柱位置及距离，根据像素坐标换算成云台坐标与姿态，从而进行射击。此外，为了整合这两项任务，引入行为树作为系统的状态机，两项任务分别为行为树的节点。

3.3 算法整体框架设计



整体框架图

主要有在官方模块基础上增加了三个模块：yolo 检测模块，grasp_caisson 模块，和 decision 模块。

3.4 算法功能模块说明

3. 4. 0 定位算法

实现描述：
实现在地图上定点

实现原理

定位由三部分组成，odom，UWB 与激光雷达，采用的是自适应蒙特卡罗定位，由于在实验过程中，我们发现 UWB 坐标定位不准，经常导致机器人撞墙，因此我们将 UWB 屏蔽了，状态设置为 false。但官方给出的解决方案是由于我们自己给出的坐标原点不同于地图的坐标原点，因此导致定位出错。

实现过程

1.前期我们采用发布坐标点，由 global planning 和 local planning 订阅，进行路径规划，每次对于目标点，只能进行一次话题发布，若多次发布目标点，会导致 local planning 规划规划混乱，第二阶段，我们直接使用决策树里面的 set_goal() 进行坐标发布。

2.通过 rivz 窗口，我们可以观察到机器人的方向，以及雷达坐标，UWB 坐标，小车坐标，

遇到的问题及解决方法

需注意激光雷达的安装方向，小车方向要与激光雷达方向一致。

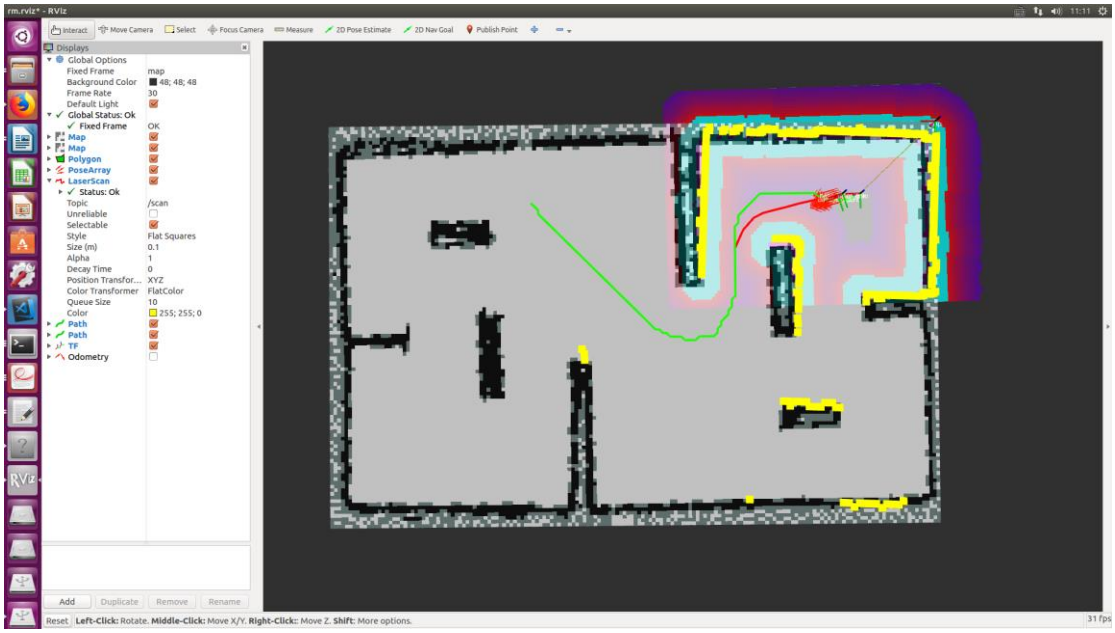
消耗物资

激光雷达

3.4.1 导航算法

实现描述

在定位的基础上，对目标点规划速度最佳行走路径。



实现原理

导航包括 global planning 和 local planning 两部分，global planning 采用的是 astar 算法，在 costmap 上计算由起点到终点之间的最短路径，然后再由 local planning 的 timed_elastic_band 进行局部规划，规划出机器人的速度向量 vector(x，y，yaw)，然后将速度通过串口发送给底盘，从而控制机器人的运动，局部规划可调节机器人的速度与加速度，通过修改 timer_elastic_band 里面的 kinematics_opt 的参数，还可调节机器人与障碍物的膨胀系数，即 tolerance_opt 的值，这些参数可根据实际进行修改。

实现过程

在进行路径规划的过程中，我们可以打开 rviz 窗口，观察由起点至终点规划出来的路径，绿色为 global planning 规划的路径，红色为 local planning 规划出来的路径，最终机器人会按照红色路径进行运动，可通过观察 rviz 窗口，判断是 global planing 还是 local planning 发生了错误，从而进行调整优化。

在给机器人进行坐标定位的时候，我们可先通过仿真判断定位点以及程序是否正确，然后再在机器人上进行实验，但由于仿真地图和真实物理地图存在偏差，因此坐标点的定位还需通过真实实验测量。

遇到的问题及解决方法

在实验过程中我们发现由于激光雷达安装位置过低，会导致机器人将弹药箱识别为障碍物，从而导致机器人无法顺利通过走道。在安装时需要注意激光雷达的安装位置，以防出现误判弹药箱为障碍块的情况

但速度不宜过大，会导致机器人撞墙或空转，而导致 odom 定位失准。

3.4.2 识别弹药箱

实现描述

识别出弹药箱，并计算出像素中心和物体中心的 x 轴相对距离

需要识别的弹药箱

实现原理

对此，我们使用了两种方法，第一阶段的时候我们使用传统方法，即 openCV 实现边缘检测，颜色识别，加上霍夫圆判断最终确定圆心。第二阶段我们使用神经网络识别箱子的方法，神经网络识别箱子我们采用的是 YOLOv3 算法，由于 YOLOv3 识别速度快，准确率高的特点，从而广泛被大家使用。

实现过程

传统方法：

1. 使用 HSV 通道进行处理

```
Mat imgHSV;
cvtColor(imgOriginal, imgHSV, COLOR_BGR2HSV);
```

2. 进行高斯滤波
3. 均值化以消除光线影响
4. 对 HSV 通道进行 inrange 和 bitwise 操作，获得红色和蓝色圆环

```
void inRange(InputArray src, InputArray lowerb, InputArray upperb, OutputArray dst);
```

参数解释：

参数 1：输入要处理的图像，可以为单通道或多通道。

参数 2：包含下边界的数组或标量。

参数 3：包含上边界数组或标量。

参数 4：输出图像，与输入图像 src 尺寸相同且为 CV_8U 类型。

bitwise:

bitwise_and、bitwise_or、bitwise_xor、bitwise_not 这四个按位操作函数

5. 腐蚀膨胀获得完整圆环

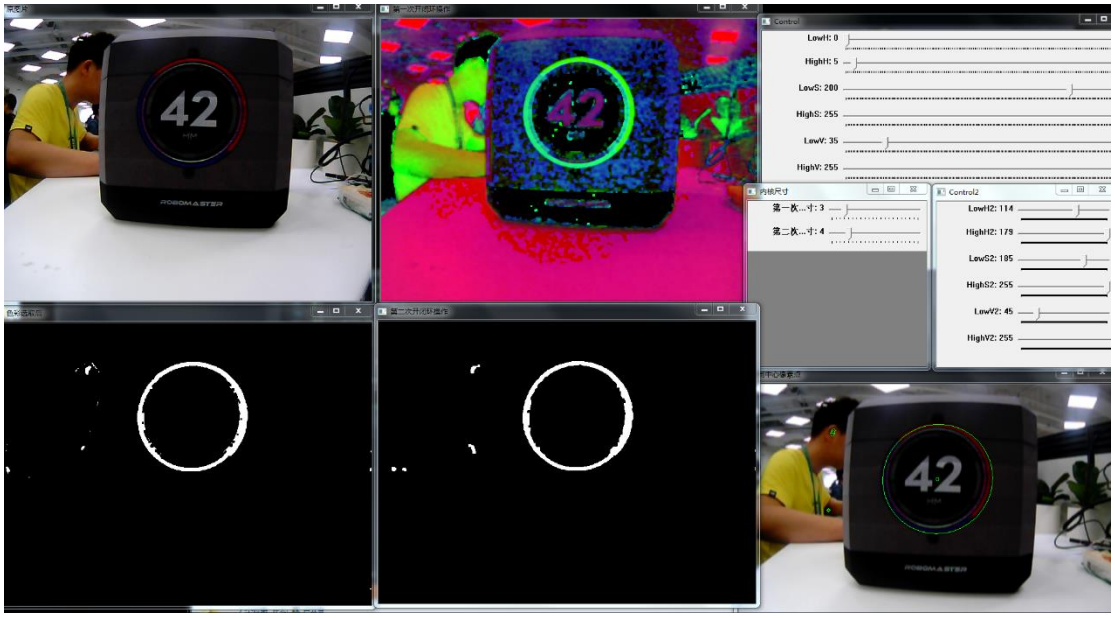
```
//设置内核(方形或椭圆,用于卷积)
    Mat element1 = getStructuringElement(MORPH_ELLIPSE, Size(5, 5));
//进行腐蚀膨胀操作
//方法一:(直接调取函数)
    erode(g_srcImage,g_dstImage, element);
    dilate(g_srcImage,g_dstImage, element);
```

//方法二:(间接调取函数)

morphologyEx(imgHSV, imgHSV, MORPH_OPEN, element);

morphologyEx(imgHSV, imgHSV, MORPH_CLOSE, element);

6. 轮廓描绘并导出圆心



圆环颜色识别效果图

神经网络识别：

1.首先我们利用车载摄像头采集了 600 张左右的图片，图片中包含了各种姿态以及各种明暗度的弹药箱，然后利用 yolo_mark 进行打标，最终得到的是带框的图片以及包含框坐标点的 txt 文件，我们将 YOLOv3 神经网络的输入长宽设为 800，由于我们只识别一种物体，因此将每一个 yolo 层前一个卷积层的 filter 设为 18（3×（1+4+物体种类）），yolov3 对于每一个 grid cell 预测 3 个边界框，YOLO v1 中是 2 个，YOLO v2 中是 5 个，将 yolo 层的 classes 设为 1。最终的边界框与实际物体边界存在差异，因此我们将视觉用于微调，先运动到坐标点，然后在由视觉进行角度调整，然后由 pc 给单片机发送命令，由单片机控制机械爪进行夹取。

2. 对于训练神经网络，图片的采集是一个难点与重点，我们前后进行了两次训练，第一次训练由于采集的图片是通过自己的手机采集的，而且图片色彩偏亮，物体较为清晰，因此最终讯来拿出来的神经网络智能识别清晰且高亮的物体，但是实际应用中，摄像头捕捉的物体大多都是色彩偏暗，分辨率偏低且不完整的，因此在第二次训练中，我们先用车载摄像头采集视频，然后利用软件截取图片，因此采集的图片比较符合真实物理环境，而且我们加多了多个弹药箱存在以及弹药箱不完整的图片，这些对于我们后期的识别都提供了极大的帮助。打标过程需要修改物体的名称 obj.names 为 caisson，打标完标签的路径都会保存在 train.txt 中。

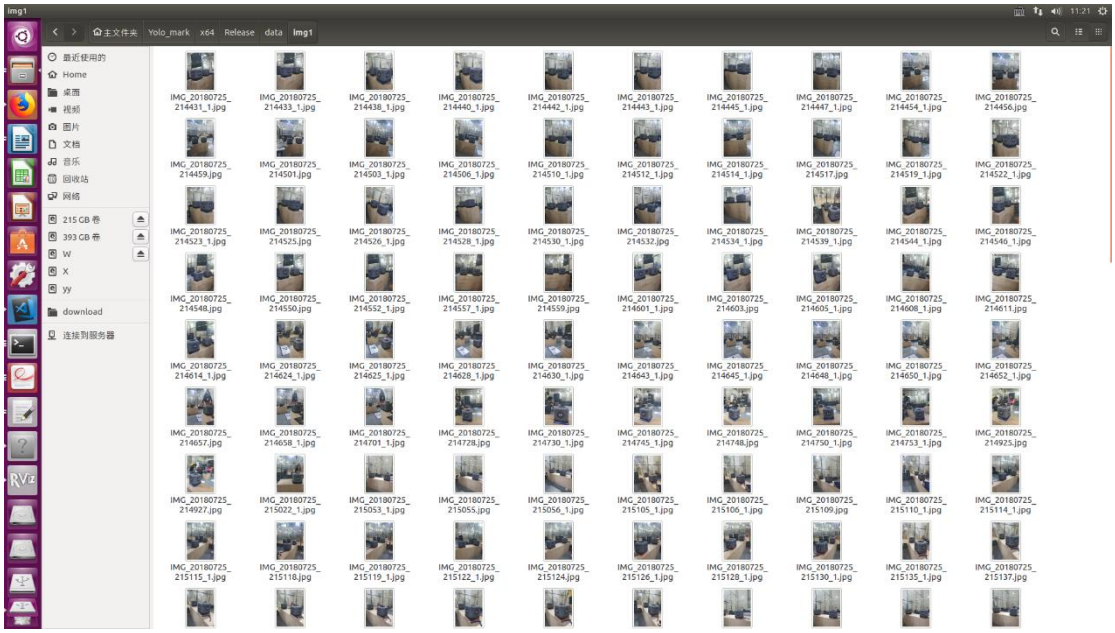


图 1　第一次用于训练的图片

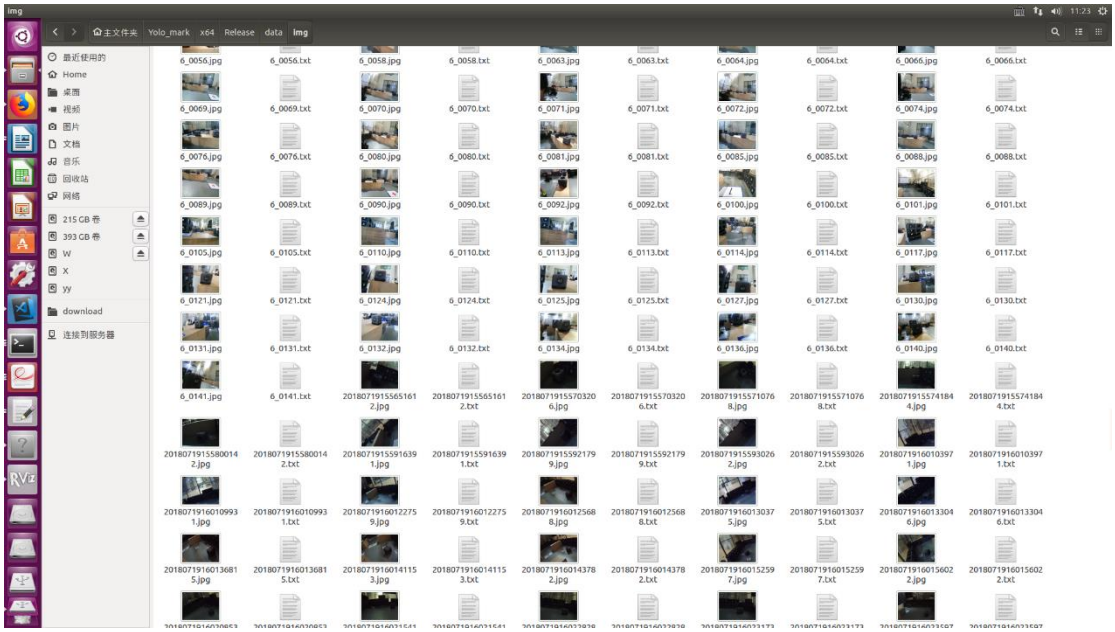


图 2　第二次用于训练的图片

在第一次训练中，神经网络输入的长和宽使用的是默认的 413×413，在第二次训练中，我将输入尺寸长和宽设为了 800×800，bitch size 设为默认的 64，subdivisions 设为 8，若设备显存很小，可将 random 设为 0，关闭多尺度训练。

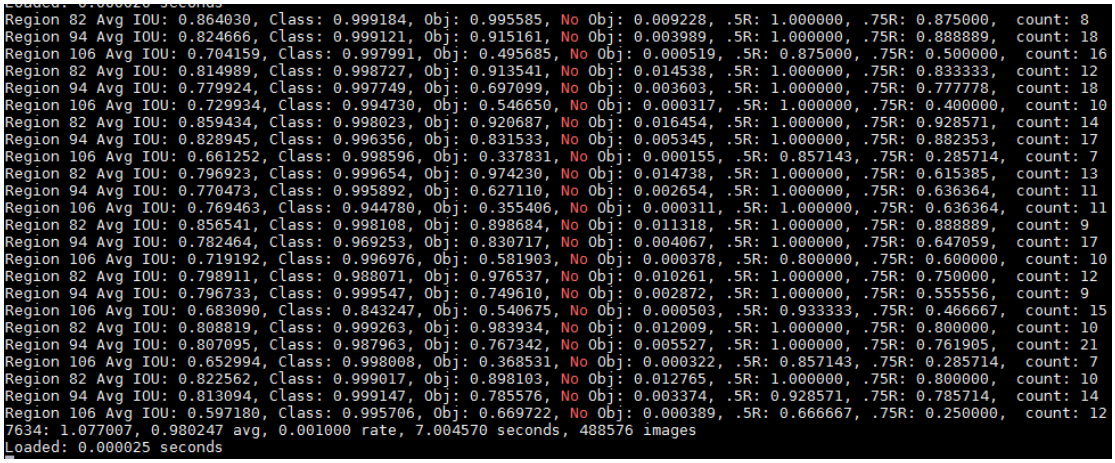


图 3　训练效果图

训练过程中的参数：

Avg IOU:当前迭代中，预测的 box 与标注的 box 的平均交并比，越大越好，期望数值为 1；

Class: 标注物体的分类准确率，越大越好，期望数值为 1；

obj: 越大越好，期望数值为 1；

No obj: 越小越好；

.5R: 以 IOU=0.5 为阈值时候的 recall; recall = 检出的正样本/实际的正样本

0.75R: 以 IOU=0.75 为阈值时候的 recall;

count:正样本数目。

在 GPU 上训练大概 5 小时后，最终的 loss 降为 0.1 时，可从 barkup 文件夹中取出 yolov3-rm.weights 进行测试，从 github 上下载 darknet，进入 darknet 目录，运行./darknet detect cfg/yolov3-rm.cfg yolov3-rm.weights data/caisson.jpg，看观察到图片中的物体被框出，预测的准确率为 98%，由于边界框并不是和物体边界完全重合，因此并能利用边界框的坐标对物体进行定位，只能利用 yolov3 进行辅助定位。

```
  50 conv     256  1 x 1 / 1     50 x  50 x 512   ->     50 x  50 x 256  0.655 BFLOPs
  51 conv     512  3 x 3 / 1     50 x  50 x 256   ->     50 x  50 x 512  5.898 BFLOPs
  52 res   49                    50 x  50 x 512   ->     50 x  50 x 512
  53 conv     256  1 x 1 / 1     50 x  50 x 512   ->     50 x  50 x 256  0.655 BFLOPs
  54 conv     512  3 x 3 / 1     50 x  50 x 256   ->     50 x  50 x 512  5.898 BFLOPs
  55 res   52                    50 x  50 x 512   ->     50 x  50 x 512
  56 conv     256  1 x 1 / 1     50 x  50 x 512   ->     50 x  50 x 256  0.655 BFLOPs
  57 conv     512  3 x 3 / 1     50 x  50 x 256   ->     50 x  50 x 512  5.898 BFLOPs
  58 res   55                    50 x  50 x 512   ->     50 x  50 x 512
  59 conv     256  1 x 1 / 1     50 x  50 x 512   ->     50 x  50 x 256  0.655 BFLOPs
  60 conv     512  3 x 3 / 1     50 x  50 x 256   ->     50 x  50 x 512  5.898 BFLOPs
  61 res   58                    50 x  50 x 512   ->     50 x  50 x 512
  62 conv    1024  3 x 3 / 2     50 x  50 x 512   ->     25 x  25 x1024  5.898 BFLOPs
  63 conv     512  1 x 1 / 1     25 x  25 x1024   ->     25 x  25 x 512  0.655 BFLOPs
  64 conv    1024  3 x 3 / 1     25 x  25 x 512   ->     25 x  25 x1024  5.898 BFLOPs
  65 res   62                    25 x  25 x1024   ->     25 x  25 x1024
  66 conv     512  1 x 1 / 1     25 x  25 x1024   ->     25 x  25 x 512  0.655 BFLOPs
  67 conv    1024  3 x 3 / 1     25 x  25 x 512   ->     25 x  25 x1024  5.898 BFLOPs
  68 res   65                    25 x  25 x1024   ->     25 x  25 x1024
  69 conv     512  1 x 1 / 1     25 x  25 x1024   ->     25 x  25 x 512  0.655 BFLOPs
  70 conv    1024  3 x 3 / 1     25 x  25 x 512   ->     25 x  25 x1024  5.898 BFLOPs
  71 res   68                    25 x  25 x1024   ->     25 x  25 x1024
  72 conv     512  1 x 1 / 1     25 x  25 x1024   ->     25 x  25 x 512  0.655 BFLOPs
  73 conv    1024  3 x 3 / 1     25 x  25 x 512   ->     25 x  25 x1024  5.898 BFLOPs
  74 res   71                    25 x  25 x1024   ->     25 x  25 x1024
  75 conv     512  1 x 1 / 1     25 x  25 x1024   ->     25 x  25 x 512  0.655 BFLOPs
  76 conv    1024  3 x 3 / 1     25 x  25 x 512   ->     25 x  25 x1024  5.898 BFLOPs
  77 conv     512  1 x 1 / 1     25 x  25 x1024   ->     25 x  25 x 512  0.655 BFLOPs
  78 conv    1024  3 x 3 / 1     25 x  25 x 512   ->     25 x  25 x1024  5.898 BFLOPs
  79 conv     512  1 x 1 / 1     25 x  25 x1024   ->     25 x  25 x 512  0.655 BFLOPs
  80 conv    1024  3 x 3 / 1     25 x  25 x 512   ->     25 x  25 x1024  5.898 BFLOPs
  81 conv      18  1 x 1 / 1     25 x  25 x1024   ->     25 x  25 x  18  0.023 BFLOPs
  82 yolo
  83 route  79
  84 conv     256  1 x 1 / 1     25 x  25 x 512   ->     25 x  25 x 256  0.164 BFLOPs
  85 upsample            2x     25 x  25 x 256   ->     50 x  50 x 256
  86 route  85 61
  87 conv     256  1 x 1 / 1     50 x  50 x 768   ->     50 x  50 x 256  0.983 BFLOPs
  88 conv     512  3 x 3 / 1     50 x  50 x 256   ->     50 x  50 x 512  5.898 BFLOPs
  89 conv     256  1 x 1 / 1     50 x  50 x 512   ->     50 x  50 x 256  0.655 BFLOPs
  90 conv     512  3 x 3 / 1     50 x  50 x 256   ->     50 x  50 x 512  5.898 BFLOPs
  91 conv     256  1 x 1 / 1     50 x  50 x 512   ->     50 x  50 x 256  0.655 BFLOPs
  92 conv     512  3 x 3 / 1     50 x  50 x 256   ->     50 x  50 x 512  5.898 BFLOPs
  93 conv      18  1 x 1 / 1     50 x  50 x 512   ->     50 x  50 x  18  0.046 BFLOPs
  94 yolo
  95 route  91
  96 conv     128  1 x 1 / 1     50 x  50 x 256   ->     50 x  50 x 128  0.164 BFLOPs
  97 upsample            2x     50 x  50 x 128   ->    100 x 100 x 128
  98 route  97 36
  99 conv     128  1 x 1 / 1    100 x 100 x 384   ->    100 x 100 x 128  0.983 BFLOPs
 100 conv     256  3 x 3 / 1    100 x 100 x 128   ->    100 x 100 x 256  5.898 BFLOPs
 101 conv     128  1 x 1 / 1    100 x 100 x 256   ->    100 x 100 x 128  0.655 BFLOPs
 102 conv     256  3 x 3 / 1    100 x 100 x 128   ->    100 x 100 x 256  5.898 BFLOPs
 103 conv     128  1 x 1 / 1    100 x 100 x 256   ->    100 x 100 x 128  0.655 BFLOPs
 104 conv     256  3 x 3 / 1    100 x 100 x 128   ->    100 x 100 x 256  5.898 BFLOPs
 105 conv      18  1 x 1 / 1    100 x 100 x 256   ->    100 x 100 x  18  0.092 BFLOPs
 106 yolo
Loading weights from yolov3-rm.weights...Done!
data/IMG_20180725_215919.jpg: Predicted in 69.075438 seconds.
caisson: 98%
```

图 4　预测单张图片识别物体的准确率。

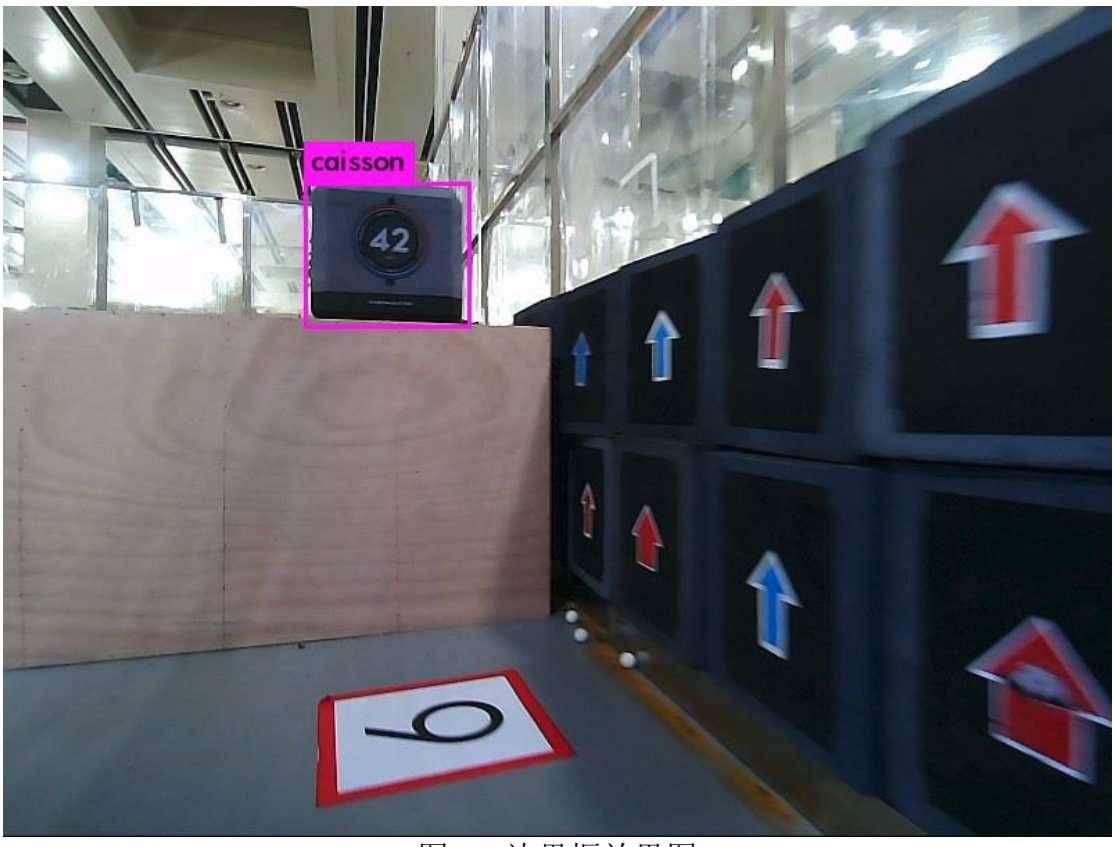


图 5　边界框效果图

遇到的问题及解决方法

传统方法在实践过程当中主要遇到以下问题：
1.难以识别较远距离物体
2.受光线影响较大
3.识别的圆中心不稳定，左右飘动

由于多次尝试过后仍然很难解决以上问题，我们决定在第二阶段的时候使用神经网络识别，遇到问题主要如下：

在实验过程中，我们可以检测到远处的多个物体，但是在我们的整个逻辑里面是检测到物体便开始调整姿态，对准物体后向前运动，并不会考虑机器人与弹药箱之间存在障碍物的情况，因此我们需要率掉远距离的弹药箱，在 src/image.c 文件中，我们找到画出边界框的函数 draw_detections（），然后根据边界框的长和宽，滤掉面积过小的弹药箱。

在摄像头视野中存在检测到两个面积相差不大的物体，我们可以计算边界框的面积，选择面积较大的弹药箱进行夹取。

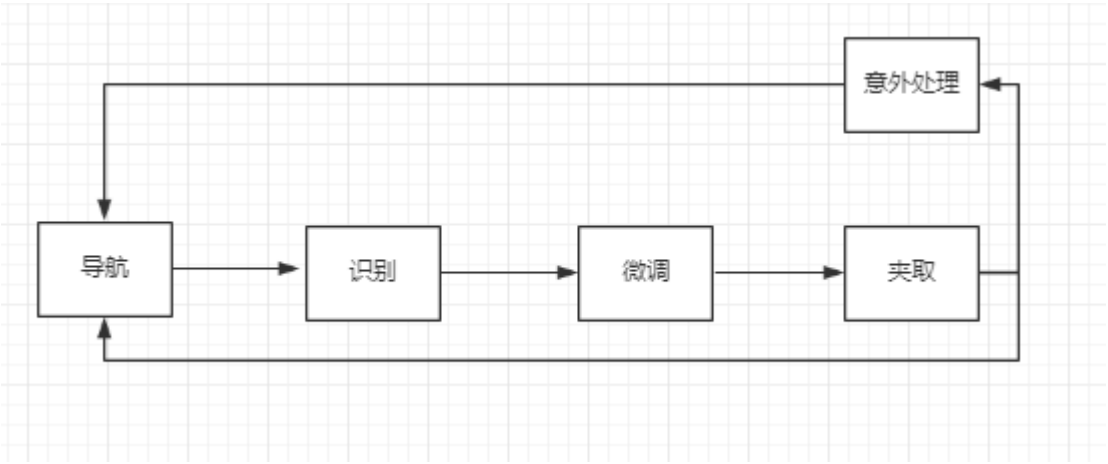
```
box_width = box_width_new;
box_width_new = right - left;
box_height = box_height_new;
box_height_new = bot - top;

if ((box_width_new * box_height_new) > (box_width * box_height) &&
            box_width_new > 120 && box_height_new >120){
    max_box_num = i;
}
```

然后再在 yolo_node.cpp 文件中对面积最大的弹药箱进行处理，通过 rostopic 发布出弹药箱的中心点的位置以及是否检测到弹药箱，然后由 grasp_caisson 接收后进行中心对准处理。

3. 4. 3 夹取弹药箱

实现描述

提前标定合适弹药箱位置，使用导航算法移动至弹药箱附近位置，然后启动检测弹药箱节点（详请见“弹药箱检测”），获得弹药箱中心与像素中心相差距离，然后进行微调，直至中心对准。对准后启动夹取节点，完成整套夹取动作。夹取动作由 PC 发送模式选择，即向上走还是向下走，夹取还是归位。完成后由 MCU 返回信号启动下一个节点。
流程图如下：



实现原理

原本方案采用 PC 直接控制 MCU 夹取，即发送三个自由度电机的转动角度（详见“嵌入式夹取电机控制”），来控制整套夹取动作，后来发现丢包率非常严重，即便采取了多次发送的方式仍然有大约 10%的丢包率；并且 PC 的时钟控制很不准确，在 PC 端的 sleep( )可能受到 CPU 占用率等因素影响，导致时间有很大的波动，因此最终容易导致还没夹取完成就复位的情况发生。

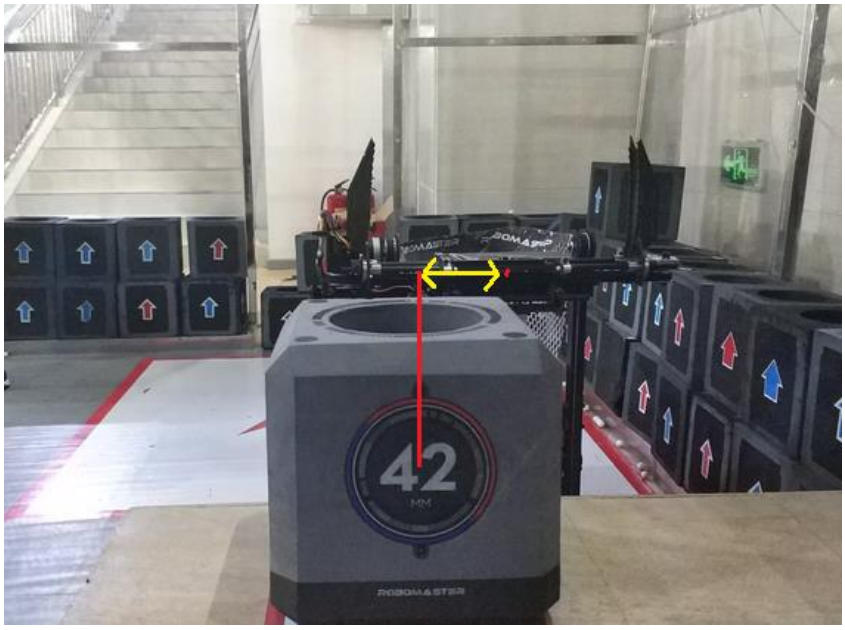
后来直接改用 PC 发送夹取模式，由 MCU 控制电机转动角度来完成整套夹取动作。

实现过程

1.导航到弹药箱附近（详见“定位算法”,“导航算法”），关键点是定准弹药箱位置，我们采取的是粗调加微调的方式选取弹药箱位置，即导航至附近（约相差 15cm）后，启动检测节点进行微调。

2.检测弹药箱（详见“弹药箱检测”），主要功能判断有没有弹药箱，若有则计算出像素中心和物体中心的 x 轴相对距离。

3.微调对准，根据得到的相对距离在 PC 端使用 PID 闭环控制，根据实际效果，发现距离不大，因此直接采用 P 环控制，使得像素中心和物体中心的 x 大小相等。



微调前

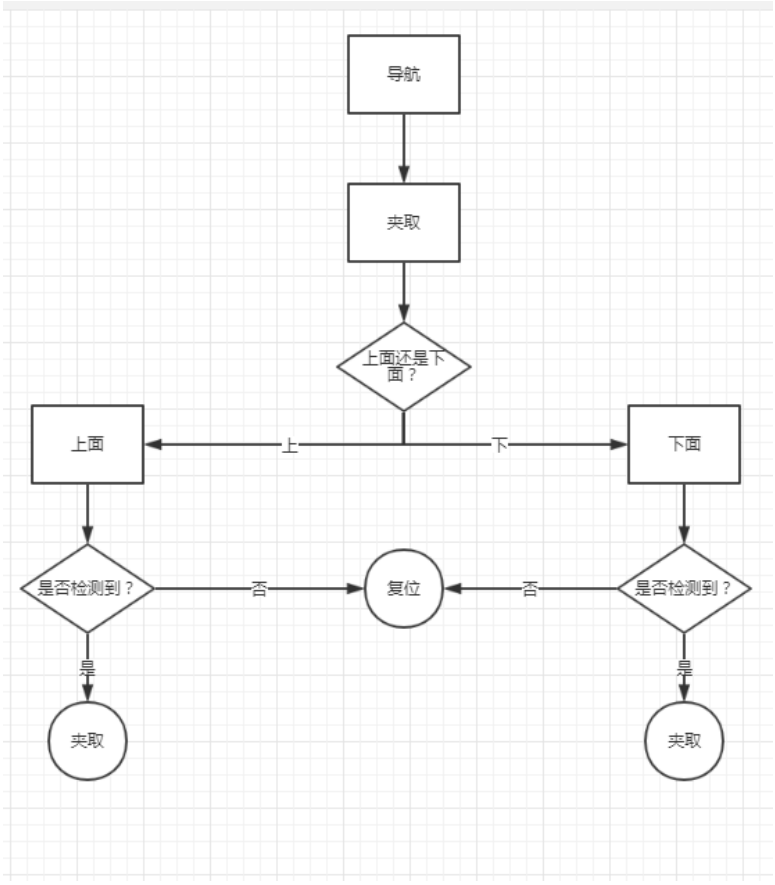
微调后

实现代码：（部分）

```
void GraspCaissonNode::pixedClose(int dis,float &x,float&y)
{
   float p = PID_P_;
   x = 0;
   y = p * (float)dis;
   if(y>0.3)
       y = 0.3;
   if(y< -0.3)
       y = -0.3;

  /* if(x > 0)
       x = -x;
   if(x < -0.1)
       x = -0.1;*/

}
```

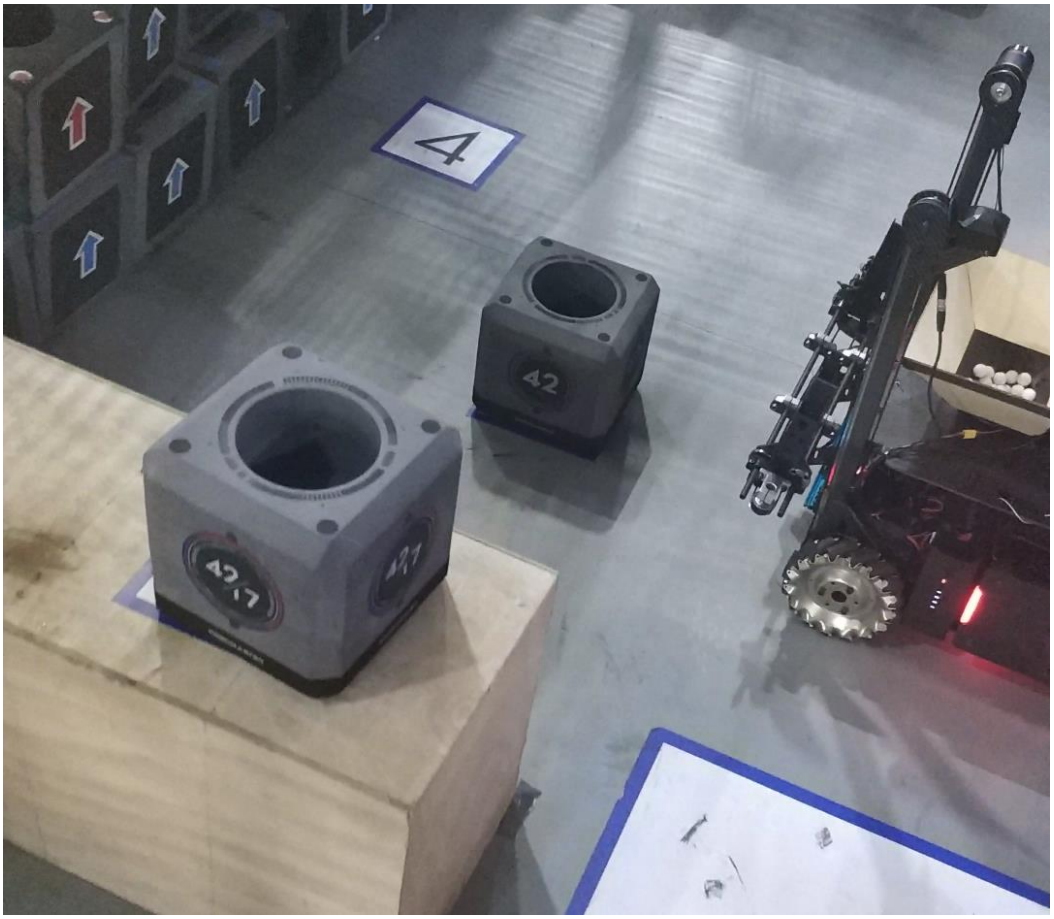
P 环微调

4.模式控制，模式控制是 PC 控制 MCU 夹取的核心，因为在实际情况中，有上下两种弹药箱，因此需要控制“该看上面还是看下面”，而且看完之后还要根据结果判断“该不该夹取”。因此我们采用下面模式逻辑控制：



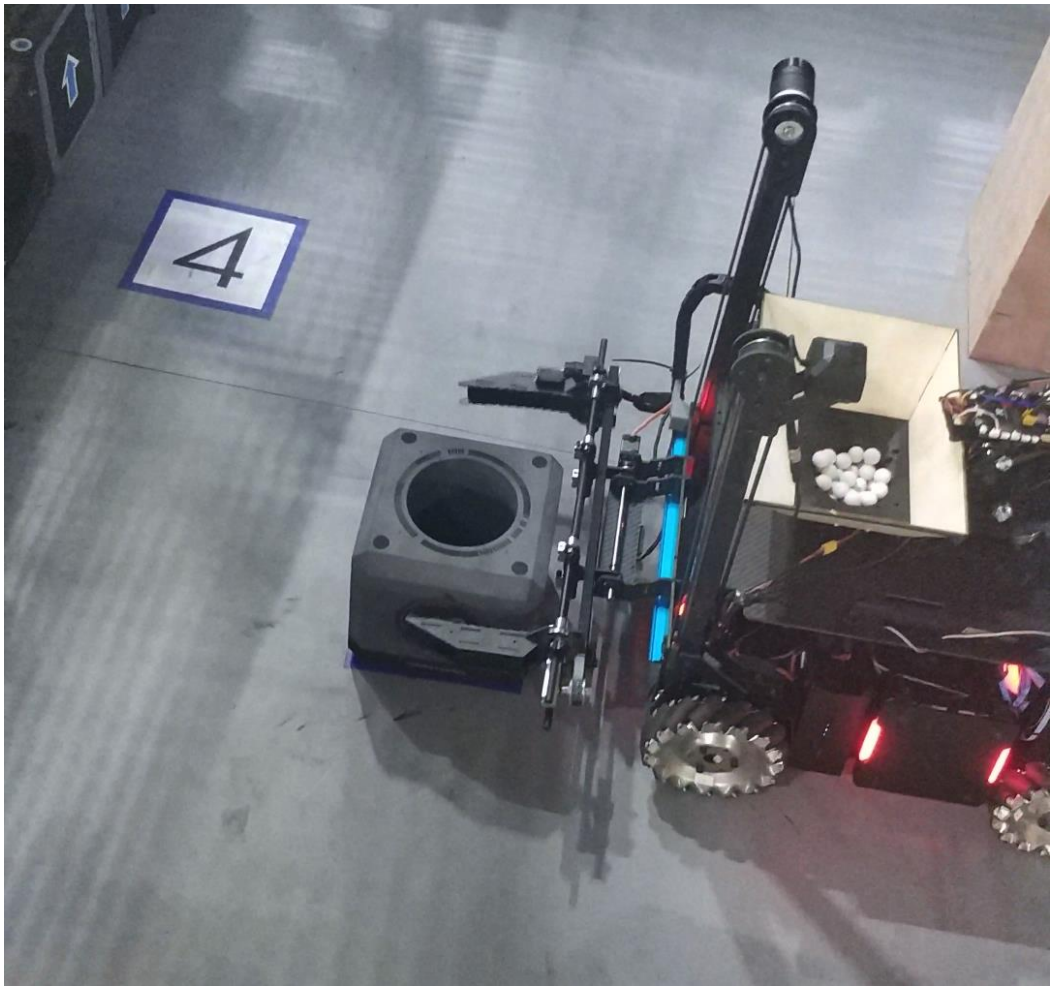
夹取逻辑图

检测上方（对应模式 1,0）



检测下方（对应模式 1,1）

夹取上方（对应模式 2,0）



夹取下方（对应模式 2,1）

实现代码：（部分）

```
if(!place_flag_)
    {
        publishSpeed(vel_pub_, 0, 0);
        for(int i = 0;i < 50; i++)
           publishClaw(claw_pub_,1,0,0);
        sleep(1);
        place_flag_ = true;
        begin_flag_ = caisson_msg_.isCheck;
    }
```

检测上方

```
bool GraspCaissonNode::getHightCaisson(){

  if(clawResult == 1)
         {
           std::cout << "Hight final" << std::endl;
           publishClaw(claw_pub_,0,0,0);

           return true;
         }

   else if(graspCaisson()){



     if(caisson_flag_ && !back_flag_)

        {
            std::cout << " Hight find" << std::endl;
            back_flag_ = true;
            for(int i = 0;i < 50;i++)
                publishClaw(claw_pub_,1,1,0);
        }

     else if(caisson_flag_ == false)
        {
           publishClaw(claw_pub_,0,0,0);
           std::cout << "Hight not" << std::endl;
           caisson_flag_ = false;
           return true;
        }



}
```

模式发送

5.意外处理，在整个导航，检测，夹取的过程中，可能出现各种各样的问题（详请看下面“遇到的问题及解决方法”），因此我们采用计时保护，即规定时间内没

有执行完毕则自动跳出状态，以提高系统稳定性。
实现代码：（部分）

```
    std::chrono::steady_clock::time_point end_time =
std::chrono::steady_clock::now();
    std::chrono::microseconds execution_duration = std::chrono::duration_cast
<std::chrono::microseconds>(end_time - start_time);

    execution_time_total += execution_duration.count();

    if(execution_time_total > execution_time_th)
      {

        place_flag_ = false;
        caisson_flag_ = false;
        back_flag_ = false;
        begin_flag_ = false;
        publishClaw(claw_pub_,0,0,0);
        std::cout <<"time out "<< execution_time_total << std::endl;
        execution_time_total = 0;
        SetNodeState(rrts::common::NodeState::FAILURE);
      }
```

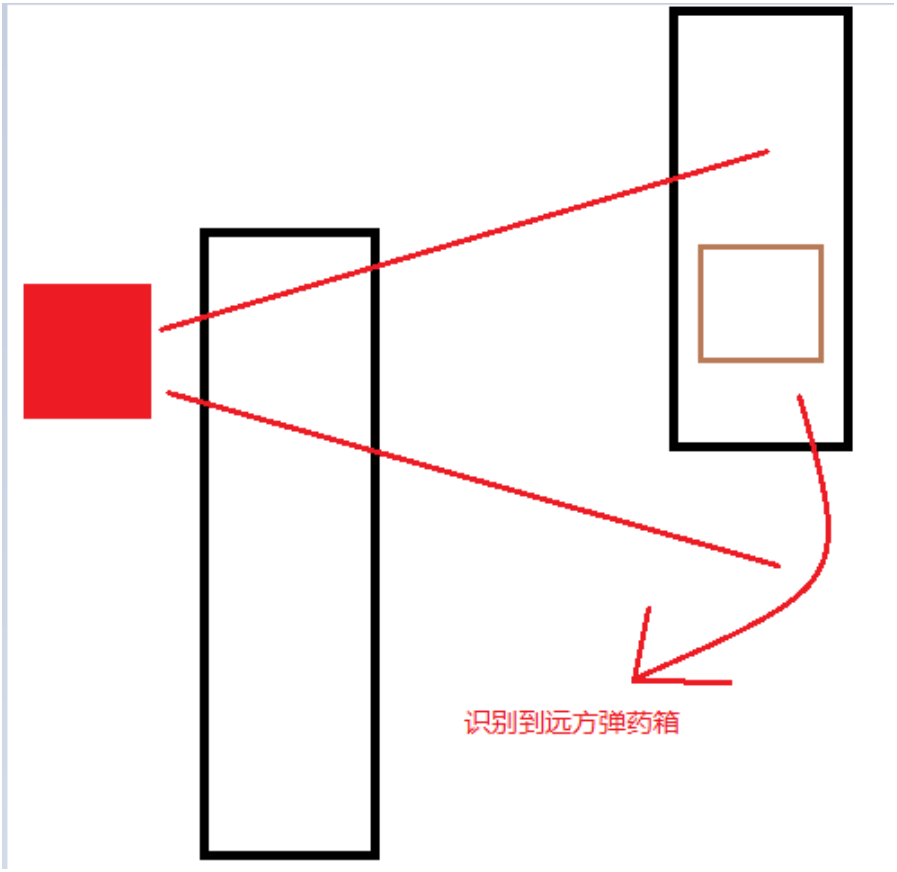
意外处理代码

遇到的问题及解决方法

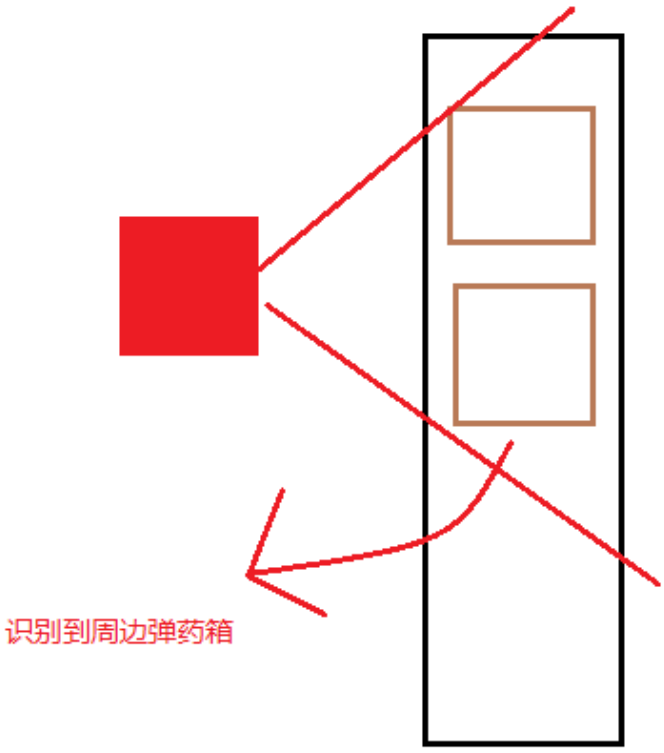
1.识别不到弹药箱：
我们第一阶段采用边缘检测加霍夫圆判断，对于近距离（20cm 空间相对距离）识别精度很高，但是远距离（20cm 以外），则受光线，弹药箱摆放位置的影响巨大。后来我们采用 yolo 算法识别，对于远距离识别非常精准（因为训练时采用整个弹药箱的正样本），但是近距离时则无法识别。

解决：对于这个问题，我们想了好多办法，但都无法解决这个算法带来的系统误差，所以我们干脆使用距离误差来弥补这个算法的不足，也即定点导航的时候不定到弹药箱非常近的位置，而是在其正前方 20cm 左右的位置然后开始识别。（详见“弹药箱识别”）

2.识别到远处：
在上一个问题中，我们采取了距离误差来弥补 yolo 算法的不足，但是却带来了新问题，也即识别到远方的弹药箱（见下图），和周边的弹药箱



识别到远方弹药箱



识别到周边弹药箱

解决：对于这个问题，我们采取面积过滤方法，即在 yolo 算法获取到的一系列数组中选取面积最大，且大于一定值（像素面积的 1/4）时才认为检测到。

3.跳不出光电触发循环：
当出现以上问题，解决方法不奏效时，便会出现夹取失败的情况，夹取失败的时候可能会导致一直卡在等待光电触发的循环里，导致整车卡死。

解决：对于这个问题，我们采用计时保护，即规定时间内没有执行完毕则自动跳出状态，以提高系统稳定性。
实现代码：（部分）

4.意外处理时复位不当导致整车卡位
使用上述方法处理卡死在光电循环的情况时，带来了新的问题，即跳出“夹取”节点时会 PC 端会发送复位命令给 MCU，当在夹取过程中复位，将会出现卡墙的情况，一旦卡墙则整车将无法动弹，我们在决赛时也因此输掉了两场比赛
（夹取失败截图）

对于这个问题，解决方案是在 MCU 端收到复位信号的时候往前走一小段距离以离开墙体，然后爪子再复位。我们发现这个问题是在热身赛后，由于时间紧迫，MCU 端没有时间对新的代码进行调试，因此我们在比赛期间并没有使用此策略。

5.夹取时遇到敌人
在夹取的过程当中，特别是靠近敌人位置的弹药箱时，很容易遇到敌人的攻击，在此情况下，夹取的 6-7 s 内将会受到致命打击。

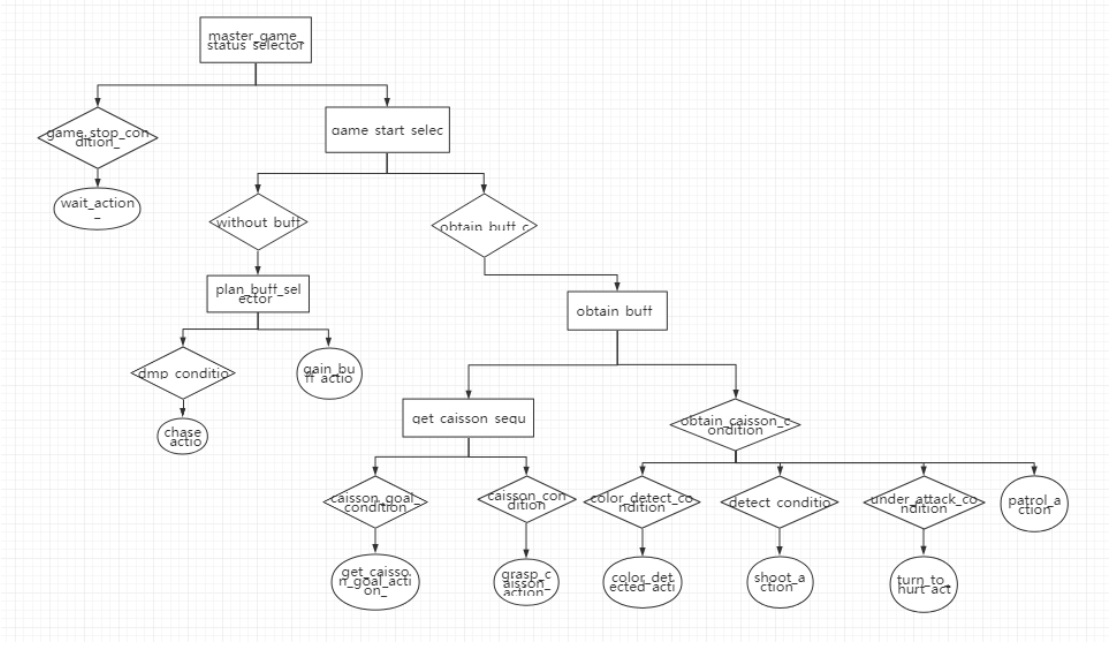
解决：对于这个问题，我们采用官方的行为树来解决（详见“单兵逻辑”），即在没有弹药且正在夹取时遇到敌人后，优先级最高的是逃跑。

消耗物资

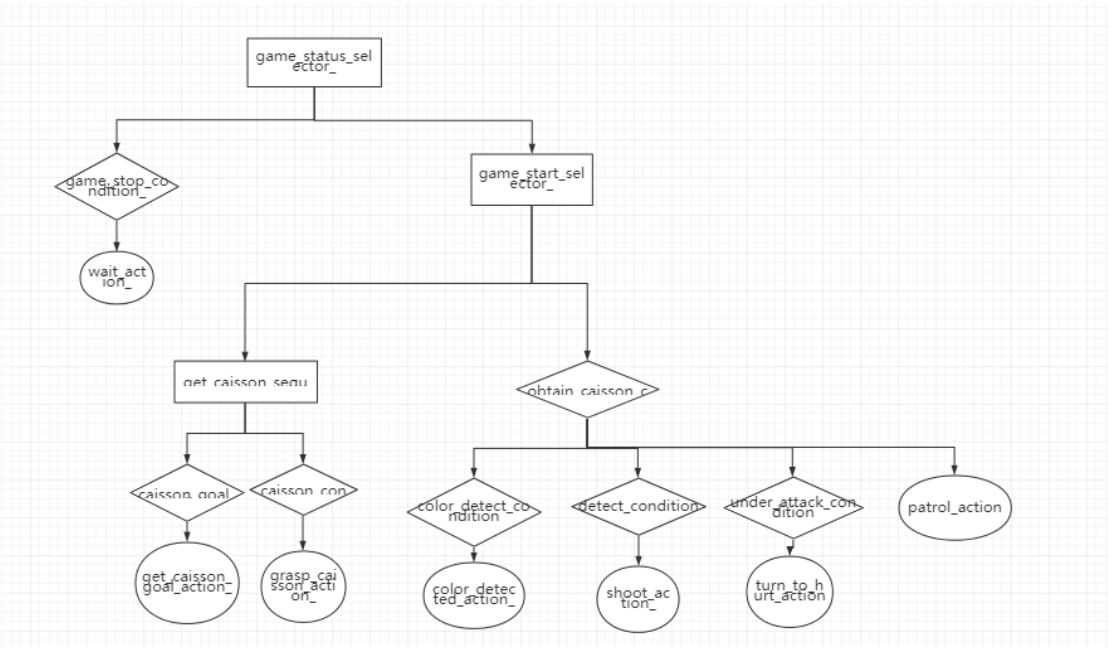
光电触发器，
爪子（详见“机械部分”）
摄像头

3.4.3 单兵逻辑

实现描述
单兵分为主机逻辑和从机逻辑



主机逻辑



从机逻辑

实现原理

对于单兵作战的机器人,我们采取通用的行为树作为机器人运行的状态机.如上图所示.行为树根节点定义为 selector 类型的节点，子节点为判定比赛是否开始的 percondition 节点。由于到达弹药箱位置和夹取弹药箱是一个顺序的过程，因此取弹药箱的节点定义为 sequence 类型节点，进攻的节点为 selector 类型，根据行为树从左到右，深度优先的规则

实现过程

从机整个行为树的运行状态如下：

(1) 比赛是否开始，若开始，转(2);否则执行 wait_action，节点返回 true;

(2) 执行取弹节点 get_caisson_sequence，满足条件的情况下（条件为当前获取的弹药箱书是否小于客户给的阈值），顺序执行 caisson_goal_condition 节点和 caisson_condition 节点，既让机器人先到达弹药箱位置，随后进行取弹操作。若 get_caisson_sequence 节点返回 true,则执行(3)；否则返回 false。

(3) 执行 obtain_caisson_condition 节点，该节点下的子节点从左到右分别为 color_detect_condition（根据颜色检测敌方装甲板），detect_condition(根据装甲板位置进行射击)， under_attack_condition(己方装甲板受到攻击检测)，patrol_action(巡航模式)。即：

color_detect_condition：是否发现敌方装甲板，是返回 true,否返回 false;

detect_condition:是否得到装甲板位置信息，是则执行 shoot_action，并返回 true;否则返回 false;

under_attack_condition:己方前后左右装甲板是否收到攻击，是则调整机器人姿势转向被攻击方向进行反击，并返回 true;否则返回 false;

patrol_action：在以上条件节点都不满足的情况下，机器人根据给的巡航点进行巡航，返回 true；

(4)从根节点循环执行以上节点。

主机整个行为树的运行状态如下：

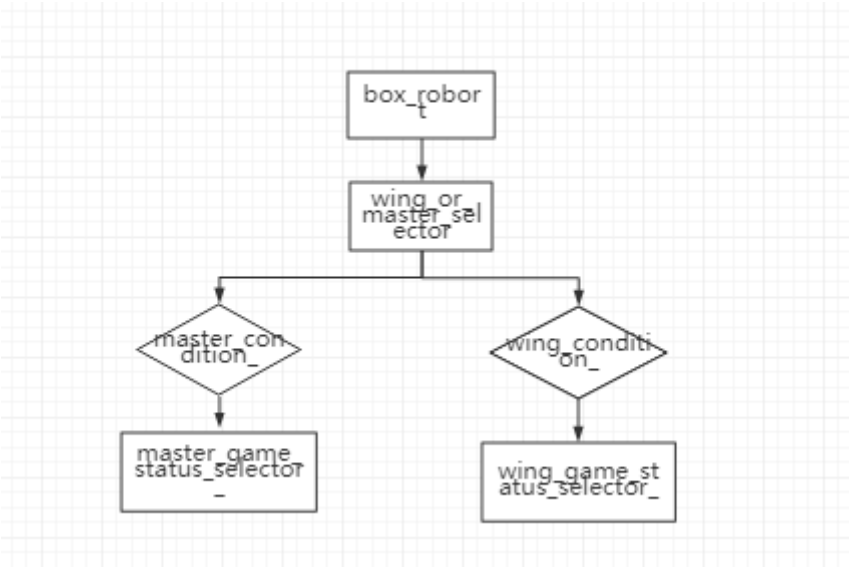
相比从机机器人，主机机器人的首要任务是抢 buff，因此，主机机器人行为树的第一左子节点为 plan_buff_selector_，该节点为 selector_类型，它的子节点为 dmp_condition_和 gain_buff_action_。dmp_condition_为判断是否受到敌人的进攻，因一开始主机并没有获取弹药，没有进攻能力，所以，当 dmp_condition_返回 true 时，马上执行 chase_action_逃跑。如果返回 false，则执行 gain_buff_action_去获取 buff。获取 buff 后，主机执行的状态与从机一样，既取弹药箱，然后去巡航进攻敌人。

3.4.4 多兵作战

实现描述

多兵作战主要将步兵机器人分为主机和从机，其中主机的首要任务是去 buff 点抢 buff，抢完之后回己方区域取弹后巡航进攻。从机启动后便在己方区域取弹之

后进行进攻。在初始位置使用雷达进行根据 4,6 是否有障碍物来决定主从，主从判断后两机不再通讯。



双机逻辑

实现原理

雷达检测：

1.雷达数据是大小为 720 的数组，并且按照角度逆时针排序，每隔 0.5 度为取一个数

2.获取对应数组，并求平均值即可知道是否有障碍物

实现过程

1.根据需要发布消息建立 lidar_check_info 的 msg（bool）

2.查找 lidar 发送的 topic，将需要的信息进行 subscribe，并回调至一个数组

3.对数组进行求平均，大于 2 的返回 true

代码实现：（部分）

```
void MTcallback(const sensor_msgs::LaserScan::ConstPtr &msg) {
   for(i = 0; i < 10; i++)
        {
          a[i] = msg->ranges[i];
        }
}



lidar_check::lidar_check()
{
  for(i = 0; i < 10; i++)
   {
    a[i] = 0;
   }

    //获取雷达数据，ranges
    ros::NodeHandle n,nh;
    ros::Publisher lidar_check_pub = n.advertise<messages::lidar_check_info>("lidar_check_info",100);
    ros::Subscriber movetip_sub_ = nh.subscribe<sensor_msgs::LaserScan>("/scan", 100, MTcallback);
}

void lidar_check::start()
{
//对ranges中1-10求平均值
   for(i = 0; i < 10; i++)
        {
          std::cout<<a[i]<<std::endl;
        }

//判断是否大于2
}
```

遇到的问题及解决方法

查找 lidar 发送的 topic 时，返回 inf 值
解决：inf 值为 std 内置数字，根据 lidar 定义，小于 30 返回 inf。因此可以将此数据舍弃，再取剩余数据平均值。

3.4.5 控制

详见嵌入式部分

3.5 测试结果

测试结果嵌套在“实现过程”中，请参考

3.6 可优化方案

由于时间仓促，我们没有很好地弄好双车通讯，因此在实战时两车很容易撞在一起，因此可优化方案为实现双车高效通讯。

4. 夏令营感想、总结

当鼠标光标到此处时,意外着 40 天的夏令营生活即将结束。总的来说，我们全组并不满意最终的结果，甚至无法接受。分析主要有以下客观与主观原因：

客观原因：

1. 整个夏令营期间，因为一些原因，组内的一些队员不得不提前离开，从嵌入式到算法到机械，三个方面我们都受到了人员的损失，导致到最后比赛时，我们只剩下六个人，从人员数量，备场气势上已经输给了其他队；
2. 全组只有一个人算是完全参加过 RM，当调试出现硬件问题时，其他人并不知道相关设备的调试和重置，这样就耽误了很多时间。

主观原因：

1. 方案设计上的缺陷，第一阶段的设计与第二阶段并没有形成迭代，相当于每个阶段都是重复造轮子，导致后期进度完全落后，没有给调试留下足够的时间；
2. 组长没有把握每个队员的特点与特长，安排工作往往对不上，各方面交流沟通不及时，导致最后只有两三个人调试两个机器人，而其他人想帮忙却帮不上。
3. 比赛心态失衡，在比赛期间，我们组犯了些低级的错误，如代码没编译完就断电了，比赛时没按拨码开关，供电出现问题……等等；

除了上述原因外，还有很多我们分析不到的，潜移莫化地影响着我们，导致我们最后的成绩并不理想。

诚然，40 天的夏令营生活我们并不是一无所获。在技术上，我们接触了视觉，ros,行为树……等等机器人相关的知识，并最终运用到了自己设计的机器人上。除此之外，在队内方案讨论、组间技术交流的过程中，我们收获了知识，收获了友谊，开阔了视野，一起刷过的夜，爆过的肝，让我们体会到一次次失败后阶段性成功的喜悦与成就感。特别感谢组内全体队员，虽然我们一直困难重重，在看不到任何希望的时候，我们还一起坚持着；特别感谢其他组的队员，在我们困难时，从物资、技术和心理上给予我们鼓励和帮助；特别感谢，夏令营的各位老师们，不厌其烦地给我们解决各种问题。

夏令营的结束，并没有结束我们对机器人技术探索的热情，每一次的失败都是下一个成功的垫脚石，未来的星海浩瀚，还等着我们去探索。

RoboMaster 大赛组委会

邮箱：robomaster@dji.com

官方论坛：http://bbs.robomaster.com

官方网站：http://www.robomasters.com

电话：075536383255（周一至周五 10:00-19:00）

地址：广东省深圳市南山区西丽镇茶光路 1089 号集成电路设计应用产业园 2 楼 202

微信

微博