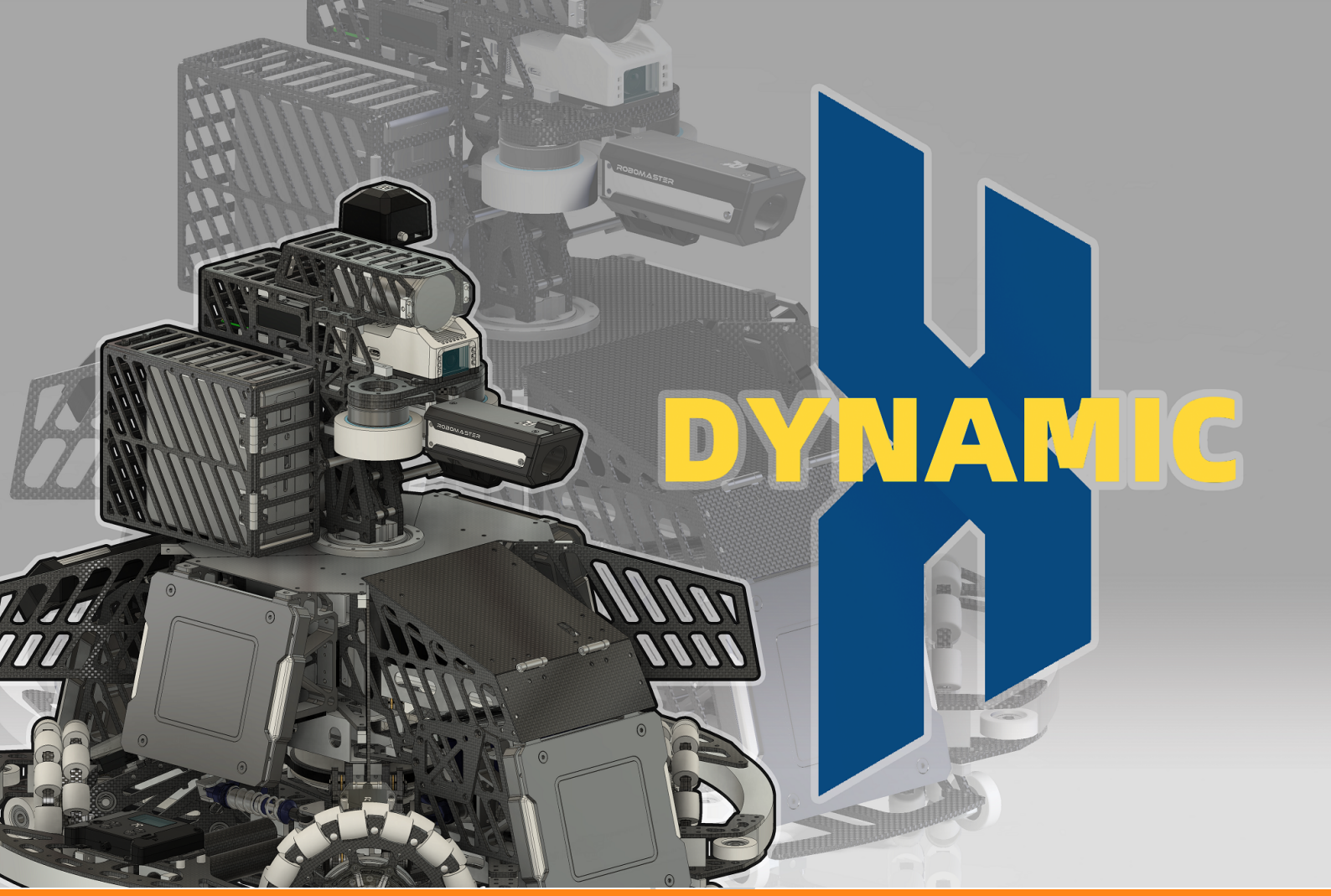


步兵机器人技术方案

RoboMaster 2022

组织：广东工业大学 DynamicX

时间：4 月 2022

目录

https://www.overleaf.com/project/623eafc63fa8c63db7ea3791

第 1 章 机器人功能定义

步兵机器人功能定义如 **图** 1.1所示。

图 1.1: 步兵机器人功能定义图

第 2 章 机器人核心参数

*/

2.1 机械参数

本赛季团队研发的步兵是中心供弹式、全向轮底盘式的步兵，其机械结构设计紧凑，合理且有效地利用空间。

2.1.1 基本参数

基本参数	参数量化
尺寸	长：543.08mm，宽：543.08mm，高：497.42mm，如 **图** 2.1所示
质量	15.2kg
重心位置	位于 Yaw 轴所在轴线上，且离地 120mm

表 2.1: 机械基本参数

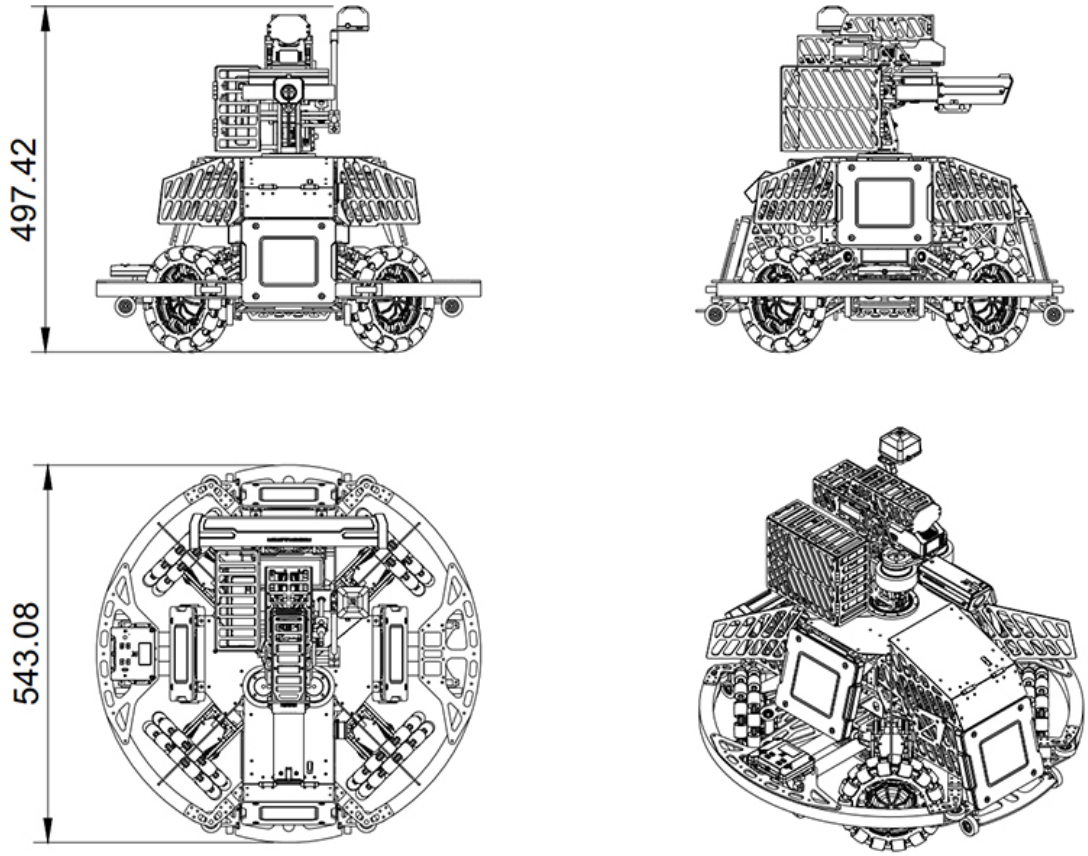


图 2.1: 机器人尺寸

2.1.2 机械执行器件

执行元件	用途	数量（个）
2006 电机	为拨盘提供动力源 驱动连杆打开弹仓	1 1
3508 电机	为整车的进退提供动能 驱使 pitch 轴转动 为弹丸提供初始动能	4 1 2
6020 电机	驱使云台周转运动	1

表 2.2: 机械执行器件

2.2 电路参数

整车有两个电容组，超级电容组和浪涌吸收电容组。其容量参数及工作电压如下所示：

- 超级电容容量：加入保护电路后参数为15.9 V / 15 F，标称容量为1896.075 J。工作电压范围为 $7V \sim 15.5V$。
- 浪涌吸收电容容量：参数为50 V / 20 mF，标称容量为25 J。工作电压范围为 $24V \sim 50V$。

车载模块及其用途、设计最大功率：

- 超级电容充电：用于给超级电容充电，设计最大功率为120 W。
- 超级电容升压：用于将超级电容升压至26 V 输出，设计最大功率为480 W。
- NUC 供电电源：用于给 NUC 供电，设计最大功率为200 W。
- IMU: 用于精确获取机器人姿态，提高运动的准确性。
- 荧光充能装置: 用于给荧光弹丸进行充能，设计功率为1.632 W。

2.3 性能参数

性能参数	参数量化
最大运动速度	平移 4.5m/s、旋转 20rad/s
最大运动加速度	平移 $4m/s^2$ 、旋转 $14rad/s^2$
最大爬坡角度	30°
云台自由度	Yaw 轴 360°，Pitch 轴俯角 30°、仰角 50°

表 2.3: 性能参数

2.3.1 飞坡

下面为飞坡全过程，为视频生成的等时间间隔照片。飞坡时性能参数：

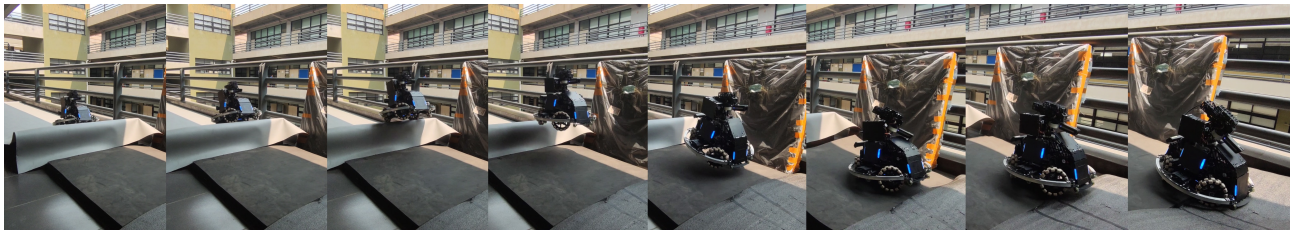
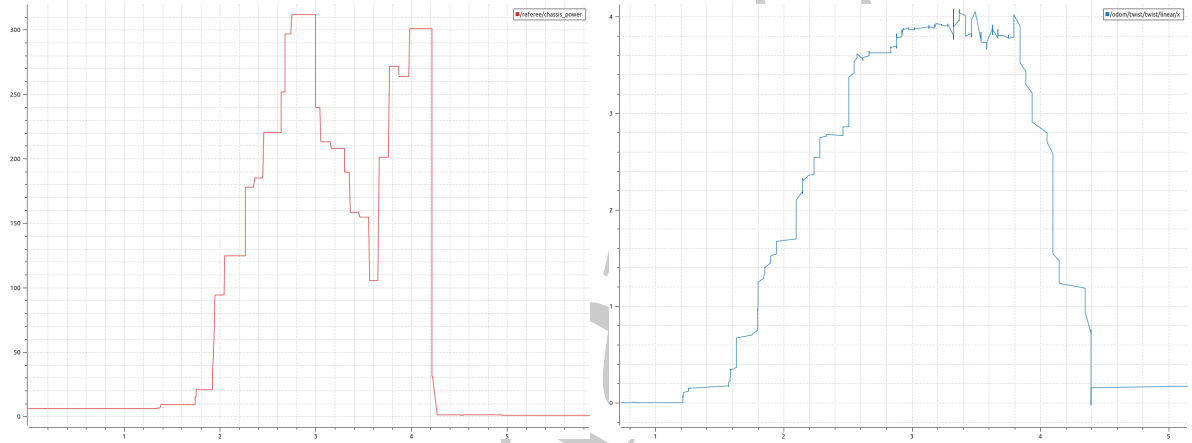


图 2.2: 步兵机器人飞坡

- 平地加速距离：3m
- 飞坡速度：4m/s
- 最大功率：300W

图 2.3(a) 和 **图** 2.3(b) 展示了飞坡时的功率和速度数据。



(a) 飞坡功率　　(b) 飞坡速度

图 2.3: 飞坡功率和速度数据

从图上可以看出，当步兵运动到坡边缘时，功率达到最大为 300w。同时可以看到出坡速度达到 4m/s 即可顺利飞过，因此只要加速距离加长即可在相对低功率下成功飞坡。

2.4 主要传感器参数

表 2.4 为传感器的型号、参数与数量

传感器名称	型号	参数	数量
大恒图像摄像头	MER-131-210U3C	分辨率 1280 *1024、帧率 (fps)210	1
IMU	BMI088	-	1

表 2.4: 步兵机器人传感器型号、参数及数量

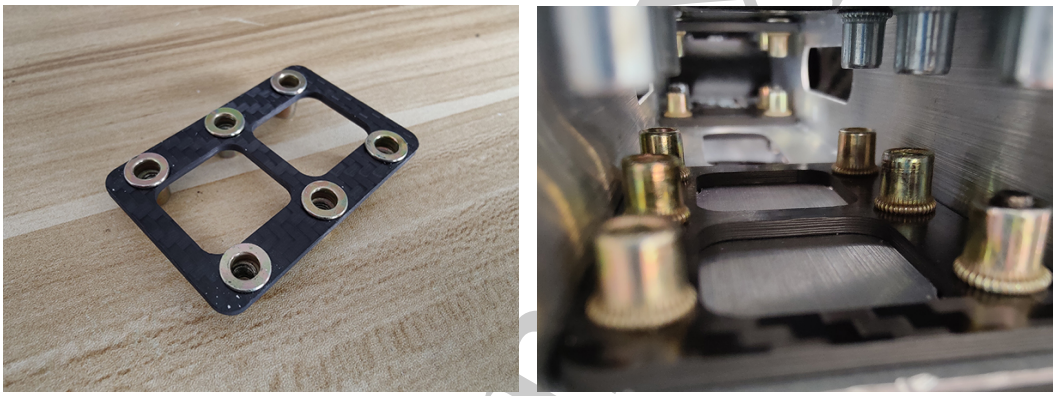
第 3 章 设计方案

3.1 机械结构设计

3.1.1 底盘设计

3.1.1.1 车架结构

车架主体采用薄壁粗铝方管，质量轻强度高。如**图** 3.1(b)所示我们在铝方管内嵌固定了拉铆螺母的碳板，这样铝管和铝管的连接就无需焊接，同时极大地简化了步兵底盘的拆装，并且由于碳板直接压在铝管上，碳板可以分散螺丝的拉力，降低铝管撕裂的可能性，但需要定期检查。



(a) 拉铆螺母固定板　　(b) 拉铆螺母装在铝管内

图 3.1: 铝管连接方式展示

如**图** 3.2(a)所示车架左右保护框采用碳管与铝方管连接，质量轻强度高且不会因为堆叠碳管而增加车体高度。

结合全向轮布局，我们将保护框设计为圆形，可有效减小因场地因素或对方机器人对步兵机器人移动和小陀螺的影响，因此保护框采用 20*20*1mm 的 6063 铝合金方管弯曲而成，弯曲后将一个带有保护框形状的槽和孔位的 2.5D 雕刻的玻纤模具扣在方管上打孔并切去多余部分，这样制成的保护框质量轻强度高且成本低。保护框与车架采用碳板连接，在承受大载荷冲击时共同吸收能量，保护车架。

如**图** 3.2(b)所示车架四周安装有落地防卡导轮，防止飞坡姿态不稳定时落地倒栽和脱落下台阶时卡住。底部 RFID 使用碳板转接，旨在方便拆装而不影响其他零部件，比如需要拆卸裁判系统。RFID 四周有碳板保护，防止步兵在下台阶时损坏 RFID 或被台阶卡住。

(a) 车架图　　(b) 落地防卡导轮

图 3.2: 车架展示

3.1.1.2 轮系结构

3.1.1.2.1 一代轮系结构 在一代轮系中，我们采用了与上海交通大学开源步兵相似的轮系结构。即移除电机出轴处的卡簧和垫圈，使得推力轴承可以顶住轴承外圈，更换内侧 10-22-6 的普通轴承，替换为同尺寸的角接触轴承。但是根据使用经验选择了带防尘壳的推力球轴承，该轴承可以降低轮系的维护难度，同时因为防尘壳的存在，三片式的推力球轴承被组合成了一个整体，便于轮系的安装。如**图** 3.3所示全向轮法兰采用胀套与输出轴连接，法兰上有 6 个 M4 螺纹孔，6 根 M4 螺丝穿过两层全向轮固定板和垫高块后与全向轮法兰固定，简化了全向轮的拆装。同时全向轮内轮廓与全向轮法兰圆柱段外轮廓过度配合，保证了全向轮与电机输出轴的同心度。全向轮棍子采用航发 6 寸低硬度全向轮棍子，该棍子与地胶的摩擦系数大。轮毂采用三层碳纤维板替换原厂的铝合金轮毂夹住滚子转轴，降低了飞坡后轮毂受到冲击变形的概率，同时降低了质量。

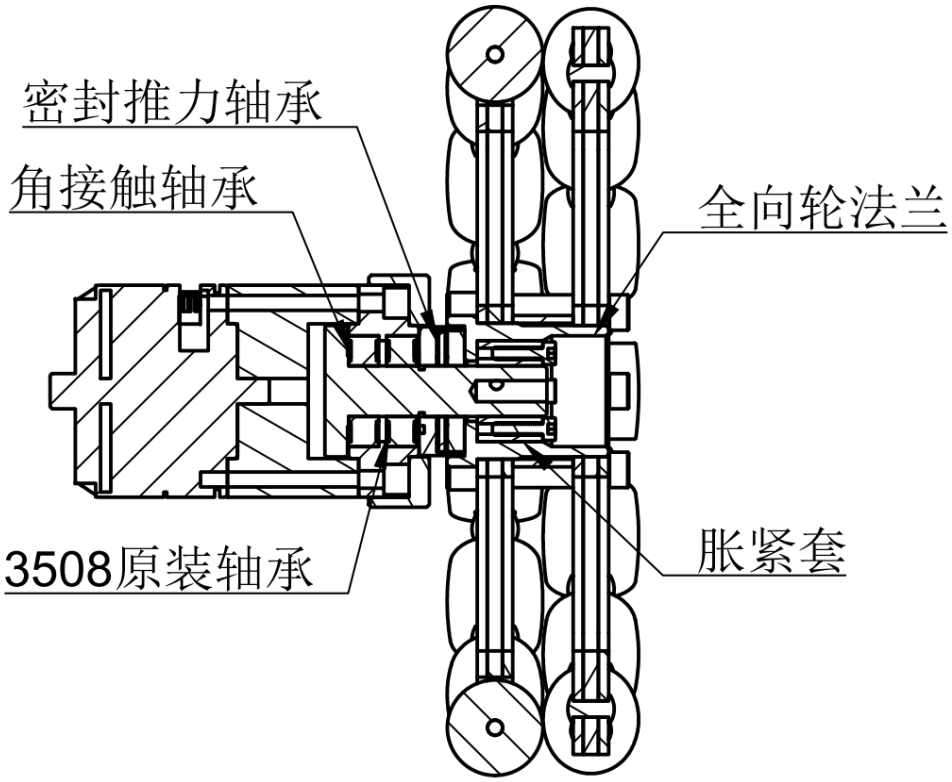
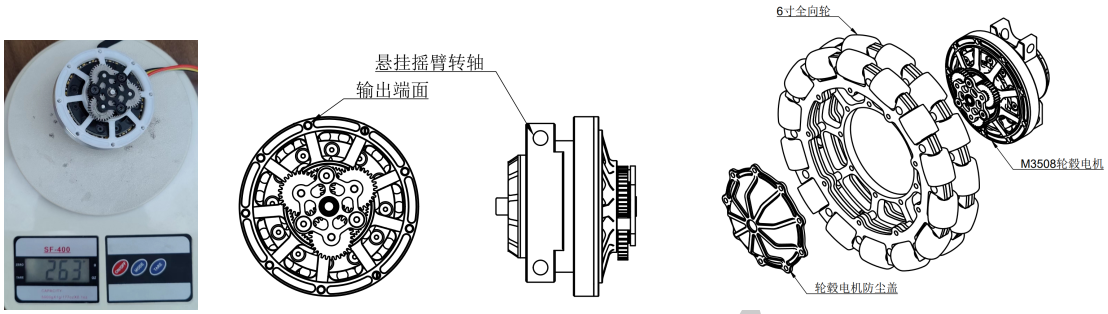


图 3.3: 轮系剖面图

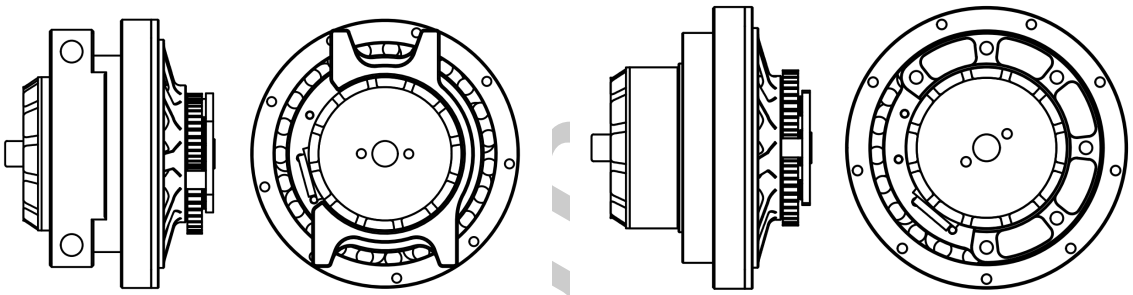
3.1.1.2.2 二代轮系结构 为了简化结构和减重，在二代轮系中我们采用了**图** 3.4所示自制的减速箱轮毂电机。



(a) 轮毂电机质量 (b) 轮毂电机图 (c) 6 寸全向轮轮毂电机爆炸图

图 3.4: 轮毂电机及其配套的全向轮

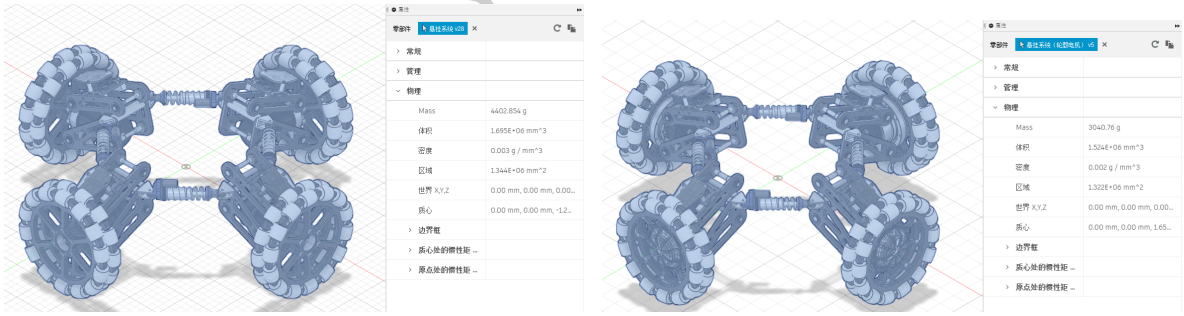
该减速器可自由设计减速比（1:11-1:21）同时如**图** 3.5所示输入端可根据需求改变固定方式，例如采用端面固定或四连杆摇臂。轮毂电机采用一个大号四点接触轴承作为输出端轴承，提高了该电机的抗冲击能力，同时简化了轴系，降低了全向轮的拆装的难度。



(a) 轮毂电机四连杆摇臂固定 (b) 轮毂电机端面固定

图 3.5: 两种轮毂电机固定方式

该电机在采用端面固定的时候仅重约 270g，并且由于减速器采用端面输出，不用像普通 3508 一样添加联轴器即可固定轮子。如**图** 3.6所示，更换轮毂电机后减重效果显著，整车可减重 1.4kg。

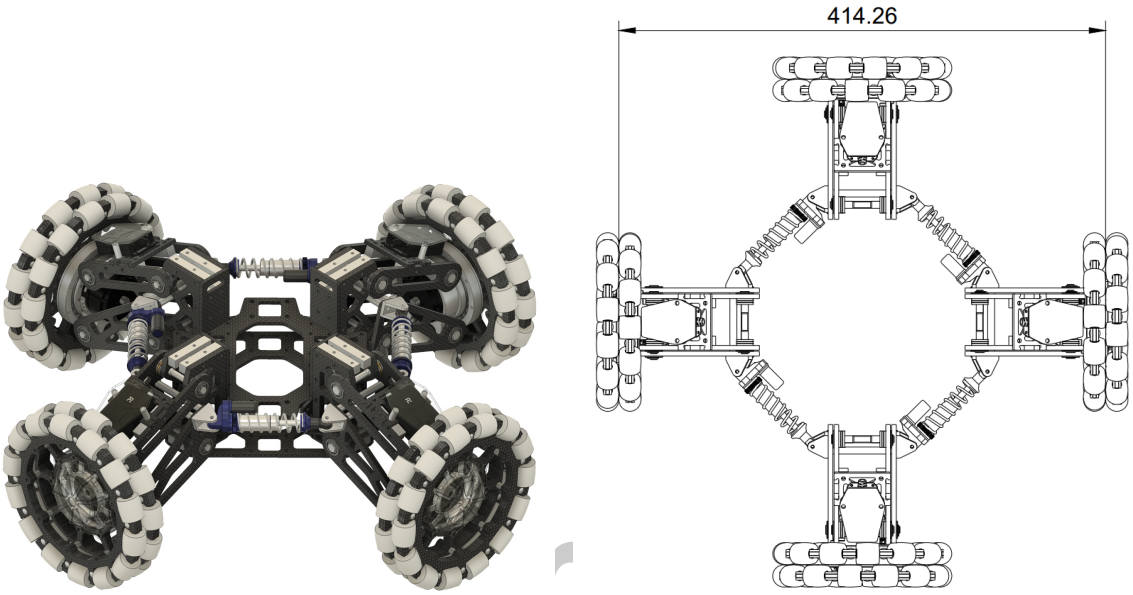


(a) 普通 3508 悬挂系统系统质量 (b) 轮毂电机悬挂系统系统质量

图 3.6: 悬挂系统质量对比

3.1.1.3 自适应悬挂结构

我们采用了与哈工大深圳开源步兵相似的环式自适应平行四边形悬挂，四个避震器水平放置介于两轮组之间。保证轮组抓地、结构简单、可以空出中心云台位置。但根据地盘布局将避震器的安装位置下移，提高了底盘的空间利用率。通过设计自适应行程，确保地盘在小陀螺上坡时不会有轮子悬空。同时，为提高小陀螺旋转速度，我们尽可能的减小了全向轮对边的轮距。



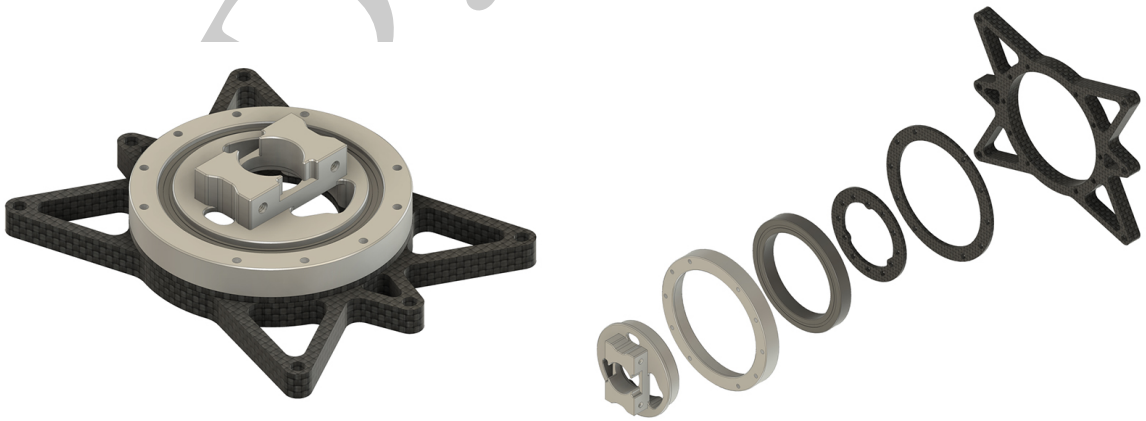
(a) 悬挂系统图　　(b) 轮距图

图 3.7: 悬挂系统展示

3.1.2 云台设计

3.1.2.1 云台架结构

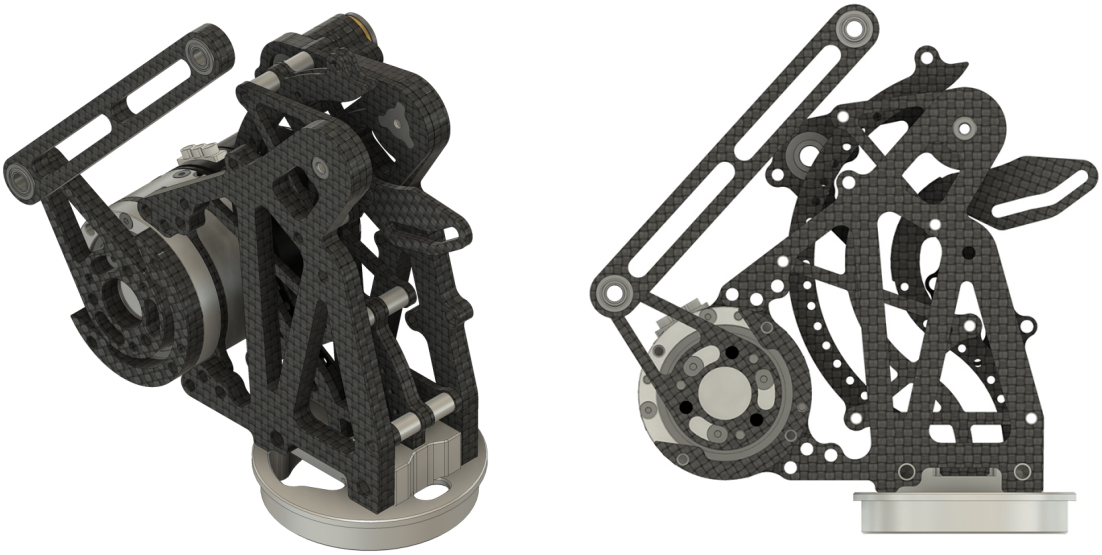
Yaw 轴轴承为了稳定性采用 RA5008 交叉滚子轴承，间隙极小，抗冲击性能好，可以多次承受步兵飞坡失败时的冲击而不失效。



(a) yaw 轴转轴　　(b) yaw 轴转轴爆炸图

图 3.8: yaw 轴转轴展示

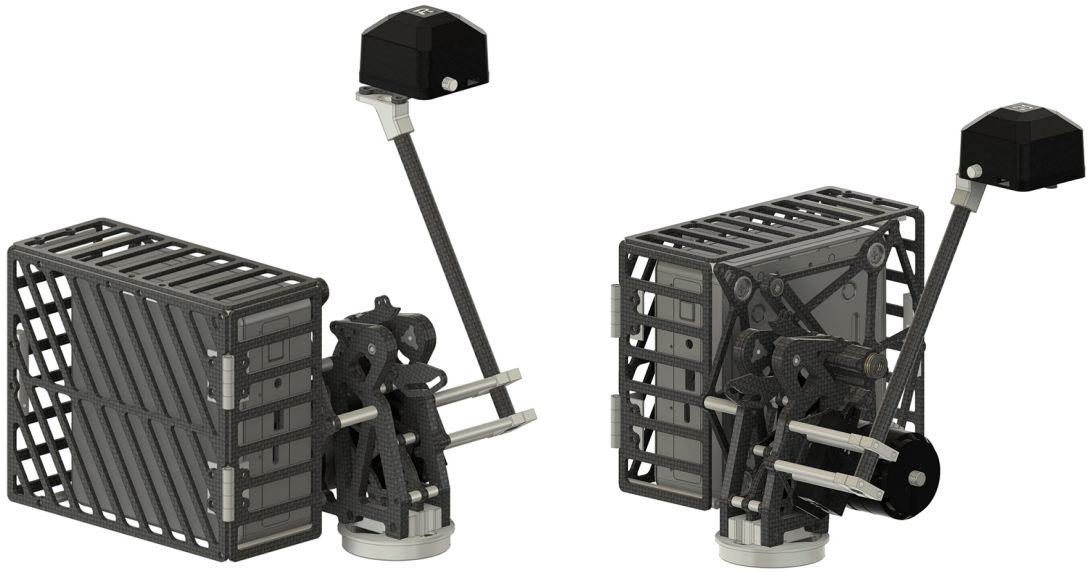
为了减小 yaw 轴转动惯量，云台架结构沿 yaw 轴分布。为配合两侧装甲板高度并满足规则限高，我们将云台 pitch 轴转轴到 yaw 轴轴承的高度降低至 90mm，依然满足云台-30 度俯角，同时减短了弹链长度，并可以满足哨兵和无人机的使用需求。云台架主体采用两块 5mm 碳板做支撑，质量轻强度高。为减小 pitch 轴惯量，我们没有选择将电机放置 pitch 轴上做配平，而是将 pitch 轴的 M3508P11（下文会提及）固定在云台支架通过连杆驱动云台俯仰，并另外添加重力补偿装置来降低电机负载。



(a) 云台支架斜二测隐藏 minipc 和 uwb　　(b) 云台支架侧视图隐藏 minipc 和 uwb

图 3.9: 云台支架展示

由于我们采用中置的弹链，minipc 挂在云台架侧面。为保护 minipc，我们在 minipc 外添加了一个碳纤维板构成的保护壳，并留出空间用来理线和放置 usb2can 等模块。由于定位模块上方 145° 内不得被导体遮挡，我们将 uwb 架高。这段距离内不需要连接其他结构，以此我采用一根 8mm 的薄壁碳管配合管夹和无人机脚架固定件组成 uwb 支架。



(a) 云台支架 minipc 保护　　(b) 云台支架 uwb 固定

图 3.10: minipc 和 uwb 安装图

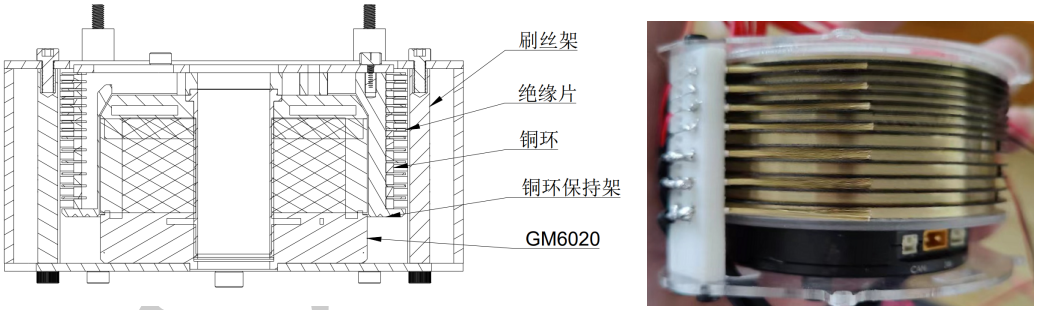
3.1.2.2 自制 6020 一体式滑环

下供弹步兵若要实现小陀螺功能通常需要将电机偏置以留出过孔滑环的位置，并使用同步带驱动云台旋转。电机偏置势必会导致 yaw 轴驱动部分体积和质量偏大，不利于车内布局，同时，同步带的弹性会降低 yaw 轴的控制带宽，过孔滑环内置的两个滚珠轴承也可能因为装配误差不与云台支架的交叉滚子轴承同轴，增加 yaw 轴旋转阻力。

(a) GM6020 滑环一体化　　(b) 偏置 GM6020 独立滑环

图 3.11: 两种下供弹云台架

因此我们提出了一种将滑环套在电机外部的方案，该方案 GM6020 可以直驱云台，提高了控制带宽；滑环转子与电机转子固定，与电机共用轴承，降低了 yaw 轴的转动阻力。同时我们可以根据不同线路对电流的要求设计滑环电刷导线数量和铜环宽度，降低了滑环的高度和重量。由于该方案电机直驱云台，拆装 yaw 轴电机时不需要考虑同步带张紧的问题，整个模块的固定仅需 7 根螺丝，简化了云台的拆装。

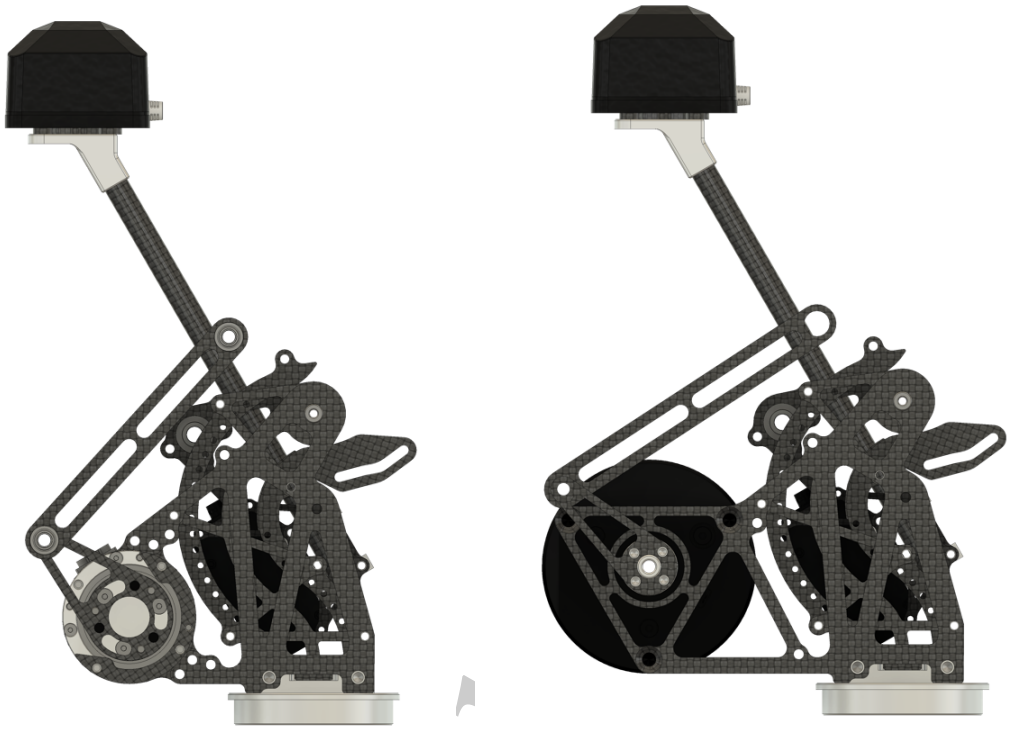


(a) GM6020 外套滑环剖面图　　(b) GM6020 外套滑环实物图

图 3.12: GM6020 外套滑环展示

3.1.2.3 自制云台电机（M3508P11）

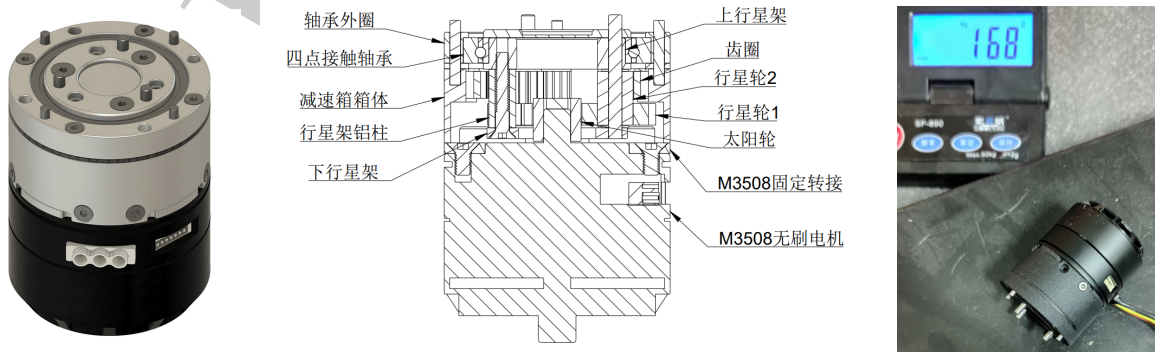
目前步兵云台的主流方案是使用 GM6020 驱动，但 GM6020 电机扭矩密度低。如使用 M3508 电机直接驱动云台则力矩太小。



(a) 云台支架 M3508 云台电机　　(b) 云台支架 GM6020

图 3.13: 两种 pitch 轴驱动方式

为提高云台响应，我们在步兵上使用了自制的 3508 云台电机，该电机仅重 170g。我们将 M3508 原装的减速箱替换成自制的减速箱，自制减速箱为 2K-H 结构，齿轮采用线切割中走丝加工制成，减速比为 11。自制的 3508 云台电机减速器背隙＜ 30Arc-sec，徒手感知不到间隙。减速器输出端采用四点接触轴承，抗冲击性能强，并且为方便固定采用端面输出。减速后最大输出力矩约为 3N · m。该云台电机虽没有绝对位置编码器，但由于使用在 pitch 轴上不需要全向转动，仅需在电机上电时旋转到限位处通过碰撞限位来确定 pitch 轴初始位置，即可实现在不额外添加传感器的前提下控制云台 pitch 轴。



(a) 3508P11 斜二测　　(b) 3508P11 剖面图　　(c) 3508P11 实物图

图 3.14: 自制 3508 云台电机展示

3.1.2.4 Pitch 轴重力补偿

重力补偿机构应用到 RoboMaster 机器人中可以优雅地解决云台 pitch 轴电机发热、惯量过大和控制效果差的问题；它纯机械的工作方式，结构简洁，零件多为标准件，设计限制少，免维护。该结构由简单而又实用的理论支撑，易于实现且通用性强，在 RoboMaster 中可以被称为一种优雅的机械机构及解决方案。

针对上述重力补偿的作用，经理论分析可知 pitch link 所受重力作用在 pitch joint 上的重力矩 τ 为

$$\tau = mgl\sin\theta \tag{3.1}$$

重力矩在 A 点的作用等价于 f_g。关系如下：

$$f_g = mg\frac{L}{h_1} \tag{3.2}$$

图 3.15 结合静力学可知，显然 f、f_z、f_g 构成的力三角形与 ΔAOB 相似。则有

$$\frac{f_g}{f} = \frac{h_2}{x} \tag{3.3}$$

将 (3.2) 代入 (3.3) 消去 f_g 得：

$$\frac{mgL}{fh_1} = \frac{h_2}{x} \tag{3.4}$$

将弹簧拉力 $f = kx$ 代入 (3.4) 消去 f 得：

$$\frac{mgL}{kxh_1} = \frac{h_2}{x} \tag{3.5}$$

其中 k 为弹簧的劲度系数。

注意到 (3.5) 左右分母均有 x，约去得：

$$\frac{mgL}{kh_1} = h_2 \tag{3.6}$$

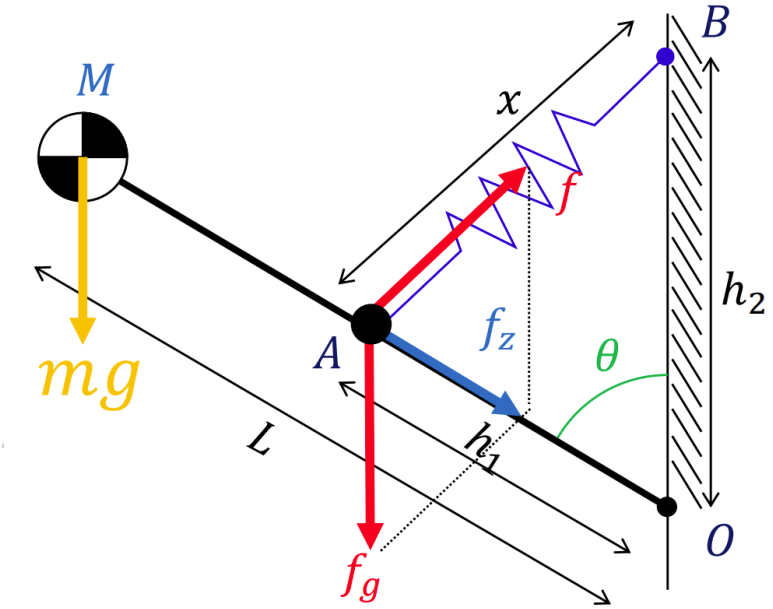


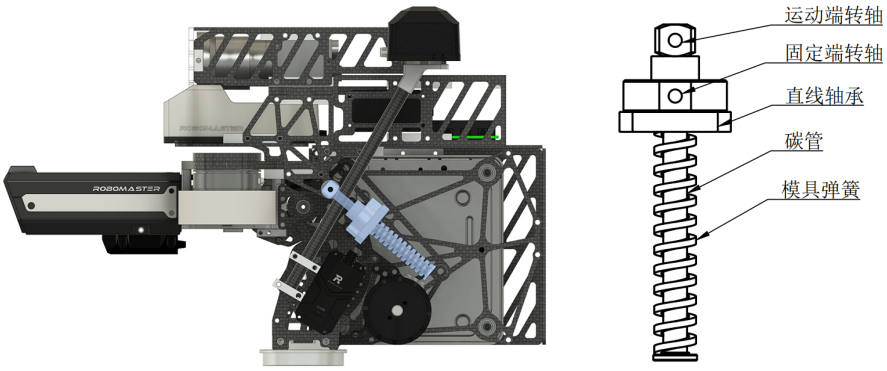
图 3.15: 机构的简图和受力分析，图源[1]

注意到 (3.6) 中各符号均代表常数，与 θ 无关，换言之：当拉簧的 k 满足 (3.7) 时，**图** 3.15 中的机构（系统）将会处于静力平衡状态。即：无论 θ 的值是多少，拉簧产生的拉力作用在 pitch joint 上的力矩与 pitch link 作用在 pitch joint 上的重力矩抵消，也就是实现了完美平衡。

$$k = \frac{mgL}{h_1h_2} \tag{3.7}$$

通过实验我们发现零原长弹簧直线轴和两个旋转轴的阻力对平衡效果有很大影响。以此在今年的重力补偿机构中，我们不再使用石墨铜套和杆端轴承，因为它们的运动阻力过大，严重影响平衡效果。在今年的机构中，

我们使用滚珠直线轴承和深沟球轴承作为代替。

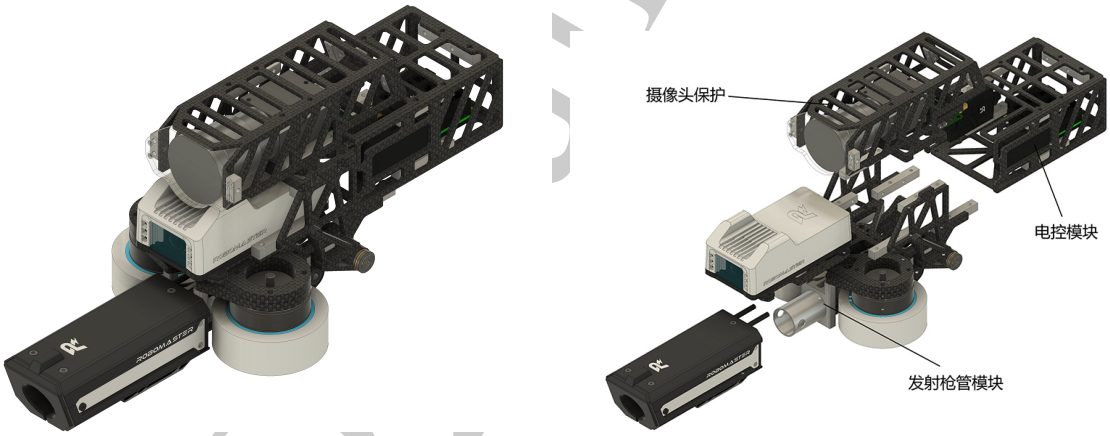


(a) 重力补偿装置安装　　(b) 零原长弹簧

图 3.16: 重力补偿实现方案

3.1.3 发射设计

发射结构使用 3508 电机（去减速箱），摩擦轮直径 60 mm，摩擦轮中心距 73 mm、最小间距 13 mm。整体设计中将发射模块、电控模块与图传摄像头分模块前中后放置，实现发射 Pitch 轴质心靠近转轴，同时也可单独对某模块进行拆卸维修而不影响其他模块。如 **图** 3.17所示。



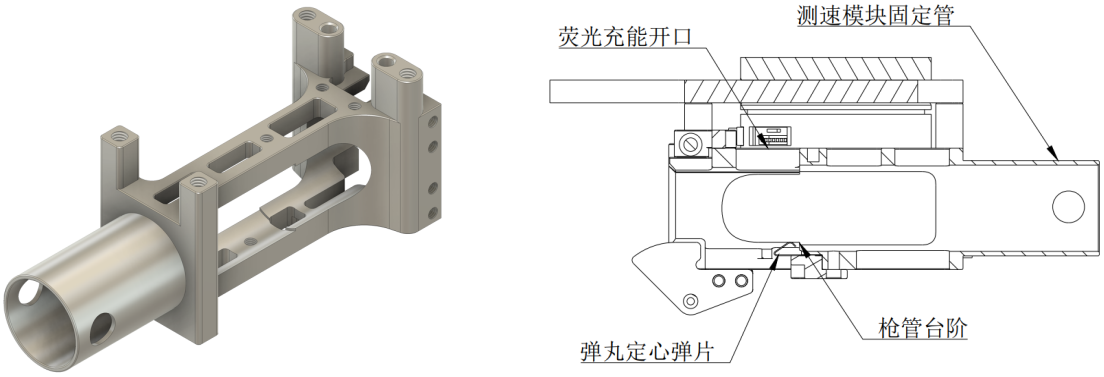
(a) 发射轴测图　　(b) 发射爆炸图

图 3.17: 发射展示

3.1.3.1 枪管结构

我们于去年提出的板堆叠发射在实际使用中由于队内加工精度不足，导致定心效果不佳，并且由于零件多且厚度薄，外包价格与整体的枪管相比并无优势。因此我们今年采用一体化枪管，与板堆叠枪管相比零件少，拆装方便。

枪管采用 6061 铝合金铣削而成。摩擦轮固定板由 4 个 m4 螺丝与铝件固定，同时由于螺纹无法起到定位作用，我们在铝件上添加了两个销钉孔，确保了摩擦轮固定板和枪管铝件的相对位置，进而保证了摩擦轮的定位精度。我们将 17mm 测速模块套于枪管前方的凸台上，通过拧紧测速模块夹紧螺丝来固定测速模块。该枪管固定方案结构简单强度高，测速模块拆装方便，我们认为是一种较为优雅的测速模块固定方案。

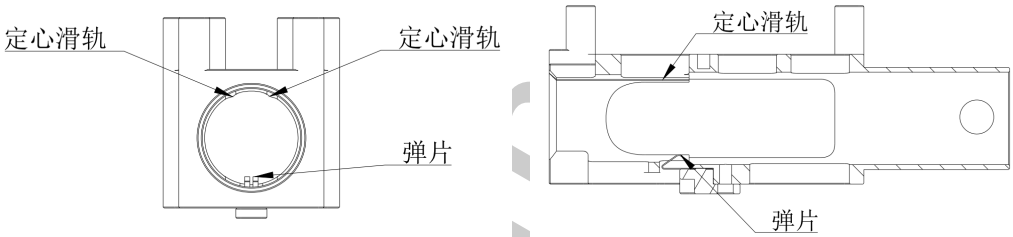


(a) 枪管斜二测图　　(b) 发射机构剖面图

图 3.18: 枪管展示

3.1.3.2 弹丸定心结构

如 **图** 3.19所展示，我们的设计中采用了 5 号电池弹片与两条定心滑轨构成近似三点定心。



(a) 三点定心正视图　　(b) 三点定心剖面图

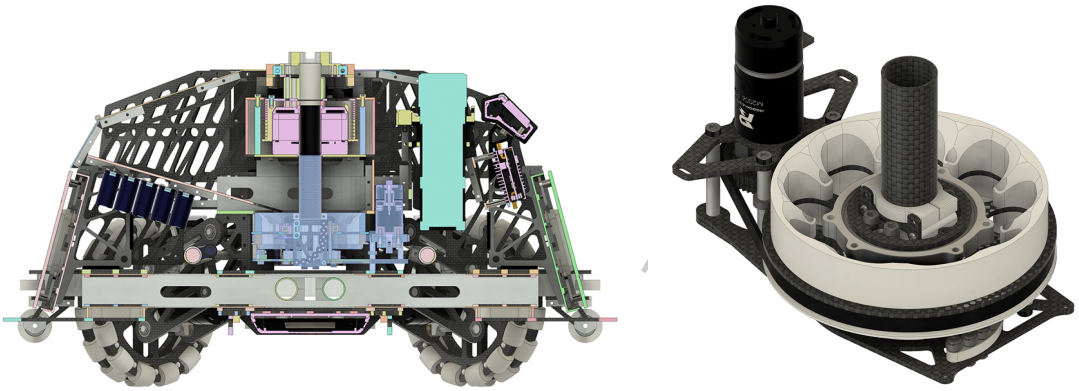
图 3.19: 弹丸定心结构展示

弹丸从待发位置到与摩擦轮接触前，弹丸会被单发限位的 5 号电池弹片向上顶在两条定心滑轨之间，实现了弹丸定心，这样可以保证弹丸每次与摩擦轮在同一位置接触，从而获得较高的发射精度。弹丸在与摩擦轮接触后枪管直径扩大，确保弹丸在加速过程中和加速后不会被外界因素干扰，避免了弹丸和枪管摩擦导致旋转。

3.1.4 中心供弹设计

3.1.4.1 拨盘结构

为减小步兵两侧轮间距，提高陀螺速度，缩短链路长度，实现步兵轻量化，降低重心等一系列考虑，我们在北京理工大学和哈工大开源中心供弹的基础上研发了小弹丸的中心供弹拨盘。



(a) 底盘剖面图　　(b) 中心供弹斜二测图

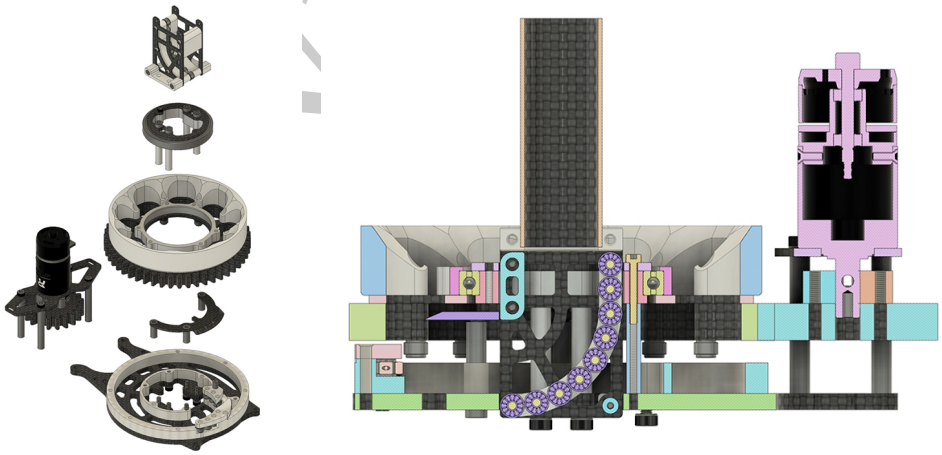
图 3.20: 中心供弹展示

该拨盘的供弹原理与开源中心供弹相同，故不在此赘述。

为减小拨盘的尺寸和重量，我们在设计中减少了拨盘的齿数。同时我们提出了一种将拨盘轴承放置在拨齿内圈的结构，这样我们可以选择小直径的薄壁深沟球轴承，降低成本的同时做到了轻量化。

中心供弹采用 M2006 通过齿轮驱动拨叉，在保证拨弹频率的前提下我们选择尽可能大的减速比来提高推弹力度，因此齿轮传动减速比为 2:5。

拨盘可根据底盘高度调整预置弹丸数量，在步兵的版本中，我们可预置三层弹丸，在高频发射时不会出现空拨问题。



(a) 中心供弹爆炸图　　(b) 中心供弹剖面图

图 3.21: 中心供弹结构

3.1.4.2 弹链结构

链路使用 2 层 1mm 碳板和 1 层 2mm 碳板堆叠，利用 2mm 碳板垫高布置微型轴承减小链路阻力。该链路避免小的拐弯半径带来的推弹阻力，并保证 pitch 轴在任何角度时弹丸相对摩擦轮的距离不会发生变化，避免误发射导致浪费弹丸。

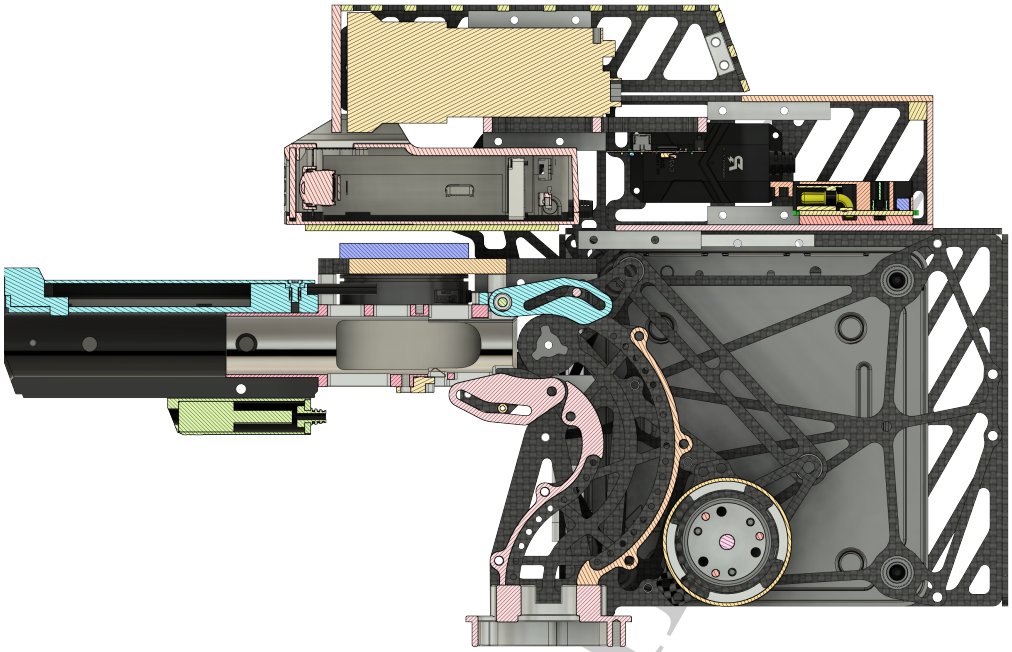
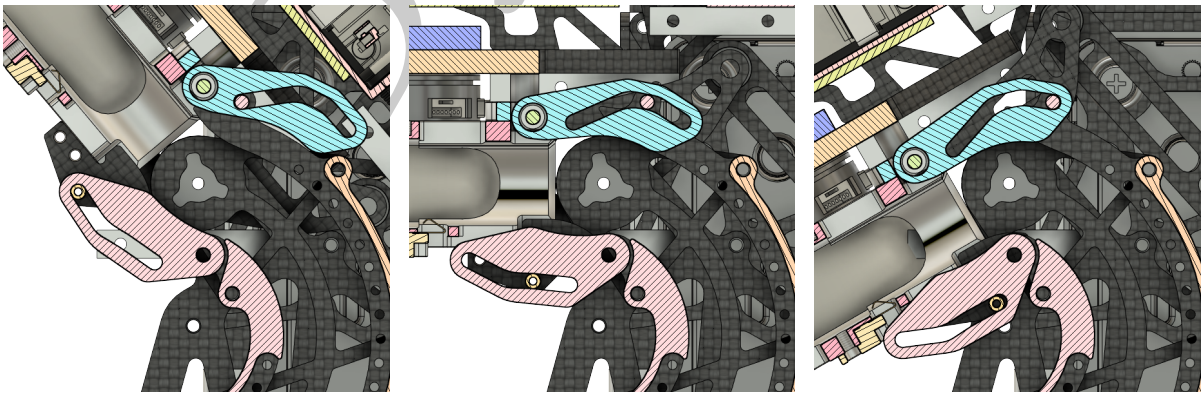


图 3.22: 云台弹链

弹链轴使用了上下两个活动滑槽，使发射机构在 30 度俯角至 50 度仰角均可以流畅发弹，并且保证弹丸到摩擦轮的距离不会发生变化。

滑槽设计通过在 Pitch 轴活动范围内均匀选取四个点，画出滑块会经过的点，同时不断调整活动滑槽的转轴和滑块的位置，来保证弹丸在滑槽中平稳运动，并且滑槽不会和云台结构干涉。在确定转轴位置和滑块经过的四个点后，将这四个点连线来拟合滑块的运动轨迹。最后在滑槽上添加圆角，避免滑块卡住。



(a) P 轴正 50 度　(b) P 轴水平　(c) P 轴负 30 度

图 3.23: P 轴弹链滑块演示

3.1.5 工艺选择

在步兵机器人整体的设计过程中，综合机械制造的合理性，再结合本学校的加工设备资源、外包零件经费等情况，在步兵机器人中采用了比较多的用 2D 雕刻加工玻纤板和碳纤板的结构设计（正常迭代使用玻纤板，最终比赛成车使用碳纤板），在一些必要的铝件上优化设计方案，设计出的铝件可以利用团队 3 轴铣床自加工，若出现复杂特征体的加工零部件，在经费充裕的情况下也可接受外包，结合 3D 打印件、部分弯管零部件、部分线切割的厚金属部件作辅助，部分工艺选择如 **图** 3.24 所示。

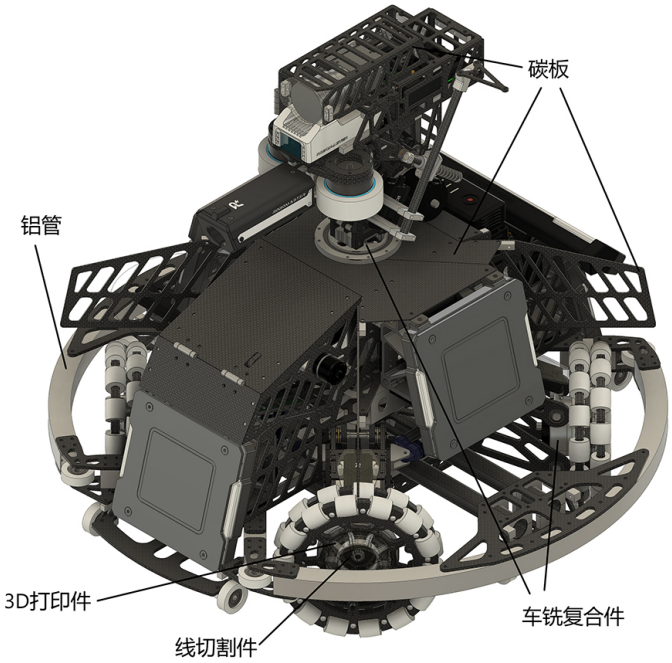
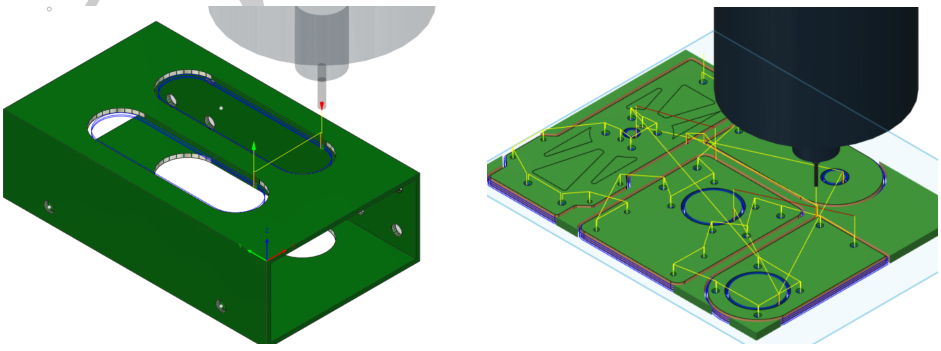


图 3.24: 步兵机器人工艺选择示意图

3.1.5.1 2D 雕刻

根据自己队伍的实际加工方式的情况，利用雕刻机可以可快速加工 2D 轮廓类的零部件，便于加快队伍在备赛期间的迭代速度，也可以节省迭代的成本，在结构设计上就尽可能使用板类零件，达到 80 % 可队内自我加工，尽量避免出现外包商的拖延而减缓进度的情况。加工如 **图** 3.25所展示。

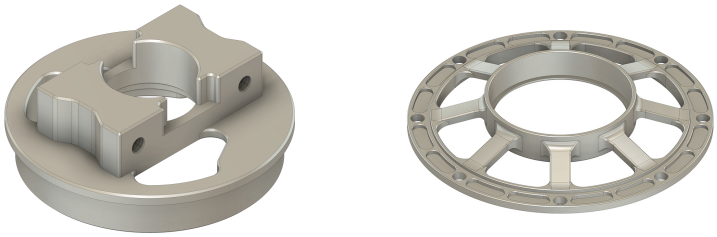


(a) 铝管　　(b) 碳板

图 3.25: 2D 雕刻

3.1.5.2 车铣复合加工

在整车设计过程中，不可避免的会出现一些因零部件的功能需求而导致外型复杂，或者是零件的尺寸精度要求高等情况，需要使用车铣复合加工，此时队内的加工设备已经无法制造，发外包商制造成了此类零部件的第一选择，比如发射的枪管铝件、轮毂电机的固定架、云台 yaw 轴的固定铝件等。如**图** 3.26所展示。

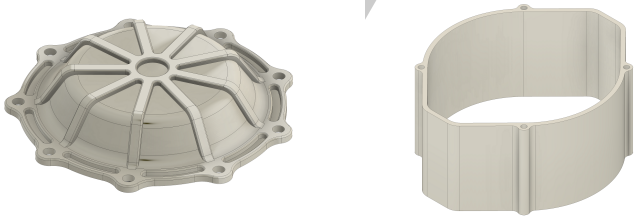


(a) 云台铝件　　(b) 轮毂电机铝件

图 3.26: 车铣复合加工

3.1.5.3 3D 打印

3D 打印作为非规则模型的加工非常便利，面对着一些拥有曲面的、强度要求不高、作平常固定非支撑件的零件，首选均为 3D 打印。如**图** 3.27所展示。

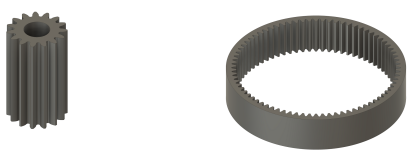


(a) 防尘盖　　(b) 滑环外壳

图 3.27: 3D 打印

3.1.5.4 线切割

对于厚度大的零部件，如自制减速器中的齿轮和齿圈，材料为模具钢，此时使用外包线切割加工更为合适。如 **图** 3.28所展示。



(a) 齿轮　　(b) 齿圈

图 3.28: 线切割

3.1.5.5 弯管

由于步兵机器人的自旋速度较快，并且在比赛场地激烈交战的通道一般比较狭窄，极易碰撞到其他机器人或者场地边缘，故外保护框设计成圆形，可以有效避免碰撞的自旋速度损耗，材料选择为铝管，利用手动弯管机加工成型。如 **图** 3.29所示。

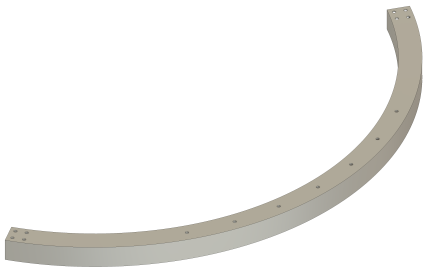
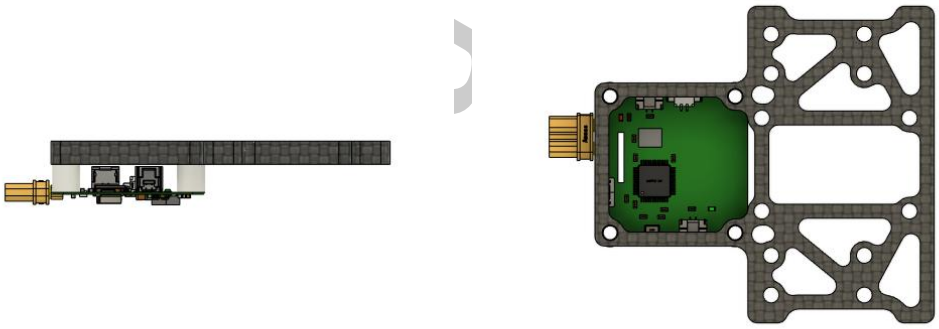


图 3.29: 保护框弯管

3.1.6 传感器的设计安装、电路板的固定及连接

机器人各个传感器与电路板都采用螺丝连接的方式，与板件刚性连接，在需要绝缘的地方使用绝缘材质连接，既保证各个传感器与电路板的正常工作，同时也保证各个硬件与机器人的固定。

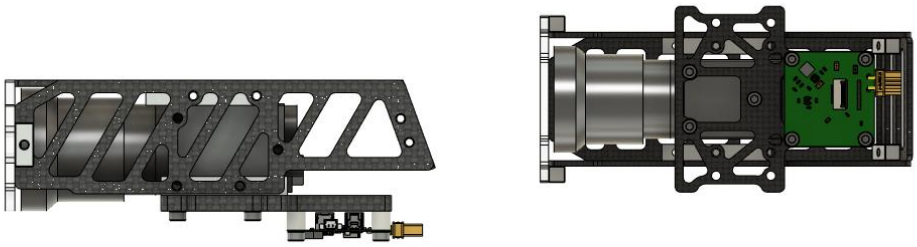


(a) IMU 固定侧视图　　(b) IMU 固定俯视图

图 3.30: IMU 固定展示

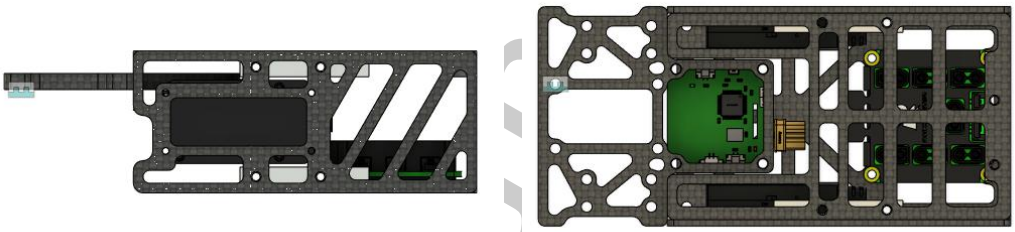
如 **图** 3.30 所展示，惯性测量单元 IMU 由四颗螺丝固定在四根带螺纹的尼龙柱上，再通过螺丝连接的方式将尼龙柱与一块高刚性的板件连接，通过这种方式，使 IMU 固定在一块高刚性的板件的下端。摄像头通过螺丝连接在 IMU 固定板件上端，使摄像头与 IMU 可以作为一个整体的模块在云台上安装与拆卸，避免控制组需要频繁地对 IMU 和相机进行联合标定。尼龙柱在此处不仅起到 IMU 的固定作用，保证 IMU 与板件的刚性连接，同时也可以起到绝缘作用，防止金属与 IMU 直接接触而导致短路损坏 IMU。

如 **图** 3.31 所展示，摄像头通过四根螺丝与摄像头固定板刚性连接，保证摄像头在机器人运动过程中的连接稳定性。摄像头四周通过 2mm 板件围成摄像头保护壳，板件通过板板连接件连接，以保证摄像头保护壳的坚固性，确保在比赛过程中对摄像头的稳定保护。在摄像头前端装有一块 3mm 亚克力板，既保证了对摄像头镜头的保护，同时不会遮挡摄像头的视角。



(a) 摄像头固定侧视图　　(b) 摄像头固定仰视图

图 3.31: 摄像头固定展示



(a) 电路板放置侧视图　　(b) 电路板放置俯视图

图 3.32: 电路板放置展示

如 **图** 3.32 所展示，机器人各个电路模块都是使用螺丝与板件固定，以云台为例，中心板通过四根螺丝固定在底板上，两个电调通过四根螺丝固定在左右两侧板上，且各个板件通过板板连接件互相连接成一个封闭的盒子，以保护盒子中的各个电路模块，避免在比赛中因为受到弹丸的击打而损坏。

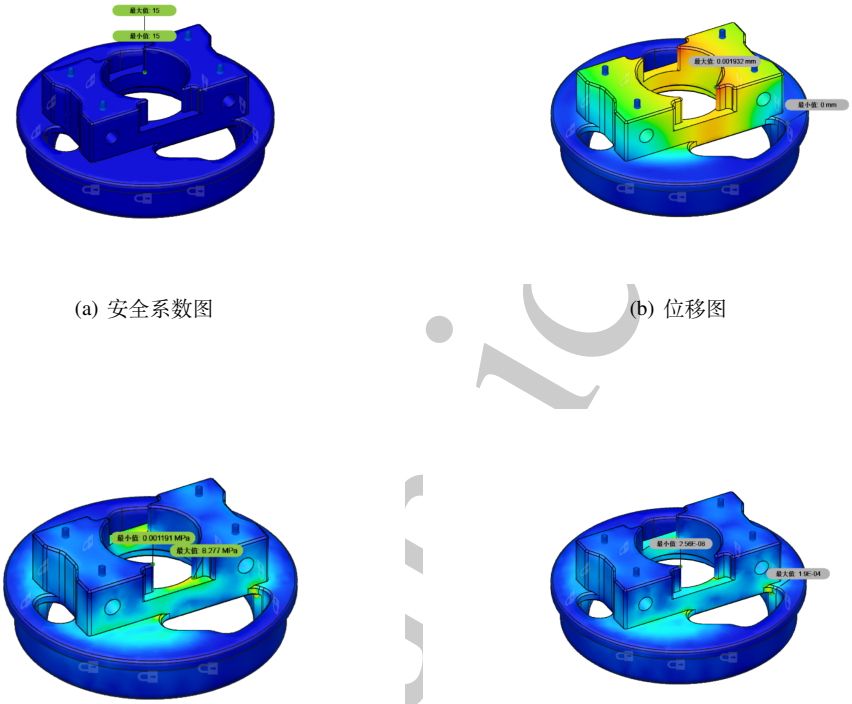
3.1.7 核心零件的有限元分析、静动力学分析

分析案例为一体式云台安装座，在设定约束条件为 4 个螺栓固定圆柱面，车体处在水平面运动状态下，载荷情况为 200N 单向力（竖直向下）1300N 单向力（竖直向上）以及 200N m 力矩（相当于 2Kg 的重物所提供的负载力在加速度大小为 10m/s^2 的运动状态下）。各种数据是从设想车体受到冲击时（例如飞坡），其负载力的竖直分量和力矩。

如 **图** 3.33(a) 所展示，仿真结果仍然具有 15 的最低安全系数，考虑实际工况的载荷比仿真载荷要低很多，且当前安全系数显示该零件至少仍可承受 15 倍当前载荷，可得该零件的强度已经很满足真实应用场景，飞坡时云台 yaw 轴与底盘连接处发生断裂的可能性小。

如 **图** 3.33(b) 所展示，在载荷情况与有限元分析一致的情况，且车体处在水平面运动状态下，有限元仿真结果的最大位移量为 0.001932mm，可见其零件刚性也十分满足真实应用场景，正常水平运动不会因为刚性问题导致云台抖动。

如 **图** 3.33(c)，如 **图** 3.33(d) 所展示，安装座底面有螺纹孔固定约束。车体在水平面静止状态下，两侧 4 个螺丝孔里边下表面有 20N 的垂直向下的单向力。仿真结果的最大应力为 8.277Mpa，最大应变基本为 0，完全符合需求。



(a) 安全系数图　(b) 位移图

(c) 应力图　(d) 应变图

图 3.33: 有限元仿真分析展示

3.2 硬件设计

3.2.1 整体硬件框图

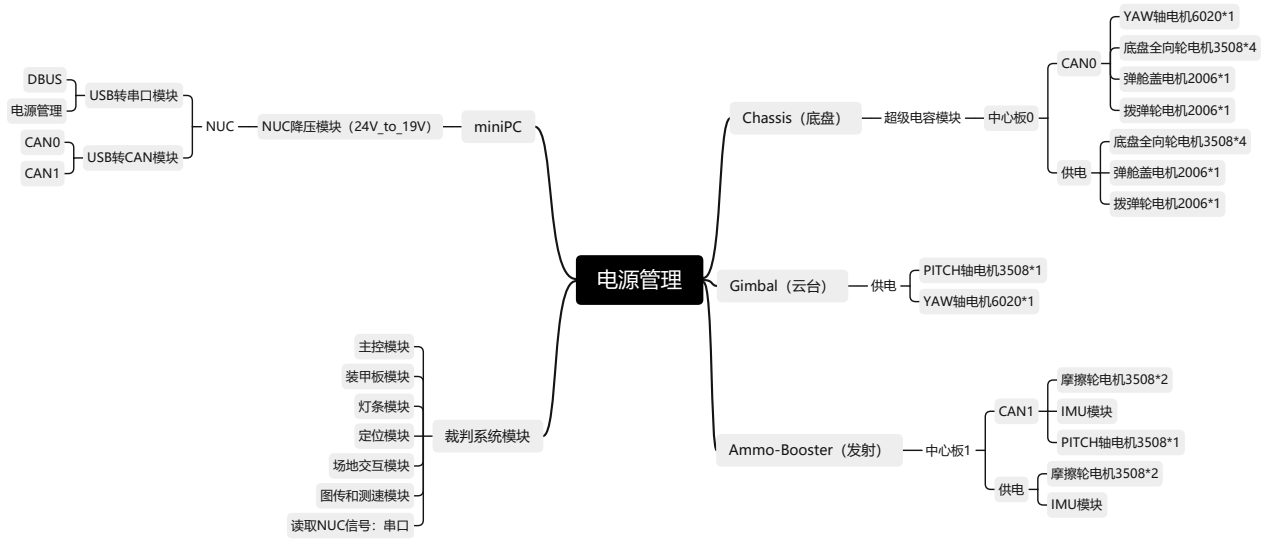


图 3.34: 整体硬件框图

图 3.34由电源管理模块出发，从通信和供电两方面介绍了我们步兵机器人的整体硬件电路连接情况。

3.2.2 超级电容

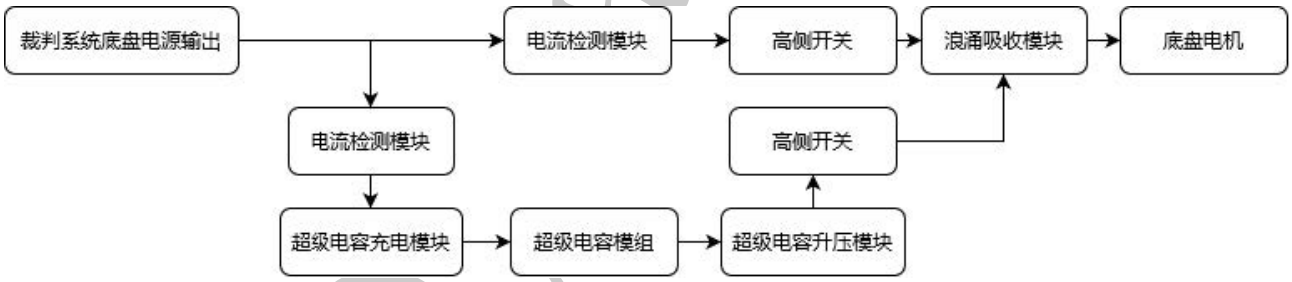


图 3.35: 超级电容拓扑图

为了实现超功率模式，我们采用了通用的拓扑结构，如 **图** 3.35 所示。这种拓扑结构的优点是能够容易地实现电容与裁判系统的功率合成以及对拓扑切换时延有一定的容错性。这种结构的主要的功率路径有两个，分别是：

1. 经过电流检测模块、高侧开关、浪涌吸收模块后直接提供给底盘电机。
2. 经过电流检测模块、超级电容充电模块给超级电容模组充电；超级电容经升压模块、高侧开关、浪涌吸收模块后提供给底盘电机。

在装机测试中，在比赛规则限定电池输出功率不超过 50W 的条件下，加入了超级电容管理模块之后，可以在保持电池输出功率在 50W 时，由电源管理测试模组提供 120W 的动力输出长达 17s 的时间。

3.2.2.1 控制器模块

控制器模块是超级电容管理模块的计算和控制单元，用于控制其它子模块的工作状态。我们控制器采用意法半导体公司的 STM32H750VBT6 芯片，该芯片采用 Cortex-M7 内核，具有六级双发射超标量流水线及分支预

测器、单精度及双精度浮点计算单元、紧密耦合内存 TCM、高速缓存 L1 cache、DSP 及 MPU 单元。我们使其工作在 480MHz 的主频下，可以达到约 1027DMIPS 的计算能力。

由于导电滑环的线数不足以再扩展出独立的通信通道，所以微控制器桥接在裁判系统与整车控制器 rm-controls 中间，通过 UART 与 rm-controls 进行通信，桥接后的拓扑结构如 **图** 3.36 所示。通信数据包附加在裁判系统数据包后面发出，数据包格式参考中容量数传协议。

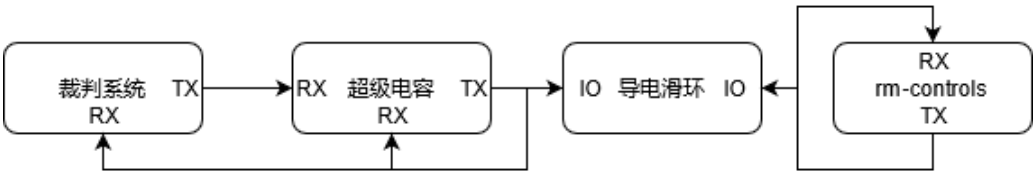


图 3.36: 桥接后的拓扑结构

为了实现充电、放电的精准功率监测，我们使用 STM32 自带的高速 16 位 ADC 配合 DMA1 外设实现对 2 路电流采样窗口、1 路电压采样窗口进行自动循环采样和数据转移。期间还需要配置 MPU 保护区域、cache 禁止缓存区域、ld 链接文件等。最终内存空间分配如下：

0x00000000	0x8000000	0x20000000	0x24000000	0x2407F000	0x30000000	0x38000000
ITCMRAM	FLASH	DTCMRAM	RAM_D1	ADCRAM	RAM_D2	RAM_D3
64KB	128KB	128KB	508KB	4KB	288KB	64KB

图 3.37: STM32 内存空间分配

由于超级电容管理模块是一个模拟、数字、功率器件混合的高系统复杂度的电路系统，并且有诸多的如有限状态机、过流过压欠压保护、裁判系统数据读取转发、功率补偿校正等任务，所以我们使用了开源的 FreeRTOS 实时操作系统完成系统的调度。为了实现更高的系统实时性，任务调度器被配置到 10kHz 的任务切换频率且使用抢占式调度器，并且配置了任务优先级为 1 的窗口看门狗防止系统卡死在某个状态下。

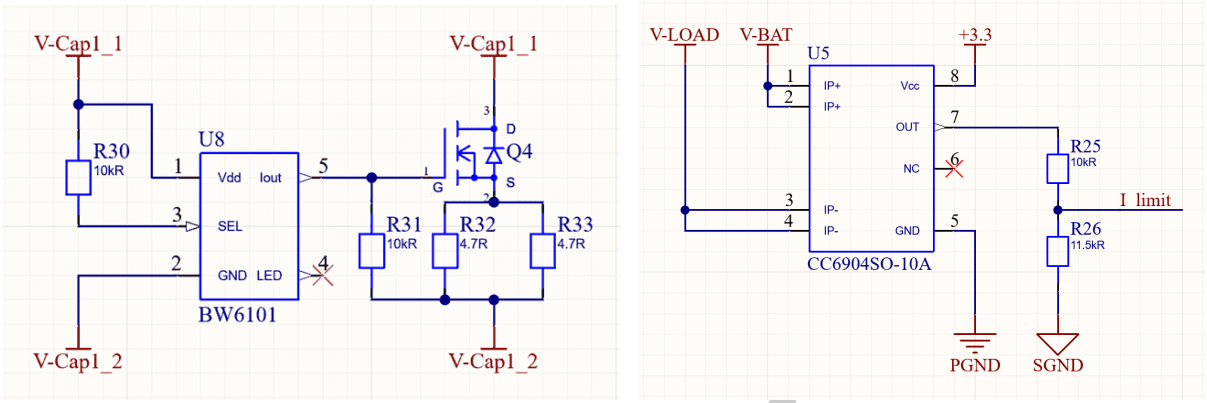
窗口看门狗很好地预防了系统的卡死，但是在调试的时候看门狗直接复位系统的方法使得卡死时的调试信息无法保存下来，而移植 FATFS 时 FLASH 空间不够，所以我们写了一个事件记录器和文件系统。每个事件发生时（如系统掉电、进入超功率模式）就会将事件标志和系统状态记录到文件系统中，后续可以通过回溯信息查找崩溃日志。

在上车调试控制器时，由于车体在运动，无法使用电脑一直连接控制器查看当前的关键系统变量和状态，所以我们写了一个轻量级的 GUI 界面（Flash 占用仅 8KB），用于绘制电压及功率曲线、任务优先级及堆栈最高水位线、崩溃日志回溯等功能。

3.2.2.2 超级电容模组

电容模组使用 2.7V/30F 电容单体，组合成三并六串的模组，模组标称参数为 16.2V/15F。如 **图** 3.38(a)，由于超级电容单体保护使用阈值为 2.65V 的比较器，在单体出现过压时比较器输出高电平控制 N-MOS 管开启，所以加入了单体保护的超级电容模组可得到新的标称参数 15.9V/15F，标称容量为 1896.075J。泄放回路为串联式，当单体触发泄放时，总是向下一个单体输出，直到该串中最后的一个电容向 GND 泄放。这种设计的优点是在单体充电平衡时，并不直接浪费所有能量，可以提高效率；缺点是最后一个泄放 MOS 管总优先承受最大的泄放电流，直到该串所有的电容电压相等，在使用过程中将使得热量集中于模组的一端，有需要情况下在布局时交叉摆放以减少这种影响。

泄放电阻采用两颗 2512 封装 4.7Ω 的电阻堆叠而成，单颗功率耗散为 2W，所以受到泄放电阻的影响，得到该模组总的连续泄放能力为 53.82W，需要加强散热以保证模块的持续稳定工作。



(a) 超级电容单体保护电路原理图　　(b) 充电电流采样电路原理图

图 3.38: 部分原理图展示

3.2.2.3 充电模块

在充电时，充电模块将负责限制电容的充电功率，且该充电模块可受控于微控制器，连续地控制充电模块的输入电流。本设计中需要控制的是充电电流，在超级电容不同电压下，充电模块恒定输出电流与输入电流差距很大，因此需要微控制器的参与来通过调整输出电流以实现对输入电流（即输入功率）的限制。

充电模块使用了专用的超级电容充电控制器 BQ24640，该控制器是 TI 为超级电容模组充电所设计的一款专用芯片，基本拓扑是同步型 Buck，集成了电流差分采样放大器，通过在电源的输出对一个模拟输入端口设置电压并在控制器内部进行比较，从而达到电流限制的目的。由于集成的差分放大器差分输入量程为 0 ~ 100mV，在差分输入只有 5mV 时，电流控制的误差将达到 ±25% 以上。为保证电流控制的精度，这里选用 10$m\Omega$ 的采样电阻。根据数据手册公式(3.8)，可求出输出电流为 $I_{charge} = 10A$。

$$I_{charge} = \frac{V_{iset}}{KV_{iset-max} \times 100mV} \tag{3.8}$$

BQ24640 电压基准 V_{iset} 的上拉电阻被微控制器的 DAC 输出上拉，在电压基准未触发时，呈现高阻状态，ISET 节点电压与 DAC 输出电压相等，此时充电模块输出电流受控于微控制器。充电模块的原理图如 **图** 3.39 所示。

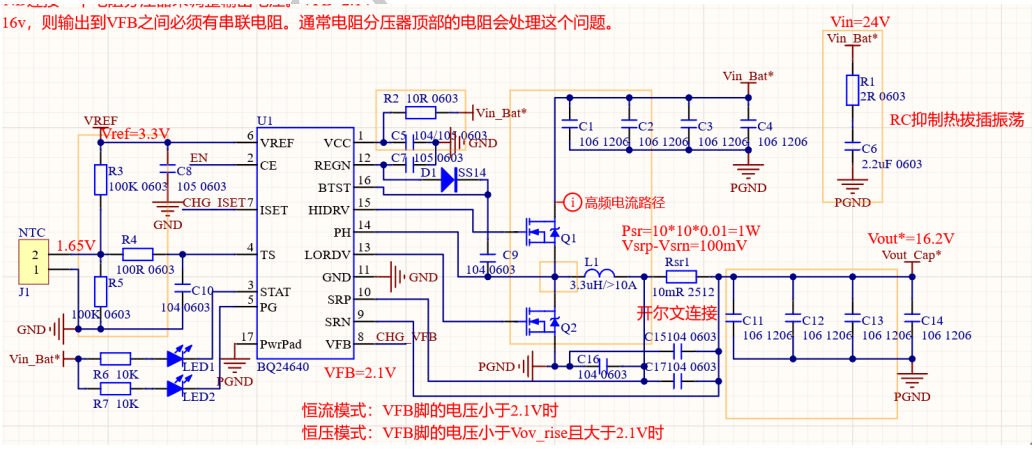
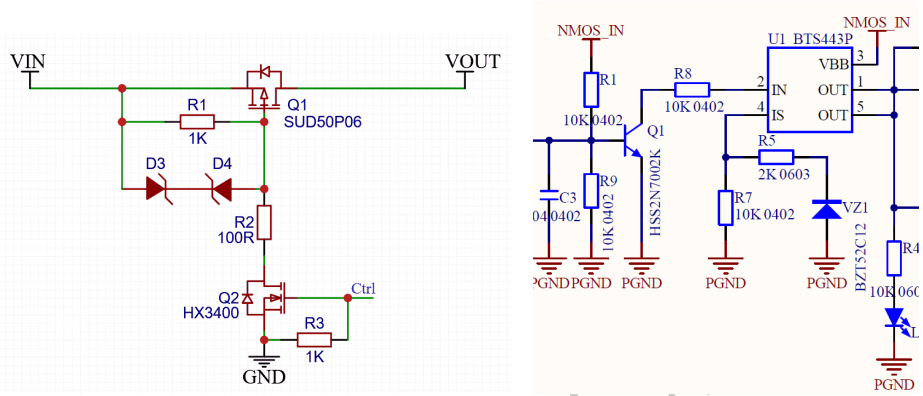


图 3.39: 充电模块原理图

3.2.2.4 高侧开关模块

高侧开关用于控制上述模块之间的连通性。我们采用了英飞凌的智能高侧驱动器 BTS443P 如 **图** 3.40(b) 。该智能高侧开关内部封装了一颗导通电阻为 16$m\Omega$ 的 NMOS 以及用于升压的电荷泵。该智能高侧开关还具有门极保护、过压保护、过流保护、过热保护、短路保护等功能，还提供了电流监视引脚（我们未使用此引脚）。



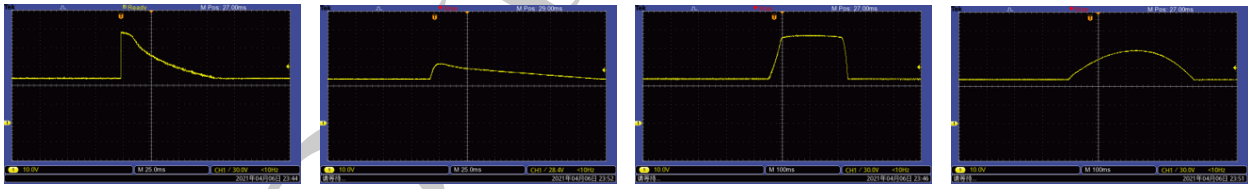
(a) 使用 PMOS 作为高侧开关　　(b) 使用 BTS443P 作为高侧开关

图 3.40: 高侧开关原理图

另外，我们还通过三极管实现反相器，将所有的模块（包括高侧开关）使能电平统一设定为低电平。这在微控制器故障或与模块使能引脚连接不稳定使引脚浮空浮空的时候可以避免模块被误开启。

3.2.2.5 浪涌吸收模块

在刚开始的时候，由于我们选择的高侧开关是一颗耐压为 40V 的 PMOS 管。在底盘电机刹车或正反转切换的瞬间高侧开关就会烧毁。所以我们通过示波器测量了单个 3508 电机在刹车和正反转切换时的电压跳动情况如 **图** 3.41(a) 和 **图** 3.41(c) 所示，最高钳位电压达到了 50V，足以过压击穿 PMOS。所以我们先设计了一个利用肖特基二极管和电解电容器作为浪涌吸收的模块如 **图** 3.42(a)。



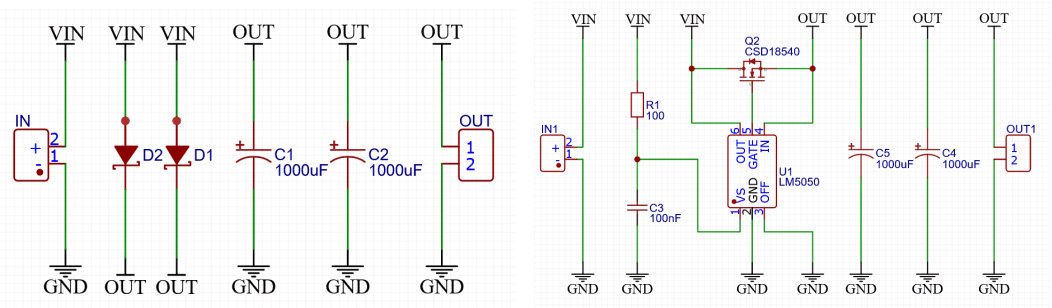
(a) 无浪涌吸收刹车　(b) 有浪涌吸收刹车　(c) 有浪涌吸收正反切　(d) 无浪涌吸收正反切

图 3.41: 有浪涌吸收模块对比

在增加了浪涌保护模块后，在浪涌保护模块的输出侧用万用表进行波形的测量。可以看到，有浪涌保护模块的电机刹车电压波形变成了 **图** 3.41(b)，正反转切换变成了 **图** 3.41(d)。从图中可以看出，最高电压大概在 40V，且在正反切的图中波形变成了圆滑的圆弧状，说明在切换的过程中电容有能量的储存，在正反转切换的瞬间不会继续从电源获取能量，实现了浪涌吸收和能量回收的效果。

在测试时，发现当负载功率达到 240W 时，我们采用的肖特基二极管 SVM1550UB 的发热严重，需要加装散热片来保证芯片不会烧毁。于是分析负载功率达到 240W 时肖特基二极管上的功率耗散，有公式(3.9)求得 $P_{diode} = 4.9W$，可见损耗率达到了 2.1%。所以我们采用了 TI 公司的 LM5050 理想二极管控制器，配合 CSD18540 MOS 管替换掉肖特基二极管。可求得此时的 $P_{dis} = 0.22W$，损耗可以忽略不计。

$$P_{diode} = \frac{P_{out}}{24V} \times U_{diode} = \frac{P_{out}^{\ 2}}{24V} \times R_{DS} \tag{3.9}$$



(a) 早期版本电路　　(b) 现在版本电路

图 3.42: 早期和现在的浪涌吸收模块

3.2.2.6 升压模块

在超级电容中，升压模块 1.0 版本的设计最大功率为电源管理额定功率的两倍，即 240W。由于升压模块有最低的输入电压限制，以及在输入输出电压压差较大时会导致输出端产生很大的纹波，且电源转换效率会变得很低，所以我们将升压模块的输入电压范围设置为 7 ~ 16V。在 1.0 版本中采用两相供电技术，其中一相在原有的纹波波谷处补充输出，首先直接补偿了原有的电压跌落，且在同样的频率下，由于两相半桥开关状态相互错开，输出纹波的频率即为开关频率的两倍，更高频的纹波简化了输出滤波的设计，单个电感中的开关电流也会减小，提高功率密度。

在此我们采用了 ADI 的多相同步升压控制器 LTC3784，单颗控制器可提供两相互补的 PWM 输出，分别驱动两对半桥，可在较小的输出电容配置时，高输出电流下仍能具有高效率和低输出纹波。为解决多相供电技术的瞬态性能比同步多路并联稍差的问题，LTC3784 提供了电流模式的误差放大器反馈环路补偿功能，通过匹配反馈网络的前馈电容和补偿节点的 RC 网络的器件参数，可以使多相供电的瞬态性能得到十分大的改善。

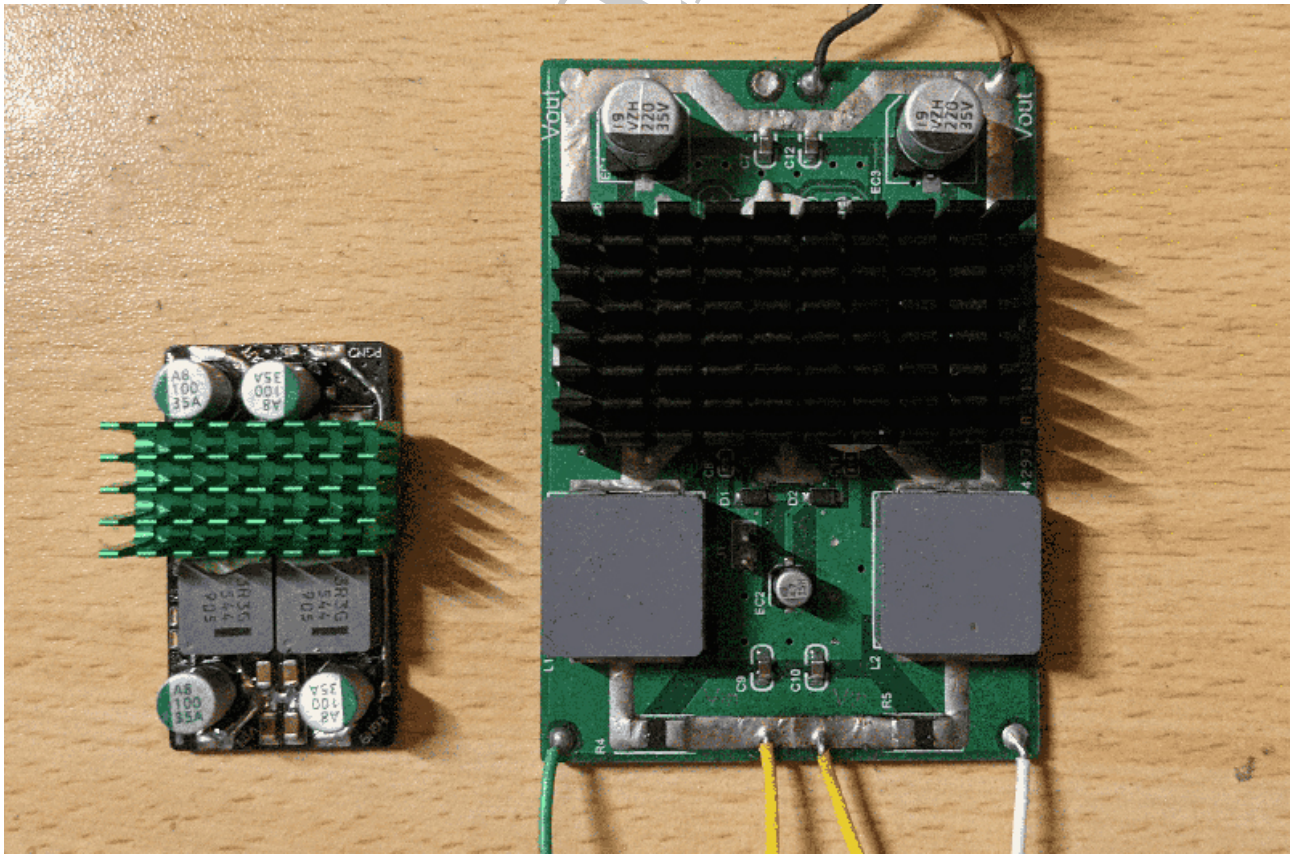


图 3.43: 升压 1.0 版本大小对比

升压模块 2.0 版本也采用 ADI 的多相同步升压控制器，但是型号上我们选取了 LTC3897 的双片四相方案，实现 480W（24V20A) 的超大功率。四相的方案选择让元器件选型、散热变得更加容易，纹波更低的同时也提高了效率。LTC3897 与 LTC3784 相比, 其在宽输入范围的同时，内置了一个理想二极管控制器和浪涌停止器，三个

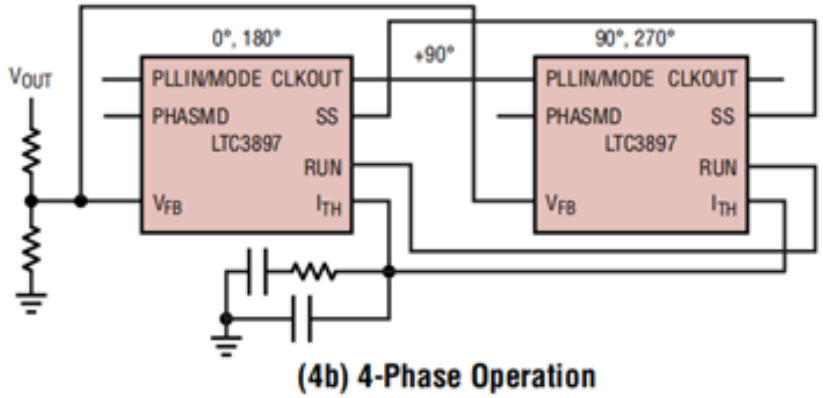


图 3.44: 四相拓扑连接图

控制器可以单独使用，这让我们得以节省一个输出端的高侧开关模块。单颗 LTC3897 实现两相 PWM 互补输出，两个并联通过芯片内部的锁相环实现频率同步，通过 PHASMD 引脚悬空实现相为 90ř 互补，让双片四相的方案得以实现。

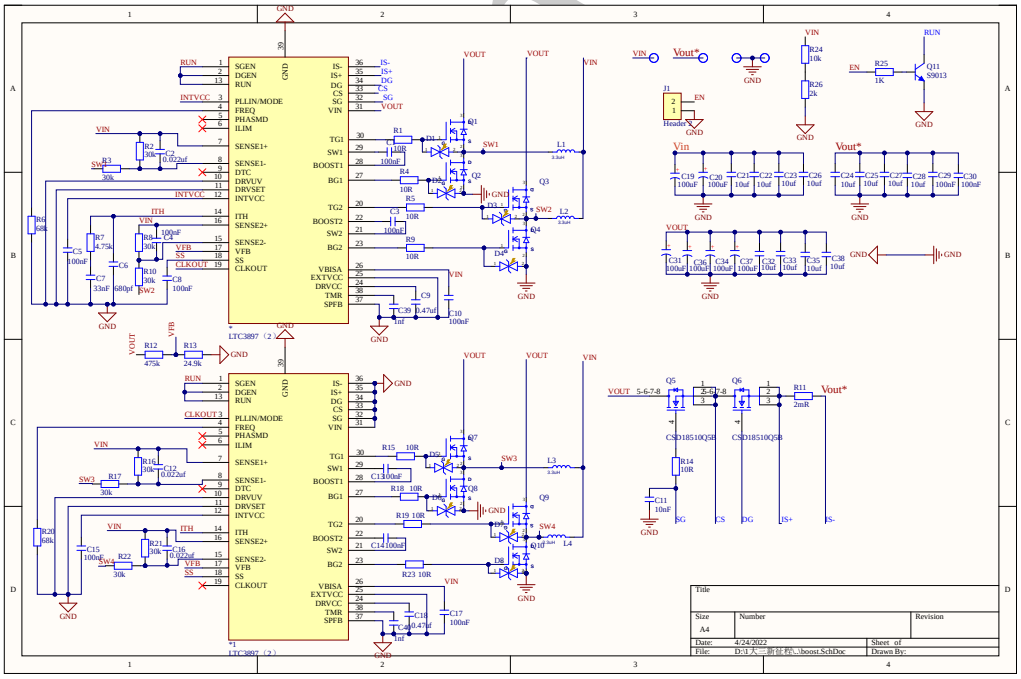


图 3.45: 升压 2.0 原理图

3.2.2.7 升压模块功率器件选型

电感选型：考虑到板面体积不能太大，这里选用了封装为 1054 的松下的 ETQP5M2R5YFC 的电感，标称的感量为 2.5uH，在加强散热时额定电流为 18.1A，饱和电流 27.2A，以下将在此基础上进行电压范围和工作频率的确定。根据电感的纹波电流容易列出不等式(3.10)。

$$\frac{I_{out(max)} \times V_{out}}{2 \times V_{in}} + \frac{V_{in}}{2 \times f_{SW} \times L} \times (1 - \frac{V_{in}}{V_{out}}) < 18.1A \tag{3.10}$$

降低频率可以降低 MOS 管开关损耗，却会增大电感的纹波电流，为提高效率和留有足够的裕量，设定输入电压为 7V，工作频率为 644kHz，此时最大的 $\Delta I_L \approx 0.6 I_{max}$，处于较良好的范围内。

MOS 管选型：设控制管，即半桥的下管损耗功率为 P_{ctrl}；同步管，即半桥的上管损耗功率为 P_{sync}，则有公式(3.11)和(3.12)：

$$P_{ctrl} = \frac{(V_{out} - V_{in})V_{out}}{V_{in}^2}(\frac{I_{out(max)}}{2})^2(1+\delta)R_{DS(ON)} + k \cdot V_{out}^3 \frac{I_{out(max)}}{2 \times V_{in}} \cdot C_{miller} \cdot f \tag{3.11}$$

$$P_{sync} = \frac{V_{in}}{V_{out}} \times (\frac{I_{out(max)}}{2})^2 \times (1+\delta) \times R_{DS(ON)} \tag{3.12}$$

将两式中的 $(\frac{I_{out(max)}}{2})^2(1+\sigma)$ 约去，又 $\frac{(V_{out}-V_{in})V_{out}}{V_{in}^2}$ 可写成 $(\frac{V_{out}}{V_{in}})^2 - \frac{V_{out}}{V_{in}}$，因为最大输入电压为 16V，输出电压为 26.4V，可得 $(\frac{V_{out}}{V_{in}})^2 - \frac{V_{out}}{V_{in}} \geq 1.016$。又 $\frac{V_{out}}{V_{in}} < 1$。根据手册，k 的经验值为 1.7，易得在本设计中 $P_{ctrl} > P_{sync}$，故在控制管选型时，需要注意结温的温升系数，选取导通电阻和米勒电容更小的 MOS 管。

最终升压模块选用 HSBB4052 作为同步管，由于该款 MOS 管的封装使用的是 3mm*3mm 的无引线扁平封装，在低电压时选用该 MOS 管作为控制管温升将超过了该款 MOS 管 150°C 的最大容许工作温度极限，所以选用了 NTMFS5C670NL 作为控制管。这两种型号都具有极低的栅极充电电荷和米勒电容，极大地减少了高工作频率下 MOS 管开关动作引起的损耗。

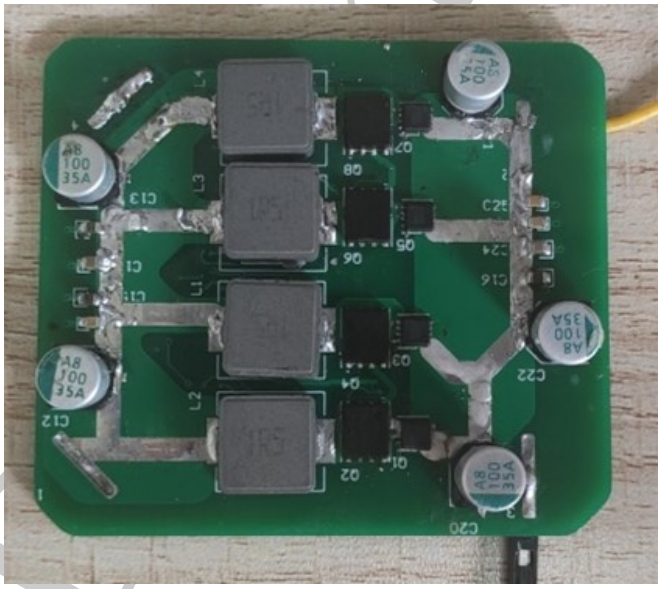


图 3.46: 升压 2.0 测试版

3.2.2.8 环路补偿电路计算

环路补偿：首先要确定前馈电容和衰减电容的大小，前馈电容 C_{ff} 是一个与反馈网络中 R_{top} 并联的电容，滤波电容 C_{flt} 则是与 R_{bot} 并联的，未添加前馈电容时，反馈网络的增益由分压电阻大小决定，其计算公式如(3.13)。在反馈网络中添加前馈电容后，电源的输出可以更高效地响应高频扰动，但前馈电容可能会引入不想要的开关噪声，因此可添加一个衰减电容用于减小引入前馈电容后的增益，避免高频段增益接近 0dB 时系统不稳定。前馈电容一般在 1000pF 以下，由于电容的阻抗特性，反馈网络的低频区仍与不加前馈电容时一致；而在中高频区域，前馈电容为输出的扰动到反馈端提供了低阻抗通路，促使电源控制器对负载顺便产生更快的响应。添加前

馈电容后在反馈网络的零点频率后引入了增益增量，当处于零点和极点频率之间时，该增量的达到最大值。反馈网络的零点 f_z 和极点 f_p 可以由公式(3.14)和(3.15)计算得出。

$$DC\ gain = 20 \cdot lg(\frac{R_{top}}{R_{top} + R_{bot}}) \tag{3.13}$$

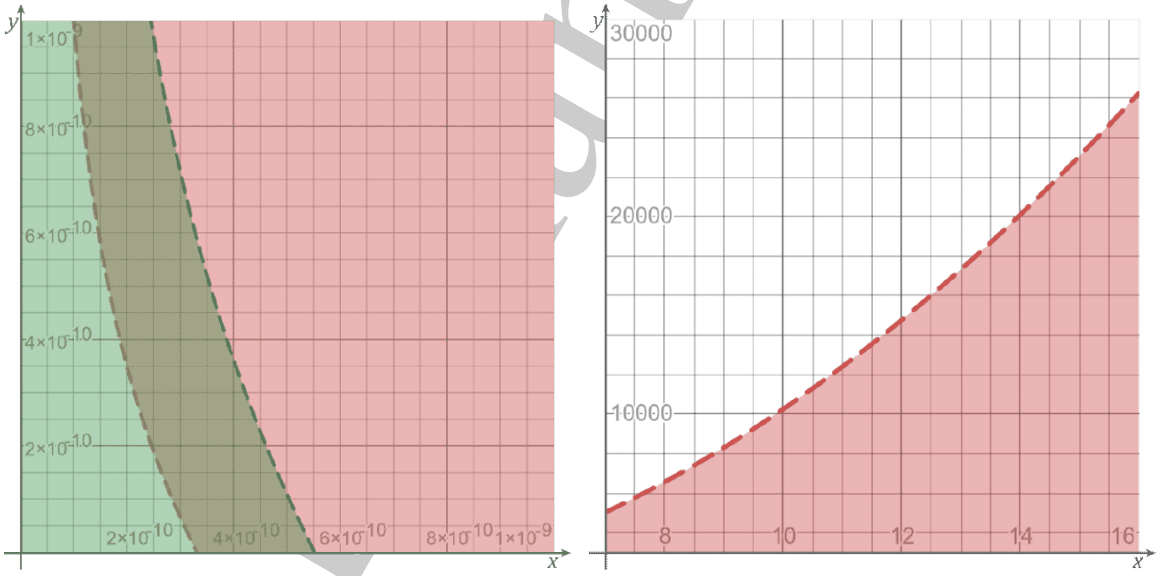
$$f_z = \frac{1}{2\pi \cdot R_{top} \cdot C_{ff}} \tag{3.14}$$

$$f_p = \frac{1}{2\pi \cdot (C_{ff} + C_{flt})} \cdot (\frac{1}{R_{top}} + \frac{1}{R_{bot}}) \tag{3.15}$$

由公式可看出零、极点的频率与前馈电容的大小成反比。要获得充足的相位裕度并充分衰减噪声，反馈网络的穿越频率应选为工作频率的 1/10 至 1/6，可得在本设计中该频率范围为 $64.4kHz \sim 107.3kHz$。当使零、极点的频率的几何平均值为反馈网络的穿越频率时，可得到最优化的响应，即可列出公式 $f_{cross} = \sqrt{f_z \times f_p}$。代入(3.14)和(3.15)有：

$$f_{cross} = \frac{1}{2\pi} \cdot \sqrt{\frac{1}{C_{ff} \cdot (C_{ff} + C_{flt})} \cdot \frac{1}{R_{top}} \cdot (\frac{1}{R_{top}} + \frac{1}{R_{bot}})}$$

可画出如 **图** 3.47(a) 曲线，横轴为 C_{ff}，纵轴为 C_{flt}，处于两范围交集内的电容值都是可用的，由于零点频率只与 C_{ff} 相关，更大的 C_{ff} 使得零点变得过低，使在低频区域的增益变得过大，系统对噪声过于敏感，所以首先选取靠近左上角的标准电容值，即取 $C_{ff} = 180pF$，$C_{flt} = 910pF$。



(a) C_{ff} 与 C_{flt} 选择范围　　(b) 环路带宽限制

图 3.47: Boost 环路补偿计算 1

由于 Boost 拓扑在连续导通模式下，环路中会出现右半平面零点（RHPZ），使瞬态响应特征变差，这是由于在负载电流变大时，占空比也会变大，在短时间内电感向负载供电的时间变短，电压反而下降得更多，需要几个周期后才能恢复。根据经验，一般的设计要求环路的带宽（或称环路的穿越频率，为避免混淆，以下均称为环路带宽）低于 1/3 的右半平面零点频率，右半平面零点频率可由公式(3.16)算出。R_{out} 为负载电阻，L 为电源选用的功率电感，当电源的输入电压和负载电阻越小，此时右半平面零点频率越低，负载电阻由输出电流直接决定，环路带宽的公式为(3.17)。

$$f_{rhpz} = \frac{R_{out}}{2\pi \cdot L} \cdot (\frac{V_{in}}{V_{out}})^2 \tag{3.16}$$

$$f_{band} < \frac{V_{out}}{6\pi \cdot L \cdot I_{out}} \cdot (\frac{V_{in}}{V_{out}})^2 \tag{3.17}$$

当电流很小时，带宽即使很高系统也能稳定，所以仅需考虑最大电流时的情况，如 **图** 3.47(b) 为最大电流时环路带宽的限制（横轴 x 为输入电压 V，纵轴 y 为带宽频率 kHz）。

确定环路的带宽限制后，现在开始进行 II 型补偿网络设计，如 **图** 3.48，由于电流型控制器的跨导型误差放大器输出并非理想电流源，具有一个大内阻 R_0，由 LTC3784 的数据手册读取到 $R_0 = 2M\Omega$，即可计算出环路的第一个极点，其余的参数由补偿网络的器件数值确定，最后应使得环路响应符合 **图** 3.49 的要求。为简化计算，这些步骤在 LTpowerCAD 中完成。

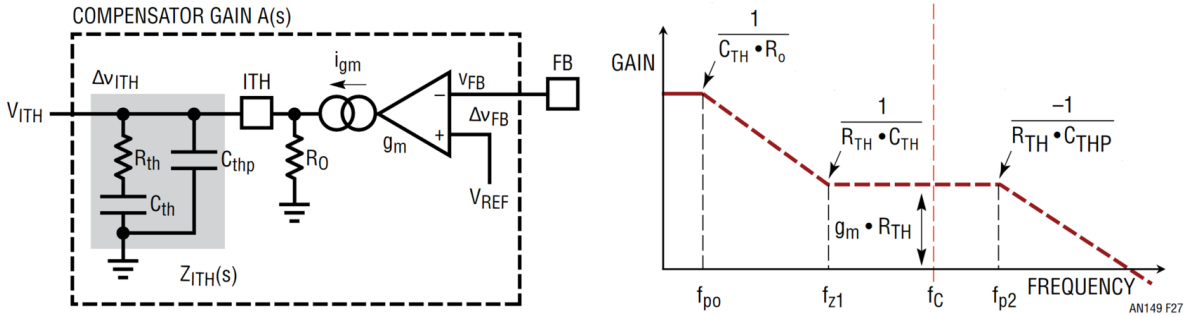


图 3.48: 环路补偿网络框图与补偿增益曲线

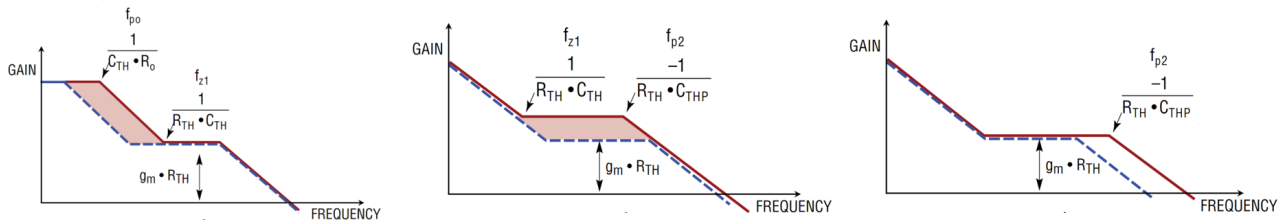


图 3.49: II 型补偿网络器件参数与补偿增益曲线关系

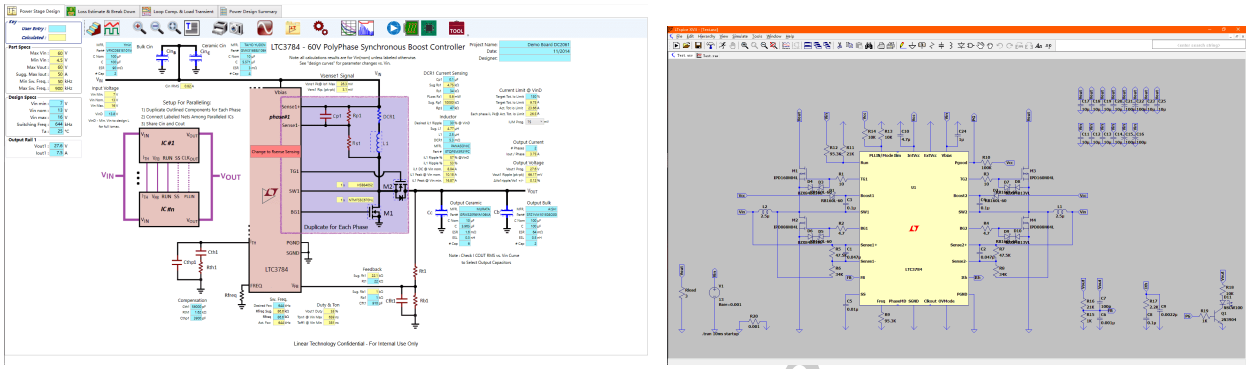
使用可视化的方法可以快速地确定参数，如 **图** 3.49，调整 C_{TH} 将影响第一极点和零点的位置，更小的 C_{TH} 值将使得增益曲线的第一个下降的斜边向由平移，使得高增益区间向高频拓展，使电源在负载阶跃中更快的达到稳态，但零点频率提高会导致目标带宽处的相位裕度不足；而更大的 R_{TH} 使得增益曲线的中频平台向上平移，且直接增大电源的带宽，使得在负载阶跃时过充和下冲减小，但过高的带宽会令相位裕度变差；更小的 C_{THP} 使第二极点的位置向高频移动，C_{THP} 作为环路补偿引脚的去耦电容，可降低补偿节点的噪声，但过大的 C_{THP} 使得带宽变小，使电源的瞬态响应变慢，应配置 C_{THP} 使第二极点频率低于工作频率且高于电源的带宽，以较好消除开关噪声的同时保证有足够的相位裕度。

改变输入电压和电流，确保系统的参数能使电源的环路是稳定的同时，在右侧估算的电流阶跃响应有足够好的效果。此处只需关注环路的稳定性即可，对于纹波的仿真将会在下一段得到一个较为理想的结果。

3.2.2.9 计算机辅助设计验证仿真

本模块的设计将使用 ADI 提供的 LTpowerCAD 以及 LTspice 等工具进行辅助设计和仿真。如 **图** 3.50(a) 所示，首先使用 LTpowerCAD 辅助确定包括环路补偿在内的参数。确定参数后，使用 LTspice 仿真电源的运行效果。如 **图** 3.50(b) 所示，搭建原理图，设置输入电压为 13.8V，仿真电感纹波电流最大的状况，以确定最大的输

出纹波电压。仿真得纹波电压为 0.031V，纹波系数仅有 1.12‰。至此，升压模块设计告一段落。

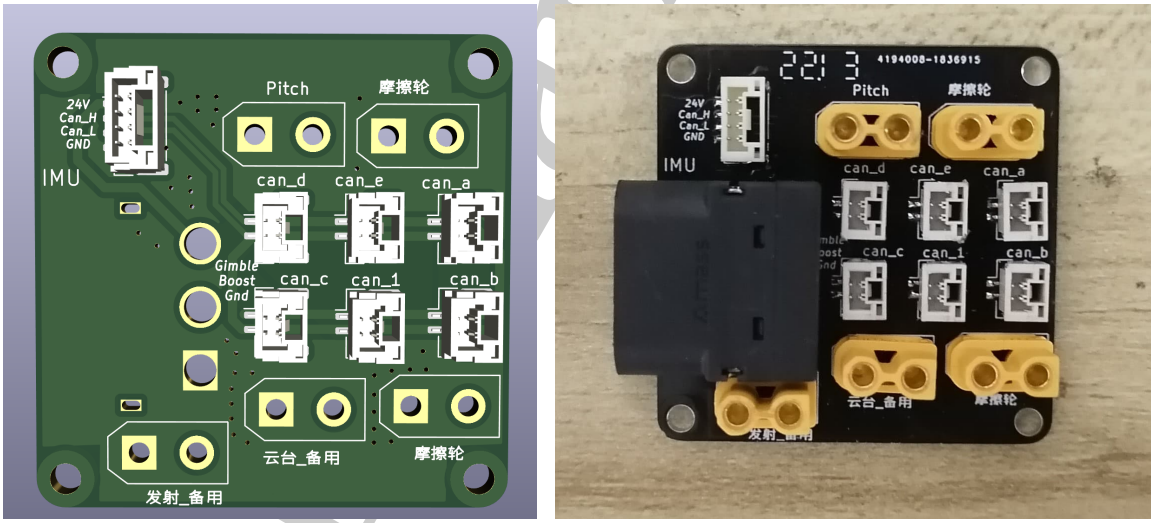


(a) LTpowerCAD 功率级设计页面　　(b) 在 LTspice 中搭建仿真原理图

图 3.50: 部分设计界面展示

3.2.3 发射中心板

在机器人设计中，IMU 模块被放置在了发射机构上，但是 IMU 模块的供电应隶属于“Gimble（云台）”而非“Ammo-Booster（发射）”，为了方便机器人线路的排布，我们设计了一款自制的中心板，输入供电端子为“MR60”端子，三端分别为 Gimble、Ammo-Booster 和 GND。自制发射中心板板载了 6 路 can 接口（GH1.25）、3 路发射供电接口（XT30）、2 路云台供电接口（XT30）以及 IMU 专属接口（GH1.25——4Pin），接口全部采用立插，极大的方便了机器人的安装与检修。



(a) 中心板 PCB 3D 图　　(b) 中心板实物图

图 3.51: 自制中心板展示

3.2.4 NUC 降压模块

由于我们步兵机器人使用的主控是 Intel NUC（以下简称为 NUC），所以需要单独为 NUC 制作一个供电电压为 19V，供电电流为 6.35A 的降压电源。主芯片我们选取 LM3150 为 Buck 控制器，LM3150 最高输入电压可达 42V，输出电流可达 12A，满足 NUC 供电需求。另外，LM3150 还工作在 COT（恒定导通时间）模式，具有优异的瞬态响应性能和低压差输出的能力。

最终设计原理图如**图** 3.52所示。实际测量中，在负载电流达到 6A 时，NUC 供电电源的效率可以达到 97.8%，模块整体平均温度被控制在 50C 以下，有非常优越的性能表现。

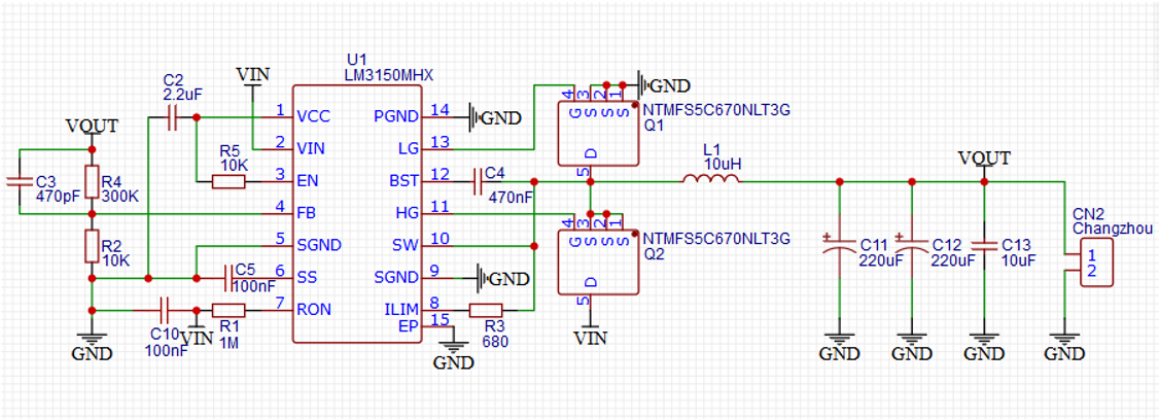


图 3.52: LM3150 原理图

3.2.5 协议转换模块

为满足不同通信协议的通信，我们设计了多种不同的协议转换模块。在步兵机器人上，我们设计了 USB 转 CAN 模块和 USB 转串口模块，模块主要用于 NUC 和其它硬件模块之间的通信和控制电机。

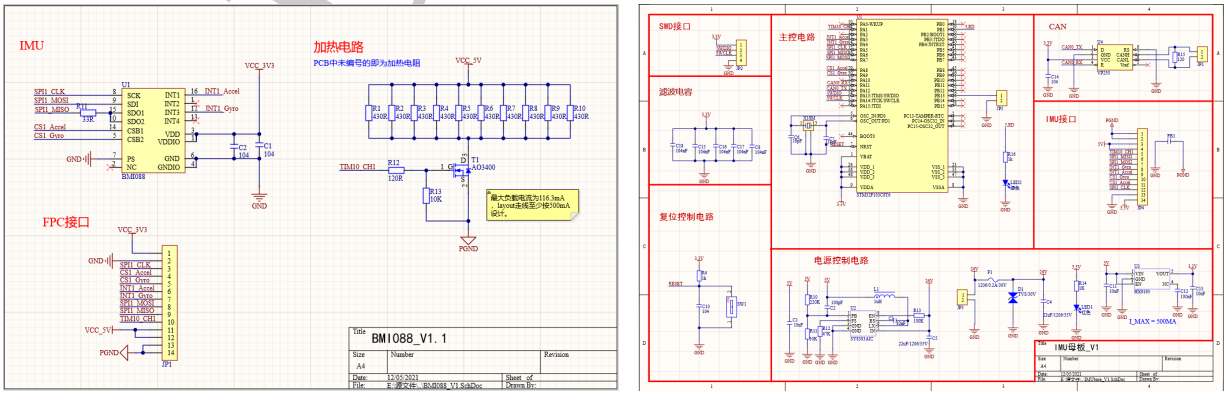
- USB 转 CAN 部分我们基于 github 的开源方案 candleLight 开发，采用 STM32F072CBT6 作为主控芯片，实现 USB 转 CAN 的功能。STM32F072CBT6 具有能够同时工作的 USB 全速外设和 CAN 外设，封装为 QFP64，是较为优秀的解决方案。
- USB 转串口部分主芯片选取 CH340E，该芯片是常用的一款 USB 总线的转接串口芯片。

由于整车需要多路 CAN 总线来保证电机回传数据包的完整性，我们选取 GL850G 作为 USB HUB 芯片实现 USB 一拖四的方案，CAN 电平转换芯片采用 MAX3051EKA 芯片，该芯片使用 3.3V 供电且封装为 SOT23-8，为 PCB 小型化提供了基础。同时我们还通过使用施密特触发器来实现上电时序，保证 USB 设备枚举的顺序在每次上电后都是一致的。

3.2.6 惯性测量单元

为提高机器人姿态获取的精密性、运动控制的准确性，我们使用了六轴惯性测量单元 BMI088。该六轴惯性测量单元具有卓越的抗震性能，能保证在冲击下的工作稳定性。为改善惯性测量单元的温飘问题，我们增加了加热电路，通过 PID 实现对陀螺仪的恒温处理。

相对应的，我们设计了专用于 BMI088 的控制板。该控制板使用 STM32F103C8T6 作为主控，通过 SPI 协议与 BMI088 通讯，并且集成了一路 can 信号转化电路，能够较便捷地对外通信。BMI088 及其控制板的设计原理图如 **图** 3.53(a)与 **图** 3.53(b)所示。



(a) BMI088 原理图　　(b) IMUbase 原理图

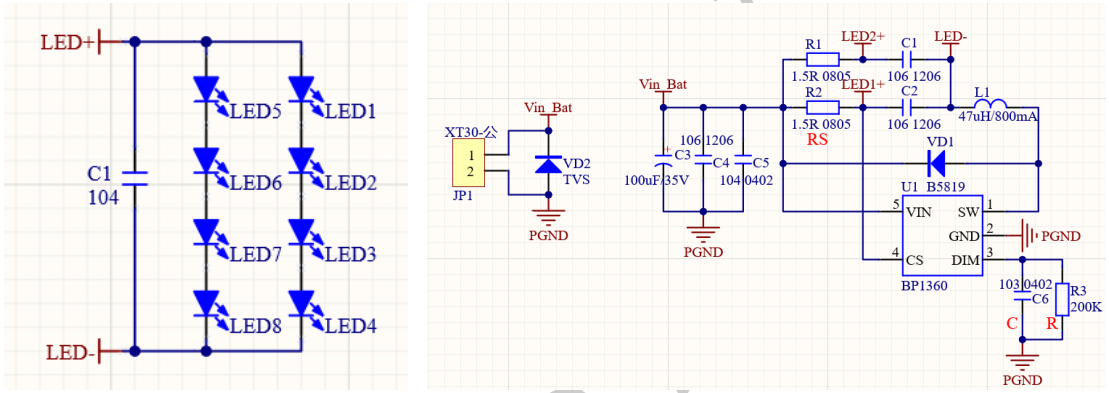
图 3.53: IMU 模块

3.2.7 荧光充能装置

比赛中需要使用荧光弹丸，弹丸在经过特定波长紫外光照射充能后，能够发出黄绿色光并显示弹丸飞行轨迹。因此需要设计一款充能装置，产生特定紫外光以进行荧光充能。我们将荧光充能装置分为两个部分：

- 紫外光 LED 灯板。采用封装为 2835 的 390-410nm 的紫外灯珠，灯珠发光角度 120 度，单颗功率 0.2W。我们的设计中采用四串两并的 LED 灯珠，并使用铝基板以保证足够的散热。
- LED 恒流驱动板。我们使用 BP1360 作为恒流驱动，在 24V 工作电压下选择 68mA 的驱动电流，以此保证紫外光灯板总体功率大于 1.5W。该芯片采用极少的外部元器件，能够快速通过采样电阻设置恒流电流的大小。

紫外光 LED 灯板和恒流驱动板的设计原理图如 **图** 3.54(a)与 **图** 3.54(b)所示。



(a) 紫外光 LED 灯板原理图　　(b) LED 恒流驱动板原理图

图 3.54: 荧光充能装置

3.3 软件设计

3.3.1 概述

`rm-controls`[2] 是一套在 PC 上运行的无下位机电控软件，基于 ros-control[3] 的硬件和仿真接口以及配套的控制器，用于开发 RoboMaster 机器人和高性能机器人。

3.3.1.1 极高的代码复用率

我们在设计代码架构时，极其注重代码复用性，将代码模块化并参数化，对于不同机器人只需要根据配置文件加载不同数量和种类的控制器。做到了一**套**代码给**所有**机器人同时使用，如：在开发好步兵的基础模块后，开发双云台哨兵时，只需要编写配置`.yaml`文件，除了决策层的开发，不需要书写或修改任何一行代码。**图** 3.55 部署了 rm-controls 的机器人，我们在 2021 年 RMUL 和 RMUC 均在全队的机器人中使用了 rm-controls。

代码 3.1 为哨兵的云台配置文件，可以看到上下两个云台的配置文件结构相同，两个云台控制器被独立地动态地加载，然后分别获取两个控制器的参数，包括它要控制的关节名称，还有对外接口 (以 rostopic 等形式) 的名称。

(a) #1 Hero (b) #2 Engineer (c) #3 Standard (Mecanum)

(d) #4 Standard (Swerve) (e) #5 Standard (Mecanum) (f) #6 Sentry

图 3.55: 部署了 rm-controls 的机器人

Listing 3.1: 哨兵上下云台部分配置

```
upper_gimbal_controller:
    type: rm_gimbal_controllers / Controller
    detection_topic: "/upper_detection"
    camera_topic: "/upper_camera/camera_info"
    yaw:
      joint: "upper_yaw_joint"
      pid: { p: 5.0, i: 0, d: 0.3, i_clamp_max: 0.0, i_clamp_min: -0.0, antiwindup: true}
    pitch:
      joint: "upper_pitch_joint"
      pid: { p: 8.0, i: 50, d: 0.3, i_clamp_max: 0.1, i_clamp_min: -0.1, antiwindup: true}
```

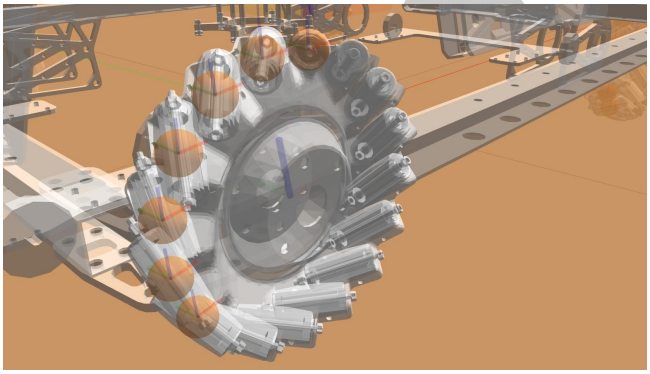
```
lower_gimbal_controller:
    type: rm_gimbal_controllers / Controller
    detection_topic: "/lower_detection"
    camera_topic: "/lower_camera/camera_info"
    yaw:
      joint: "lower_yaw_joint"
      pid: { p: 5.0, i: 0, d: 0.3, i_clamp_max: 0.0, i_clamp_min: -0.0, antiwindup: true}
    pitch:
      joint: "lower_pitch_joint"
      pid: { p: 5.0, i: 100, d: 0.2, i_clamp_max: 0.4, i_clamp_min: -0.4, antiwindup: true}
```

3.3.1.2 多刚体动力学仿真

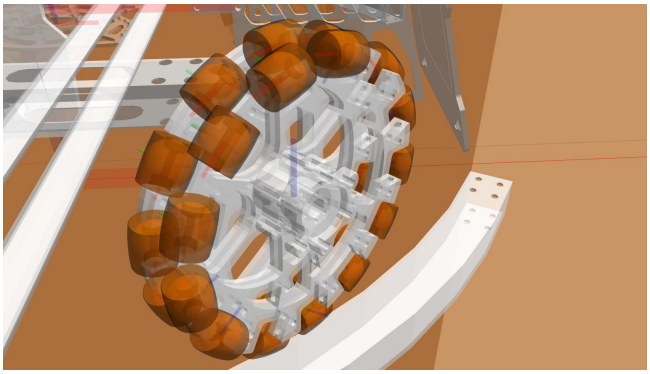
所有机器人都能在 Gazebo 进行多刚体动力学仿真，且仿真和实车运行的代码不需要相互移植，是同一份代码甚至同一个二进制文件。除了发射控制器，其他模块和控制器在开发时均会在仿真中调试和测试，打断点时可以做到时间暂停，在机械组还没有加工装配好车时已经可以把代码测试好，完成参数的粗调，大大提高开发和迭代效率。如：2022 赛季全向轮机器人在加工装配好后只花不到一天时间就测试调试完所有程序。**图** 3.56 为全向机器人在 RMUC 场地中的仿真。

图 3.56: 全向轮步兵机器人仿真

除此之外我们还有真实的棍子仿真，麦轮与全向轮的每一个棍子在 Gazebo 中都有自己的坐标系与碰撞箱。



(a) 麦轮棍子仿真



(b) 全向轮棍子仿真

图 3.57: 麦轮和全向轮棍子仿真

3.3.1.3 良好的兼容性

硬件接口基于 Linux 的 SocketCAN 和 sysfs，意味着可以在 Jetson AGX 和妙算等带有 CAN 总线的 ARM 设备上运行，同时我们还制作了 usb 转 CAN 设备，从而支持在任意 x86 平台上面运行，如：Intel NUC 和队员的笔记本电脑，如 **图** 3.58。在 RMUC 中我们以 Intel NUC 为主，也被官方在 2021 年高中生暑期营中部署在妙算 2 上使用。

(a) Jetson AGX Xariver (b) Jetson AGX NX (c) DJI Manifold 2 (d) Intel NUC

图 3.58: rm-controls 兼容的部分计算设备

3.3.1.4 丰富的调试工具

依托 ROS 生态，可以使用多种可视化调试工具，对电机数据，计算结果和图像远程查看。还可以使用 dynamic_reconfigure 进行动态调参。

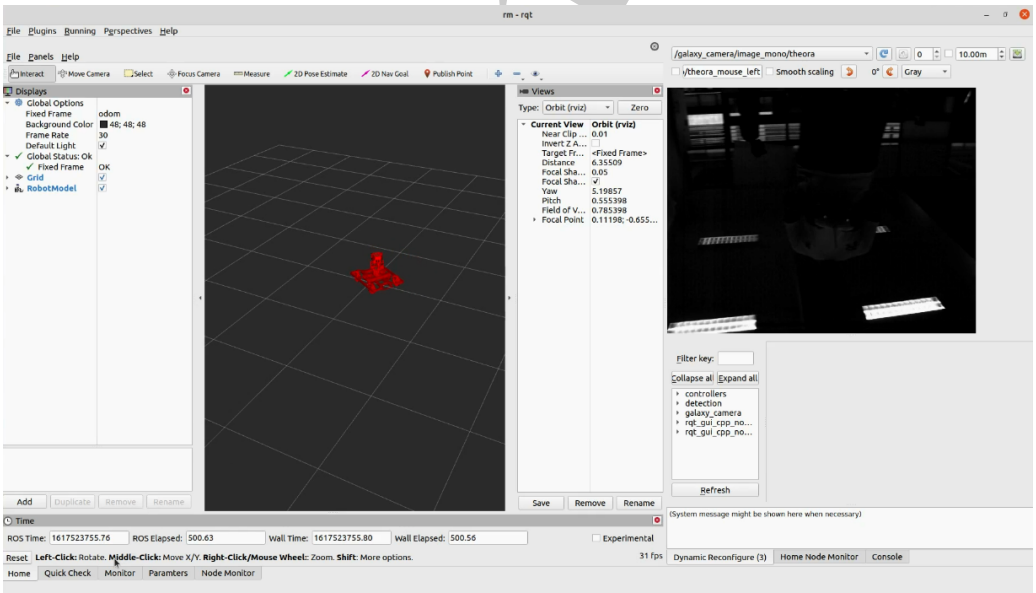


图 3.59: rqt 调试软件

图 3.59 展示了我们搭建的调试界面，由 rqt 的支持，我们不需要编写代码就可以构建自己的调试图形界面。

3.3.2 系统架构

3.3.2.1 系统层级与各层功能

程序由多个 ROS package 构成，其中有元包：`rm_control` 提供了底层硬件和仿真的通用接口，元包 `rm_controllers` 则为中间层各模块控制器，还有普通机器人操作包 `rm_manual` 和哨兵决策包 `rm_fsm`。

- rm_control

- rm_msgs: 自定义的 ROS 话题消息、服务、动作
- rm_common: 常用函数、算法、裁判系统收发、ui
- rm_description: 所有机器人的 URDF，定义了：机器人各坐标系关系、电机与关节的映射、关节的限位、仿真需要的物理属性
- rm_hw: 同名节点通过 SocketCAN 与执行器进行通信获取数据并发送指令，在实车运行时提供硬件接口给控制器
- rm_gazebo: 同名 Gazebo Plugin 在仿真运行时提供硬件接口给控制器

- rm_controllers
  - rm_chassis_contorllers: 麦克纳姆轮、舵轮、平衡车的底盘控制器
  - rm_gimbal_controllers: 内置枪管角解算模型的云台控制器
  - rm_shooter_controllers: 操作摩擦轮，拨弹盘完成发射控制器的控制器
  - rm_calibration_controllers: 校准执行器位置的控制器
  - robot_state_controller: robot_state_publisher[4] 的高性能版，高频维护 tf
- rm_dbus: dbus 数据接收节点
- rm_detection: 装甲板和能量机关视觉识别
- rm_stone: 矿石和障碍块视觉识别
- rm_track: 选择并预测目标的位置速度，并发送给云台控制器
- rm_manual: 普通机器人决策
- rm_fsm: 哨兵机器人决策

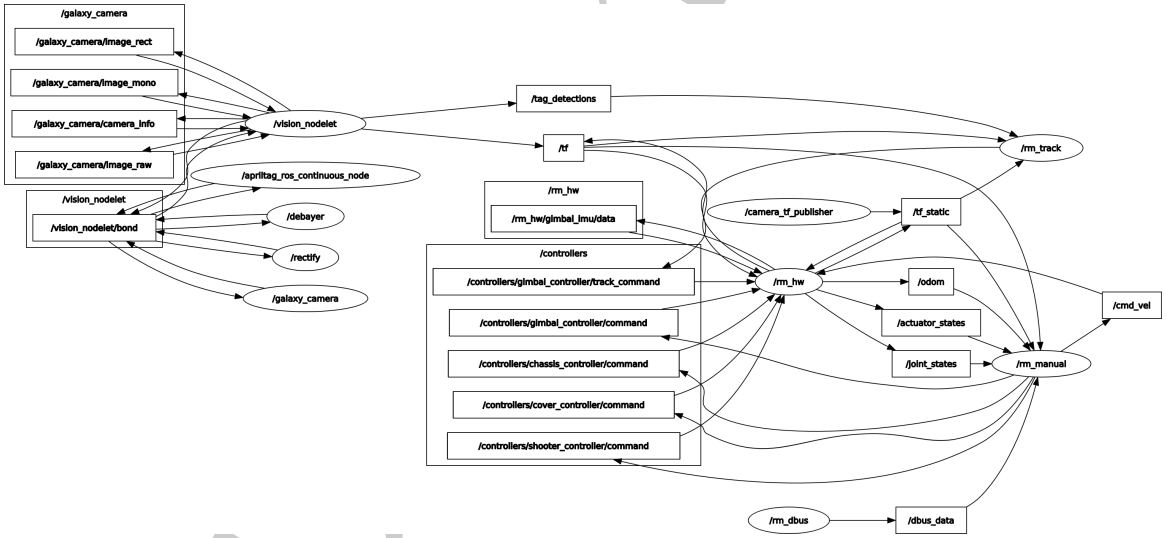


图 3.60: 步兵机器人节点图

图 3.60展示了步兵机器人的 ROS 节点图，隐藏了不活动部分，并且没有显示 ROS 服务。它还有以下节点：

- exposeurevalue_seting: 设置相机曝光值
- vision_nodelet_manager: 运行了所有视觉相关代码

`rm_hw` 中运行的控制器有：

- 状态控制器（发布器）
  - robot_state_controller (robot_state_controller/RobotStateController)
  - joint_state_controller (joint_state_controller/JointStateController)
- 主控制器
  - chassis_controller (rm_chassis_controllers/MecanumController)
  - orientation_controller (rm_orientation_controllers/Controller)
  - gimbal_controller (rm_gimbal_controllers/Controller)

- shooter_controller (rm_shooter_controllers/Controller)
- cover_controller (effort_controllers/JointPositionController)

- 校准控制器
  - trigger_calibration_controller (rm_calibration_controllers/JointCalibrationController)
  - cover_calibration_controller (rm_calibration_controllers/JointCalibrationController)
  - gimbal_calibration_controller (rm_calibration_controllers/JointCalibrationController)

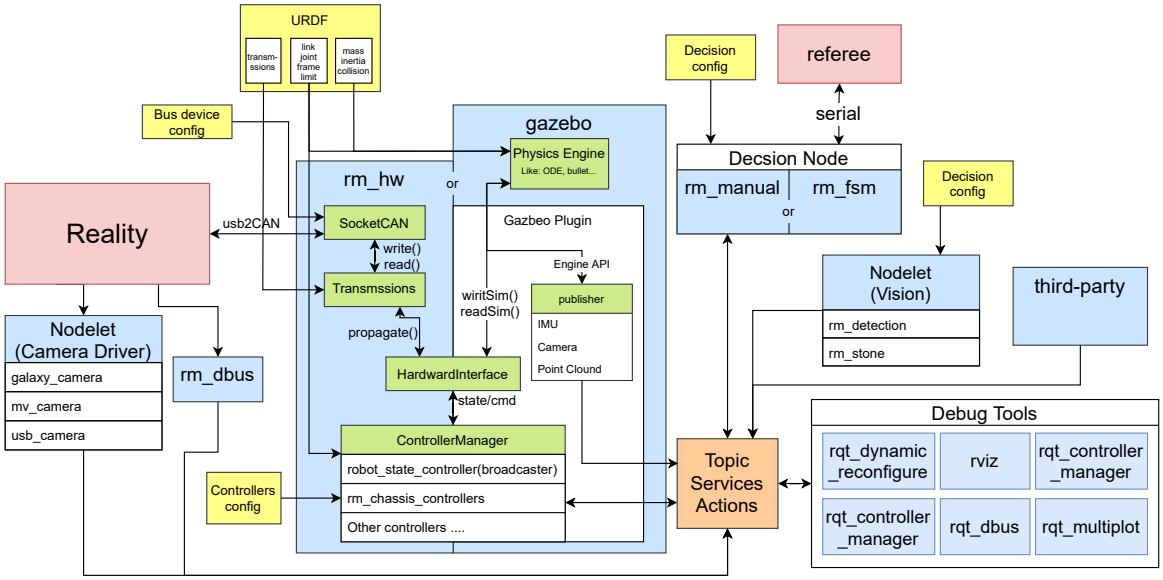


图 3.61: rm-contorls 框架示意框图；浅蓝色代表一个 Node（进程）；浅绿色代表类名或相关机制；浅黄色表示配置文件和数据；橙色为 ROS 通信中的话题、节点、动作；红色为实车。

图 3.61 展示了上述包的工作方式。所有控制器由 `ControllerManager` 管理，可以被初始化、开始、停止，并以 1khz 的频率更新，`ControllerManager` 可以被加载运行在实车的 `rm_hw` 节点或以 Gazebo Plugin 的形式加载进 `gazebo` 节点。控制器从 `HardwareInterface` 获得各传感器和执行器的 handle，进行控制，同时通过 ROS 的话题、服务和动作接受其他节点的指令和发布自己的状态。相机驱动和视觉算法通过 `nodelet`[5] 运行在同一个节点中并实现“零拷贝”。决策层通过串口读取裁判系统数据，并根据裁判系统数据和操作手指令通过话题、服务和动作对上述节点和第三方节点（如 `move_base` 路径规划、`moveit` 轨迹规划）进行操作。配置文件和数据由 `rosparam` 加载到 ROS 参数服务器上，各节点都能查询获取。多种调试工具也通过 ROS 的话题、服务和动作进行交互。

3.3.2.2 常用控制器

3.3.2.2.1 robot_state_controller 控制器 robot_state_controllers 根据 URDF 和从 `JointStateInterface` 获取到的关节位置计算机器人的正运动学然后将计算结果存入 robot_state 中的 `tf_buffer`并以一定频率发布在话题 `/tf` 上。

硬件接口

1. `JointStateInterface` - 用于获取所有 joint 的位置
2. `RoboSateInterface` - 用于维护 `tf_buffer`

订阅话题

1. `/tf` (tf/tfMessage)

   ROS tf 机制的默认话题接口

发布话题

1. /tf (tf/tfMessage)
   ROS tf 机制的默认话题接口

3.3.2.2.2 rm_chassis_controllers 控制器 rm_chassis_controllers 有 RAW、FOLLOW、GYRO 和 TWIST 四种状态，它根据速度与加速度指令计算通过逆运动学出底盘各个电机的转速，经过 PID 控制器得到各电机扭矩，并进行功率限制；还能通过电机返回数据通过正运动学解算底盘里程计并发布。

硬件接口

1. JointStateInterface - 用于获取底盘 joint 的位置和速度
2. EffortJointInterface - 用于发送底盘 joint 的扭矩指令
3. RoboSateInterface - 用于高频维护 odom -> base_link 的变换关系。

订阅话题

1. /cmd_chassis (rm_msgs/ChassisCmd)
   设定底盘的模式、加速度、最大功率。
2. /cmd_vel (geometry_msgs/Twist)
   设定底盘的速度。

发布话题

1. /odom (nav_msgs/Odometry.msg)
   底盘里程计信息（速度、位置、协方差）

3.3.2.2.3 rm_gimbal_controllers 控制器 rm_gimbal_controllers 有RATE、TRACK、DIRECT 三种状态，它根据命令对 yaw 和 pitch 两个 joint 进行 PID 控制，还能获取视觉目标数据使用子弹发射模型解算、预测和跟踪目标。

硬件接口

1. JointStateInterface - 用于获取云台 joint 的位置和速度
2. EffortJointInterface - 用于发送云台 joint 的扭矩指令
3. RoboSateInterface - 用于获取云台和视觉目标与世界坐标系在当前和历史的变换关系

订阅话题

1. /command (rm_msgs/GimbalCmd)
   设定云台的模式、pitch 和 yaw 轴转速、跟踪目标、指向目标及坐标系。
2. /track_command (rm_msgs/TrackCmd)
   接收视觉识别数据。

发布话题

1. error (rm_msgs/GimbalError)
   子弹模型计算的以当前云台角度射击对目标的距离误差

3.3.2.2.4 rm_shooter_controllers 控制器 rm_shooter_controller 有 STOP、READY、PUSH、BLOCK 四种状态，它根据命令通过 PID 控制左右摩擦轮转速和拨弹盘的位置从而发射弹丸同时还实现了卡弹检测及接触卡弹。

硬件接口

1. JointStateInterface - 用于获取摩擦轮、拨弹盘的速度和拨弹盘的位置
2. EffortJointInterface - 用于发送摩擦轮和拨弹盘的扭矩指令

订阅话题

1. command (rm_msgs/ShootCmd): 设定控制器状态、子弹速度、射击频率。

3.3.2.2.5 rm_calibration_controller 控制器 由于部分电机断电后零点会发生改变，rm_calibration_controller 启动后将按一定速度运动直到卡到机械限位，对电机位置进行归零。

硬件接口

1. `EffortJointInterface` - 根据获得的校准速度，给目标关节发布一个达到该校准速度所要用的作用力指令。
2. `ActuatorExtraInterface` - 用于获取目标执行器的基准点，当前位置，是否已经校准成功状态，是否停止状态的信息。

提供服务

1. `is_calibrated` (control_msgs/QueryCalibrationState)
   当向此服务发出一个请求时，此服务会返回校准目标是否成功校准。

3.3.2.2.6 控制器配置文件 代码 3.2展示了步兵机器人的所有控制器的配置文件。将此配置文件里的参数加载到参数服务器上后，各个控制器可以各自从参数服务器上获取自己想要的参数。因参数过多，不做一一讲解。其中可以看到云台电机，拨盘电机，弹仓盖电机的校准控制器拥有相同结构的参数配置。三个校准控制器是同一份代码，被独立地动态加载，分别获取需要校准的电机，对应的 joint 等信息。

Listing 3.2: 步兵控制器配置

```
controllers:
  joint_state_controller:
    type: joint_state_controller / JointStateController
    publish_rate: 100
  robot_state_controller:
    type: robot_state_controller / RobotStateController
    publish_rate: 100

  gimbal_calibration_controller:
    type: rm_calibration_controllers / JointCalibrationController
    joint: pitch_joint
    actuators: [ pitch_joint_motor ]
    search_velocity: 6.28
    threshold: 1e-2
    pid: { p: 0.3, i: 0, d: 0.0, i_clamp_max: 0.0, i_clamp_min: 0.0, antiwindup: true }
  trigger_calibration_controller:
    type: rm_calibration_controllers / JointCalibrationController
    joint: trigger_joint
    actuators: [ trigger_joint_motor ]
    search_velocity: 3.1415
    threshold: 0.2
    pid: { p: 0.3, i: 0, d: 0.0, i_clamp_max: 0.0, i_clamp_min: 0.0, antiwindup: true }
  cover_calibration_controller:
    type: rm_calibration_controllers / JointCalibrationController
    joint: cover_joint
    actuators: [ cover_joint_motor ]
    search_velocity: 6.28
    threshold: 1e-2
    pid: { p: 0.6, i: 0, d: 0.0, i_clamp_max: 0.0, i_clamp_min: 0.0, antiwindup: true }
```

```yaml
chassis_controller :
  type:  rm_chassis_controllers /OmniController
  # ChassisBase
  publish_rate: 100
  enable_odom_tf: true
  publish_odom_tf: false
  power:
     effort_coeff : 10.0
     vel_coeff : 0.0060
    power_offset: -8.41
  twist_angular: 0.5233
  timeout: 0.1
  pid_follow: { p: 20, i: 0, d: 0.6, i_max: 0.0, i_min: 0.0, antiwindup: true, publish_state : true }
  twist_covariance_diagonal: [ 0.001, 0.001, 0.001, 0.001, 0.001, 0.001 ]

  # OmniController
  chassis_radius: 0.208
  wheel_radius: 0.07625
  left_front :
    joint: "left_front_wheel_joint"
    pid: { p: 0.8, i: 0, d: 0.0, i_max: 0.0, i_min: 0.0, antiwindup: true, publish_state : true }
  right_front:
    joint: "right_front_wheel_joint"
    pid: { p: 0.8, i: 0, d: 0.0, i_max: 0.0, i_min: 0.0, antiwindup: true, publish_state : true }
  left_back:
    joint: "left_back_wheel_joint"
    pid: { p: 0.8, i: 0, d: 0.0, i_max: 0.0, i_min: 0.0, antiwindup: true, publish_state : true }
  right_back:
    joint: "right_back_wheel_joint"
    pid: { p: 0.8, i: 0, d: 0.0, i_max: 0.0, i_min: 0.0, antiwindup: true, publish_state : true }

orientation_controller :
  type:  rm_orientation_controller / Controller
  publish_rate: 100
  name: "gimbal_imu"
  frame_source: "odom"
  frame_target: "base_link"

gimbal_controller:
  type: rm_gimbal_controllers / Controller
  yaw:
    joint: "yaw_joint"
    pid: { p: 10, i: 0, d: 0.4, i_clamp_max: 0.3, i_clamp_min: -0.3, antiwindup: true, publish_state : true }
  pitch:
    joint: "pitch_joint"
```

```
    pid: { p: 20.0, i: 0, d: 0.45, i_clamp_max: 0, i_clamp_min: 0, antiwindup: true, publish_state : true }
  feedforward:
    gravity: 0.0
    enable_gravity_compensation: false
    mass_origin: [  0.0,0.0,0.0  ]
  bullet_solver :
    resistance_coff_qd_10: 0.45
    resistance_coff_qd_15: 1.0
    resistance_coff_qd_16: 0.7
    resistance_coff_qd_18: 0.55
    resistance_coff_qd_30: 3.0
    g: 9.81
    delay: 0.1
    dt: 0.001
    timeout: 0.001
    publish_rate: 50

shooter_controller:
  type: rm_shooter_controllers / Controller
  publish_rate: 50
  friction_left :
    joint: "left_friction_wheel_joint"
    pid: { p: 0.001, i: 0.01, d: 0.0, i_clamp_max: 0.01, i_clamp_min: -0.01, antiwindup: true, publish_state
        : true }
  friction_right :
    joint: "right_friction_wheel_joint"
    pid: { p: 0.001, i: 0.01, d: 0.0, i_clamp_max: 0.01, i_clamp_min: -0.01, antiwindup: true, publish_state
        : true }
  trigger:
    joint: "trigger_joint"
    pid: { p: 50.0, i: 0.0, d: 3.0, i_clamp_max: 0.0, i_clamp_min: 0.0, antiwindup: true, publish_state :
        true }
  push_per_rotation: 9
  push_qd_threshold: 0.90
  block_effort: 2.0
  block_speed: 0.1
  block_duration: 0.05
  block_overtime: 0.5
  anti_block_angle: 0.2
  anti_block_threshold: 0.1
  qd_15: 420.0
  qd_18: 480.0
  qd_30: 740.0
```

3.3.3 运行流程

如 **图** 3.62，ros-control 提供这样的机制：执行器（Actuator）的编码器等传感器数据被读取后通过 TransmissionsInterface（详见3.3.3.2）映射成关节 (Joint) 等机器人状态，将这些状态接口（详见3.3.3.1）提供给控制器；经过控制器计算后得到关节指令经过限制，再映射为电机的指令，发送给电机。控制器管理器可以实时加载、开始和停止各种控制器（以动态库的形式编译）。

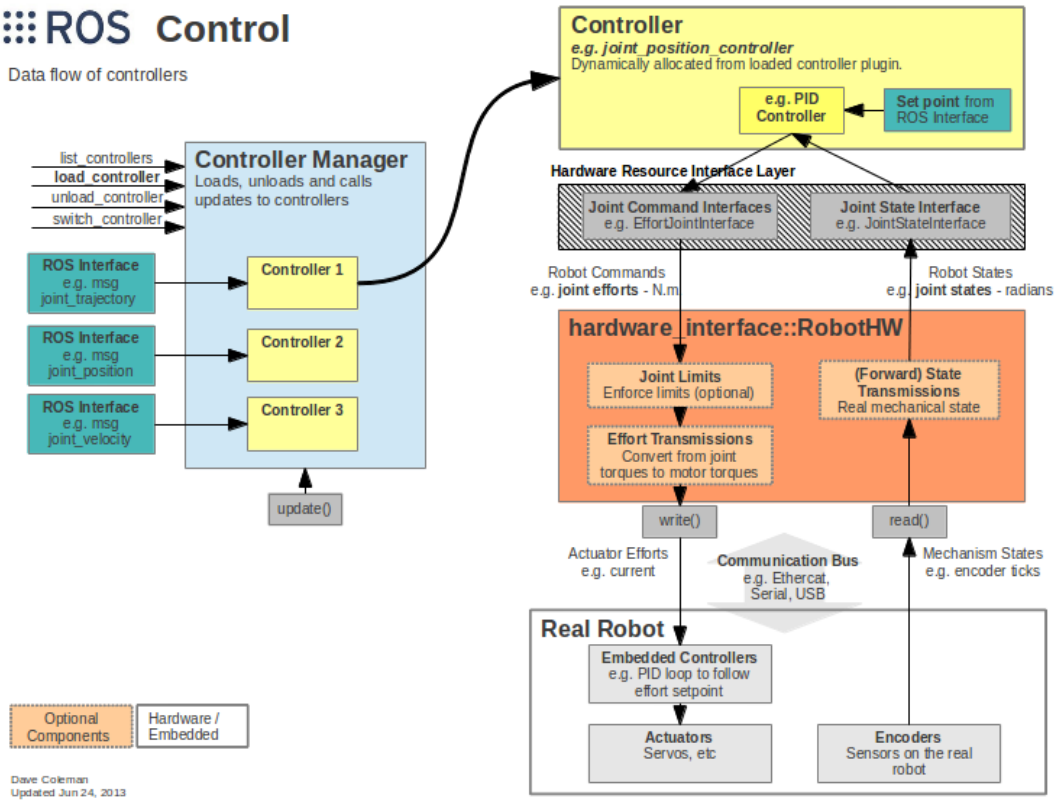


图 3.62: ros-control 框图，图源[6]

rm-control 和 rm-controllers 是基于 ros-control 开发的，它们的运行流程可分为三步：1、硬件层（底层）读取执行器的编码器等传感器数据，并映射成关节将数据传递给控制器；2、控制器计算出关节的力矩指令，并传回给硬件层；3、硬件层将关节指令再次映射为电机指令，并将指令发送出去。以上三步构成一个控制循环，并以 1KHZ 的频率循环运行。

其他的 ROS 软件包是并行运行的，没有特定的先后流程顺序。

3.3.3.1 硬件接口

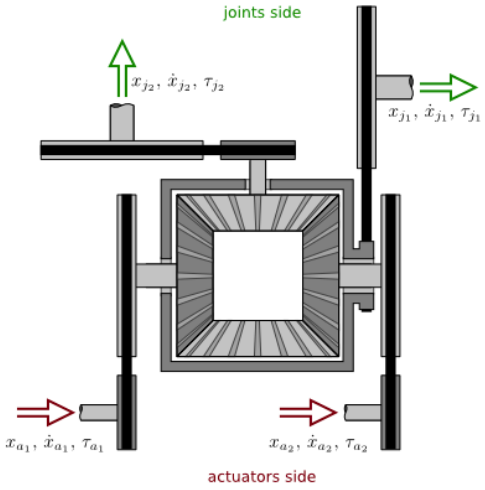
提供给控制器常见硬件接口

- Joint Command Interface - 关节指令发送接口
  - Effort Joint Interface - 用于向指令是力矩/力的执行器 (如: RoboMaster 3508) 对应的关节发送指令
  - Velocity Joint Interface - 用于向指令是速度执行器 (如: 部分舵机) 对应的关节
  - Position Joint Interface - 用于向指令是位置执行器 (如: 大部分舵机) 对应的关节
- Joint State Interfaces - 关节状态获取接口，用于获取关节的位置、速度和作用力（力或扭矩）
- Actuator State Interfaces - 执行器状态获取接口，用于获取执行器的位置、速度和作用力（力或扭矩）
- Actuator Command Interfaces - 执行器指令发送接口，和关节指令接口类似

- Force-torque sensor Interface - 力-力矩传感器接口
- IMU sensor Interface - IMU 传感器接口

3.3.3.2 Transmissions

Transmissions 是实际机器人执行器于关节的状态和指令映射，有简单减速比（改变正负可以将电机反向）、差速器（常见于机械臂末端两个关节）、双执行器（两个电机带动一个关节）。**图** 3.63 展示了差分 Transmissions 的简图和计算方法。



	Effort	Velocity	Position
Actuator to joint	$\tau_{j_1} = n_{j_1}(n_{a_1}\tau_{a_1} + n_{a_2}\tau_{a_2})$	$\dot{x}_{j_1} = \frac{\dot{x}_{a_1}/n_{a_1} + \dot{x}_{a_2}/n_{a_2}}{2n_{j_1}}$	$x_{j_1} = \frac{x_{a_1}/n_{a_1} + x_{a_2}/n_{a_2}}{2n_{j_1}} + x_{off_1}$
	$\tau_{j_2} = n_{j_2}(n_{a_1}\tau_{a_1} + n_{a_2}\tau_{a_2})$	$\dot{x}_{j_2} = \frac{\dot{x}_{a_1}/n_{a_1} - \dot{x}_{a_2}/n_{a_2}}{2n_{j_2}}$	$x_{j_2} = \frac{x_{a_1}/n_{a_1} - x_{a_2}/n_{a_2}}{2n_{j_1}} + x_{off_2}$
Joint to actuator	$\tau_{a_1} = \frac{\tau_{j_1}/n_{j_1} + \tau_{j_2}/n_{j_2}}{2n_{a_1}}$	$\dot{x}_{a_1} = n_{a_1}(n_{j_1}\dot{x}_{j_1} + n_{j_2}\dot{x}_{j_2})$	$x_{a_1} = n_{a_1}[n_{j_1}(x_{j_1} - x_{off_1}) + n_{j_2}(x_{j_2} - x_{off_2})]$
	$\tau_{a_2} = \frac{\tau_{j_1}/n_{j_1} - \tau_{j_2}/n_{j_2}}{2n_{a_1}}$	$\dot{x}_{a_2} = n_{a_2}(n_{j_1}\dot{x}_{j_1} - n_{j_2}\dot{x}_{j_2})$	$x_{a_2} = n_{a_2}[n_{j_1}(x_{j_1} - x_{off_1}) - n_{j_2}(x_{j_2} - x_{off_2})]$

图 3.63: 差分 Transmissions，图源[7]

下列代码为步兵机器人 URDF 中云台 pitch 轴执行器 `pitch_joint_motor` 与云台 pitch 关节 `pitch_joint` transmission，由于 pitch 轴是 6020 电机直驱，减速比为 1，又实际电机转向于关节定义的转向相反，取减速比为-1，在校准时实测得偏移为 1.559rad。有了这一个的映射，云台控制器只使用关节状态不需要考虑不同机器人电机安装位置、方向和初始值。

Listing 3.3: pitch 轴 transmission

```
<transmission name="trans_pitch_joint">
    <type>transmission_interface/SimpleTransmission</type>
    <actuator name="pitch_joint_motor">
        <mechanicalReduction>-1</mechanicalReduction>
    </actuator>
    <joint name="pitch_joint">
        <offset>1.559</offset>
        <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
    </joint>
</transmission>
```

3.3.4 重点功能

3.3.4.1 Gazebo 仿真

得益于 ros_control 的机制，控制器可以被加载到硬件接口上，也可以被加载到 Gazebo 模拟器中，如 **图** 3.64。值得一提的是实车硬件和仿真用的控制器都是同一份代码，不存在移植或重新编译的过程，甚至使用的是同一个二进制文件（由服务器 CI 编译测试并发布到 apt 源并安装）。这帮助我们在机械装配时，可以先提前测试程序，在机械组装配完成机器人后，只需要根据实车测试更改部分参数，而不需要更改任何代码，提高了开发效率。

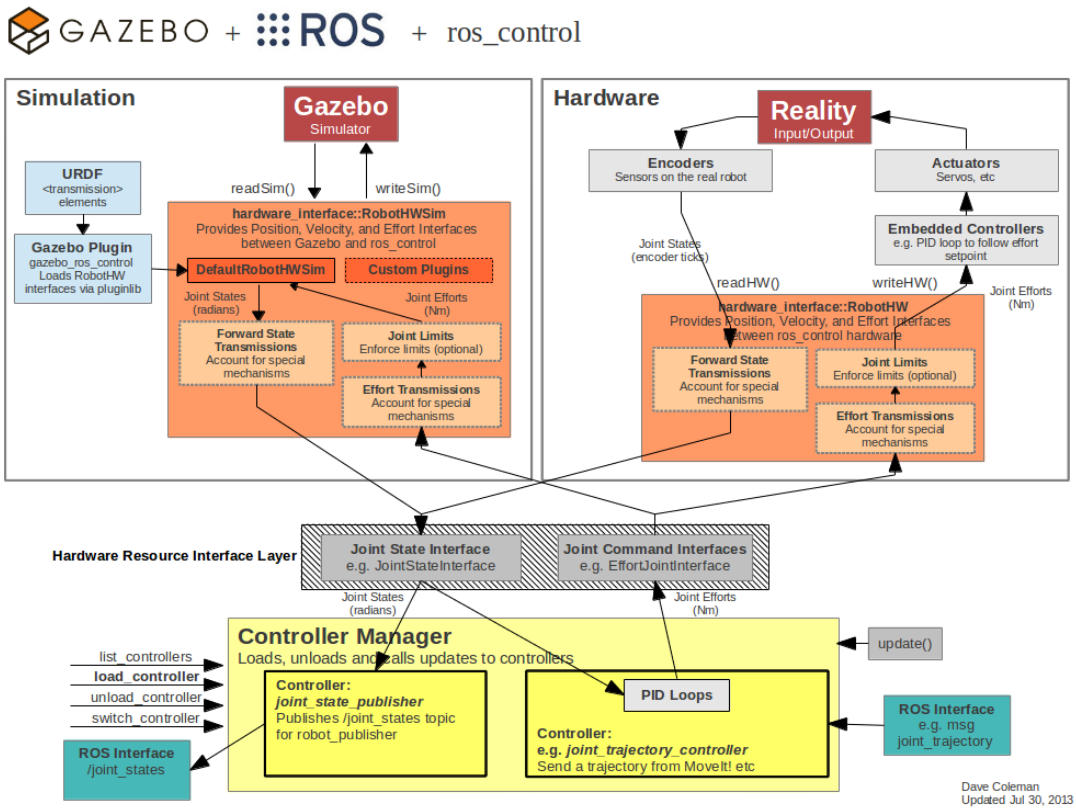


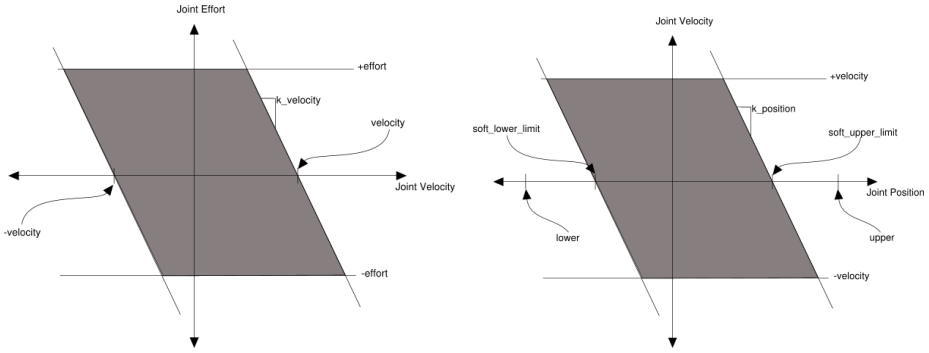
图 3.64: Gazebo + ROS + ros_control 简图，图源[8]

3.3.4.2 软限位

软限位从软件上限制关节的运动范围，为了使关节到达硬限位之前对关节进行限制，避免关节频繁撞击硬限位导致硬件损坏。其具体的实现方式是在接近软限位时，采用串级 P 控制器，先通过改变力矩对速度进行限制，再通过改变速度来限制位置。

当关节位置接近软上限时，`safety_controller`改变其所在关节最大输出力矩的上下限，实际上就是对输出力矩采用 P 控制器，以此来对该关节最大速度进行限制。关节最大速度则限制了关节的运动范围，这一步等价于对输出速度采用 P 控制器，再将其串起来变成串级 P 控制器。具体参见 **图** 3.65。其中，`safety_length_min`是软下限位置，它的取值是硬件上限减去一个较小的偏移量，`safety_length_max`是软上限位置，它的取值是硬件上限加上一个较小的偏移量。`k_velocity`决定了力矩界限的尺度，`k_position`决定了速度界限的尺度。

下列代码为步兵机器人 URDF 中云台 pitch 轴的软限位设置，其中有三个参数，`threshold` 是软限位与硬限位之间的差值，`pitch_lower_limit` 是硬限位最小值，`pitch_upper_limit` 是硬限位最大值。`k_position` 和 `k_velocity` 需要根据执行器属性及实际需要设置，`soft_lower_limit` 是软限位的最小值，按照上述说明，它应



(a) 力矩限制　　(b) 速度限制

图 3.65: 软限位的限制方式，图源[9]

该等于 `pitch_lower_limit+threshold`，使 pitch 轴触碰到硬限位之前启用软限位。同理，`soft_upper_limit` 是软限位的最大值，它应该等于 `pitch_lower_limit-threshold`。

Listing 3.4: pitch 轴软限位

```
<xacro:property name="threshold" value="0.1"/>
<xacro:property name="pitch_lower_limit" value="-0.71"/>
<xacro:property name="pitch_upper_limit" value="0.45"/>
<safety_controller k_position="100" k_velocity="0.1"
                   soft_lower_limit="${pitch_lower_limit+threshold}"
                   soft_upper_limit="${pitch_upper_limit-threshold}"/>
```

3.3.4.3 枪口热量限制

枪口热量限制通过限制子弹发射频率来控制枪管热量，以达到不会因超热量而扣血的目的。我们采用的限制方案是当枪口剩余热量高于某个阈值时，发射频率为期望射频; 当枪口剩余热量低于某个阈值时，令发射频率随着枪口实际热量的增大而线性减小; 当枪口剩余热量等于一颗子弹的热量时，令发射频率为安全射频（在安全射频下，枪口热量几乎保持稳定）; 当枪口剩余热量小于一颗子弹的热量时，令发射频率为零。参见 **图** 3.66。其中，x 表示当前枪口热量，a 表示枪口热量上限，b 表示一颗子弹的热量，c 表示枪口热量冷却速度，d 表示热量限制的阈值，e 表示期望的发射频率，y 表示实际的发射频率。

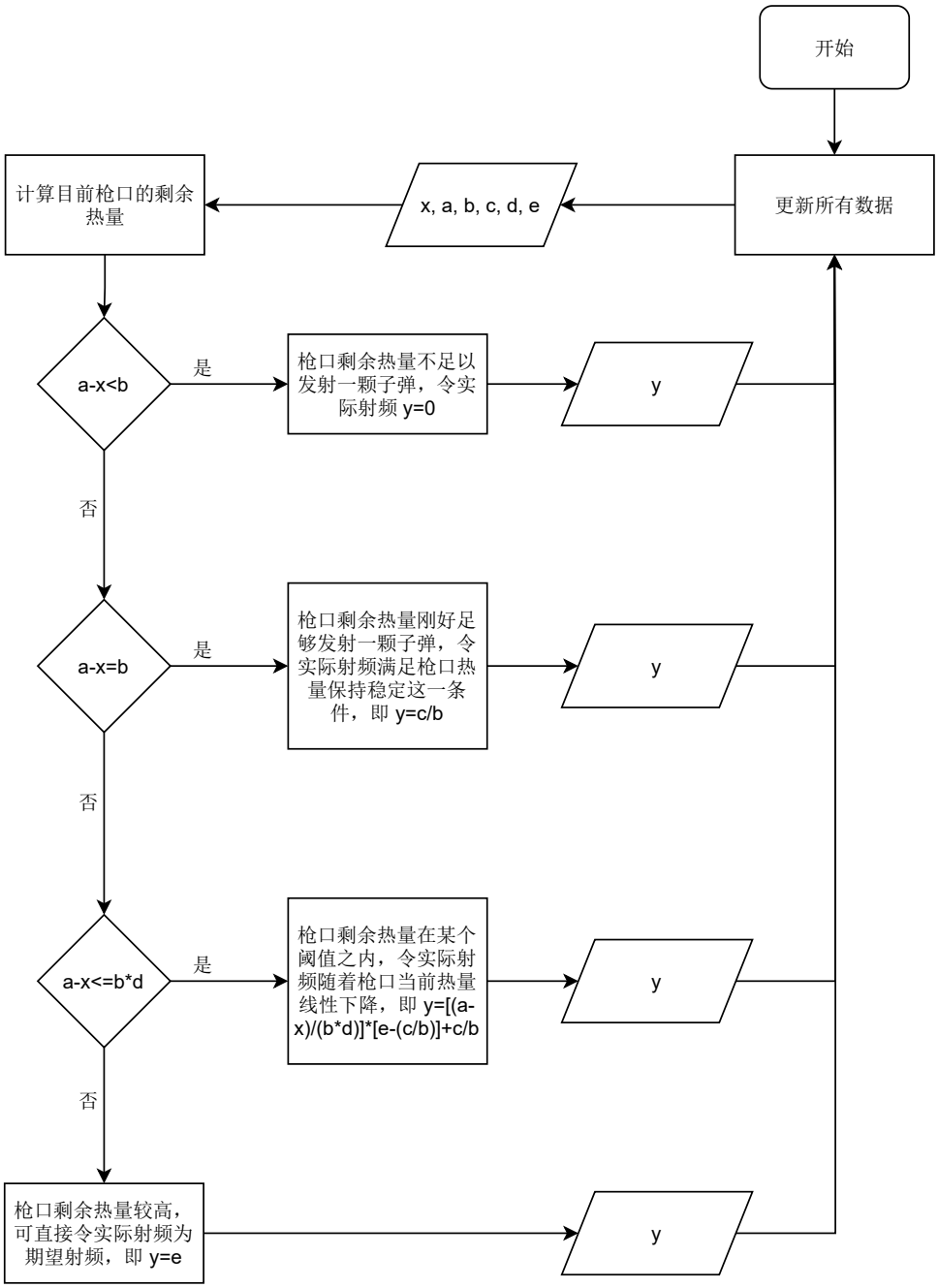


图 3.66: 枪口热量限制方案

3.3.4.4 软件底盘功率限制

软件功率限制通过限制电机输出力矩来控制底盘功率，以达到在底盘剧烈运动、实际功率即将超过给定的最大功率时，将实际功率限制到最大功率附近以下的目的。

3.3.4.4.1 理论

电机的输出功率为：

$$P_{\text{out}} = \tau\omega \tag{3.18}$$

电机的输入功率为输出功率和损失功率的和，可表示为：

$$P_{\text{in}} = P_{\text{out}} + k_1\tau^2 + k_2\omega^2 \tag{3.19}$$

其中 k_1 和 k_2 为常数。

3.3.4.4.2 实现

我们根据3.19来实现功率限制。已知：最大功率 $P_{\max}$、各电机当前转速 ω_{real} 和电机 PID 计算出来的原始力矩指令 τ_{cmd} 。当 τ_{cmd} 将使 P_{in} 高于 $P_{\max}$ 时，设有一缩放系数 k，令 $\tau'_{\text{cmd}} = k\tau_{\text{cmd}}$，使得 τ'_{cmd} 满足：

$$P_{\max} = \sum|\omega_{\text{real}}\tau_{\text{cmd}'}| + k_1\sum\tau'^2_{\text{cmd}} + k_2\sum\omega^2_{\text{real}} \tag{3.20}$$

由 $\tau'_{\text{cmd}} = k\tau_{\text{cmd}}$ 与(3.20)整理可得关于缩放系数 k 的一元二次方程:

$$(k_1\sum\tau^2_{\text{cmd}})k^2 + (\sum|\omega_{\text{real}}\tau_{\text{cmd}}|)k + ((k_2\sum\omega^2_{\text{real}} - P_{\max}) = 0 \tag{3.21}$$

则可由上式计算出 k 的值：

$$k = \frac{-\sum|\omega_{\text{real}}\tau_{\text{cmd}}| + \sqrt{\sum(\omega_{\text{real}}\tau_{\text{cmd}})^2 - 4k_1(\sum\tau^2_{\text{cmd}})(k_2\sum\omega^2_{\text{real}} - P_{\max})}}{2k_1\sum\tau^2_{\text{cmd}}} \tag{3.22}$$

最终给电机的力矩指令即为 $\tau'_{\text{cmd}} = k\tau_{\text{cmd}}$。**图** 3.67展示了步兵机器人在静止状态下进入高速小陀螺后再进入慢速小陀螺最后在高速小陀螺状态下进行平移的功率（最上）、速度指令（中间）和实际速度（最下）的曲线，可见在高速小陀螺状态下，功率被限制在了红线附近之下，实际速度比速度指令小，进入慢速小陀螺后，功率降低，此时实际速度与速度指令相等，最后在高速小陀螺状态下平移时，功率依然被限制在了青色线 (最大功率) 附近以下。

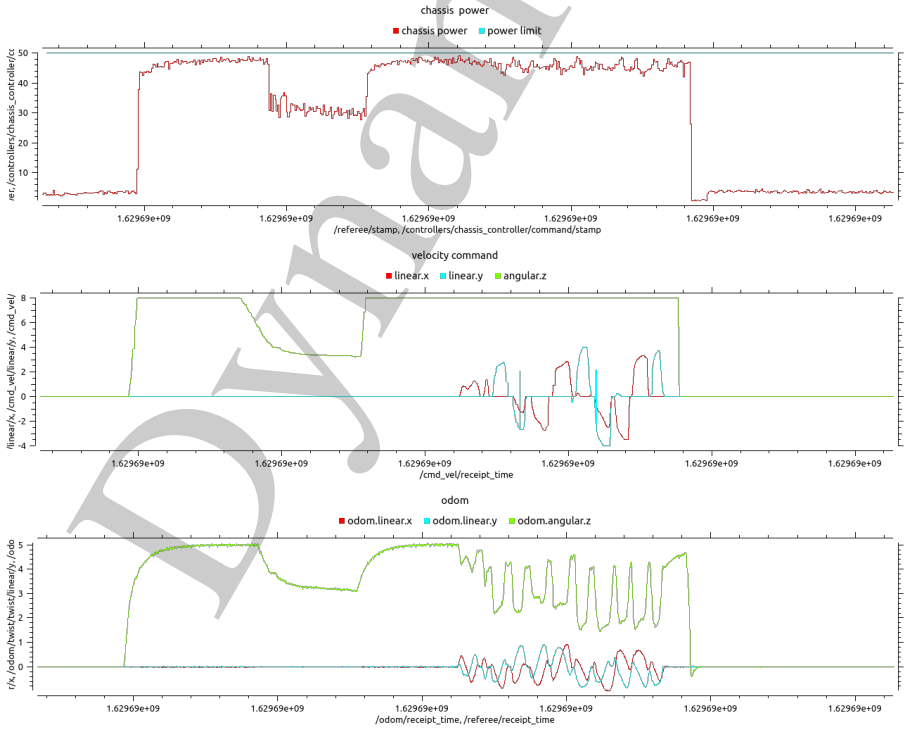


图 3.67: 底盘功率限制效果

3.3.4.5 相机图像与 imu 数据时间戳同步

相机图像与 imu 数据时间戳同步的意义在于消除云台快速运动时目标在世界坐标系位姿的剧烈抖动。imu 数据用于估计云台和相机在世界坐标系的姿态，配合其他坐标系变换信息得到准确位姿。如果两者不同步，就

意味着相机坐标系到世界坐标系的转换关系，和相机坐标系到目标坐标系的转换关系不同步。那么目标在世界坐标系的位姿的准确性也无从谈起。在云台快速运动时表现为位姿发生抖动。

为了使相机图像与 imu 数据时间戳同步，我们采用的方案是：imu 向相机发送触发信号。当相机收到触发信号时，采集图像并传输给 miniPC。同时 imu 在触发相机时通过 can 总线告知 miniPC 已触发相机这一信息，程序收到这一信息后，会将收到这一信息时的时间戳认为是 imu 数据与相机图像的采集时间，并将触发相机时的触发时间与 imu 姿态发布出去。相机驱动节点内部有一个 FIFO 队列缓存收到的触发时间，并在收到一帧图像后从队列中取出一个触发时间将其作为图像的时间戳并将图像发布出去。流程图如**图** 3.68所示。

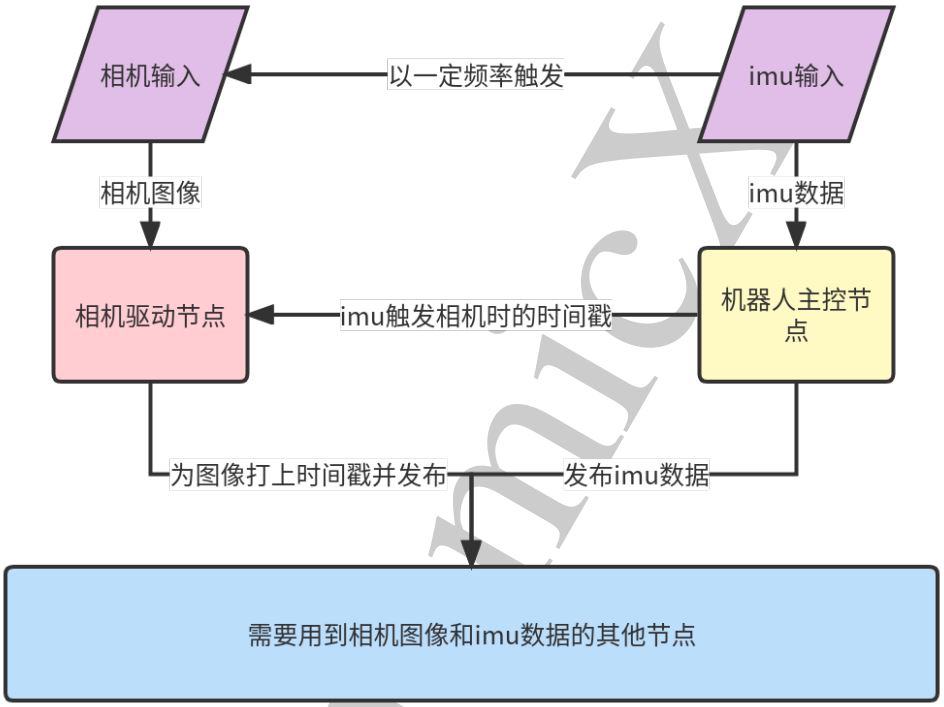


图 3.68: imu 与相机硬件同步方案

3.3.4.6 卡尔曼滤波预测目标位置

我们建立了装甲板在世界坐标系匀速直线运动的状态模型和量测模型，使用卡尔曼滤波对目标装甲板在世界坐标系的位置和速度进行最优估计，并进行预测。状态量如下：

$$x = \begin{bmatrix} x_n & \dot{x}_n & y_n & \dot{y}_n & z_n & \dot{z}_n \end{bmatrix}^T \tag{3.23}$$

x_n、y_n、z_n 为目标在世界坐标系的坐标，$\dot{x}_n$、$\dot{y}_n$、$\dot{z}_n$ 为目标在世界坐标系的速度。过程模型如(3.24)所示：

$$\boldsymbol{x}_{k+1} = \boldsymbol{F}_k \boldsymbol{x}_k + \boldsymbol{\Gamma}_k \boldsymbol{w}_k, \quad \boldsymbol{w}_k \sim N\left(\boldsymbol{0}_{3\times 1}, \boldsymbol{Q}_k\right) \tag{3.24}$$

其中：

$$\boldsymbol{F}_k = \begin{bmatrix} 1 & \Delta t & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{\Gamma}_k = \begin{bmatrix} \frac{1}{2}\Delta t^2 & 0 & 0 \\ \Delta t & 0 & 0 \\ 0 & \frac{1}{2}\Delta t^2 & 0 \\ 0 & \Delta t & 0 \\ 0 & 0 & \frac{1}{2}\Delta t^2 \\ 0 & 0 & \Delta t \end{bmatrix} \tag{3.25}$$

量测模型为：

$$z_k = \boldsymbol{H}_k \boldsymbol{x}_k + \boldsymbol{v}_k \tag{3.26}$$

采用 $r_n = [x_n, y_n, z_n]^T$ 作为量测向量，则：

$$\boldsymbol{H}_k = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \boldsymbol{v}_k \sim N\left(\boldsymbol{0}_{3\times 1}, \boldsymbol{R}_k\right) \tag{3.27}$$

卡尔曼滤波器在 rm 的运用已经非常成熟[10]。

3.3.5 软件测试

3.3.5.1 持续集成与持续部署

如 **图** 3.69所示我们将大部分代码放在 github 托管，并借助 GitHub Actions 对代码进行持续集成（CI）。开发人员每次将代码推送到远程仓库后，Github Actions 会自动化运行若干个单元测试。如果自动化测试发现新代码和现有代码之间存在冲突，CI 可以更加轻松地快速修复这些错误。

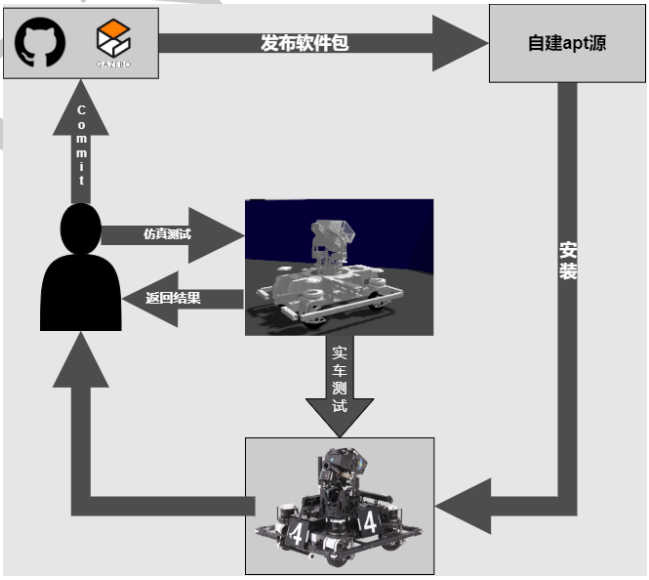


图 3.69: 持续集成与持续部署流程

GitHub Actions 有以下这些：

- CI - 相关单元测试
- Doxygen-docs 生成 api 文档页面并发布
- Format - 检查代码格式
- bloom-release - 在推新 tag 时自动把新版本信息和打包文件扔给官方源
- deb-package - 自动打包 deb

我们还自行搭建了软件源，在 CI 通过之后，会自动将最新代码发布到软件源。当在某台机器人上开发稳定之后，其他机器人可以使用安装或升级的方式获得最新功能的软件。这就是持续部署。除了在每次推送代码都会运行的 CI 以外，我们还创建了名为 rm_ci 的软件包，它会在固定的时间自动运行一系列的单元测试对我们的代码进行功能性测试，便于发现代码里在功能上出现的问题。

图 3.70展示了借助 GitHub Actions 对代码进行持续集成。

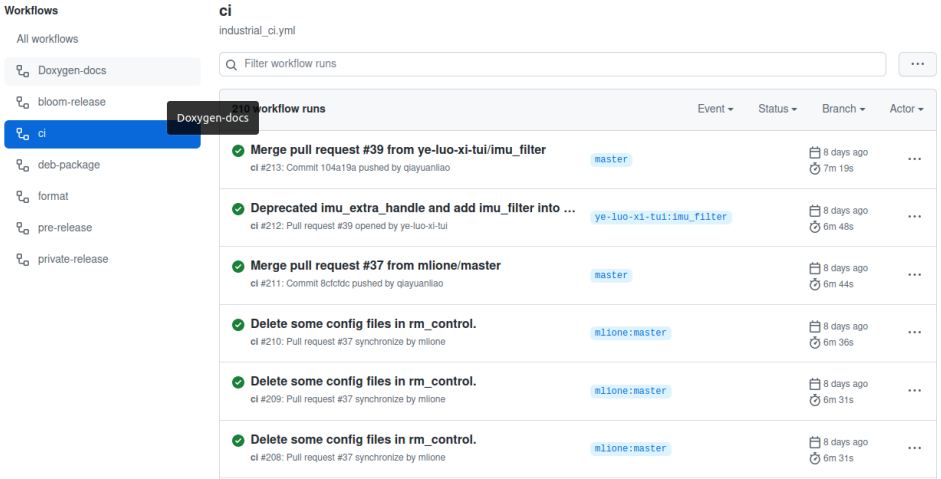


图 3.70: 借助 GitHub Actions 进行持续集成

以下展示使用两行指令完成底盘控制器的安装或更新。

Listing 3.5: 在机器人上更新软件

```
sudo apt update
sudo apt install ros-noetic-rm-chassis-controllers
```

图 3.71为 rm_ci 运行某个单元测试后测试通过的结果。

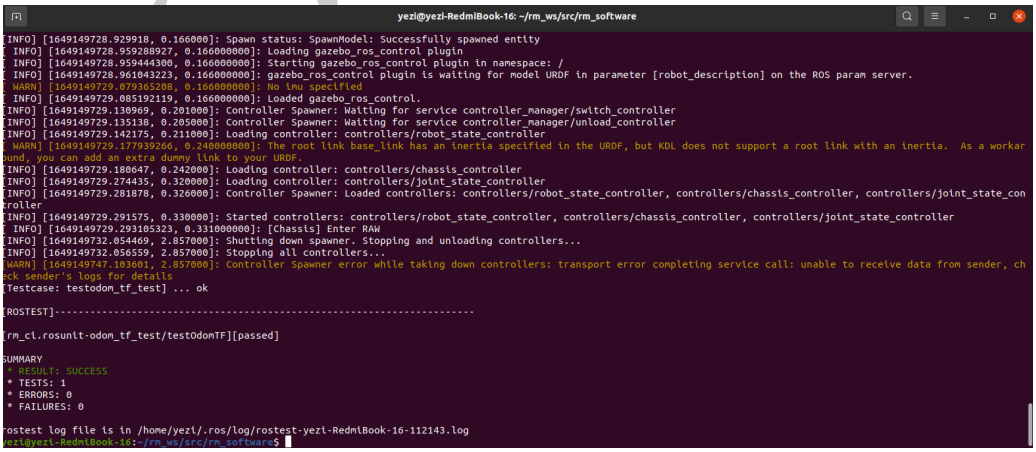


图 3.71: 单元测试运行结果

3.3.5.2 自瞄解耦测试

击打移动装甲板这一工作可以主要分为装甲板识别以及装甲板位置预测这两个部分，在上赛季的开发中，装甲板位置预测的程序需要等到装甲板识别稳定后才能开始测试，并且当击打移动装甲板的命中率低时，我们需要花较多的时间去找是识别的问题还是预测的问题。以上两个问题都使得开发效率低下。

在这一赛季中，我们使用 AprilTag 代替装甲板进行识别，用以测试除识别以外的包括筛选、拓展卡尔曼滤波预测目标位置等其他功能。我们使用第三方的包对 AprilTag 进行识别，识别准确。这使得我们能够将装甲板识别算法的开发与预测算法的开发解耦，使二者的开发与测试可以同时进行，并且在出现问题时能够及时定位问题所在，极大地提高了我们的开发效率。

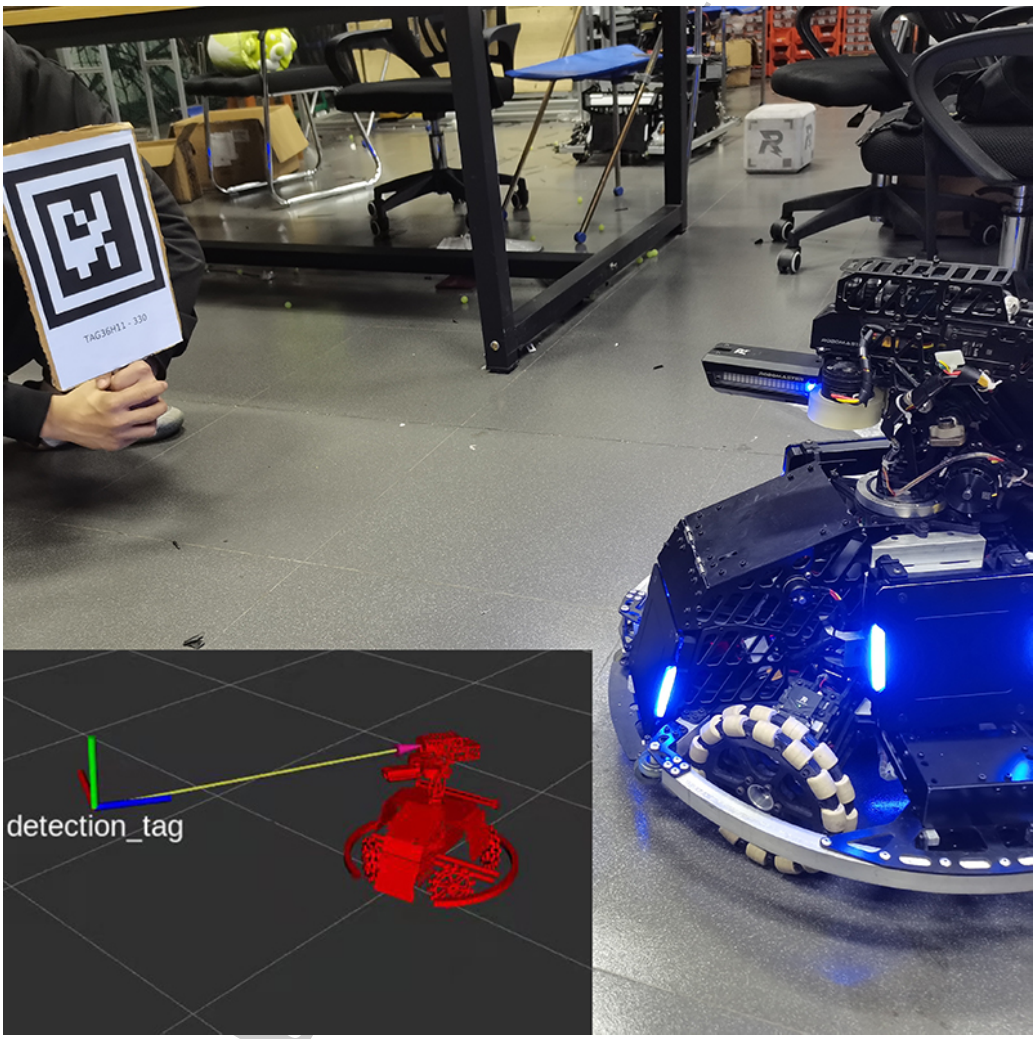


图 3.72: 识别并跟踪 AprilTag

3.4 算法设计

3.4.1 装甲板自瞄

3.4.1.1 算法相关的主要理论

3.4.1.1.1 色彩分割 在 RGB 颜色系统中，摄像头捕捉到的色彩图像由 R、G、B 三个通道组成，三个通道分别表示红、绿、蓝三种颜色的强度。通过对各个通道分离，并分别使用合理的阈值进行筛选，可对图像中指定颜色进行筛选。

由于 RGB 颜色系统对亮度等光照条件鲁棒性不足，我们一般会将图像转换到 HSV 颜色系统下进行处理. 定义：$R' = R/255, G' = G/255, B' = B/255, C_{\max} = \max(R', G', B'), C_{\min} = \min(R', G', B'), \Delta = C_{\max} - C_{\min}$，则从 RGB 转换为 HSV 的转换公式如下：

$$H = \begin{cases} 0^\circ & ,\Delta = 0 \\ 60^\circ \times (\frac{G'-B'}{\Delta} + 0) & ,C_{\max} = R' \\ 60^\circ \times (\frac{B'-R'}{\Delta} + 2) & ,C_{\max} = G' \\ 60^\circ \times (\frac{R'-G'}{\Delta} + 4) & ,C_{\max} = B' \end{cases} \tag{3.28}$$

$$S = \begin{cases} 0 & ,C_{\max} = 0 \\ \frac{\Delta}{C_{\max}} & ,C_{\max} \neq 0 \end{cases} \tag{3.29}$$

$$V = C_{\max} \tag{3.30}$$

与 RGB 相似，HSV 的色彩分割同样通过对各通道进行阈值筛选，生成的二值图即可分割出特定的色彩。例如: 设定阈值 $100 < h < 124, 43 < s < 255, 46 < v < 255$, 当满足以上三个条件时，输出 255，不满足时输出 0，即可得到色彩分割后的二值图，其中白色区域即位原图中蓝色的区域。

3.4.1.1.2 形态学处理 形态学图像处理（简称形态学）是指一系列处理图像形状特征的图像处理技术。形态学的基本思想是利用一种特殊的结构元来测量或提取输入图像中相应的形状或特征，以便进一步进行图像分析和目标识别。形态学方面，我们主要涉及到腐蚀、膨胀，及其应用：边界提取。在以下描述中，我们定义图像中前景为 1，背景为 0。

膨胀 将结构元 s 在图像 f 上滑动，把结构元锚点位置 (一般为结构元的几何中心) 的图像像素点的灰度值设置为结构元值为 1 的区域对应图像区域像素的最大值。用公式表示如下：

$$dst(x, y) = \max_{(x',y'):element(x',y')\neq 0} src(x + x', y + y') \tag{3.31}$$

其中 $element$ 表示结构元，(x, y) 为锚点 O 的位置，x' 和 y' 为结构元值为 1 的像素相对锚点 O 的位置偏移，src 表示原图，dst 表示结果图。膨胀运算用公式符号表示为:$f \oplus s$。膨胀能使物体边界扩大，具体的膨胀结果与图像本身和结构元素的形状有关。膨胀常用于将图像中原本断裂开来的同一物体桥接起来，对图像进行二值化之后，很容易使一个连通的物体断裂为两个部分，而这会给后续的图像分析（如要基于连通区域的分析统计物体的个数〉造成困扰，此时就可借助膨胀桥接断裂的缝隙。

腐蚀 将结构元 s 在图像 f 上滑动，把结构元锚点位置的图像像素点的灰度值设置为结构元值为 1 的区域对应图像区域像素的最小值。用公式表示如下：

$$dst(x, y) = \min_{(x',y'):element(x',y')\neq 0} src(x + x', y + y') \tag{3.32}$$

腐蚀运算用公式符号表示为：$f \ominus s$。腐蚀能够消融物体的边界，而具体的腐蚀结果与图像本身和结构元素的形状有关。如果物体整体上大于结构元素，腐蚀的结构是使物体变 “瘦” 一圈，而这一圈到底有多大是由结构元素决定的：如果物体本身小于结构元素，则在腐蚀后的图像中物体将完全消失：如物体仅有部分区域小于结构元素〈如细小的连通 3，则腐蚀后物体会在细连通处断裂，分离为两部分。

开闭运算 由于腐蚀和膨胀运算会存在处理后物体的大小会发生变化，为了缓解这种变化，我们一般会将腐蚀和膨胀结合使用。对图像 f 用同一结构元 s 先膨胀再腐蚀称之为闭运算，记为:$f \bullet s = (f \oplus s) \ominus s$。同理开运算记为：$f \circ s = (f \ominus s) \oplus s$

边界提取 轮廓是对物体形状的有力描述，对图像分析和识别十分重要。要在二值图中提取物体轮廓，只需要将所有物体内部的元素点置为背景点。具体做法即为逐行扫描二值图，如果发现前景点的 8 个邻域都是前景点，这该点为物体的内部点，将其设置为背景点。实际上这相当于采用一个 3*3 的结构元对原图像进行膨胀，再用膨胀后的图像减去原图像。

3.4.1.1.3 N 点透视位姿求解 如果场景 (或物体) 的三维结构已知，利用多个控制点在三维场景中的坐标及其在图像中的透视投影坐标即可求解出摄像机坐标系与表示三维场景结构的世界坐标系之间的绝对位姿关系，包括绝对平移向量 t 以及旋转矩阵 R，该类求解方法统称为 N 点透视位姿求解 (Perspective-N-Point，PNP 问题)，即 **图** 3.73 所示。

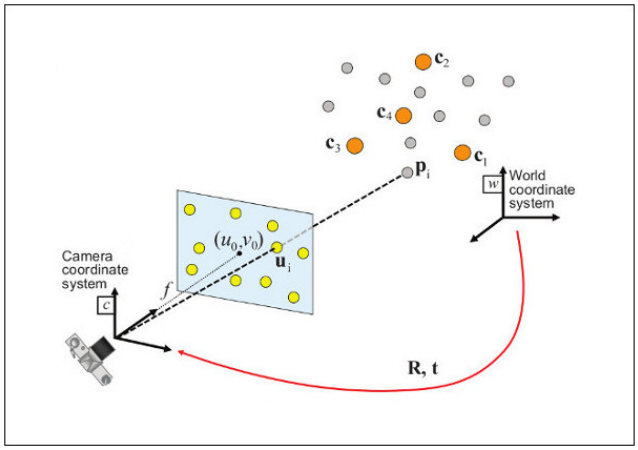


图 3.73: N 点透视位姿求解示意图

N 点透视位姿求解的最核心计算公式 (忽略畸变矩阵) 如下：

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{3.33}$$

这里的控制点是指准确知道三维空间坐标位置，同时也知道对应图像平面坐标的点。对于透视投影来说，若单目摄像头的内部参数已知 (通过相机标定获得内参)，要使得 PNP 问题有确定解，需要至少三组控制点，而当控制点在同一三维平面上时，需要至少四组控制点。

3.4.1.2 机器学习

3.4.1.2.1 支持向量机 支持向量机 (Support Vector Machine：SVM) 是由超平面定义的一种二分类分类模型。换句话说，给定标记的训练数据，算法输出最佳超平面，用来对新示例进行分类。SVM 的学习策略就是间隔最大化。在数学层面上其可以转化为一个求解凸二次规划的问题 (迭代与优化问题)。SVM 的学习算法就是求解凸二次规划的最优化算法。

对于给定的超平面 (w, b) 和训练的数据集 T, 定义超平面 (w, b) 关于样本点 (x_i, y_i) 的几何间距为：

$$\gamma_i = y_i(\frac{wx_i + b}{\|w\|}) \tag{3.34}$$

函数间距为：

$$\hat{\gamma}_i = y_i(wx_i + b) \tag{3.35}$$

其中：$x_i \in R^n$, 是样本点各维特征组成的向量;$y_i \in -1, +1$, 是样本点的类标签；$\|w\|$ 为 w 的模。并且记:

$$\gamma = \min \gamma_i, i = 1, 2, ..., N \tag{3.36}$$

$$\hat{\gamma} = \min \hat{\gamma}_i, i = 1, 2, ..., N \tag{3.37}$$

由上文中的式 (3.34)、(3.35)、(3.36)、(3.37) 可得：

$$\gamma = \frac{\hat{\gamma}}{\|w\|} \tag{3.38}$$

其中 γ 的实际意义就是确信度 (并不代表概率),γ 越大说明样本点离超平面 (w, b) 的距离越大，即该样本点越确定是某一类的。

对于硬间隔最大化 SVM(线性可分) 情况：SVM 算法其实就是在寻找具有最大" 间隔" 的超平面，并且需要保证每个样本点不被错分。这其实就是一个关于参数 w 和 b 的优化问题，即：

$$\max_{(w,b)} \gamma \tag{3.39}$$
$$\text{s.t.} \frac{y_i * (w * x_i + b)}{\|w\|} \geq \gamma \ , i = 1, 2, ...N$$

等价于：

$$\min \frac{1}{2}\|w\|^2 \tag{3.40}$$
$$\text{s.t.} y_i * (w * x_i + b) \geq 1 \ , i = 1, 2, ...N$$

上述优化问题可以通过拉格朗日算法进行求解，由于篇幅有限，不过多描述。一般情况下硬最大间隔 SVM 的条件过于苛刻，因此一般使用软最大间隔 SVM(允许少量样本点分类错误), 而且通常我们需要引入核函数，将线性不可分的样本映射到线性可分。

值得注意的是，SVM 并非概率输出模型，即得到的结果并不存在先成的置信度，与置信度相近的参数只有预测样本到超平面的距离，如果直接使用该参数进行阈值筛选会十分不合理以及不鲁棒。

3.4.1.2.2 卷积神经网络 神经网络顾名思义就是模仿人脑神经元构造去构造的一种机器学习算法。由于近些年计算机设备算力的不断发展，催生出了深度学习，即网络层数不断加深。诞生了不少十分优秀的网络结果，例如:YOLO 系列，GAN 等等。

全连接层 全连接层作为神经网络中重要的一种结构框架，是在模仿人脑神经元之间的连接方式的一种数学模型。由下图所示: 各层之间相互为输入和输出的关系，其之间的映射关系是线性的 (在不加入激活函数的情况下)。可以表示下面的矩阵运算：

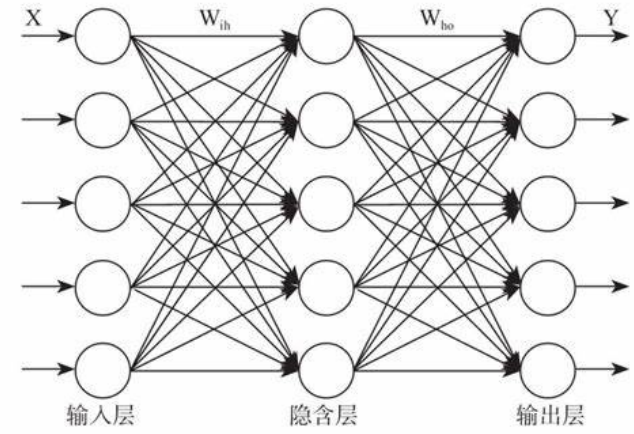


图 3.74: 全连接层网络示意图

$$H = XW_n + b_n \tag{3.41}$$

其中 H 为输出，X 为输入，W_n 为连接矩阵，其大小与输入和输出的神经元个数有关，b_n 为对应的偏置。

激活函数 全连接层运算为简单的线性运算，在这种情况下多层的线性运算可以等价为一层线性运算，而且单纯的线性模型很难拟合实际当中复杂的非线性模型，因此需要加入非线性运算环节，即激活函数。激活函数模拟神经元的工作原理，低于兴奋阈值时不做响应，高于阈值时才会发出响应。

常用的激活函数有三种，分别是阶跃函数、Sigmoid 和 ReLU。分别如下图从左到右所示:

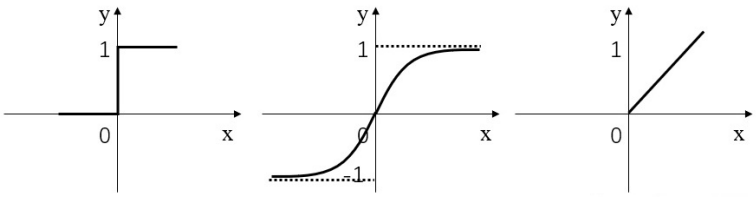


图 3.75: 三种常用的激活函数

卷积层 不难看出，全连接层的感受野等于输入的大小，这在对图像的处理上十分不合理，我们希望模型在对图像的处理上感受野是局部的，着重提取局部的纹理特征。卷积运算通过卷积核控制每次运算的感受野大小，并且实现了参数共享以达到减少训练参数的目的。卷积运算与全连接类似，如下所示：

$$s(i,j) = (X * W)(i,j) = \sum_m \sum_n x(i+m, j+n) w(m,n) \tag{3.42}$$

其中，m 和 n 分别为卷积核的长和宽。W 表示卷积核，X 表示图像。

池化层 池化 (pooling)，是一种降采样操作 (subsampling)，主要辅助卷积操作。目标是降低 feature maps 的特征空间，或者可以认为是降低 feature maps 的分辨率。因为 feature map 参数太多，而图像细节不利于高层特征的抽取。常用的池化有两种：第一种是最大值池化 (Max pooling)：，$n \times n$ 的 max pooling 就是取 n*n 个像素点中最大值并保留。平均值池化 (Average pooling): n * n 的 average pooling 就是取 n*n 个像素点中平均值值保留。

以上只是简单的描述了一下卷积神经网络中最基本的几种网络模块，像反向传播，批归一化等等十分重要的操作由于篇幅有限，就不再展开描述。

3.4.1.3 算法说明

3.4.1.3.1 装甲板自瞄算法说明 装甲板体积小，单纯依靠操作手进行打击难度极大，因此为机器人开发自瞄算法尤为重要。自瞄，顾名思义就是通过计算机自动识别装甲板位置并进行自主打击的功能，本章节说明自瞄算法当中获得目标相对机器人姿态的算法实现。我们的识别算法大致流程如 **图** 3.76所示。

接下来，我们会将装甲板的自瞄算法拆分，按实现顺序逐一介绍。

3.4.1.3.2 操作手交互 算法当中的部分参数，部分功能可以通过操作手现场更改，以使得自瞄有更好的灵活性。代码层面的实现上，在自瞄算法这一边，自需要编写一个 ROS 的服务，供电控、UI 或者裁判系统调用修改相关参数即可。我们实现了四个方面的参数或功能修改。

第一个是**敌对颜色的修改**。在机器人上电时，程序会读取裁判系统的己方颜色并调用自瞄当中的服务，修改敌对颜色。当然，自瞄算法的每一个计算循环开始都会重新读取这些可供操作手修改的参数，因此操作手也能在现场修改敌对颜色，虽然并没有什么用。

第二个是**击打目标的修改**。我们将自瞄目标分为了两个：装甲板和能量机关。原因是装甲板和能量机关的筛选以及判断条件都不相同。操作手可以通过键盘修改击打目标，本质上就是在自瞄循环开始创建不同类型的类，以达到不同识别流程的效果。

第三个是**装甲板目标的修改**。我们还对装甲板识别目标进行了分类：前哨站、哨兵和所有。三种情况分别就是只识别前哨站、只识别哨兵和不对目标进行筛选。实现的本质就是通过卷积神经网络对装甲板的 id 进行识别，再通过可修改的 id 参数作为判断条件即可。

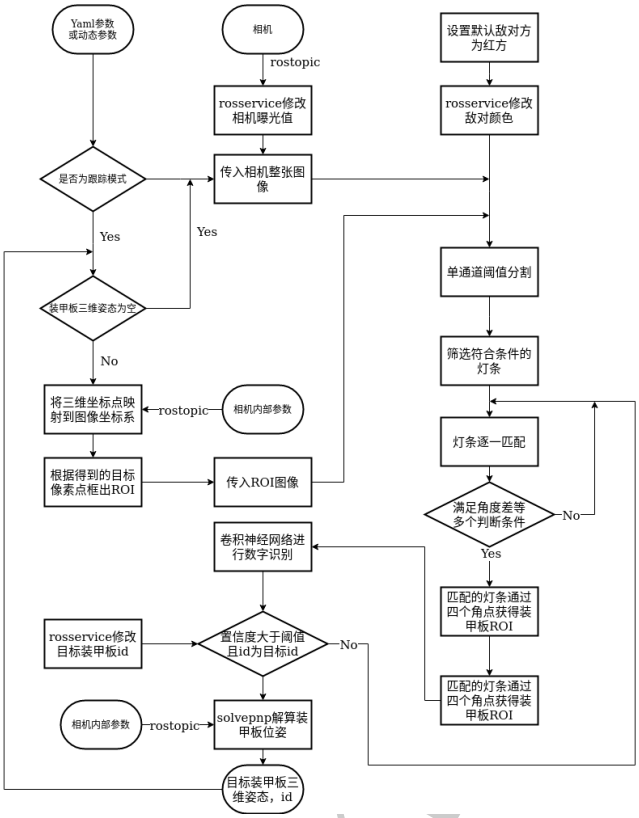


图 3.76: 装甲板自瞄流程图

第四个是**相机曝光值的修改**。由于比赛现场的光照条件无法完全确定，并且不同的曝光值，灯条的形状会发生变化导致一些判断条件发生变化。因此我们设置了 5 个级别的曝光等级以及每个等级对应的判断参数，供操作手在比赛前的 3 分钟准备时间将相机曝光修改为最佳的值，更好的确保自瞄正常运行。

第五个是**跟踪模式**。代码会默认启用跟踪模式，在该模型下，每次的识别都会依据上一次的识别结果，根据上一次识别的装甲板 ROI 获得一个相对较大的 ROI，本轮识别只会对这一 ROI 范围里的图像进行处理（如 **图** 3.77所示），直到目标丢失。如果目标丢失或在不在跟踪模式下，每次的识别就会在整张图像上进行计算处理。

3.4.1.3.3 色彩分割 装甲板两边的灯条是用于初步确定装甲板大致位置的重要信息，由于灯条属于发光体，并且为纯蓝或者纯红，可以通过图像处理当中的色彩分割初步确定装甲板位置。我们实现了两种色彩分割的方案，一种是单通道阈值分割，另一种为多通道的阈值分割。理论上，多通道的阈值分割的鲁棒性更好。

程序会在阈值分割前确定敌对颜色，再根据敌对颜色选用不同的参数作为阈值。我们通过大量的测试进行动态调参，以确定最终的参数值。对于单通道阈值分割，我们的阈值设置为 120；对于多通道阈值分割，我们的阈值设置为：$85 < h_{\text{blue}} < 115{,}97 < s_{blue} < 255{,}230 < v_{blue} < 255{,}29 < h_{red} < 255{,}115 < s_{red} < 255{,}156 < v_{red} < 255$。赛场上代码色彩分割后的效果图如 **图** 3.77所示。

3.4.1.3.4 灯条筛选及配对 在进行阈值分割后，我们还需进一步筛选出真正的灯条。首先会对二值图当中的所有轮廓进行边界提取然后进行四边形拟合，有内轮廓的四边形会被筛掉。对于拟合出来的四边形，我们会计算它们的长宽比，倾斜角度，拟合四边形面积与实际轮廓像素面积的比例，通过这三个值再进一步筛掉不符合的轮廓。

接下来就是灯条两两配对了，从左到右对灯条进行两两配对，我们会通过之前说到的三个参数进行判断，若为同一装甲板的灯条，理论上两个灯条的参数会十分相近。两个灯条配对成功后，我们取两个灯条内侧的四个点形成 ROI，并对该 ROI 进行宽度的压缩和高度的拉长以获得装甲板的 ROI，其中的伸缩比例由灯条的拟合四边形长宽决定。效果如 **图** 3.78所示。

图 3.77: 色彩分割效果图

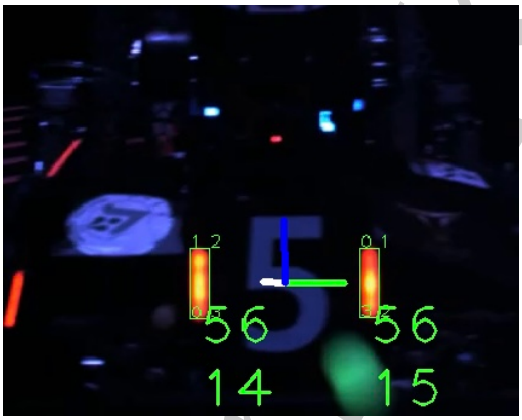
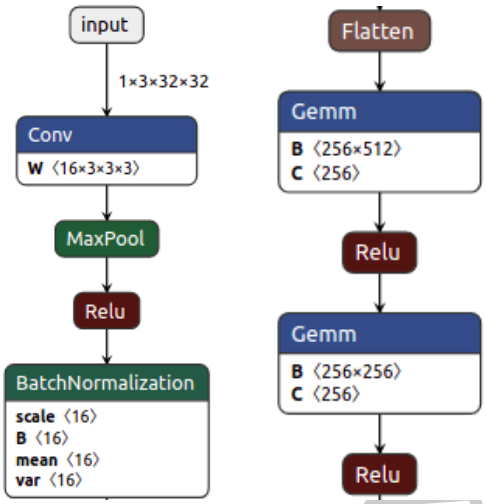


图 3.78: 灯条拟合效果图

3.4.1.3.5 装甲板 ROI 筛选及 id 确认 获得了装甲板的 ROI 后，我们还需要进一步进行判断，我们首先会计算装甲板 ROI 的长宽比，以及角度的大小，通过这两个参数在形状层面上进行筛选。随后，我们会将装甲板的 ROI 进行透视变换，方便后续的数字识别。我们通过使用卷积神经网络进行数字识别，获得装甲板 id 以及置信度。我们会通过置信度大小以及操作手需要识别的装甲板 id 进行筛选，若置信度过低或者不是操作手目标 id，则筛掉。如果该装甲板 ROI 被筛选掉，则会继续进行下一对的灯条筛选及配对，直到出现一个符合的装甲板 ROI。装甲板 ROI、识别 id 和置信度如 **图** 3.79所示。

图 3.79: 装甲板 ROI，id 及置信度效果图

对于设计的卷积神经网络框架，由于任务只是简单的 32*32 的数字识别。因此网络框架只是简单的由四层结构相同卷积模块和三层全连接层构成。对于全连接模块，其网络框架如 **图** 3.80(b)所示。对于卷积模块，其网络架构如 **图** 3.80(a)所示。前两层二维卷积核大小为 3，后两层大小为 5，卷积核个数依次为 16，32，64，128。激活函数都采用 Relu 函数，最大值池化，并且在激活函数后进行了批归一化处理。我们采用 openvino 对训练好



(a) 卷积模块示意图　　(b) 全连接层模块示意图

的模型进行部署，使用异步计算，由于模型参数较少，平均每次推算用时为 1ms。但是使用神经网络预测，对数据集的制作以及模型鲁棒性要求高，在某些极端情况下，仍会出现误识别。

3.4.1.3.6　获得装甲板位姿　到这里程序会读取相机的内参矩阵以及装甲板的实际尺寸大小，并且将装甲板 ROI 的四个点作为装甲板在图像坐标系的点进行 solvepnp，获得装甲板在相机坐标系下的位姿。并且将位姿通过自定义消息类型发送给电控，让云台进行跟踪。

3.4.2 能量机关视觉算法

3.4.2.1 算法库与接口说明

能量机关视觉算法计算待击打装甲板相对机器人的位姿，从而交由电控进行瞄准和击打。算法相关理论基于数字图像处理、计算几何和机器学习三大方面，并主要依赖**表** 3.1中开源库进行编写。

名称	所用模块	用途
OpenCV	imgproc	图像处理与几何抽象器
CGAL	Polygons 与 Arrangements	几何分析器
OpenCV	calib3d	三维重建器
OpenCV	ml	统计推理器

表 3.1: 视觉算法主要依赖

如赛季规划中所设计，算法流程图如 **图** 3.80所示。

3.4.2.2 原理阐述与公式推导

本节对 **图** 3.80中的关键部分进行解释。部分图中未给予枚举的逻辑在本节也会阐述。

3.4.2.2.1　色彩调制　色彩调制作为预处理部分最关键的项目，直接决定后续逻辑能否顺利进行。这里采用基于非极大值抑制的饱和度增强算法。此算法的另一个“副作用”就是实现灯条区域的明度归一化，这非常有利于解决各个自发光灯条由于亮度差异而造成的图像部分泛白情况。在普遍的算法实现中，应对泛白区域的手段往往是降低通道相减结果的决策阈值，而这往往是“按下葫芦浮起瓢”的根源，即噪声更加容易被误识别。对于原始的 BGR 输入我们首先进行下采样以减少后续计算复杂度。下采样是从当前图片中采样并获得行数和列数均缩小一半的输出。在 OpenCV 的实现[11]中其包含了高斯模糊和舍弃偶数行、列两个过程，可以较好地保留关键信

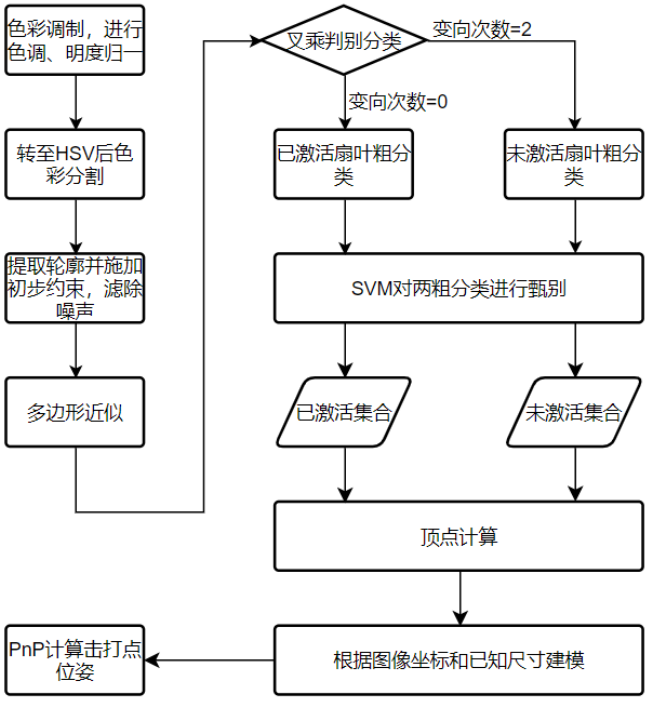


图 3.80: 能量机关识别算法流程图

息并同时大大减少数据量。然后对每个像素选择 RGB 三个通道中的最小值，将目标颜色通道减去该最小值并将另外两个通道置零。经过此部分处理将得到纯净的高饱和度“单通道”图像。最后，将图像还原为原始尺寸。

3.4.2.2.2 色彩分割 色彩调制所得结果虽然保持三通道彩色格式，但概念上是单通道、灰度的，因为它只有一个色彩通道有意义。仅仅依赖于单通道进行分割 (二值化) 只能在亮度 (像素值) 上施加约束。通过将图像转换至 HSV 色彩空间就可以在色调、饱和度、明度三个层次进行约束。定义：$R' = R/255$, $G' = G/255$, $B' = B/255$, $C_{\max} = \max(R', G', B')$, $C_{\min} = \min(R', G', B')$, $\Delta = C_{\max} - C_{\min}$, 则从 RGB 转换为 HSV 的转换公式如下：

$$H = \begin{cases} 0^\circ & ,\Delta = 0 \\ 60^\circ \times (\frac{G'-B'}{\Delta} + 0) & ,C_{\max} = R' \\ 60^\circ \times (\frac{B'-R'}{\Delta} + 2) & ,C_{\max} = G' \\ 60^\circ \times (\frac{R'-G'}{\Delta} + 4) & ,C_{\max} = B' \end{cases} \tag{3.43}$$

$$S = \begin{cases} 0 & ,C_{\max} = 0 \\ \frac{\Delta}{C_{\max}} & ,C_{\max} \neq 0 \end{cases} \tag{3.44}$$

$$V = C_{\max} \tag{3.45}$$

值得一提的是，OpenCV 的实现返回的结果在红色区存在分段，因此在处理红色目标时，必须使用`inRange()`函数分段处理，再使用`bitwise_or()`进行合并。因此我们首先将色彩调制的结果取反，再转换至 HSV，从而避免如上的弊端。对于蓝色和红色的目标，取反的结果分别是亮柠檬黄和亮青色，更加利于 H,S,V 三个分量高低阈值的选取。使用`inRange()`函数就可以得到高质量的二值图像。

3.4.2.2.3 轮廓筛选 首先在输入二值图上提取轮廓集合，然后进行逻辑判断。简称能量机关未被激活的扇叶为“锤子”，已激活全部亮起的扇叶为“扇子”。取各个外轮廓的最小外接矩形，通过限制最小面积和长宽比，可以

初步滤除非“锤子”和非“扇子”的部分。通过比较轮廓所围面积和最小外接矩形的面积，可以区分“锤子”和“扇子”。以上筛选过程不可以将被剔除的轮廓直接从容器中删除，因为它们中尚含有能量机关的中心“R”标志，这个组分在后续逻辑中可以继续发挥作用。因此，我们使用了标记机制，标记这些已经确认可能属于“锤子”或“扇子”的轮廓，后续算法在检测到标记后就直接跳过，从而避免新建容器存储一开始就被筛除的轮廓。

3.4.2.2.4 多边形近似 多边形近似，又称多边形逼近，旨在以尽可能少的点和尽可能高的精度获得对原始轮廓的高度概括，尤其是其本身就来源于规则几何图形时。多边形逼近属于抽稀算法，既减少数据量又利于后续算法设计。Douglas-Peukcer 算法[12]是经典的抽稀算法，其通过参数 ϵ 控制原始点与简化后的曲线间的最大距离，非常适合对凸轮廓进行近似或者说得到凸的近似结果。OpenCV 中很好的实现了此算法。CGAL 在其 2D Polyline Simplification[13]组件中同样提供了抽稀算法，在是否移除点的问题上其使用代价函数进行计算，从而得以给出更加贴近的近似结果；在停止条件上也提供了更精细的控制，例如可以设置最终保留的点的数量。CGAL 性能强大，在非凸的轮廓上可以得到更好的结果，和 OpenCV 所提供的函数优势互补。使用多边形近似旨在利用**叉乘判别法**进行特征分析。下文将叙述叉乘判别法的原理。

3.4.2.2.5 叉乘判别法 叉乘具有一个非常好的性质，就是不满足交换律。事实上其遵循如下性质：

$$\vec{a} \times \vec{b} = -\vec{b} \times \vec{a} \tag{3.46}$$

因此，可以利用叉乘来评价多边形的凹凸性。设 2D 多边形顶点按曲线正向组成有序序列 A，将 A 中点的坐标两两相减将得到一有序向量序列 B。将 B 中向量两两叉乘，将得到一实数序列 C。根据 C 中负数的个数或者说变号的次数就可以得到该多边形凹凸性的扼要描述。凸多边形对应为 0 个，而凹多边形有几个“凹”就有几次变号。此方法方便快捷，避免了在长宽比等约束的精细参数上煞费苦心，和顶点个数等其他简单约束相组合可以形成简练而充分的约束，有力排除干扰。显然，使用六边形近似，“扇子”的近似结果是凸的；“锤子”具有八个顶点，近似结果应当是对应变号次数为 2 的凹多边形，两次变号来源于“锤头”和“锤柄”的交界处。进行拟合后使用叉乘判别法，就可以选择出高置信度的“锤子”和“扇子轮廓”，并将它们装入额外的容器，随后就可以标记它们所对应的原始轮廓。“锤头”和“锤柄”的交界处所对应的拟合所得多边形的两个顶点会被专门记录用于后续使用，并传送到分析阶段，使得“分析器”可以直接裁切装甲板部分 (即“锤头”) 进行处理而不用理会其他部分。

3.4.2.2.6 分类器 对于叉乘判别法的结果，依然可以举出反例，例如沙漏形的轮廓可能会成为漏网之鱼。尽管实际测试中这样的情况几乎从未发生，提供一个可选的分类器环节未尝不可。我们选择 One Class SVM 或朴素贝叶斯作为分类器。前者适合于负样本难以采集的情形，而后者在速度上颇具优势，适合特征之间条件独立的情形和小规模的样本数量。使用多边形近似阶段的结果 (或稍作处理后) 作为分类器的输入，进一步利用了其简练的优势。One Class SVM（OCSVM）是基于 SVM 的一类分类。它依赖于识别由所有数据点组成的最小超球面（半径为 r，中心为 c）。这种方法又称为支持向量数据描述（SVDD）。OCSVM 适合于只有正样本数据或其他样本难以采集的情形，以确定输入是否隶属于当前类为目标。形式上，问题可以定义为以下约束优化形式:

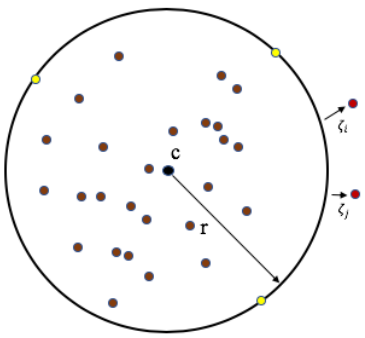


图 3.81: OCSVM 示意图

$$\min_{r,c} r^2 \text{ subject to, } \|\Phi(x_i) - c\|^2 \leq r^2 \quad \forall i = 1, 2, \ldots, n \tag{3.47}$$

然而，上述公式具有高度限制性，并且对异常值的存在很敏感。因此，允许存在异常值的公式如下所示:

$$\min r^2 + \frac{1}{\text{rg},} \sum_{i=1}^{n} \zeta_i \tag{3.48}$$

$$\text{subject to, } \|\Phi(x_i) - c\|^2 \leq r^2 + \zeta_i \forall i = 1, 2, \ldots, n \tag{3.49}$$

从 Karush-Kuhn-Tucker (KKT) 最优条件，我们得到:

$$c = \sum_{i=1}^{n} \alpha_i \Phi(x_i) \tag{3.50}$$

其中 α_i 是以下优化问题的解:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i \kappa(x_i, x_i) - \sum_{i,j=1}^{n} \alpha_i \alpha_j \kappa(x_i, x_j) \tag{3.51}$$

$$\sum_{i=1}^{n} \alpha_i = 1 \text{ and } 0 \leq \alpha_i \leq \frac{1}{m} \text{ for all } i = 1, 2, \ldots, n \tag{3.52}$$

3.4.2.2.7 主成分分析 一个很重要的需求就是得到“锤子”的中轴线的表达式。为保证结果准确，必须使用主成分分析 (PCA) 来实现，因为先前的最小外接矩形很可能存在歪斜。总之，我们想得到如 **图** 3.82所示的结果。下面推导一种 PCA 的实现，即基于特征值分解协方差矩阵的 PCA 算法，而这也正是 OpenCV 所使用的。

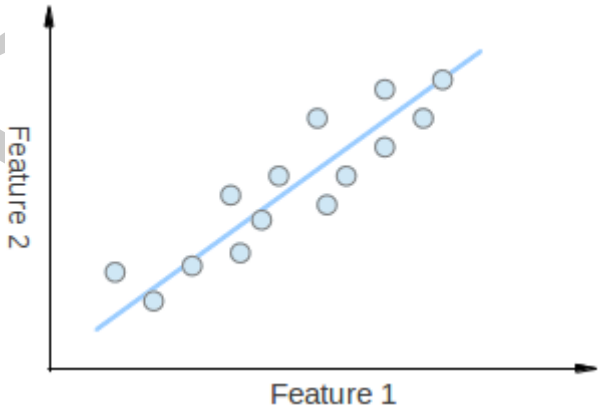


图 3.82: PCA 示意图

设有 m 条 n 维原始数据，将其按列组成 m 行 n 列矩阵 $X_{m\times n}$，然后将 X 的每一列进行零均值化，即减去这一列的均值，得到 B:

$$U_{1\times n}[j] = \frac{1}{m} \sum_{i=1}^{m} X[i, j] \tag{3.53}$$

$$B = X - \mathbf{1}_{m\times 1} * U \tag{3.54}$$

求出协方差矩阵 C：

$$C_{n\times n} = \frac{1}{m-1} B^T * B \tag{3.55}$$

求出协方差矩阵的特征值及对应的特征向量：显然 C 是对称矩阵，因此即求其对角化：

$$Q_{n\times n}^T C_{n\times n} Q_{n\times n} = \Lambda \tag{3.56}$$

至此推导完毕，我们已经获得了最为关键的特征值及特征向量。对于我们的需求情形，取最大的特征值对应的特征向量和点集的中心 (平均值)，即得到所求直线的点向式，如 **图** 3.83所形容。

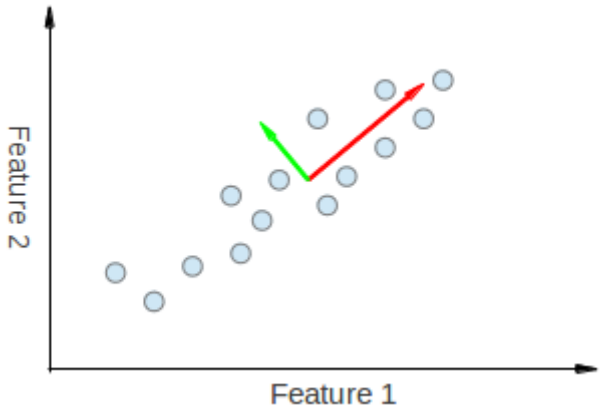


图 3.83: 特征向量示意图

3.4.2.2.8 几何光学先验 这里仅叙述原理，设计目的和应用场景及其他细节考量会在优化方案中详细叙述。由高斯公式：

$$\frac{1}{l'} + \frac{1}{l} = \frac{1}{f} \quad (l', l, f > 0) \tag{3.57}$$

其中 l 是物距，l' 是像距，f 是镜头焦距。得垂轴放大率：

$$\beta = \frac{l'}{l} \tag{3.58}$$

设传感器 (CMOS) 上每个 cell 的尺寸为 $\mu \times \mu$，物体实际尺寸为 S，得图像上物体的尺寸 s 为：

$$s = \frac{S * \beta}{\mu} \tag{3.59}$$

s 的物理意义是在 l 的物距下成像，物体在图像上能占据的最大尺寸 (此时物体垂直光轴)。

3.4.2.3 算法性能、优缺点分析

能量机关视觉算法同样建立在 ROS 框架之上，以 Topic 形式接收和发送图像和位姿等数据，主要有预处理、识别和数据准备、几何分析和位姿解算三个阶段。将以上流程写在一个回调函数之中势必颇有“堵塞”之感—在全体逻辑结束之前相机发出的数据将被完全忽略，而综合以上分析，能量机关视觉算法显然具有一定复杂度。因此我们采取多节点的形式，使用三个节点分别执行图像预处理、识别和数据准备、几何分析和位姿解算三个部分。三个节点因此可以独立运行，在概念上实现“流水线式”并行，在实测中有效提升了性能。

将一个前后相关的算法拆分为多节点的副作用就是使得中间数据的传递变为一件不再那么简单的事情，数据同步上也需要更加谨慎。此外，若随着算法的改进需要加入诸如反馈等机制，必须增加额外的通信。

3.4.2.4 优化方案

考虑到图像作为中间数据，其传输会造成很大开销，我们使用 nodelet 来实现每个节点，实现图像的零拷贝传输。合理分配节点负责的工作也是优化的一环，我们希望它们“负载均衡”，避免避免比较复杂的识别部分过于庞大。

因此我们选择将识别阶段所得的“锤子”和“扇子”ROI 打包发送给负责分析和解算的节点，其中各个 ROI 通过`warpAffine()`函数计算返回。我们使用一个大的 ROI(简称“全局 ROI”) 来承载这些“锤子”和“扇子”ROI(简称“小 ROI”)，然后将其发送给分析阶段进行顶点计算和位姿求解。

为了避免因为每次“小 ROI”的尺寸变化使得“全局 ROI”需要重新分配内存而不能直接往其写入数据，我们引入几何光学先验，预先计算“小 ROI”的最大尺寸，从而为“全局 ROI”分配一个固定的大小，保证最坏情况下也不用重新分配内存。

在原理阐述章节的叉乘判别法部分，我们已经指出了可以计算得到“锤头”和“锤柄”的交界从而裁切装甲板部分。一个问题就是为什么不直接把装甲板部分裁切下来发送给分析阶段。

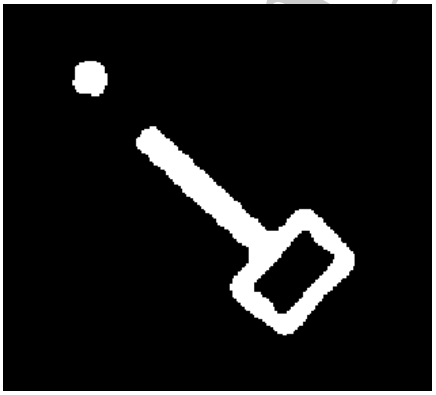
一方面，ROI 的裁切不能保证 100% 的准确性，这是由于识别过程中参数计算可能错误 (存在一定误差)。预设裁切结果永远正确过于冒进，更重要的是，即使裁切错误，只要其偏离程度不太大，装甲板部分就依然能在分析阶段被正确识别，依靠记录下的反变换矩阵信息就依然能成功得到装甲板顶点在原始图像下的图像坐标；而依赖先验的装甲板尺寸强行裁切装甲板部分则很有可能直接切在装甲板之上，导致后续逻辑全部失效。归根结底，就是因为几乎不可能保证裁切装甲板的精度可以对抗识别过程中的浮动误差，而将整个扇叶 ROI 切下本质上是留了足够大且不盲目大的合适的裕量。

另一方面，识别过程得到的一些信息需要传递给分析阶段，例如仿射变换的逆，“小 ROI”的实际尺寸等等，将它们直接印在 ROI 上立即就满足了同步的需求，而这恰好需要一点额外的空间。

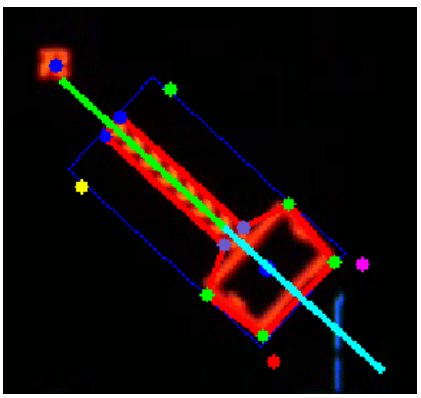
最后，图像传输是通过 nodelet 实现零拷贝的，性能上不会有太大损失。

3.4.2.5 算法结果

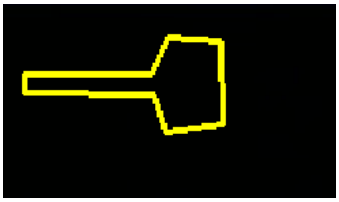
图 3.84分别展示了二值化结果、PCA 结果、多边形近似结果 (校正为水平) 以及 ROI 裁切结果 (校正为水平，左上角的一些不规则彩色像素就是记录的反变换以及实际尺寸等信息)。



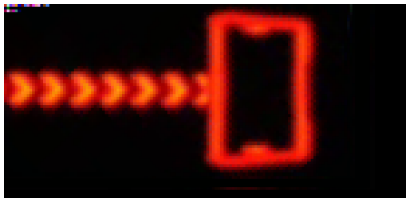
(a) 二值化结果



(b) PCA 结果



(c) 多边形近似结果



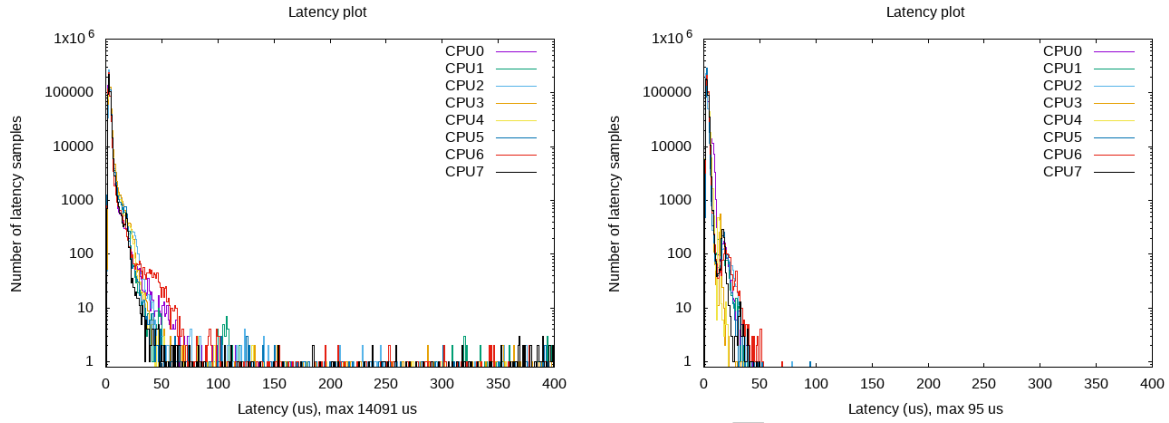
(d) ROI

图 3.84: 能量机关识别算法结果

3.5 其它

3.5.1 实时性

我们为 Linux 内核打上 PREEMPT_RT[14] 补丁。如 **图** 3.85 所示，我们使用了`cyclictest` 进行了 500000 个循环的测试，打补丁后的最大延迟有三个数量级的下降，且频次更加靠左（延迟更小），更加集中，可见实时性有了较大的提升，经过 CAN 回环和实车等测试，满足 RM 需求。



(a) 未打实时补丁　　(b) 已打实时补丁

图 3.85: 在 Intel NUC 上在压力下的实时性测试结果，横轴为延迟大小，纵轴为频次

3.5.2 自定义 UI

3.5.2.1 界面展示

以准备阶段的操作手界面为例，如 **图** 3.86所示。

图 3.86: 英雄操作手自定义 UI 界面

3.5.2.2 功能说明

整个自定义 UI 分为四大部分，分别用于显示操作模式，超级电容电量，射击辅助线，以及警告信息。

3.5.2.2.1　操作模式　位于操作手界面**左上方**，共显示 3 条信息，分别是 **CHASSIS**，**TARGET**，**EXPOSURE**。

- **CHASSIS** 用于显示机器人的底盘和超级电容的模式。以提醒操作手当前底盘的工作状态。

- 内容：显示底盘的模式。对于英雄机器人来说，底盘常用到的模式有**跟随模式（FOLLOW），小陀螺模式（GYRO）和扭腰模式（TWIST）**三种；
- 颜色：显示超级电容的工作状态。对于英雄机器人来说，超级电容共有**充电模式（绿色），放电模式（橙色），普通模式（黄色）和初始模式（白色）**四种。

- **TARGET** 用于显示机器人的视觉识别目标，以提醒操作手当前自瞄的对象。
  - 内容：显示视觉识别的装甲板类型。对于英雄机器人来说，视觉识别的类型有**全部装甲板（ALL）、基地或前哨站装甲板（BASE）**两种；
  - 颜色：显示视觉识别的装甲板颜色。有**红色（红色）、基地蓝色（蓝色）**两种；
- **EXPOSURE** 用于显示机器人的视觉曝光等级。以方便操作手调整视觉曝光度。
  - 内容：显示视觉曝光等级。有 **0** **4** 共 5 个等级；
  - 颜色：无特殊含义。

3.5.2.2.2 超级电容电量 位于操作手界面**正下方**，显示超级电容剩余电量。

- 内容：超级电容的**剩余电量占总电量的百分比**。
- 颜色：当剩余电量低于 30% 时，为**橙色**；剩余电量大于 60% 时为**绿色**；其余情况为**黄色**。

3.5.2.2.3 警告信息 位于操作手界面**正上方**，共显示 2 条信息，分别是 **SPIN**，**ARMOR**。

- **SPIN** 机器人没有进入小陀螺模式时，以闪烁的形式提醒操作手。
  - 内容：**"PLEASE SPIN!!"**；
  - 颜色：无特殊含义。
- **ARMOR** 装甲板受击时，以闪烁的形式提醒操作手受击方向。
  - 内容：在受击方向显示一个圆形，在一定时间后消失；
  - 颜色：无特殊含义。

3.5.2.3 配置文件说明

在机器人的配置文件中添加需要显示的信息，设置显示位置及颜色，代码就会自动读取机器人的对应信息，转换为对应字符串和颜色显示在自定义 UI 界面中。

裁判系统的上行频率有限，因此为了不出现串口堵塞的问题，我们将整个配置文件分为`trigger_change`，`time_change`,`fixed`,`flash`四部分，分别用于显示不同类型的 UI。UI 配置文件如下所示。

Listing 3.6: hero.yaml

```
rm_manual:
  ui:
    trigger_change:
      - name: "chassis"
        config: { start_position: [ 400, 750 ], size: 15, width: 2, title: "chassis: " }
      - name: "target"
        config: { start_position: [ 400, 700 ], size: 15, width: 2, title: "target: " }
      - name: "exposure"
        config: { start_position: [ 400, 650 ], size: 15, width: 2, title: "exposure:" }
    time_change:
      - name: "capacitor"
        config: { start_position: [ 900, 100 ], size: 15, width: 2, delay: 0.5 }
    fixed:
      - name: "4.5m_str"
        config: { type: "string", size: 10, width: 1, color: "yellow",
```

```
            start_position: [ 1000, 375 ], content: "4.5" }
  - name: "5.3m_str"
    config: { type: "string", size: 10, width: 1, color: "yellow",
            start_position: [ 1000, 393 ], content: "5.3m" }
  - name: "5.3m_line"
    config: { type: "line", width: 2, color: "orange",
            start_position: [ 949, 391 ], end_position: [ 971, 391 ] }
  - name: "mid"
    config: { type: "line", width: 2, color: "yellow",
            start_position: [ 960, 302 ], end_position: [ 960, 540 ] }
flash:
  - name: "spin"
    config: { start_position: [ 900, 750 ], size: 15, width: 2,
            color: "yellow", content: "please spin!!", delay: 0.8 }
  - name: "armor0"
    config: { type: "circle", width: 3, radius: 50, color: "yellow", delay: 0.75 }
  - name: "armor1"
    config: { type: "circle", width: 3, radius: 50, color: "yellow", delay: 0.75 }
  - name: "armor2"
    config: { type: "circle", width: 3, radius: 50, color: "yellow", delay: 0.75 }
  - name: "armor3"
    config: { type: "circle", width: 3, radius: 50, color: "yellow", delay: 0.75 }
```

3.5.2.3.1 trigger_change 该类 UI 只在触发机器人状态变化时刷新，在英雄中对应上文提到的**操作模式** UI。下面对参数作出解释：

- name：用于实现程序内部索引，一般不作更改；
- config：UI 的各项配置参数。
  - start_position：字符串显示的位置；
  - size：字体大小；
  - width：字体线条宽度；
  - title：字符串标题；

3.5.2.3.2 time_change 该类 UI 按固定频率刷新，在英雄中对应上文提到的**超级电容电量** UI。下面对参数作出解释：

- name：用于实现程序内部索引，一般不作更改；
- config：UI 的各项配置参数。
  - type：UI 的类型，常用的有字符串 string 和直线 line，可选类型详见裁判系统串口协议；
  - start_position：对于字符串来说代表显示位置，对于线段来说代表线段起始点的坐标，详见裁判系统串口协议；
  - end_position：对于线段来说代表线段终止点的坐标；
  - color：字体颜色，可选颜色详见裁判系统串口协议；
  - width：线条宽度；
  - delay：刷新的时间间隔；

3.5.2.3.3 fixed 该类 UI 只向裁判系统发送一次数据包，不进行额外的刷新操作，在英雄中对应上文提到的**射击辅助线** UI。下面对参数作出解释：

- name：用于实现程序内部索引，一般不作更改；
- config：UI 的各项配置参数。
  - start_position：字符串显示的位置；
  - size：字体大小；
  - color：线条颜色；
  - width：线条宽度；

3.5.2.3.4 flash 该类 UI 按指定的时间间隔以闪烁形式刷新，在英雄中对应上文提到的**警告信息** UI。下面对参数作出解释：

- name：用于实现程序内部索引，一般不作更改；
- config：UI 的各项配置参数。
  - type：UI 的类型，常用的有字符串 string 和圆形 circle，可选类型详见裁判系统串口协议；
  - start_position：对于字符串来说代表显示位置，对于圆来说代表圆心坐标，详见裁判系统串口协议；
  - color：线条颜色，可选颜色详见裁判系统串口协议；
  - width：线条宽度；
  - delay：闪烁频率；
  - content：警告内容；
  - radius：仅当 UI 类型为圆形时生效，为圆的半径；

3.5.3 通过裁判系统数据的读取对机器人控制的优化

3.5.3.1 功能说明

控制程序通过串口读取裁判系统数据使得机器人的各模块参数与比赛时的性能体系或是时间节点相匹配，或是通过裁判系统的数据对超级电容中的电流采样模块所进行校准。

3.5.3.2 各模块参数与性能体系的匹配

- **底盘模块**：比赛阶段，决策层收到裁判系统中关于底盘上限功率的数据后，将数据发送到 chassis controller 中，chassis controller 根据上限功率算出底盘电机的最大输出力矩，并作为 PID 的输入，从而达到功率限制的目的；
- **发射模块**：比赛阶段，决策层收到裁判系统中关于 42mm 弹丸初速度上限的数据后，将指令发送到 shooter controller 中，shooter controller 根据 42mm 弹丸初速度上限读取配置文件中对应的摩擦轮转速，并作为 PID 的输入，从而保证 42mm 弹丸初速度不超过上限。
- **视觉模块**：比赛阶段，视觉模块根据裁判系统中的敌方颜色，设定识别灯条的颜色。

3.5.3.3 校准超级电容的电流采样模块

三分钟准备阶段，超级电容的主控开始大功率充电，同时通过裁判系统数据中的底盘实时功率，对电流采样模块反馈的数据进行校准，保证比赛阶段电流采样模块反馈的数据的准确性。

3.5.4 调试工具

rqt 是一个基于 Qt 的框架，用于 ROS 的 GUI 开发。本团队利用各种插件组合成自定义界面，写成配置文件后日常调试中使用。下面对该自定义配置界面进行介绍。

3.5.4.1 Home

本界面如 **图** 3.87所示。界面左侧用于显示机器人的姿态，通过 rviz 插件，在日常调试中常用于观察机器人的 tf 坐标系以及机器人模型是否正确。界面右侧通过订阅相应话题，可以实时观察从摄像头传回的图像。右下方加入了动态调参的插件，便于调整参数。在日常调试中常用于调试视觉参数。

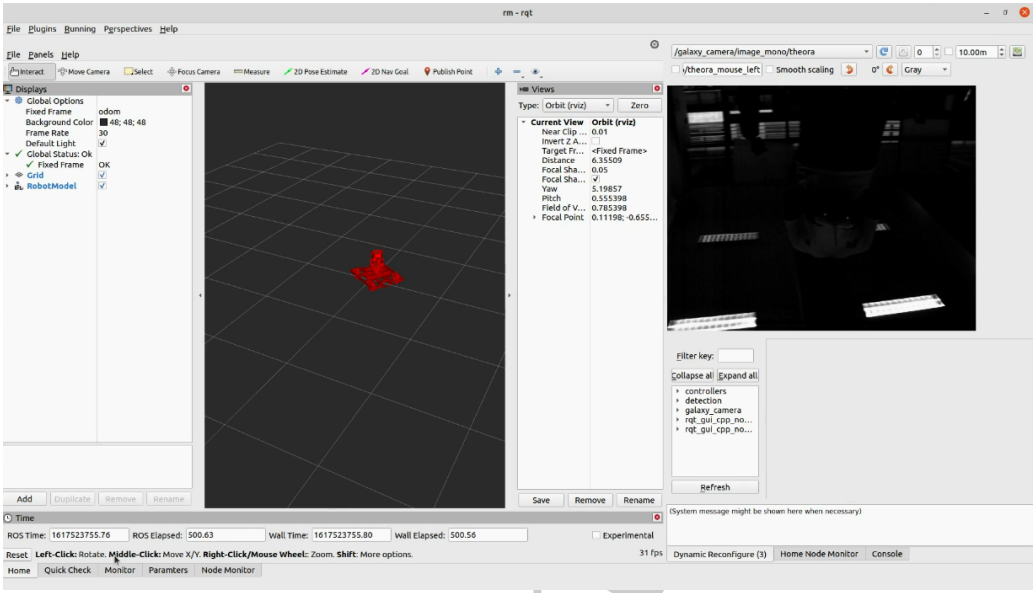


图 3.87: Home 界面

3.5.4.2 Quick Check

本界面用于快速检查代码运行状态，如 **图** 3.88所示。界面左侧显示各个节点的 CPU 和内存占用情况，以及输出的各项信息。界面右侧可以看到整个 tf 树的结构以及各个节点之间的通讯关系。在日常调试中通常用于观察输出的调试信息或者检查某个节点有没有在运行。

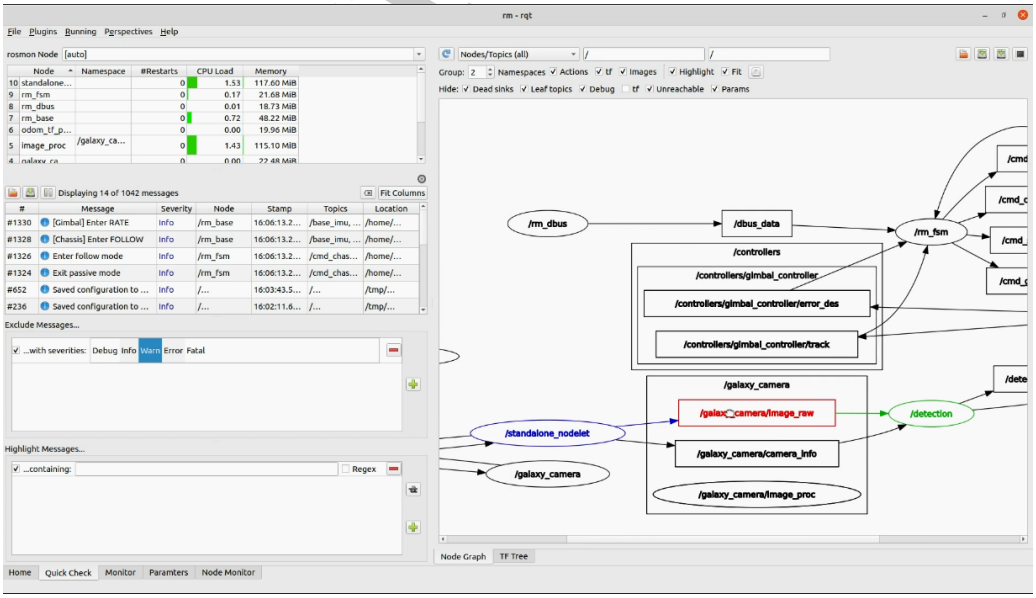
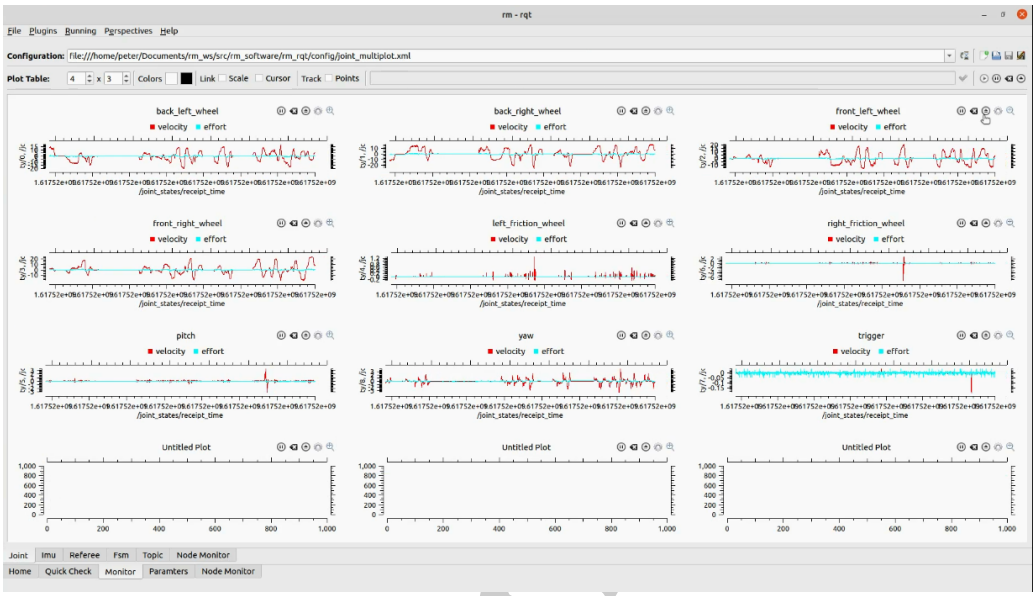


图 3.88: Quick Check 界面

3.5.4.3 Monitor

本界面显示机器人读取到的各项信息。分为 4 个子界面。分别是 **Joint, Imu, Referee, Topic**

3.5.4.3.1 Joint 本界面用于显示电机的速度和力矩，如 **图** 3.89所示。并通过 rqt_multiplot 插件绘制出电机的速度或力矩图像。在使用过程中，不仅可以对单个电机的数据进行清空或记录，也可以对全部电机的数据同时清空或记录，还可以单独放大某个电机的图像，方便观察。



(a) 多电机

(b) 单电机

图 3.89: Joint 界面

3.5.4.3.2 Imu 本界面用于显示 IMU 返回的数据，同样也是通过 rqt_multiplot 插件绘制出 IMU 返回的姿态信息。功能与 Joint 界面类似，此处不再重复说明。

3.5.4.3.3 Referee 本界面用于显示裁判系统的各项信息，如**图** 3.90所示。同样也是通过 rqt_multiplot 插件绘制出裁判系统数据图像。在实际调试中常用于调试底盘功率限制或枪口热量限制。

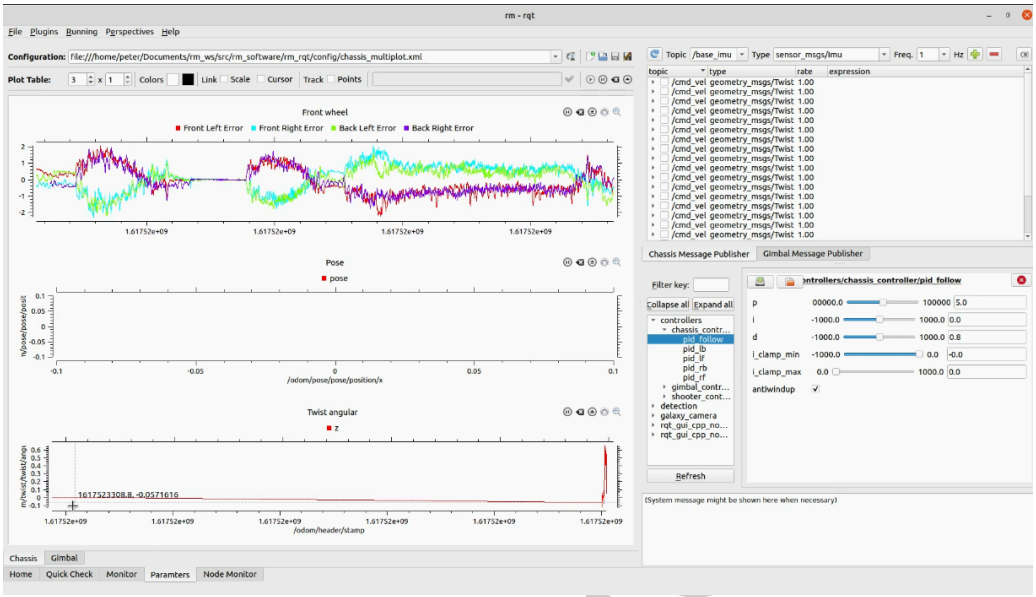


图 3.90: Referee 界面

3.5.4.3.4 Topic 本界面用于显示机器人运行时所有话题下的消息，如**图** 3.91所示。通过勾选相应话题即可读取到该话题上消息的发送频率、消息的值以及消息类型等等。在实际调试中常用于判断问题出现在哪个包。

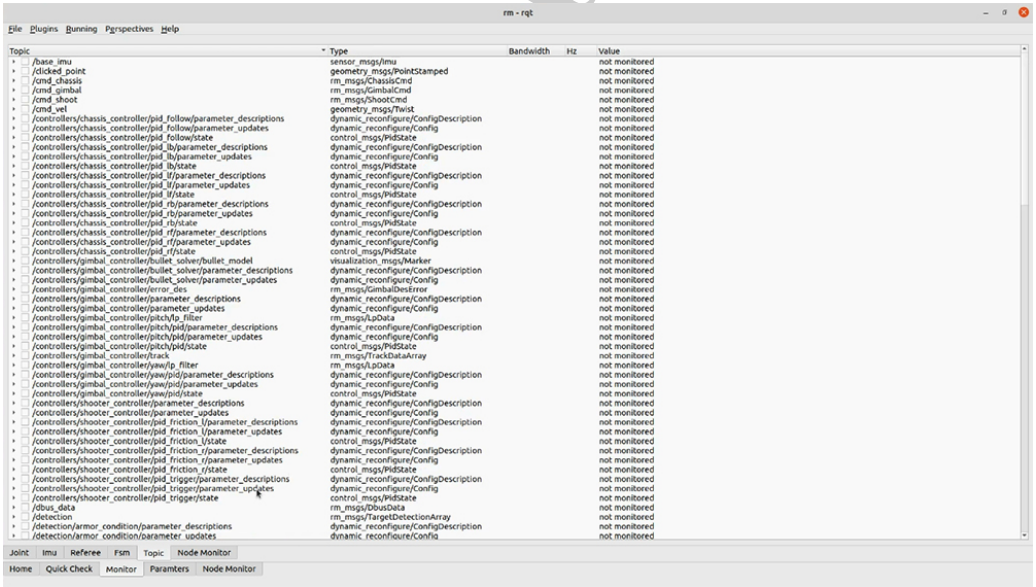


图 3.91: Topic 界面

3.5.4.4 Parameters

本界面用于调试控制器参数。分为 2 个子界面。分别是 **Chassis**，**Gimbal**

3.5.4.4.1 Chassis 本页面用于调试底盘控制器的参数，如 **图** 3.92所示。在界面左侧可以观察到底盘电机的各项数据，如误差、姿态等，也可以手动添加其他电机信息。界面右侧的上方可以直接向底盘订阅的话题发送命令消息；下方则可以对底盘 PID 参数进行动态调整，方便调试。

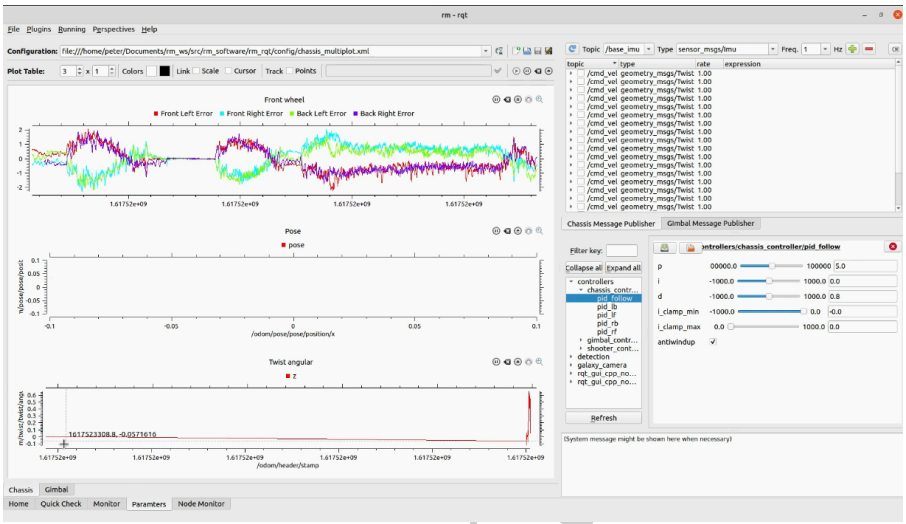


图 3.92: Chassis 界面

3.5.4.4.2 Gimbal 本页面用于调试云台控制器的参数，如 **图** 3.93所示。具体功能与 **Chassis** 界面类似，在此不再重复说明。

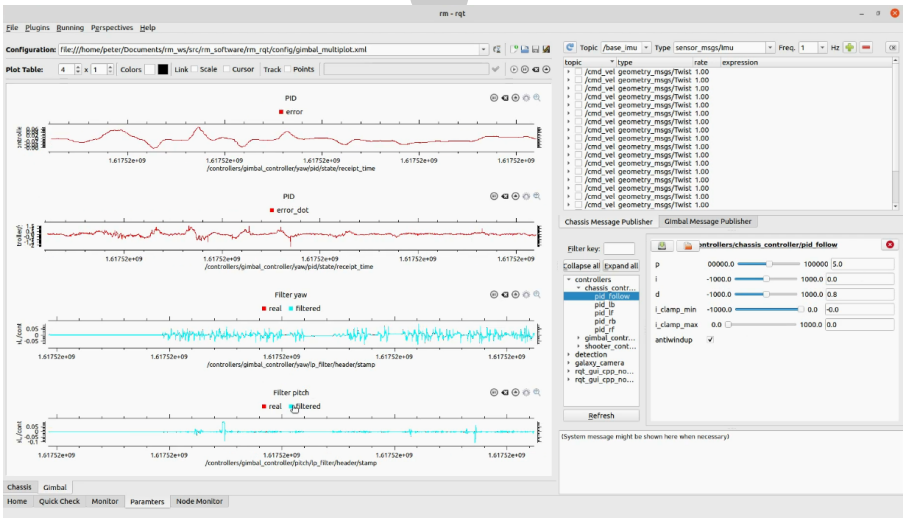


图 3.93: Gimbal 界面

3.5.4.4.3 Shooter 本页面用于调试发射控制器的参数。具体功能与 **Chassis** 界面类似，在此不再重复说明。

3.5.5 手眼标定与联合标定

3.5.5.1 手眼标定

手眼标定是指标定相机坐标系与机器人坐标系的转换关系，可分为 Eye-in-Hand 和 Eye-to-Hand。在 RoboMaster 中，需要确定相机与云台的精确的相对位置关系，这样才能提高自瞄的准确度。如果相机坐标系与机器人坐标系的转换关系不准确，那么程序对目标在世界坐标系的位姿估计实际上是不准确的。在 urdf 中，相机坐标系与机器人坐标系的转换关系是根据图纸中的数据来确定的，但由于加工与装配的误差，urdf 中的这一转换关系与实际机器人中的存在误差。为了精确确定相机坐标系与云台坐标系的转换矩阵，我们使用第三方的 ROS 软件包 easy_handeye[15] 来进行手眼标定。

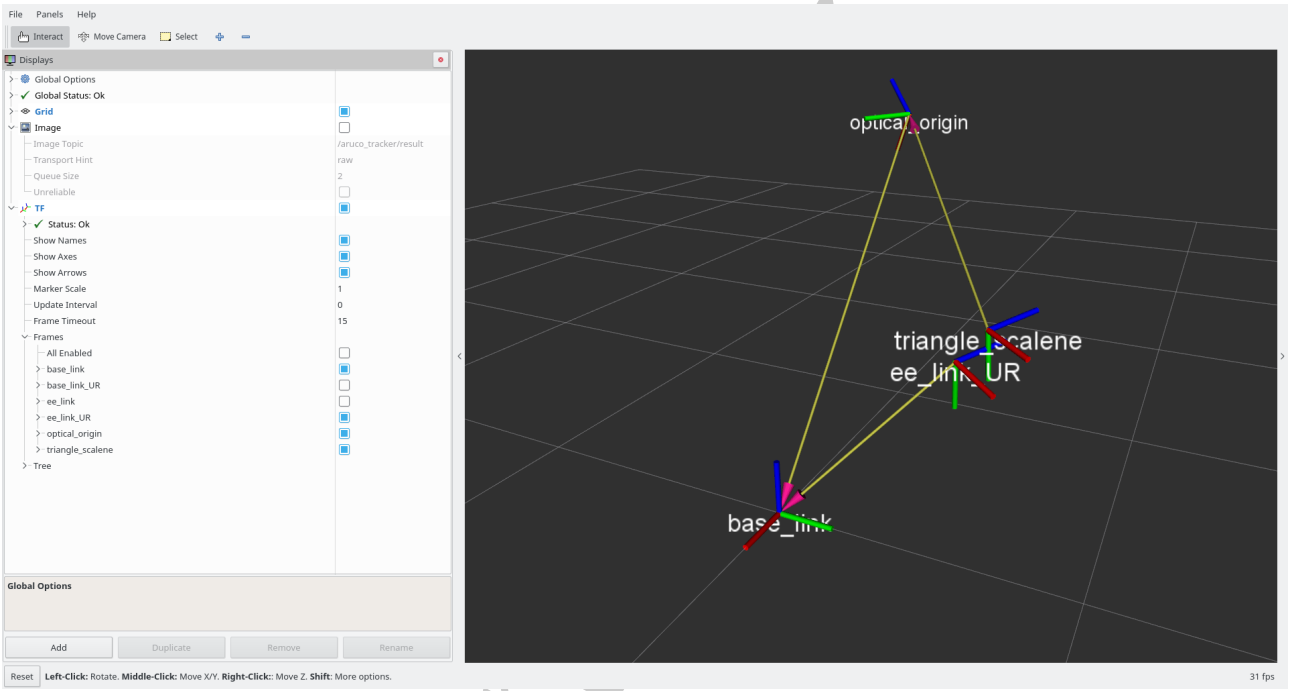


图 3.94: 使用 easy_handeye 进行手眼标定

3.5.5.2 联合标定

联合标定是指标定相机坐标系与 imu 坐标系的转换关系，进行联合标定的目的同样是为了更精确地估计目标在世界坐标系的位姿，以及云台在世界坐标系的姿态。正如手眼标定中所提到的原因，urdf 中相机坐标系与 imu 坐标系的转换与实际机器人中的存在误差。借助 imu 反馈数据，可以得出 imu 坐标系与世界坐标系的相对姿态关系。在进行联合标定后，我们可以通过坐标系变换来得到相机在世界坐标系的姿态。如果又进行了手眼标定，那么就能精确得到相机在世界坐标系的位姿，同时也可更精确地估计目标在世界坐标系的位姿。

我们使用开源软件包 kalibr[16] 很方便地进行相机与 imu 的联合标定。

第 4 章 研发迭代过程

4.1 版本迭代过程记录

版本号	功能详细说明	完成时间
V0.5	模块化方案测试：舵轮救援结构测试、下供弹链测试、中心供弹拨盘测试、自制 6020 一体式滑环测试、z 轴云台简易版测试	2021.11.09
V1.0	第 1 代底盘设计：经过对比舵轮、麦轮与全向轮底盘的优劣性，最后选择使用全向轮 X 型分布底盘，再结合自适应悬挂、圆形保护外框、下供弹链、中心供弹拨盘、一体式滑环等结构设计与加工装配	2022.01.07
V1.5	第 1 代云台发射设计：下供中部弹链、板堆叠枪管等结构设计及加工装配	2021.01.23
V2.0	第 2 代底盘设计：调整底盘高度、自制滑环的普通滑环的替换、3508 电机与轮毂电机的替换	2022.02.25
V2.5	第 2 代云台发射设计：发射枪管改用一体式、优化云台 pitch 轴各模块布局、pitch 轴使用自制减速箱	2022.03.20

表 4.1: 版本迭代过程记录

4.2 重点问题解决记录

问题描述	问题产生原因	问题解决方案 & 实际解决效果	机器人版本号	解决人员
中心供弹拨盘易发生形变结构脱落	因为尝试使用的 u 型轴承（本着减轻重量的目的）时表现出明显刚性不足	用单个薄壁深沟球轴承替换多个 U 形轴承，拨盘刚度大大提高且重量增加不大	V0.2	（机械）张楚杰
中心供弹拨盘高速旋转时同步带出现滑齿现象	同步带选型错误、同时包角不足	将拨盘更换为大模数玻纤齿轮传动，解决了滑齿的问题。	V0.3	（机械）庾日熙
弹丸在分层片位置卡住	分层片高度设计略高，导致弹丸在拨盘高频拨动时被顶起	降低分层片固定高度，同时调整分层片外形。	V0.4	（机械）杨浩晖
步兵底盘 RFID 被地面大弹丸卡住	车体高度过低，导致在经过大弹丸时被卡住	改变悬挂初始角度，提高底盘高度；同时调整 RFID 安装板，用沉头螺丝替代垫高柱。步兵底盘最终为 48mm，高于大弹丸。	V2.0	（机械）张楚杰

接下页

问题描述	问题产生原因	问题解决方案 & 实际解决效果	机器人版本号	解决人员
自制 6020 一体式滑环	良品率低且占据中部空间较大会影响拨盘上方弹丸预留位置	图纸上修改 yaw 轴布局，使得其结构可兼容自制滑环和外购标准滑环，实现若出现损坏也可相互替换，同时拨盘上方空间得到一定程度的释放	V2.0	（机械）张楚杰
步兵自瞄摄像头安装不合理	云台架高度过大，导致摄像头需要安装在图传侧面；摄像头固定在理线盒上，导致调整云台线路要重新标定摄像头	降低云台架高度，将摄像头放置在图传上方，独立于理线盒。更换电调、中心板等不需要重新标定摄像头。	V2.5	（机械）张楚杰
步兵弹链滑槽形状不合理	设计经验不足	重新设计滑槽外形，均匀选取滑块会经过的四个点，将点连线拟合滑槽轨迹。发射机构在 30 度俯角至 50 度仰角均可以流畅发弹。	V2.5	（机械）张楚杰
V1.5 版本下供云台 pitch 轴转动惯量大，与 21 年上供弹步兵云台 pitch 轴转动惯量相近	GM6020 放在 pitch 轴上用于配平	将 pitch 轴驱动电机置于云台架上，并用重力补偿机构配平云台。新版云台 pitch 轴转动惯量约为 21 年上供弹步兵云台的二分之一	V2.5	（机械）张楚杰
V1.5 版本发射机构以 20m/s 的速度发射 100 发弹丸，会出现较多发弹丸左右偏离弹道	板堆叠枪管的加工精度低及装配零件较多	将板堆叠式枪管改为一体式枪管设计，同时改变弹丸定心方式	V2.5	（机械）张楚杰
V2.5 版本发射机构发射弹丸后出现磨蹭枪管内上壁问题	安装摩擦轮电机的高度或定心位置有问题，导致弹丸在摩擦轮内受挤压发射时运动速度不与摩擦轮安装平面垂直	调整摩擦轮安装高度和定心片的位置	V2.5	（机械）张楚杰
全图装甲板识别帧率约为 90Hz, 有很大的提升空间	识别代码话题之间的图像通信存在多次拷贝现象，导致帧率降低	添加 nodelet，以实现节点之间的图像零拷贝。添加 nodelet 后帧率提升为 120Hz	V2.5	（视觉）何毅城
imu 与相机的时间戳实际上不同步	can 总线读取多个 imu 触发时，只有一个被上层控制器读取	将 imu 的滤波和数据处理下放到硬件抽象层，解决了这个问题	V2.5	（控制）叶振羽

表 4.2: 重点问题解决记录

第 5 章 团队成员贡献

姓名	基本信息（专业、年级、队内角色）	主要负责工作内容描述	贡献度（每组单独按 100% 计算，共 5 组）
张楚杰	机械设计制造及其自动化、大三、机械组组长	整体机器人的底盘、云台、发射结构设计等	（机械）50%
杨浩晖	机械电子工程、大二、机械组成员	云台和发射 1 代版本结构设计、整车加工装配等	（机械）30%
冯军华	机械电子工程、大二、机械组成员	整车加工装配	（机械）10%
庾日熙	机械设计制造及其自动化、大二、机械组成员	拨盘结构迭代、z 轴云台简易测试架等设计及其加工装配测试	（机械）10%
陈麒铵	自动化类、大二、电路组组长	超级电容主控模块与惯性测量单元模块设计	（电路）20%
曾文俊	集成电路设计和电子设计自动化 IC 班、大三、电路组成员	超级电容设计迭代及整体电路模块方案设计	（电路）20%
麦振博	微电子科学与工程、大二、电路组成员	整体机器人的电路连接、NUC 降压模块设计	（电路）15%
陈梓豪	电子科学与技术、大三、电路组成员	超级电容升压模块的设计与调试、荧光充能装置设计	（电路）15%
韦浩亮	信息工程、大二、电路组成员	超级电容模组的设计与调试	（电路）15%
侯昱帆	光电信息科学与工程、大三、电路组成员	超级电容充电模块的设计与调试	（电路）15%
何毅城	自动化、大三、视觉组组长	装甲板识别算法设计和部署	（视觉）50%
沈佳辰	光电信息科学与工程 IC 班、大三、视觉组成员	能量机关视觉算法设计	（视觉）50%
廖洽源	机械电子工程（创新班）、大三、控制组组长	指导控制组成员进行调试测试，检查代码错误，优化代码等	（控制）50%
叶振羽	自动化、大二、控制组成员	步兵程序的调试与优化，全向轮底盘的运动学及代码实现，自瞄系统的设计与实现	（控制）50%
梁伟聪	机械电子工程、大三、机械组成员及队长	负责队内各兵种开发方案研讨、进度安排等，统筹把握各兵种的开发任务	（管理）50%
李钦鹏	机械电子工程（创新班）、大三、项目管理	负责队内兵种机器人的技术进度汇报及管理，催促兵种正常进度开展及测试兵种方案执行	（管理）35%
李翰霆	电子科学与技术、大二、项目管理	负责队内兵种机器人的进度 DDL 催促及协助兵种各形态测试文档撰写	（管理）15%

表 5.1: 团队成员贡献

参考文献

[1] YUN S H, SEO J, YOON J, et al. 3-dof gravity compensation mechanism for robot waists with the variations of center of mass[C]//2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). [S.l. : s.n.], 2019: 3565-3570.

[2] Rm-controls[Z]. https://github.com/rm-controls/. (Accessed on 08/15/2021).

[3] CHITTA S, MARDER-EPPSTEIN E, MEEUSSEN W, et al. Ros_control: A generic and simple control framework for ROS[J/OL]. The Journal of Open Source Software, 2017. http://www.theoj.org/joss-papers/joss.00456/10.21105.joss.00456.pdf. DOI: 10.21105/joss.00456.

[4] Robot_state_publisher - ROS Wiki[Z]. http://wiki.ros.org/robot_state_publisher. (Accessed on 08/15/2021).

[5] Wiki.ros.org/nodelet[Z]. http://wiki.ros.org/nodelet. (Accessed on 08/20/2021).

[6] Ros_control - ROS Wiki[Z]. http://wiki.ros.org/ros_control. (Accessed on 08/15/2021).

[7] Transmission_interface: transmission_interface::DifferentialTransmission Class Reference[Z]. http://docs.ros.org/en/jade/api/transmission_interface/html/c++/classtransmission__interface_1_1DifferentialTransmission.html.

[8] Gazebo : Tutorial : ROS control[Z]. http://gazebosim.org/tutorials/?tut=ros_control.

[9] Pr2_controller_manager/safety_limits - ROS Wiki[Z]. http://wiki.ros.org/pr2_controller_manager/safety_limits. (Accessed on 08/21/2021).

[10] WANG H, JI Z. 基于卡尔曼滤波的目标识别跟踪与射击系统设计[Z]. 2021. DOI: 10.13140/RG.2.2.24410.47040.

[11] PyrDown[Z]. https://docs.opencv.org/4.2.0/d4/d86/group__imgproc__filter.html#gaf9bba239dfca11654cb7f50f889fc2ff. (Accessed on 12/12/2022).

[12] DOUGLAS-PEUCKER ALGORITHM[Z]. https://cartography-playground.gitlab.io/playgrounds/douglas-peucker-algorithm/. (Accessed on 12/12/2022).

[13] 2D Polyline Simplification[Z]. https://doc.cgal.org/latest/Polyline_simplification_2/index.html#Chapter_2D_Polyline_simplification. (Accessed on 12/12/2022).

[14] RTwiki[Z]. https://rt.wiki.kernel.org/index.php/Main_Page.

[15] Easy_handeye: automated, hardware-independent Hand-Eye Calibration[Z]. https://github.com/IFL-CAMP/easy_handeye/blob/master/README.md.

[16] Camera IMU calibration[Z]. https://github.com/ethz-asl/kalibr/wiki/camera-imu-calibration.