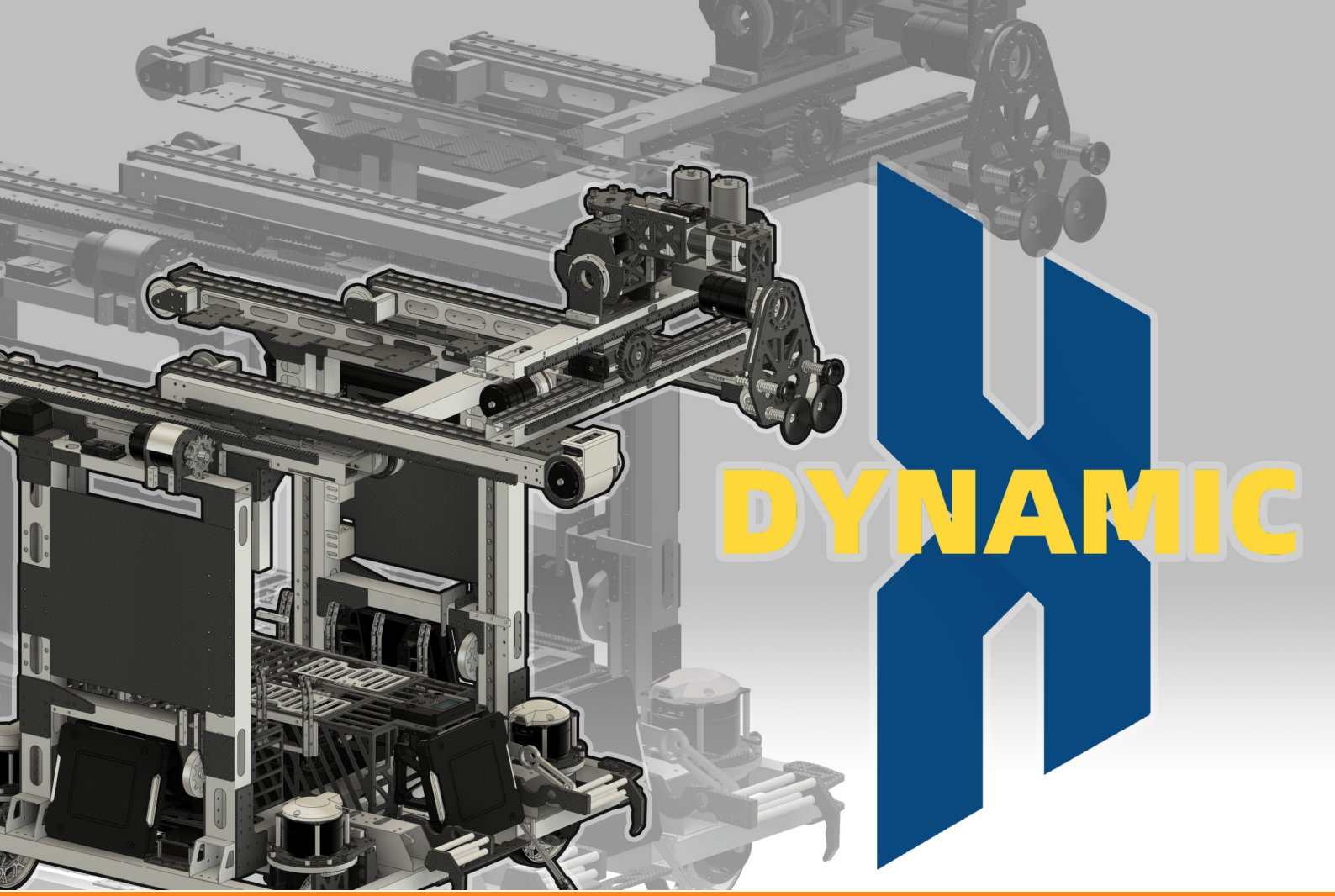


工程机器人技术报告

RoboMaster 2022

组织：广东工业大学 DynamicX

时间：8 月 2022

目录

第 1 章 简述

作为 Robomaster 赛场的不可或缺的一部分，工程机器人的重要性不言而喻，特别在现行规则下，其工作的效率及其稳定性直接决定着队伍的步兵以及英雄的作战能力。工程机器人主要有两大任务：（1）夹取矿石并兑换；（2）救援阵亡机器人。其中，夹取矿石尤为重要，一旦工程机器人故障，无法夹取矿石时，全队便会陷入吃低保的局面。面对具有稳定夹矿、兑矿功能的队伍，失误方将完全处于劣势，几乎没有胜利的可能。

相对于 RMUC2021 的比赛规则，2022 赛季对工程机器人的需求进一步提高。在国赛八进四阶段，兑换站会根据累计兑换金币数量后而改变其位姿，最终阶段将具有 x、y、z、roll、pitch、yaw 上的自由度，这对工程机器人兑换机构的设计与控制提出了较上赛季更高的要求。

1.1 主要思路

考虑到工程机器人在很小的尺寸下需要实现非常多的功能且需要应对国赛六自由度兑换站，我们定下了用一个灵活的机构实现大部分功能思路。结合我队的电控框架较为先进，并且在 2021 赛季中就已经搭建了一套机械臂控制程序并将其应用到我队 2021 赛季上场的工程机器人上。同时结合我队机械组上赛季已经具有设计机械臂经验，对多种形式的机械臂方案有一定了解，所以我们定下了使用机械臂作为主要机构的大方向。

1.1.1 需求分析

任务	需求分析	结构设计
取矿	保持底盘不动，快速横移间隔取矿且可以灵活取得大、小资源岛及地面的矿石 具有极高的空接成功率 可以同时取得多个矿石取兑换	取矿机构选用六轴机械臂 机械臂末端使用吸盘吸取矿石 使用矿石仓
兑换	可以在兑换站具有六自由度时完成兑换 有多个矿石时机械臂可以在同一位置快速连续夹取兑换	兑换机构选用六轴机械臂 为保证兑换时的效率，平移机构采用齿 + 双齿条传动
救援	使用救援卡救援全部机器人 将阵亡机器人拖拽回复活区	救援卡伸缩机构置于底盘 底盘前端增加拖拽机构
铺路	使用障碍块为己方提供行动便利，举起障碍块保护前哨站	机械臂一、四、五轴使用更换高减速比的 3508 电机
操作	操作手取矿、兑换、救援时有良好视野	在机械臂三轴安装云台，使得图传可以随机械臂升降与平移

表 1.1: 需求分析

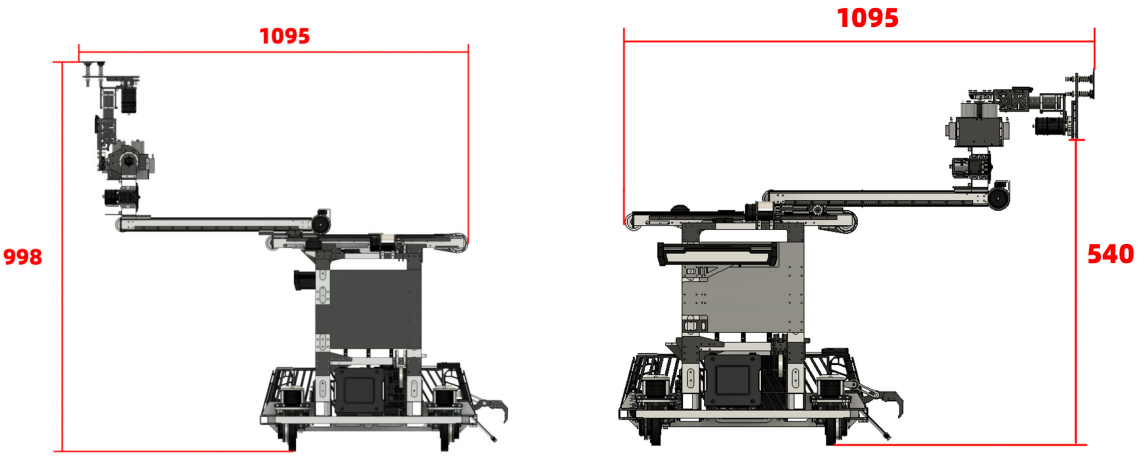
如 **表** 1.1，根据对 RMUC2022 规则的分析，设计者将工程机器人的主要功能分为取矿、兑换、救援三个大部分。另外补给、铺路作为备用计划的一部分，其结构设计及布局管理也纳入整体设计进行考量，除此以外，为了让操作手更加便利地操作机器人，操作的视野需求也被纳入整体设计中。

1.1.2 矛盾分析

如**表** 1.2, 在设计时经常会出现需求与规则相矛盾的情况，该表列举相关状况并提供了解决方案。

要求	矛盾	解决方案
伸展到最大尺寸空接矿石	最大伸长时不得超过机体 400mm 最大伸展尺寸不得超过 1100mm	空接时机械臂向前伸展如**图** 1.1(a) 吸取掉落到资源岛的矿石时机械臂向后伸展如**图** 1.1(b)

表 1.2: 矛盾分析



(a) 空接　　(b) 资源岛

图 1.1: 机械臂伸展示意图

1.2 技术方案

1.2.1 机械臂

在决定使用机械臂作为机器人的主要执行机构后，我们对机械臂的一系列伸展尺寸进行了分析。

不同高度夹取分析

- 机械臂若按照把爪子末端假设成一个点，并且该点与矿石中心重合，再按照最终该点所在位置为目标区域中心，机械臂底座高度为 300mm。
- 兑换站：高度方向需伸展 400mm，水平方向需要移动 253mm 。
- 小资源岛（槽内）：高度方向需伸展 400mm，水平方向需要移动 160mm。
- 大资源岛（槽内）：高度方向需伸展 200mm，水平方向需要移动 300mm。
- 掉落在地面的矿石：高度方向需伸展 -200mm，水平方向需要移动 150mm。
- 障碍块（不考虑工具外形）：高度方向需伸展 -160mm ，水平方向需要移动 150mm。

1.2.2 底盘设计

在本赛季规则中，场地中起伏路段的面积大量增加，在起伏路段上无论是机器人的移动抑或是救援其它机器人时所需要的力都更大，如果采用麦轮，在麦轮向前平移运动时，是依靠相对应的轮组抵消对应左右方向的力，而留下前进方向的力，这样的形式必然会带来电机功率的损耗，而如果采用舵轮，舵轮底盘的最大速度效率方向是任意方向，则不会损耗相对应的功率因为所有的功率都用在了期望运动的方向上，能极大地提升牵引力。工程机器人的运动速度会直接影响队伍的支援问题，无论是取矿、兑换、救援、卡位等都需要一个速度更快的工程机器人。同时由于工程机器人不限制底盘功率，故工程机器人使用舵轮底盘。

1.2.3 取矿机构设计

在 2021 赛季全国赛中，东北大学的工程机器人[1]创新性地使用吸盘来获取矿石，其工程机器人在比赛中展现了较传统的使用夹爪的工程机器人更为强大的空接能力。

传统的夹爪取矿机构需要矿石侧面接触到夹爪才能对矿石进行夹取，这势必会造成一定高度的损失，而使用吸盘来对矿石进行获取，这可以是空接高度无限接近 1000mm，这一点高度的优势在分寸必争的空接争夺中极为重要，几乎决定了矿石的归属权。同时，使用吸盘来进行空接时，我们只需要保证矿石掉落在吸盘上就可以保证获取到矿石，相对传统夹爪机构需要传感器来感知矿石掉落后驱使夹爪夹紧来进行空接，吸盘不需要严格地调整机械臂的姿态，所以吸盘的成功率更高且更为简洁。因此我们将吸盘融合到 6 轴机械臂上。

综上所述，本赛季我们在机械臂的末端使用吸盘对矿石进行获取。

1.3 战场战术定位

1.3.1 战场定位

- 经济辅助：引进经济体系之后，工程机器人就成为了获取资源岛矿石进行兑换获取经济优势的强力辅助。
- 战略性防御：在某些情况下，工程机器人可利用自己较厚的血量来对己方前哨站进行遮挡保护，同时也可以在机器人作战中作为掩体或者障碍来影响机器人战斗的战况。
- 战场救援：工程机器人还具有战场救援的功能，可以在己方作战机器人阵亡之后进行复活救援。
- 战场辅助：工程机器人在己方机器人与敌方机器人产生战斗时，可以起到限制敌方机器人移动、堵住敌方机器人逃跑路线的作用

1.3.2 战术定位

- 抢占先机：在双方对局开始的时候，工程机器人主动出击，尽快占领大资源岛禁区并着手准备抢夺金矿石来获得经济优势。
- 经济优先：在双方对局的前期，要尽量保证己方取得经济优势，将工程机器人较多的精力放在争夺矿石获得经济优势上。
- 适时保护：在双方对局期间，工程机器人实时与作战机器人保持交流，在特定的场景对包括己方前哨站和作战机器人在内的单位进行一定程度的保护
- 输出优先：在双方对局的中后期，在保证有一定经济基础的前提下，优先考虑借助工程机器人的力量协助作战机器人的输出，达到血量优先，占领获胜先机的目的。

第 2 章 其他学校工程机器人分析综述

2.1 东北大学 TDT 战队

2.1.1 各项功能的完成度及技术水平

- 可完成所有功能。空接成功率极高。

2.1.2 值得借鉴之处

- 采用吸盘结构吸取矿石，在空接高度方面胜过一切夹爪车。其强力的空接能力极大地弱化了吸盘车在抢夺地面矿石或非正常姿态的岛上矿石不占优势的影响。
- 整车初始状态重心低，行进过程平稳。
- 前后延伸机构采用铝管配合自制轴承盒结构，轻量化而实现高效，反应速度快。
- 救援机构中采用的局部齿轮作为转动限位，不容易损坏其他位置结构。这一设计省去了需要单独设计限位的空间而达到同等限位效果。
- 图传与左右平移一体，在运动时同步运作，保证视角在移动时始终正式前方，给予操作手更好对位体验。

2.1.3 有待改进之处

- 矿仓为向上敞开式矿仓且翻转需靠末端转轴吸盘挑拨来完成，撞击，剧烈晃动易使矿石出现非正常姿态甚至掉落。调整矿石姿态需要花相对较长的时间。
- 其吸盘结构在取矿时对于矿石姿态要求过高，吸取非正常形态的矿石有一定的困难。

2.2 哈尔滨工程大学创梦之翼战队

2.2.1 各项功能的完成度及技术水平

- 可完成所有功能。
- 其设计结构紧凑，空间利用率高，质量轻。

2.2.2 值得借鉴之处

- 上层结构采用具有前方 45° 自由度的摆臂（动力为气缸）使得升降机构得以后移与末端夹爪等机构平衡整车重心，该结构动作简单利于快速争抢资源岛上金矿石且根据其报告，结构未有明显变形且拥有足够刚度。此外整车初始状态重心低，行进过程平稳。
- 存储翻转一体矿仓充分利用结构展开时的垂直空间可达到存双矿石且在兑换时不必为了翻转，将后存的矿石扔到地上这类行为。

2.2.3 有待改进之处

- 受限于摆臂结构与夹爪车定位空接高度过低，在与吸盘车争抢空接矿石时略占劣势。
- 其矿仓翻转机构跟随前后延伸机构运动导致存完第一个矿石后若没调整好矿石高度将导致底部与资源岛面壁干涉而无法向前延伸，拖延争抢第二枚矿石时间。
- 依据其报告中存储翻转一体式矿仓，其赛场上初期仅为底盘上四面亚克力板作为矿仓且与翻转机构分离，导致存储第一枚矿石由于争抢过程来不及将上层结构收回放置矿石而是直接使其下落，矿石落于矿仓时出现非正常姿态居多。

第 3 章 机器人功能定义

工程机器人功能定义如 **图** 3.1所示。

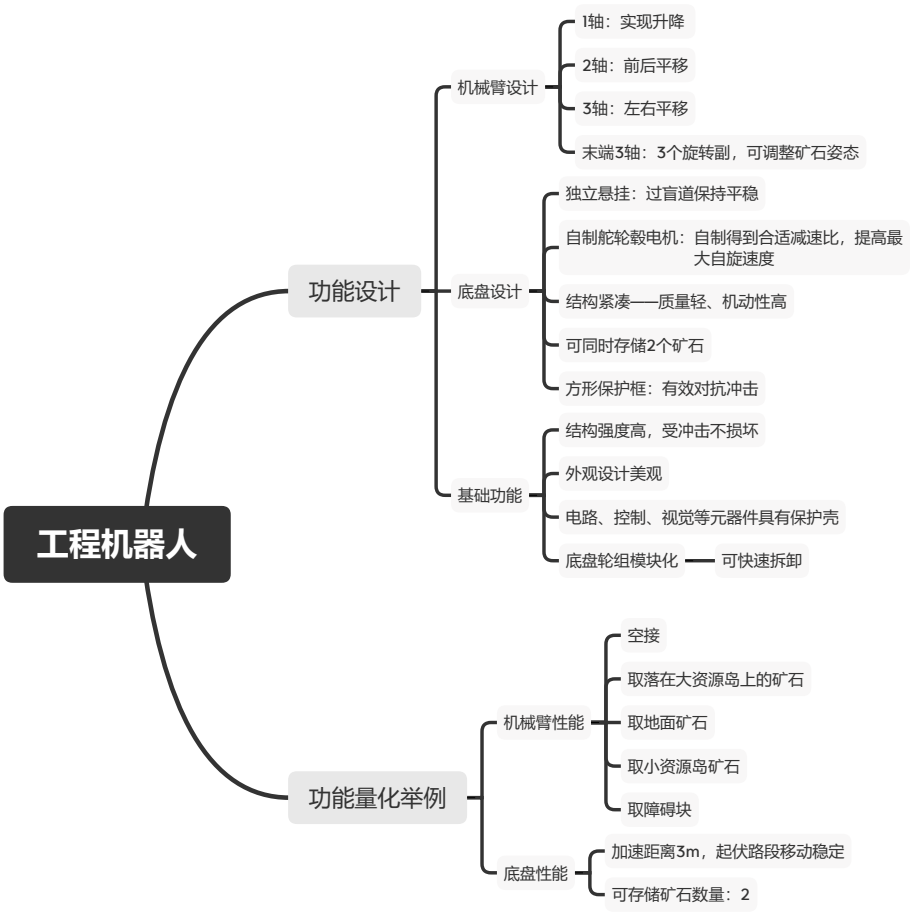


图 3.1: 工程机器人功能定义图

第 4 章 机器人核心参数

4.1 基本参数

4.1.1 机械参数

工程机器人重量，长、宽、高、重心高度等

名称	参数
重量（kg)	31.2kg（含裁判系统）
长、宽、高 (mm)	590*590*595（收缩尺寸）918*995*995(伸展尺寸）
重心高度 (mm)	216
运动速度	平移 4.76m/s 旋转 16.2rad/s
运动加速度	平移 6m/s^2 旋转 15rad/s^2

表 4.1: 工程机器人机械参数

4.1.2 云台

云台的固定形式为与机械臂 link3 刚性固定，可随着末端机械臂位置的变化来调整视野的高度，以机械臂向前为 x 轴正方向，竖直向上为 z 轴正方向，云台相对于 joint3 的相对位置为（148, 0, 98），如**图** 4.1所展示。

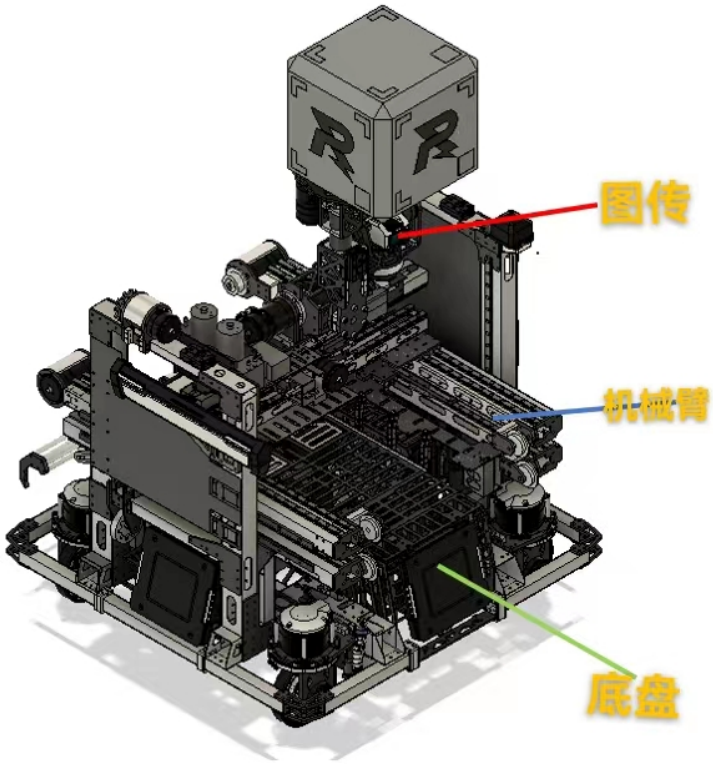


图 4.1: 云台位置示意图

4.1.3 机械臂

机械臂是七轴六自由度机械臂，其中 joint1，joint2 以及 joint3 依靠两个电机驱动同一轴。机械臂的 Link 和 Joint 的定义如 **图** 4.2，它们的参数如**表** 4.2、**表** 4.3所示。

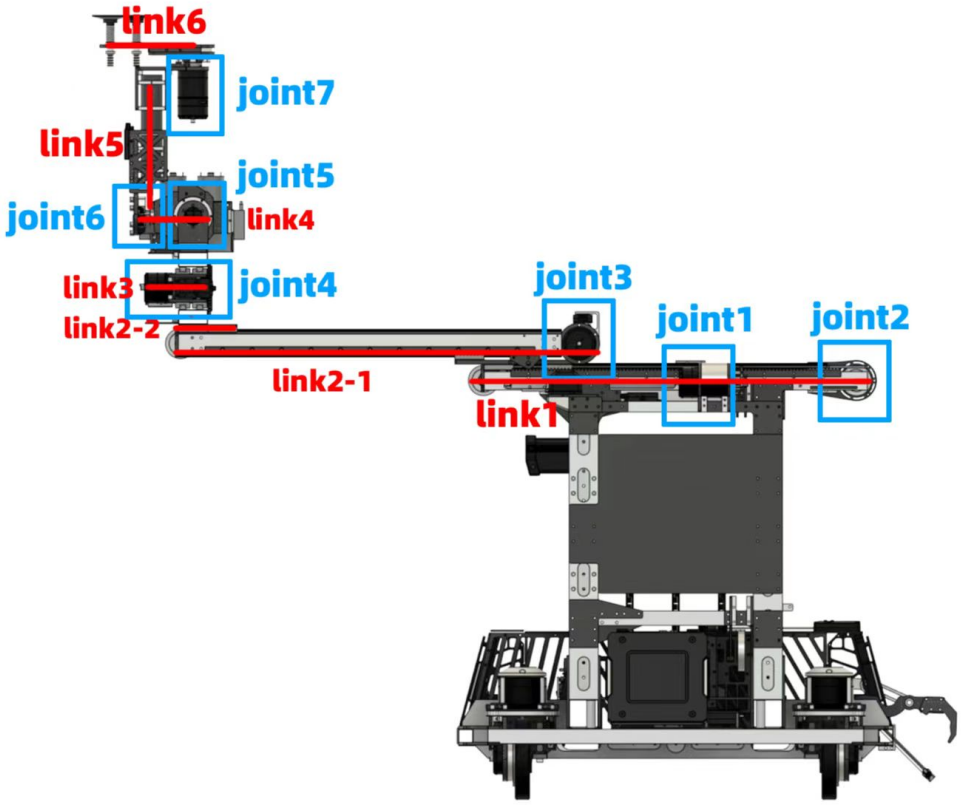


图 4.2: 机械臂的 link 和 joint 示意图

连杆名称	质量（Kg）	臂长（mm）
Link1	2.324	128
Link2_1	0.961	162
Link2_2	0.961	162
Link3	0.953	113
Link4	0.897	95
Link5	0.023	-
Link6	0.961	162

表 4.2: 机械臂 Link 参数

4.2 电路参数

车载模块及其用途、设计最大功率：

- NUC 供电电源：用于给 NUC 提供19 V 电源，设计最大功率为200 W。
- 机械臂开关：用于根据底盘电压控制机械臂是否上电，设计最大功率为480 W。

工作电压范围：

关节名称	力或力矩	限位角度或限位距离	速度
Joint1	609.75 (N)	+0.24,-0 (m)	0.4 (m/s)
Joint2	110 (N)	+0.33,-0.35 (m)	1.9 (m/s)
Joint3	258 (N)	+0.31,-0.12 (m)	0.964 (m/s)
Joint4	85.47 (N)	+0.31,-0.31 (m)	2.92 (m/s)
Joint5	13.3 (N m)	+3.84,+0.0 (rad)	18.8 (rad/s)
Joint6	13.3 (N m)	+0.9,-0.9 (rad)	18.8 (rad/s)
Joint7	5 (N m)	+4.5,-1.3 (rad)	50 (rad/s)

表 4.3: 机械臂 Joint 参数

- 在设计 NUC 供电电源时，将其工作最低电压设定为22 V，最高电压则为电池满电电压，一般约为24 V 左右。

4.3 执行器

表 4.4为驱动器的种类与数量。

电机种类	用途	数量
M3508	机械臂 jont2、3、4、7 驱动	6
M3508（改减速比）	底盘轮系、机械臂 joint1、5、6 驱动	8
M3508（无减速箱）	云台 yaw 和 pitch 轴	2
M2006	救援、拖拽驱动	2
M6020	舵轮航向驱动	4

表 4.4: 工程机器人执行器种类及数量

第 5 章 机械设计

5.1 机械结构设计

机械整体设计主要将工程机器人分为机械臂和底盘两部分。

5.1.1 机械臂

如**图** 5.1所展示，机械臂基本自由度由 3 个转动副和 3 个移动副组成。

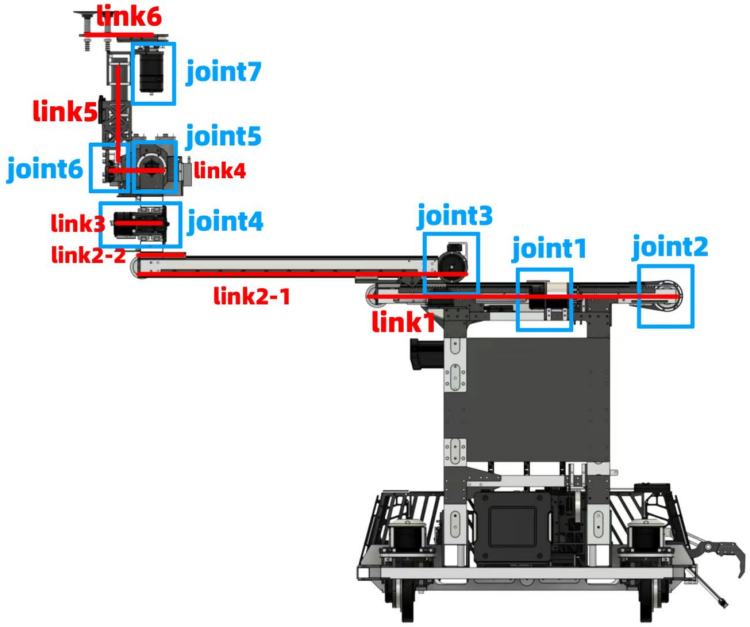


图 5.1: 机械臂

机械臂每个自由度由各电机独立驱动，机械分别采用了链传动，齿轮齿条传动，改电机减速比后直连等方式。

5.1.1.1 Joint1 与 Link1

一轴为了增大承载能力，所以采用了链传动，电机选型采用 M3508（改减速比 1：51），持续输出的最大力矩为 8.05Nm，另外外加恒力弹簧，由此来抵消机械臂的整体重量，使得在无外力作用下，一轴可以悬停在任意位置，并且有效的提高了升降的速度。

为了增大一轴的支撑刚度，升降架顶端用一根 20*30 的铝管连接。为配合两侧装甲板高度并满足规则限高，我们将升降架放置在距离中心 200mm，依然满足矿仓存储矿石的要求。升降支架主体采用两根 40*30 的铝管支撑，质量轻强度高。为减小 1 轴惯量，我们选择将恒力弹簧分别连接 1 轴和 2 轴之间来抵消机械臂的重力，起到了重力补偿的效果。1 如**图** 5.3所展示。

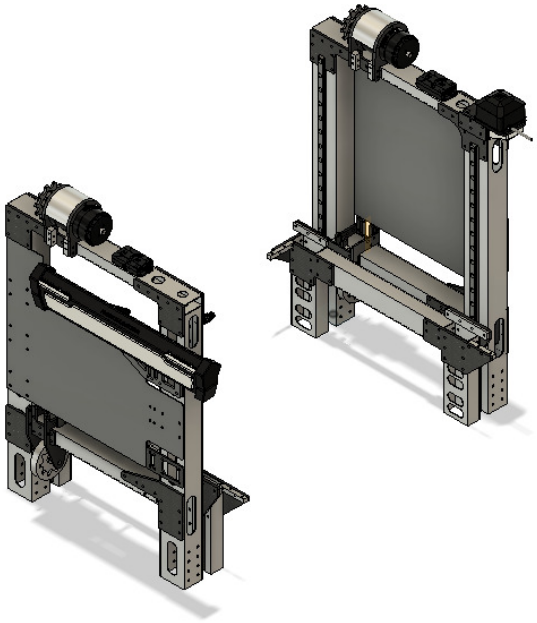


图 5.2: Link1

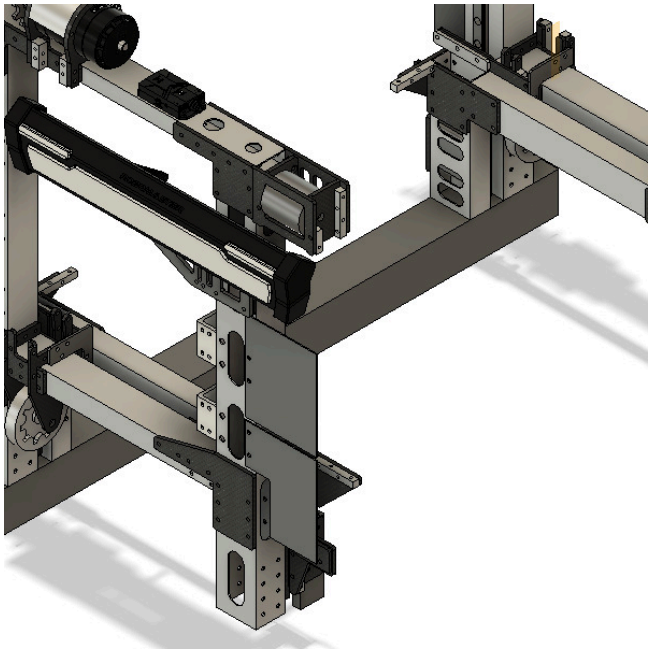


图 5.3: 恒力弹簧图

5.1.1.2 Joint2、Joint3 与 Link2

Joint2-平移副-齿轮齿条加链条传动。缺点：齿轮易磨损。齿轮和齿条之间存在间隙，导致校准的时候存在偏差。链条使用久之后容易松动。2 轴整体的质量太大。考虑到加速度的附加，两层各自通过两个电机驱动。如图所示，第一层链条带动中间滑块移动，齿轮连接中间滑块，同时和上下齿条啮合，在链条带动齿轮向前移动同时，第二层铝管的齿条也向前移动，因此可以实现双倍行程的效果。第二层延伸轴链条带动上层滑块移动，因此实现了空接和取大资源岛在机器人前后方向。

Link2 整体结构设计思路如**图** 5.4所展示。

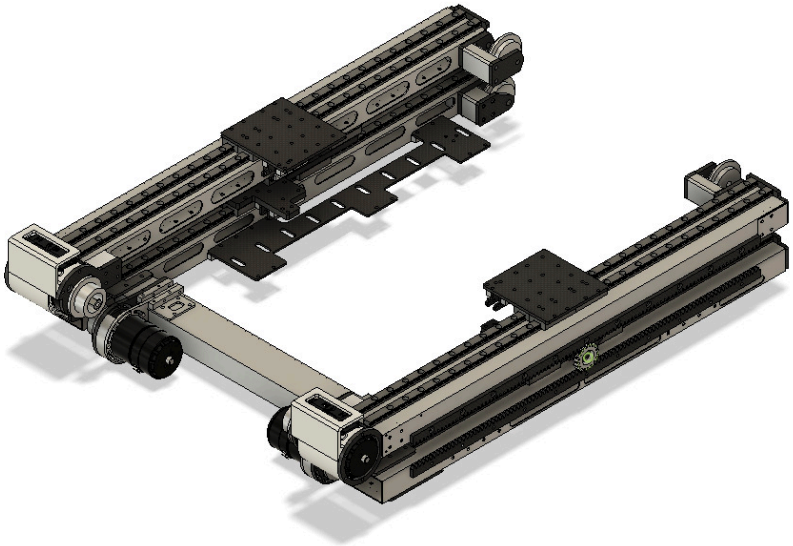


图 5.4: Link2 图

5.1.1.3 Joint4 与 Link3

Joint4-移动副-齿轮传动 3，齿轮位于上下两层铝管之间，与电机通过联轴器直连，齿条分别连接上下两层铝管上。电机驱动带动齿轮旋转，从而驱动齿条相对移动，实现双倍行程的效果。如**图** 5.5所展示。

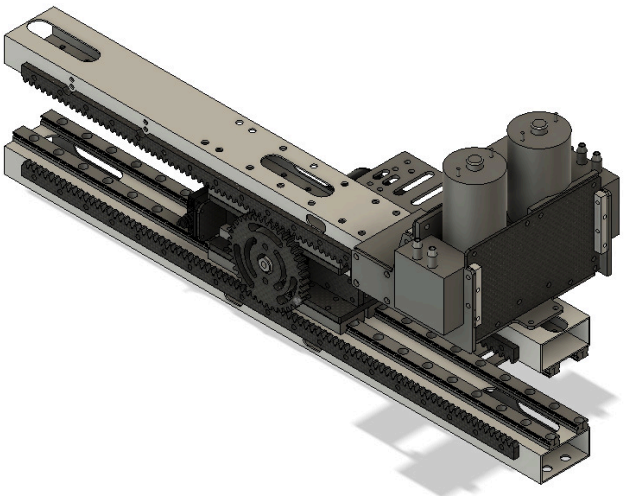


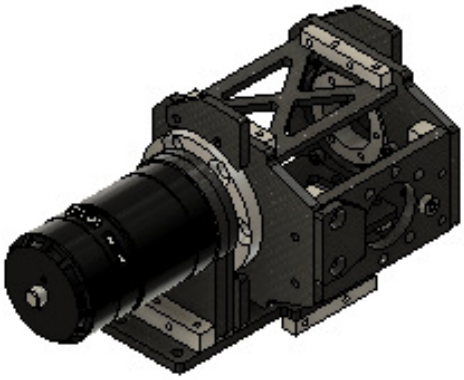
图 5.5: Link3

5.1.1.4 Joint5 与 Link4

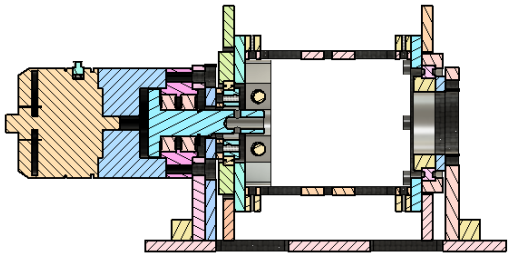
Joint5-转动副-电机直连，设计预计机械臂末端伸出机体 500mm 左右，力臂较大，需要抬起 2.25kg 的障碍块，静态所需扭矩较大，电机选型 M3508（减速比 1：51），最大持续输出力矩 8.05N m, 缺点：改减速比后静态游离缝隙大。电机通过螺丝固定在碳板上，再用打印件作辅助固定，因机械臂运动中会出现较大的径向力和轴

向力，力会集中作用在电机轴处，故设计时电机轴外部放置交叉滚子轴承，内部改装电机放置角接触轴承，使其能承受电机轴方向内外双向的力，保护电机不被破坏。

Link4 结构复杂，由 2 块侧板与铝件组成，其中中间部分为 Link5 的电机，如**图** 5.6(a)。



(a) Link4 图 (含部分 Link3)



(b) Joint4 截面图

图 5.6: Joint4 与 Link4 展示图

5.1.1.5 Joint6 与 Link5

Joint6-转动副-电机直连，电机选型 M3508（减速比 1：51），最大持续输出扭矩 8.05N m，3508 电机的固定、安装推力和角接触轴承、承受双向力等方式与 Joint5 一样。如**图** 5.7所展示。

Link5 为 3508 电机输出轴。

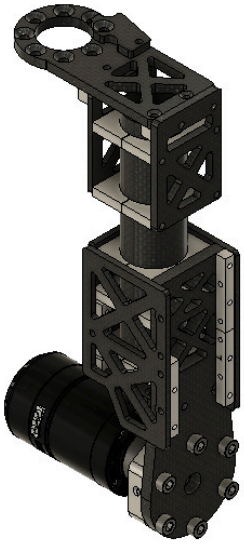


图 5.7: Link5 图

5.1.1.6 Joint7 与 Link6

Joint7-转动副-电机直连，电机选型 M3508（减速比 1：19），最大持续输出扭矩 3N m，3508 电机的固定、安装推力和角接触轴承、承受双向力等方式与 Joint5 一样。如**图** 5.8所展示。

Link6 为 3508 电机输出轴。

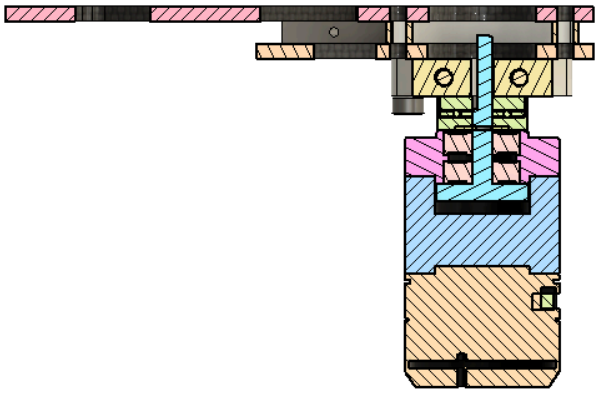


图 5.8: Joint7 截面图

5.1.1.7 吸盘

吸盘为两个直径 50mm 的硅胶吸盘，通过弹簧缓震延长矿石与吸盘的接触时间，使吸盘抽真空的时间增加，因此提高了空接矿石的成功率。外加两块碳板增加吸盘的刚性，避免吸附矿石后出现一个矿石前倾的情况。

图 5.9: 吸盘图

5.1.1.8 云台

考虑到操作手需要广阔的视野和精准的对位，所以选择将云台安装在 Link3 上可以随着机械臂前后平移和左右平移。云台设计的 yaw 轴与 pitch 轴均为电机直连，电机选型为 M3508（无减速箱），设计中以轻量化贯彻始终，故选择拆除减速器的 M3508，单个重量约 80g，设计结构简单为，如**图** 5.10所示。

5.1.2 底盘

工程机器人的底盘根据井字形结构设计，主要由主框、救援机构、保护框、轮组、矿仓四部分组成。如**图** 5.11所示。

5.1.2.1 主框

主框是底盘的主要构成部分，承担着承载矿石仓和机械臂的作用。主框主要包括主梁，碳板，救援机构等。

主梁采用薄壁粗铝方管，但经过测试后发现上交步兵底盘使用装螺母的内嵌件进行方管连接虽然有一定的便利之处，可若经常拆装，3D 打印的内嵌件会磨损坏导致无法固定住螺母，进而使得螺母在内嵌件内打滑，导致无法拆卸的问题。因此我们采用 2mmta 碳板套进主梁内 (如**图** 5.12(b))，并配合使用拉铆螺母保证连接处的连接强度和刚度，具有质量轻，成本低，易于拆卸的优点。

5.1.2.2 救援机构

救援机构分为刷卡救援机构和拖拽救援机构两部分。

救援卡救援机构采用齿轮齿条传动，通过线轨实现远距离输出，使用 M2006 电机进行驱动。如**图** 5.13(a)所示。

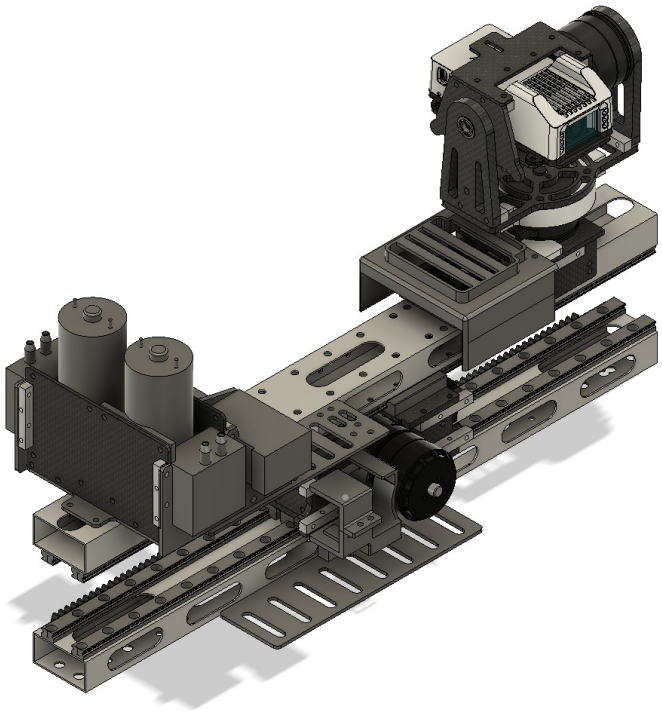


图 5.10: 云台安装示意图

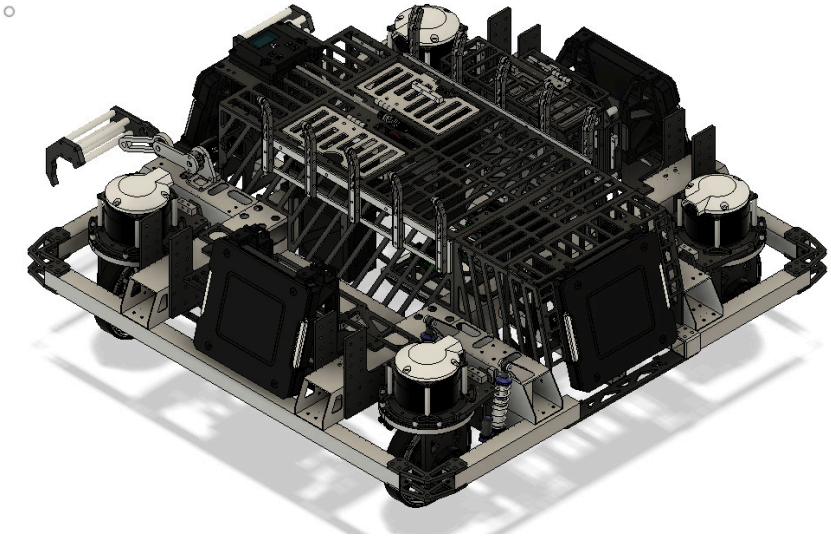


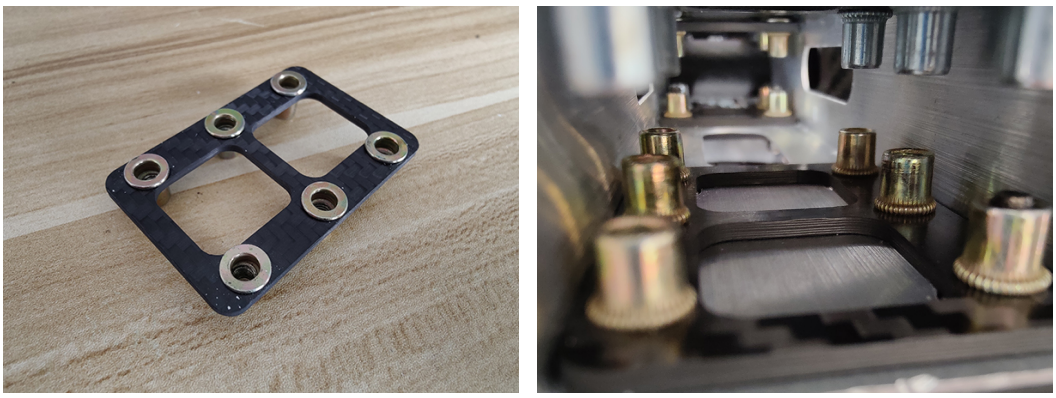
图 5.11: 底盘

拖拽救援通过 2006 驱动 4 连杆来驱动爪子旋转，利用四连杆自锁的原理，使拖拽的过程不太容易受到外界的影响。如**图** 5.13(b)所示。

5.1.2.3 保护框

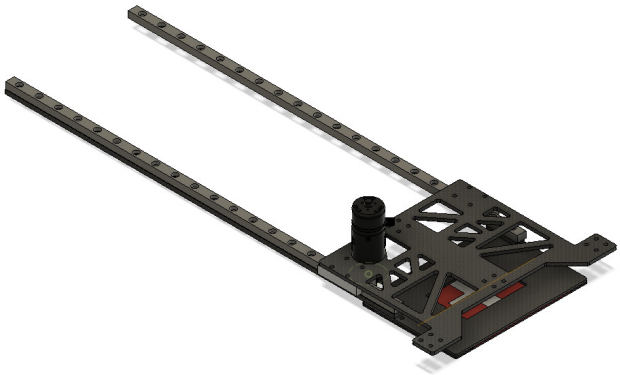
保护框主要是起着保护轮组和主框的作用，避免机器人受到冲击后直接损坏机器人的内部机构。保护框主要由细铝方管制成框架，防触底，活动板组成。

框架由 20*20*1 的 2 根长铝方管和 4 根短铝方管组成；各管之间使用 4mm 的碳板连接，减小冲击；四个角装有包胶轴承，减小撞击时的静摩擦力。



(a) 拉铆螺母固定板　　(b) 拉铆螺母装在铝管内

图 5.12: 铝管连接方式展示



(a) 刷卡救援机构图

(b) 拖拽救援机构图

由于机器人前重后轻，因此前后加装了贴近地面的防触底轮（包胶轴承），降低机器人翻车的趋势。

5.1.2.4 轮组

轮组系统负责机器人的正常运动以及适应战场地形，包括车轮支座、车轮、悬挂等机构。

轮组与东北大学的舵轮步兵轮组原理、结构相似，主要更改了 3508 电机的减速器，使其体积更小，质量更轻，转速更快。如**图** 5.13

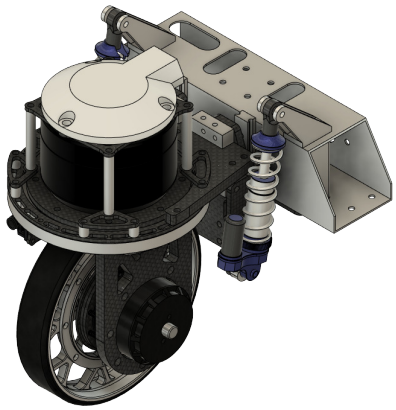


图 5.13: 轮组

5.1.2.5 矿弹仓

为了使机器人在有能力取得多个矿石的情况下，可以同时取得多个矿石并兑换，设计者在底盘中间设计矿石仓存储矿石。

矿石仓侧壁使用轴承，并且侧壁呈 135°，保证了矿石可以顺利存储在矿仓里，机械臂能够快速，连续地夹取并兑换矿石。矿石仓的底部放置了 1 块 200*300mm 的碳板并且表面贴上铁氟龙胶布进行润滑，保证了机械臂存取矿石时的容错率。后方通过 1 块 2mmta 碳板连接来保证侧壁的刚性，避免存矿过程中对矿仓的损坏。如**图** 5.14。

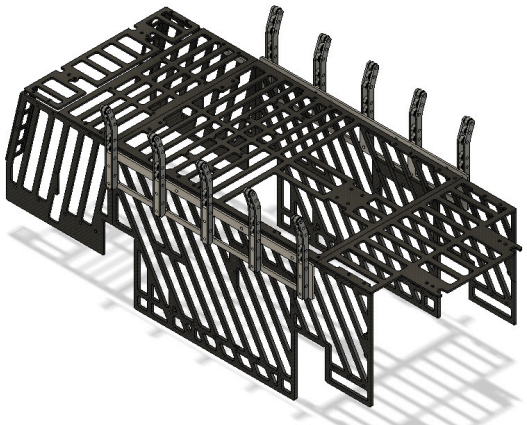


图 5.14: 矿仓

5.2 工艺选择

在工程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资源情况，在工程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，在一些必要的铝件上优化设计方案，设计出的铝件都可以利用团队三轴铣床自加工，避免复杂特征体的出现影响加工难度。结合 3D 打印件与部分车削的轴类零件作辅助。

5.2.1 2D 雕刻

工程机器人的大部分零件均为板类，如 **图** 5.15 所展示。底盘除去铝管主框架外，剩下的轮组均为碳板结构；机械臂也采用的是碳板加铝管的设计。工程整体设计时工艺选择 90% 均可自加工完成，实现少外包。

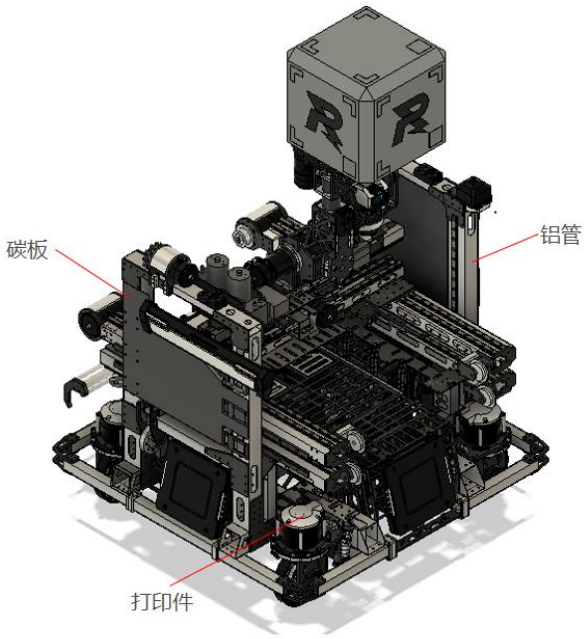
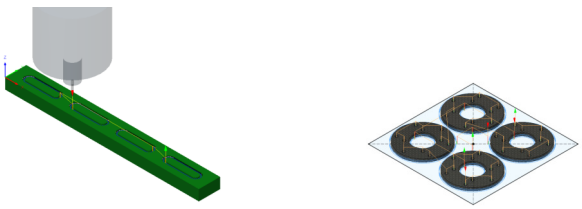


图 5.15: 工程机器人使用的材料指示图

因为设计时选择加工方式为 2D 雕刻，接着利用 fusion360 出简单刀路即可。铝管同理，铝管可看作由 4 个板面组成的整体，加工时就可以当作板子去雕刻，加工完成一面就翻转 90°。如此类推，就可以实现团队自行加工，无论是玻纤、碳纤板还是铝管。因为 2D 雕刻是最简单高效的加工方式，也是我们的工艺选择的首选。加工如 **图** 5.16 所展示。



(a) 铝管　　(b) 碳板

图 5.16: 2D 雕刻

5.2.2 3D 打印

对于一些拥有曲面的、强度要求不高、作平常固定非支撑件的零件，首选 3D 打印作为加工方式。

(a) 电机保护壳　　(b) 滑环固定座

图 5.17: 3D 打印

5.2.3 铣削和车削

对于既要满足强度要求，又拥有着复杂特征结构的特殊零件，就需要选择三轴铣削加工工艺，如**图** 5.18所展示。对回转体则采用车削工艺，如**图** 5.19所展示。

(a) 轮毂　　(b) 悬臂固定铝件

图 5.18: 铣削

(a) 齿轮　　(b) 齿圈

图 5.19: 车削

第 6 章 硬件设计

6.1 整体硬件框图

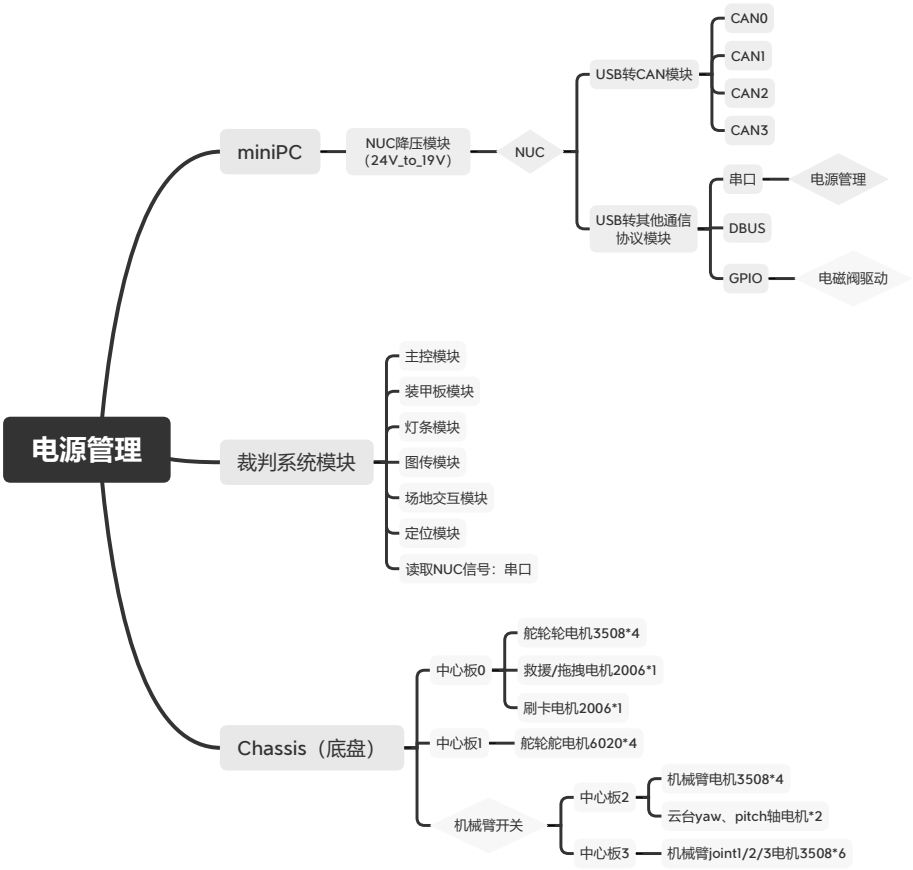


图 6.1: 整体硬件框架

图 6.1由电源管理模块出发，从供电和通信两方面介绍了我们工程机器人的整体硬件电路连接情况。

6.2 自研硬件模块

6.2.1 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。在工程机器人上，我们设计了 USB 转 CAN 模块和 USB 转 XXX 模块（包含串口、DBUS、GPIO），模块主要用于 NUC 和其它硬件模块之间的通信和控制电机。这些模块均使用 GH1.25 系列的接头作为板级连接件，以匹配官方的各类电机、模块。

- 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、DBUS 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。
- USB 转 GPIO 部分，我们采用 ATTINY85V-10SU 作为主芯片进行协议转换，该芯片是一款 Arduino 处理器，能够满足转换需求。

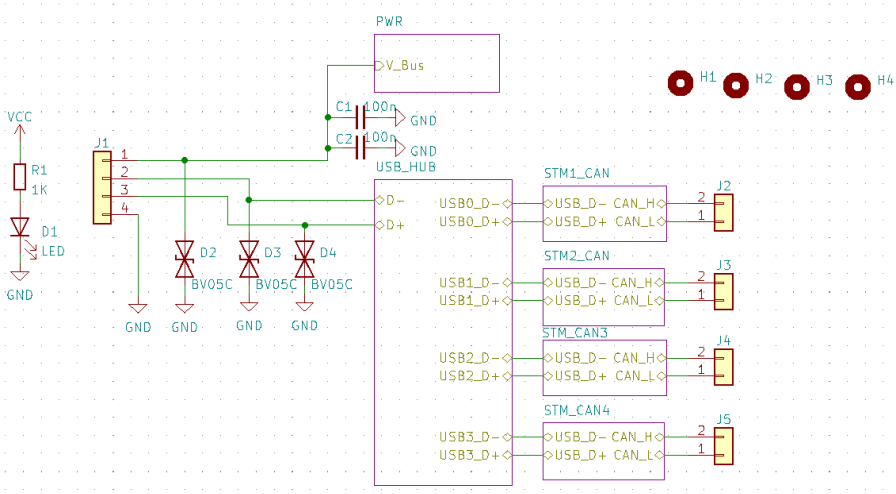


图 6.2: 协议转换模块主原理图

在实际测试过程中，各模块的通信均正常，丢包率极大的减少甚至没有出现丢包等不良通信状况。但另一方面，该模块会出现一定的发热，约莫 50°C 左右。

6.2.2 NUC 降压模块

由于我们工程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** 6.3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 50°C 以下，有非常优越的性能表现。

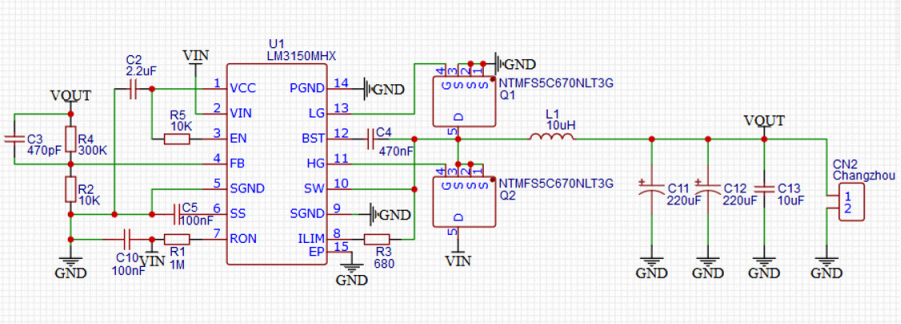


图 6.3: LM3150 原理图

6.2.3 机械臂开关模块

为达到控制机械臂上电与否的目的，需要制作一个可控的电源开关。如 **图** 6.4 所示，我们设计了一个以继电器为主体的电源开关模块。该模块采用松乐的小型大功率继电器，其控制、输入、输出端口的电压阈值可达48 V，满足设计需求；模块还集成了 TVS 二极管，具有较高的安规性能。

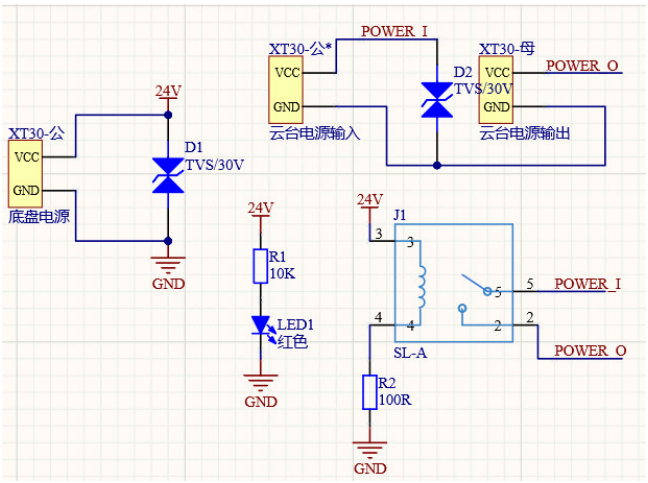


图 6.4: 机械臂开关原理图

在实际测试中，本模块对于控制端的响应速度十分迅速，且导通压差极小，工作效果极佳。

6.2.4 电磁阀驱动模块

由于使用吸盘进行取矿，故需要使用电磁阀对其吸力进行控制，最终我们针对电磁阀设计了一款驱动模块。该驱动模块如 **图** 6.5 所示，其原理是通过 J1 或 J2 输入 TTL 逻辑电平信号，控制电子开关 BTS443P 的通断，从而实现对电磁阀的控制。

BTS443P 是带保护的汽车高端功率开关芯片，它具有 1：1 的输入输出比率，其最大电流可达21 A、负载电压可达36 V，满足本次的设计需求，故采用该方案。此外，它还不需要额外电源供电，使得整体方案更为精炼。

实际测试中，本模块对于控制端的响应速度十分迅速，能够对电磁阀进行快速的开启与关断。

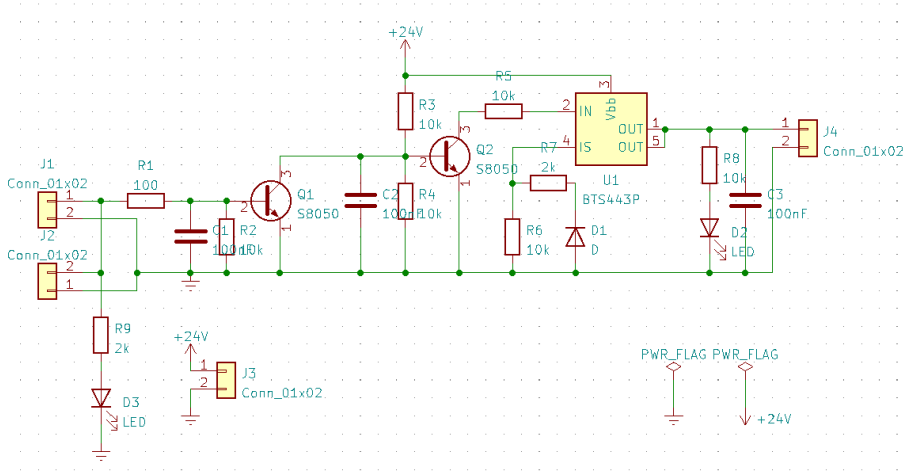


图 6.5: 电磁阀驱动模块原理图

6.3 关键器件选型

6.3.1 NUC 降压模块方案

我们大部分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。在早期，我们采用 TI 公司的 LM25116 为 NUC 供电，但是在使用中发现当电池电压低于 23V 时 NUC 带重载会卡死（电压轨道塌陷），原因是 LM25116 的占空比无法达到很大。所以我们选择了 LM3150 作为 Buck 控制器。

LM3150 是美国国家半导体还未被 TI 收购时期设计生产的产品。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。

6.3.2 USBHUB 方案

由于机器人使用 NUC 作为主控，所以与其他模块、电机通信时，必须进行通信协议的转换。为达到尽量减少所接插的 USB 口之数量、提升抗震性能，有必要研发 USB 分线器并在此基础上进行协议转换。市面上常见的 USBHUB 有双通道、四通道以及七通道的专用芯片方案，根据实际需求，我们选取了 SL2.1S 芯片实现一拖四的 USBHUB 方案。

SL2.1S 是高性能、高集成度、低功耗的 USB2.0 集线器主控芯片。该芯片采用 STT 技术，并使用单电源供电（供电电压为 5v），只需在外部电源引脚添加滤波电容。该芯片完美支持 USB2.0 高速（480MHz）、USB2.0 全速（12MHz）和低速模式（1.5MHz），完全满足通信速率的要求。

第 7 章 rm-controls

7.1 通用部分

7.1.1 概述

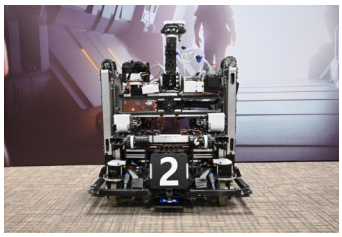
`rm-controls`[2] 是一套在 PC 上运行的无下位机电控软件，基于 ros-control[3] 的硬件和仿真接口以及配套的控制器，用于开发 RoboMaster 机器人和高性能机器人。

7.1.1.1 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** 7.1 2022 赛季部署了 rm-controls 的机器人。

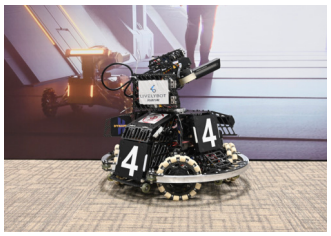
代码 7.1 为哨兵的云台配置文件，可以看到上下两个云台的配置文件结构相同，两个云台控制器被独立地动态加载，然后分别获取两个控制器的参数，包括它要控制的关节名称，还有对外接口 (以 rostopic 等形式) 的名称。

(a) #1 Hero



(b) #2 Engineer

(c) #3 Standard (Omin)



(d) #4 Standard (Omin)

(e) #5 Sentry

(f) #6 Aerial

(g) #7 Darts

图 7.1: 2022 赛季部署了 rm-controls 的机器人

Listing 7.1: 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}
```

7.1.1.2 多刚体动力学仿真

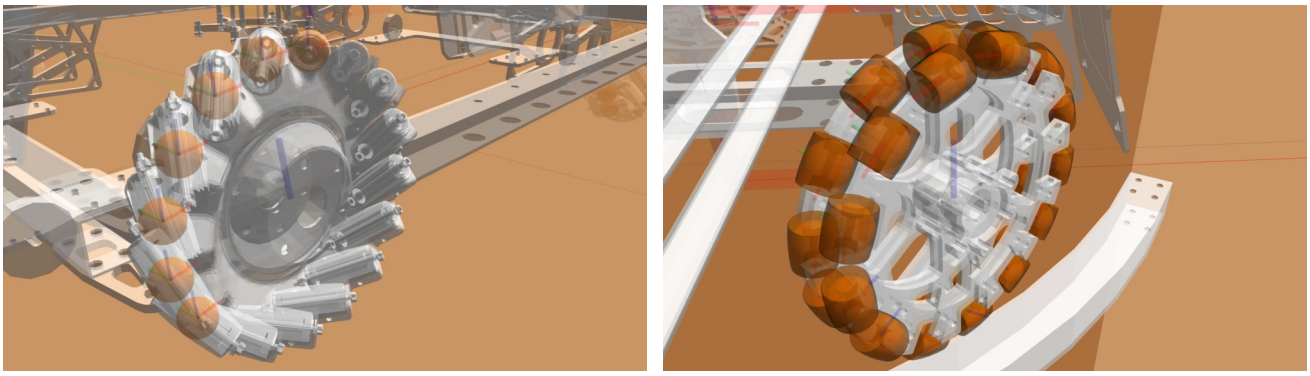
所有机器人都能在 Gazebo 进行多刚体动力学仿真，且仿真和实车运行的代码不需要相互移植，是同一份代码甚至同一个二进制文件。除了发射控制器，其他模块和控制器在开发时均会在仿真中调试和测试，打断点时可以做到时间暂停，在机械组还没有加工装配好车时已经可以把代码测试好，完成参数的粗调，大大提高开发和迭代效率。如：2021 赛季舵轮机器人和 2022 赛季全向轮机器人均在加工装配好后只花不到一天时间就测试调试完所有程序。**图** 7.2 为全向机器人在 RMUC 场地中的仿真。

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

7.1.1.3 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** 7.4。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。

图 7.2: 全向轮步兵机器人仿真



(a) 麦轮辊子仿真　　(b) 全向轮辊子仿真

图 7.3: 麦轮和全向轮辊子仿真

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

图 7.4: rm-controls 兼容的部分计算设备

7.1.1.4 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 7.5 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。

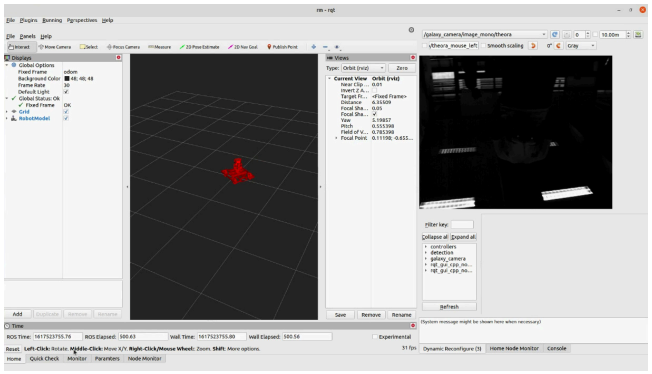


图 7.5: rqt 调试软件

7.1.2 Prerequisite

7.1.2.1 ros-control

如 **图** 7.6，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 映射成关节 (Joint) 等机器人状态，将这些状态接口提供给控制器；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）。

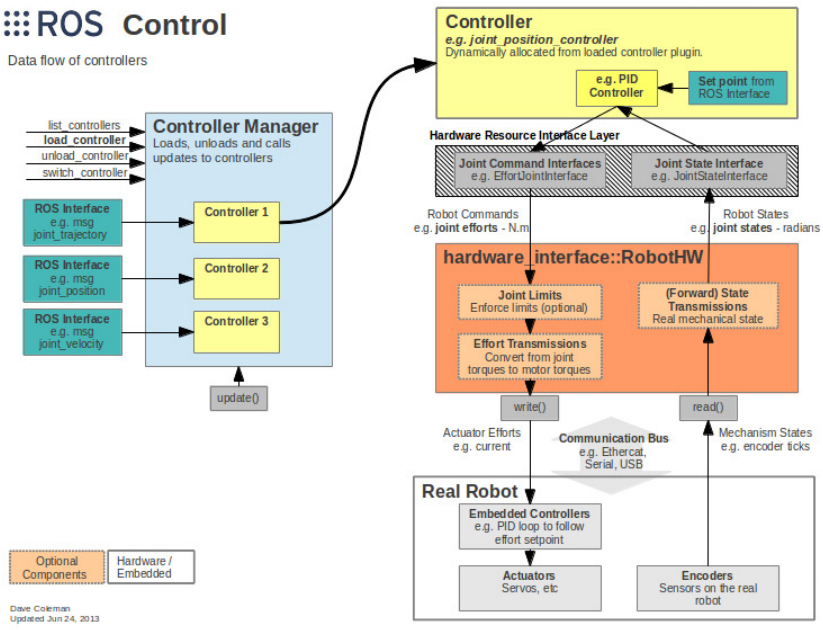


图 7.6: ros-control 框图，图源[4]

7.1.2.2 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

7.1.2.3 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** 7.7 展示了差分 Transmissions 的简图和计算方法。

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 7.2: pitch 轴 transmission

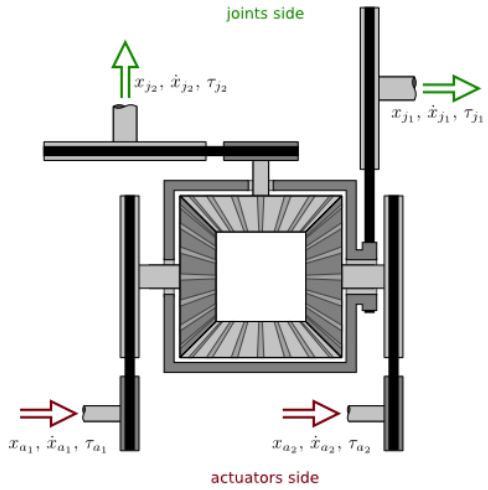
```
<transmission name="trans_pitch_joint">
    <type>transmission_interface/SimpleTransmission</type>
    <actuator name="pitch_joint_motor">
        <mechanicalReduction>-1</mechanicalReduction>
    </actuator>
    <joint name="pitch_joint">
        <offset>1.559</offset>
        <hardwareInterface>hardware_interface/EffortJointInterface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

7.1.2.4 Gazebo 仿真

得益于上述的机制，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** 7.8。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）。

7.1.3 实时性

我们为 Linux 内核打上 PREEMPT_RT[7] 补丁。如 **图** 7.9 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。



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

图 7.7: 差分 Transmissions，图源[5]

7.1.4 USB 转 CAN

由于一般 x86 平台不具有 CAN 外设，也不具有 SPI 等简单外设接口，无法使用 MCP2515 等 SPI 转 CAN 芯片。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。

由于整车需要至少 2 路 CAN 总线来保证电机回传数据包的完整性，我们采用了如**图** 7.10所示的 GL850G 作为 USB HUB 芯片实现 USB 一拖四的方案。在执行器较少的机器人上只需要 2 路 CAN，还有两个设备分别用于接收 DBUS 遥控器信号和裁判系统信号；在工程机器人和舵轮机器人上面我们会用将四路 usb 都转为 CAN 从而得到 4 路 CAN。另外，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。

如**图** 7.11 所示为使用 STM32 实现 USB 转 CAN 部分电路。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。图中的 R6、R7 电阻用途为更改 STM32 的 Boot 模式，从而使 STM32 能够通过更改 R6、R7 的短接模式而在 DFU 烧录模式与 Flash 模式下切换，方便烧录固件。

在 Ubuntu 系统中，通过 aptitude 获取 can-utils 软件包就可以将插入的 USB2CAN 设备映射成网卡设备。可以通过标准的网络接口启用 CAN 网卡。can-utils 软件包还提供了一系列如 candump、cansend 等用于调试系统总线的指令。

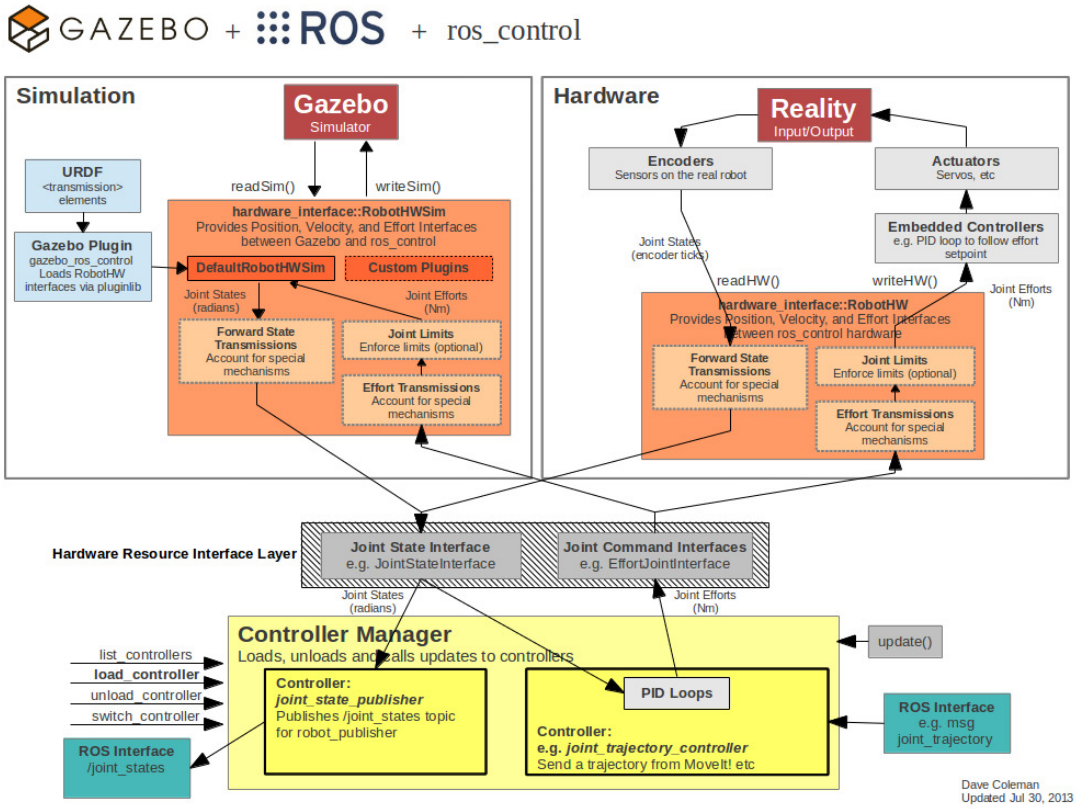
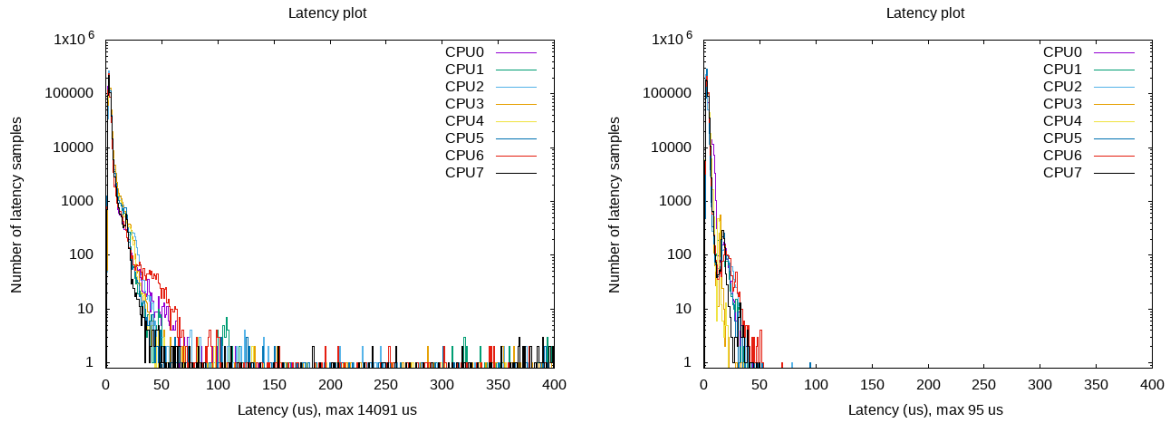


图 7.8: Gazebo + ROS + ros_control 简图，图源[6]



(a) 未打实时补丁　　(b) 已打实时补丁

图 7.9: 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

7.1.5 程序框架

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

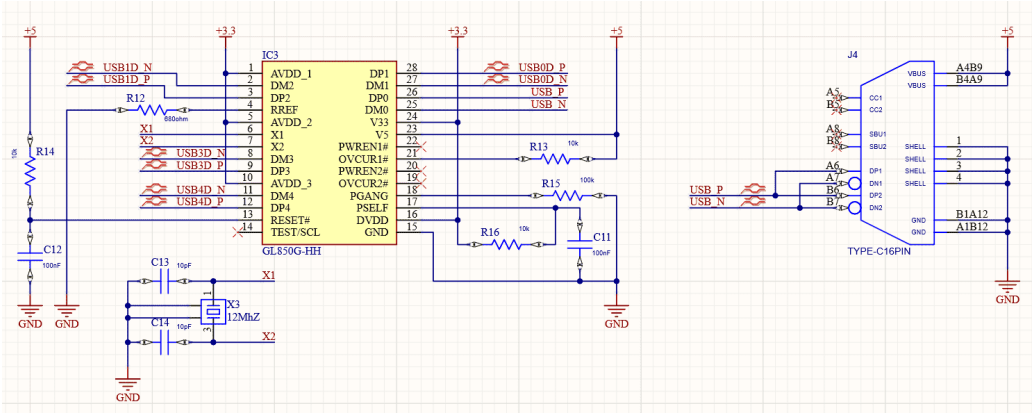


图 7.10: USB HUB 部分电路

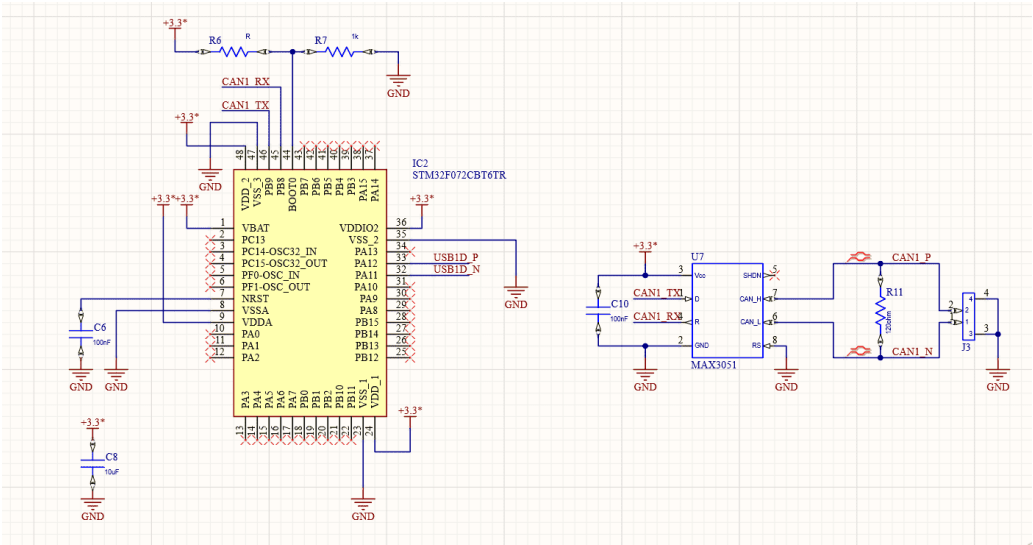


图 7.11: STM32 实现 USB 转 CAN 电路

  - 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_dbus: dbus 数据接收节点
  - 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_controllers: 麦克纳姆轮、舵轮、平衡车的底盘控制器
  - rm_orientation_controllers: 获取 imu 姿态以修正整车实际姿态的控制器
  - 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[8] 的高性能版，高频维护 tf
  - gpio_controller: 读写 gpio 的控制器
  - mimic_joint_controller: 使指定 joint 的位置模仿另一 joint 的控制器
- 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_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

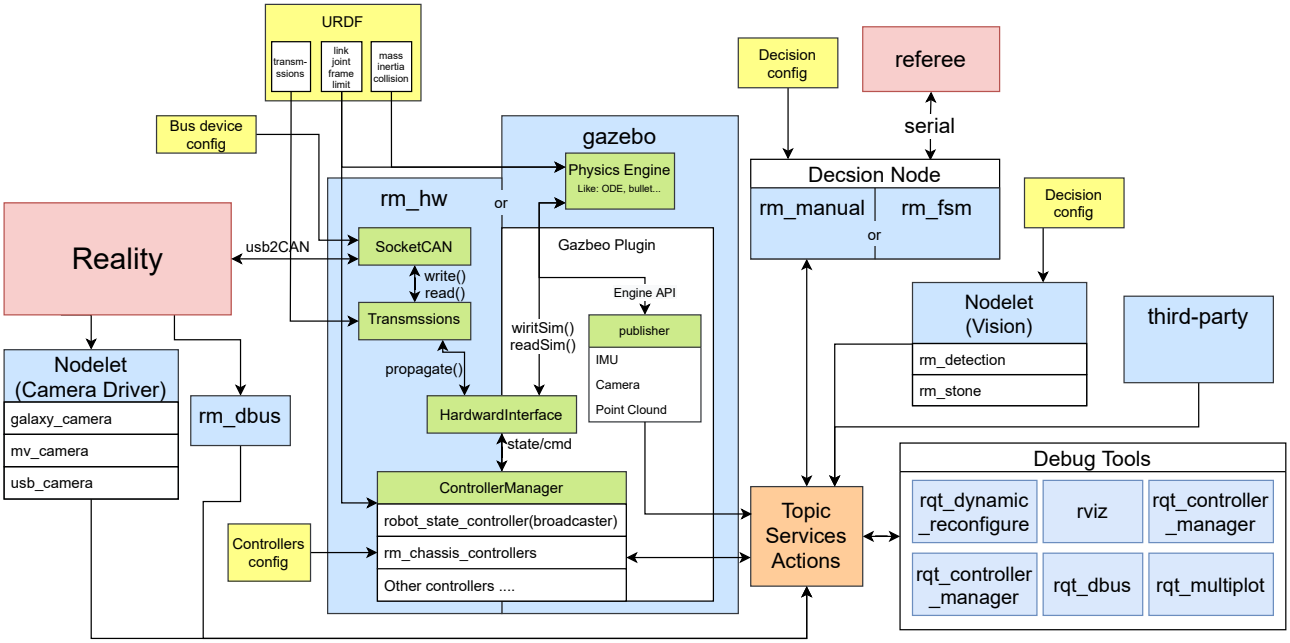


图 7.12: rm-contorls 框架示意框图；浅蓝色代表一个 Node（进程）；浅绿色代表类名或相关机制；浅黄色表示配置文件和数据；橙色为 ROS 通信中的话题、节点、动作；红色为实车。

图 7.12 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[9] 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。

7.1.6 常用控制器

7.1.6.1 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

7.1.6.2 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通过逆运动学计算出底盘各个电机的转速，经过 PID 控制器得到各电机扭矩指令，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）

7.1.6.3 rm_orientation_controller 控制器

rm_orientation_controller 会获取 Imu 的姿态，并利用这一姿态修正整个机器人的估计姿态。

硬件接口

1. `ImuSensorInterface` - 用于获取 Imu 姿态。
2. `RoboSateInterface` - 用于高频维护两个指定 link 坐标系之间的变换关系。

订阅话题

1. `/data` (sensor_msgs/Imu)

   获取 imu 姿态。

发布话题

1. `/tf` (tf/tfMessage)

   ROS tf 机制的默认话题接口

7.1.6.4 rm_gimbal_controllers 控制器

rm_gimbal_controllers 有`RATE`、`TRACK`、`DIRECT` 三种模式，它根据命令对 yaw 和 pitch 两个 joint 进行 PID 控制。在`TRACK`模式下还能使用子弹发射模型来解算 yaw 和 pitch 两个 joint 的目标位置。

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、指向目标及坐标系。

2. `/track` (rm_msgs/TrackData)

   接收`TRACK`模式下需要击打目标的位置和速度。

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

7.1.6.5 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。

7.1.6.6 mimic_joint_controller 控制器

对于工程机械臂的 joint1、2、3 来说，这些轴需要两个电机去驱动同一个轴，为了保证两个电机位置数据一样，mimic_joint_controller 启动后模仿电机将实时获得被模仿电机的位置数据并一直跟踪此位置数据。

硬件接口

1. `EffortJointInterface` - 用于发送模仿电机的扭矩指令
2. `JointStateInterface` - 用于获得被模仿电机的位置数据

7.1.6.7 gpio_controller 控制器

gpio_controller 用于读写 gpio。可根据命令控制 gpio 输出电平进而控制机器人真空泵电源的通断，从而实现对机器人吸盘的控制，还可以用于检测输入电平的高低。

硬件接口

1. `GpioStateInterface` - 用于获取 Gpio 当前状态
2. `GpioCommandInterface` - 用于发送 Gpio 的控制指令

订阅话题

1. `/command` (rm_msgs/GpioData)

   指定需要控制的 gpio 的名字和输出电平。

发布话题

1. `gpio_states` (rm_msgs/GpioData)

   使用到的所有 gpio 的名字、类型、电平。

7.1.6.8 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

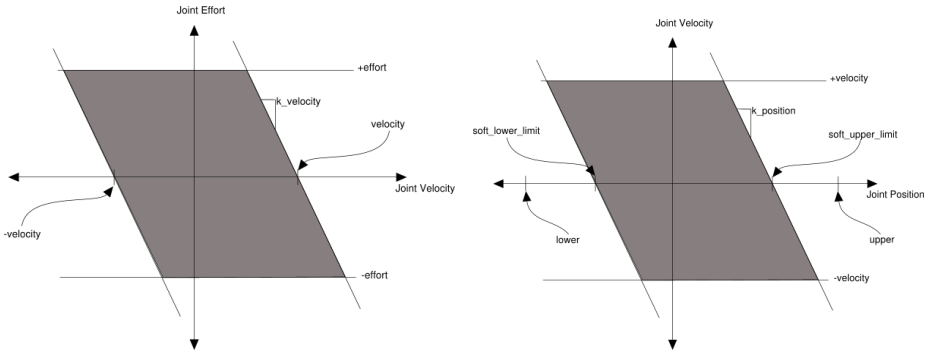
1. `is_calibrated` (control_msgs/QueryCalibrationState)

   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。

7.1.7 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** 7.13。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。



(a) 力矩限制　　(b) 速度限制

图 7.13: 软限位的限制方式，图源[10]

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 7.3: 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}"/>
```

7.1.8 软件测试

7.1.8.1 持续集成与持续部署

如 **图** 7.14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。

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

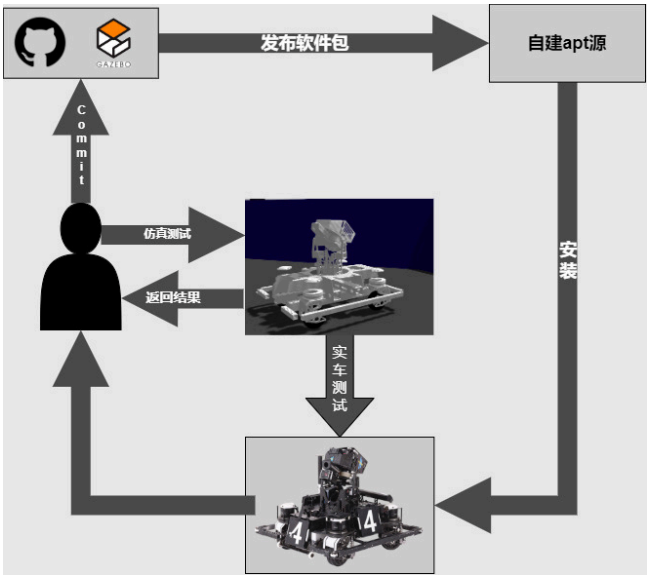


图 7.14: 持续集成与持续部署流程

- 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 7.15展示了借助 GitHub Actions 对代码进行持续集成。

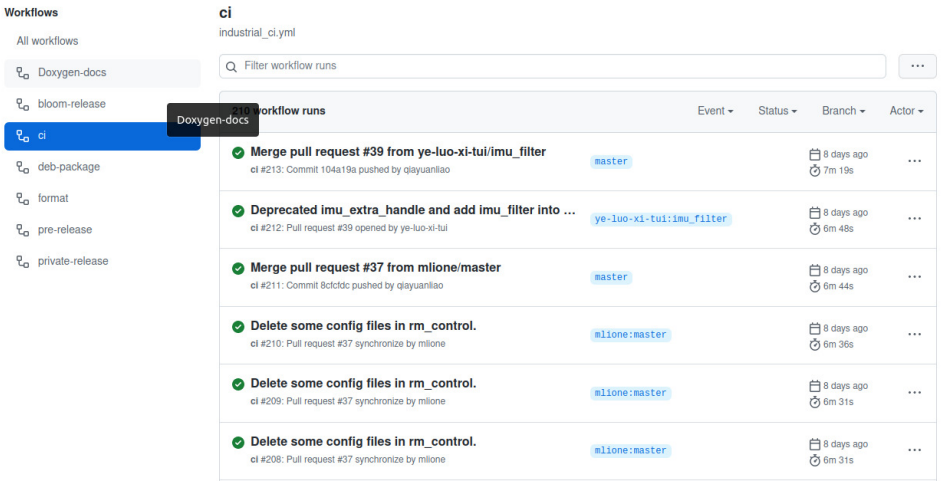


图 7.15: 借助 GitHub Actions 进行持续集成

以下展示使用两行指令完成底盘控制器的安装或更新。

Listing 7.4: 在机器人上更新软件

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

图 7.16为 rm_ci 运行某个单元测试后测试通过的结果。

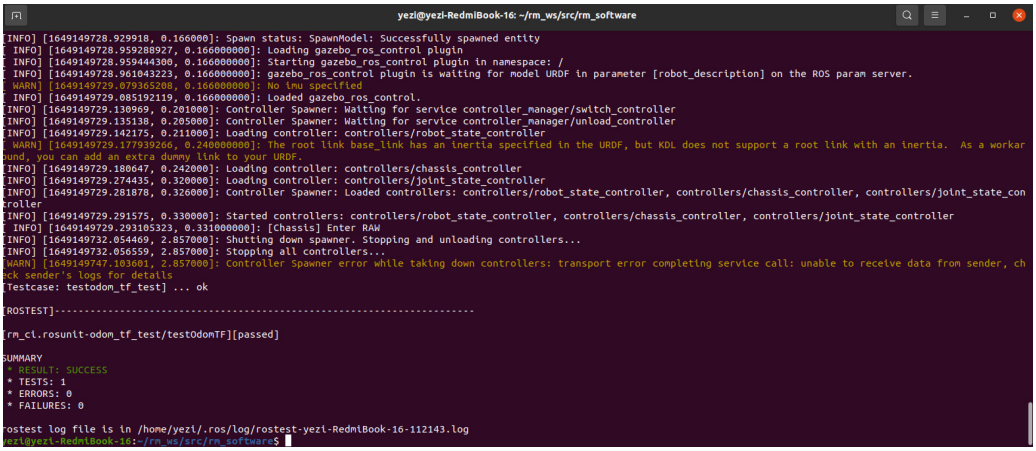


图 7.16: 单元测试运行结果

7.2 工程机器人

7.2.1 概述

工程机器人有专用的元包`rm_enginner`，包含以下包:

- engineer_arm_config: 用于机械臂运动规划的配置文件和 launch 文件
- engineer_arm_ikfast_plugin: 机械臂的逆运动学求解器
- engineer_middleware: 根据配置文件通过 topic 和 actions 的方式依次执行步骤序列的中间层

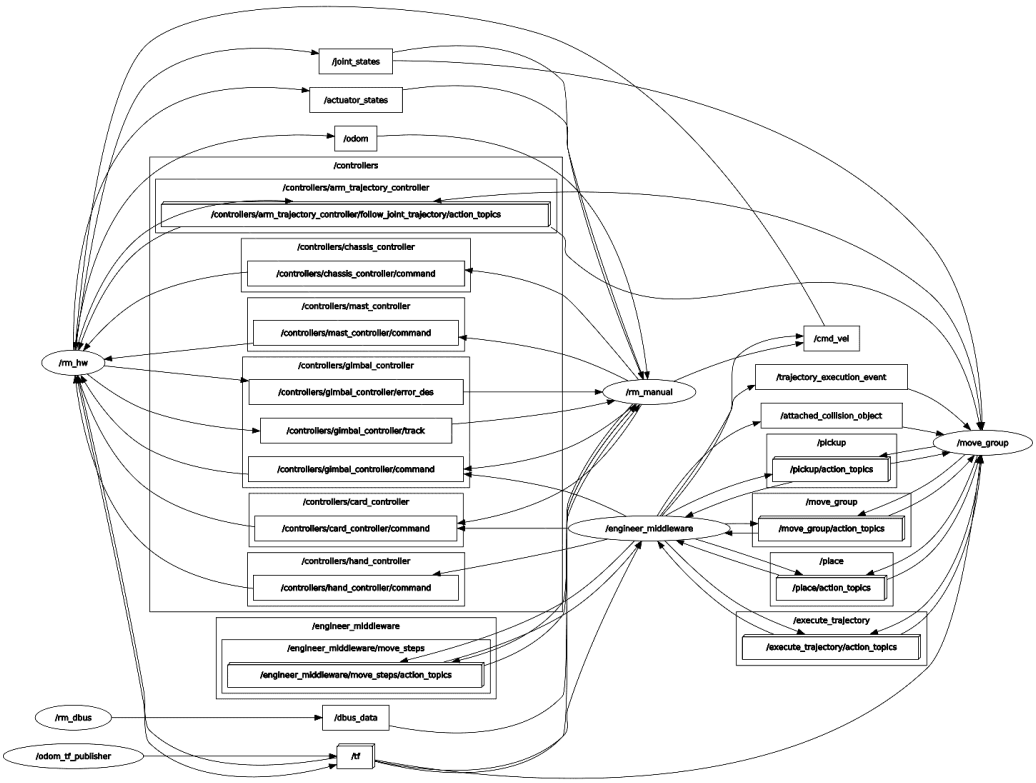


图 7.17: ROS 节点图

图 7.17 展示了工程机器人的 ROS 节点图，隐藏了不活动部分，并且里面没有显示 ROS 服务，除了 5.1 提到的节点，它还有以下三个节点：

- move_group: 机械臂轨迹规划并将轨迹发送给控制器
- engineer_middleware: 根据执行的步骤序列的名称得到动作序列通过话题、服务和动作顺序执行底盘、云台、机械臂、爪子的运动
- rm_maunal: 根据操作手和裁判系统数据通过话题、服务和动作来启停控制器、对控制器发送指令、开始和停止执行步骤序列

`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)
  - arm_trajectory_controller (effort_controllers/JointTrajectoryController)
  - gimbal_controller (rm_gimbal_controllers/Controller)
  - joint_mimic_controller (mimic_joint_controller/MimicJointController)
  - gpio_controller (gpio_controller/Controller)
  - drag_controller (effort_controllers/JointPositionController)
  - card_controller (effort_controllers/JointPositionController)
- 校准控制器
  - joint1_calibration_controller (rm_calibration_controllers/JointCalibrationController)
  - joint1_mimic_calibration_controller (rm_calibration_controllers/JointCalibrationController)
  - joint2_calibration_controller (rm_calibration_controllers/JointCalibrationController)
  - joint2_mimic_calibration_controller (rm_calibration_controllers/JointCalibrationController)
  - joint3_calibration_controller (rm_calibration_controllers/JointCalibrationController)
  - joint4_calibration_controller (rm_calibration_controllers/JointCalibrationController)
  - joint5_calibration_controller (rm_calibration_controllers/JointCalibrationController)
  - card_calibration_controller (rm_calibration_controllers/JointCalibrationController)
  - drag_calibration_controller (rm_calibration_controllers/JointCalibrationController)
  - yaw_calibration_controller (rm_calibration_controllers/JointCalibrationController)
  - pitch_calibration_controller (rm_calibration_controllers/JointCalibrationController)

7.2.2 MoveIt 机械臂运动规划

7.2.2.1 概述

我们使用了 MoveIt[11] 进行机械臂的运动规划。**图** 7.18 展示了通过 RViz 等可视化工具调用 MoveIt 的 `move_group` 实现在界面中拖动机械臂末端到达某一位姿后电机规划和执行机械臂本体就会运动到该位姿。而在实际机器人中，`engineer_middleware` 通过 `move_group` 提供的 actions 发送目标并让 MoveIt 进行运动规划并通过 arm_trajectory_controller 提供的 `follow_joint_trajectory` actions 使机械臂运动起来，见**图** 7.18。

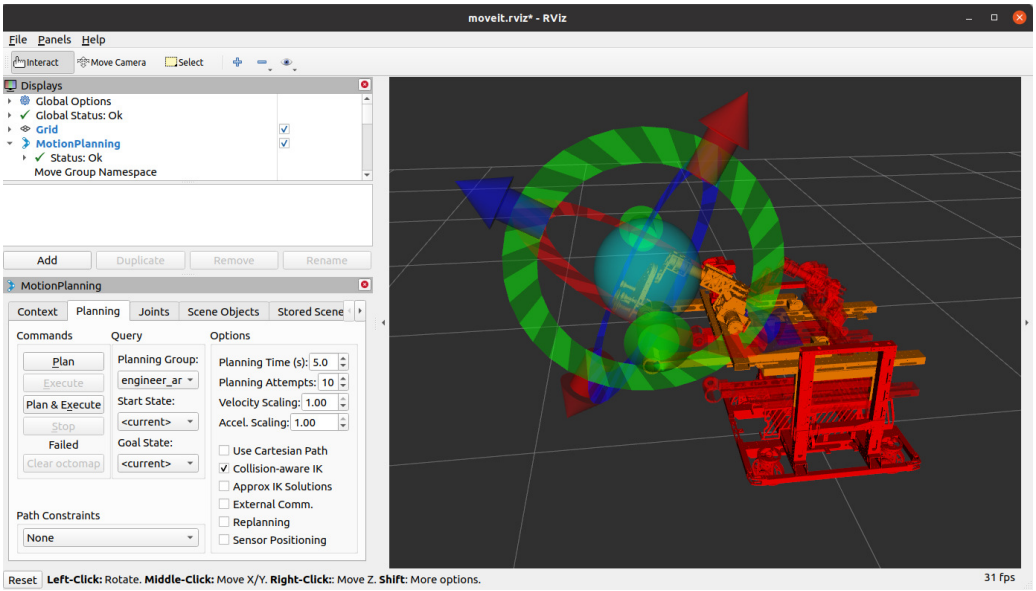


图 7.18: 通过 Rviz 调用 MoveIt

7.2.2.2 engineer_arm_ikfast_plugin

为了更好地完成抓取和兑换动作，机械臂末端三轴交于同一点，拥有六个自由度，我们使用 OpenRAVE[12] 提供的 `IKFast` 运动学编译器，IKFast 可以根据 URDF 分析任意链状机构的运动学，并自动生成优化过的求解器 c++ 代码，求解一次逆运动学只需要数微秒。

7.2.3 engineer_middleware 动作序列

7.2.3.1 概述

`rm_manual` 被设计为“将操作手的键鼠操作映射为机器人模块的指令”和一些简单的逻辑，并不适合实现动作序列。因此我们开发了一个中间层，`rm_manual` 通过 actions 通知 `engineer_middleware` 需要执行什么动作序列，并可以获取动作序列的执行进度，同时可以随时停止执行；`engineer_middleware` 根据动作序列的执行状态通过 rostopic 发送指令控制底盘、云台、吸盘，并通过 actions 调用 MoveIt 让机械臂运动。

7.2.4 动作序列构成

每个动作序列（`steps`）由多个 `step` 构成，每个 `step` 可以同时存在多种运动，每个运动都有它的属性。

7.2.4.1 通用运动属性

- 超时：当超过这个时间，运动执行失败
- 容许误差：当误差小于这个值，运动执行成功

7.2.4.2 MoveIt 运动属性

- 通用运动属性
- 速度：对最大速度的缩放

- 加速度：对最大加速度的缩放

7.2.4.3 机械臂末端运动属性

- MoveIt 运动属性
- 位置 (double[3])：期望位置
- 姿态 (double[3])：期望姿态（欧拉角）
- 坐标系：期望位态对应的基坐标系
- 笛卡尔：运动时是否走直线

7.2.4.4 机械臂关节运动属性

- MoveIt 运动属性
- 关节角度 (double[n])：期望 n 个关节的角度

7.2.4.5 场景规划属性

- 场景名称：决定了执行当前动作时候的场景状态

7.2.4.6 吸盘属性

- 状态：决定了吸盘的当前状态

7.2.4.7 底盘运动属性

- 位置：期望底盘运动到的位置
- yaw：期望底盘运动到的角度
- 坐标系：期望位姿的基坐标系

7.2.5 配置文件示例

7.5 展示了在大资源岛抢矿的时候，开始进入预备姿势对位并加入大资源岛的场景规划，操作手对位后，吸盘打开，机械臂末端前伸吸到矿石后举起矿石的动作。其中使用了 anchor 来化简配置文件；值得注意的是，机械臂运动和运动均可以指定坐标系，这样配合 `tf` 可以非常方便地接入视觉。

Listing 7.5: 取大资源岛矿石动作序列之一

```
common:
  speed:
    slowly: &SLOWLY
      speed: 0.2
      accel: 0.2
      timeout: 10
    normally: &NORMALLY
```

```
    speed: 0.5
    accel: 0.5
    timeout: 6.
  quickly: &QUICKLY
    speed: 1.
    accel: 1.
    timeout: 4.

gripper:
  open_gripper: &OPEN_GRIPPER
    state: false
  close_gripper: &CLOSE_GRIPPER
    state: true

tolerance:
  normal_tolerance: &NORMAL_TOLERANCE
    tolerance_joints: [ 0.009, 0.009, 0.009, 0.006, 0.1, 0.1, 0.1 ]
  bigger_tolerance: &BIGGER_TOLERANCE
    tolerance_joints: [ 0.014, 0.02, 0.02, 0.009, 0.2, 0.2, 0.2 ]
steps_list :
  MID_BIG_ISLAND0:
  - step: "add␣scene␣of␣sky␣island"
    scene_name: "BIG_STONE_ISLAND"
  - step: "gimbal"
    gimbal:
      <<: *BIG_STONE_POS
  - step: "ready"
    arm:
      joints: [ 0.238, 0.170, 0.3, 0,1.6878, 0, 0 ]
      <<: *NORMALLY
      <<: *NORMAL_TOLERANCE

  MID_BIG_ISLAND00:
  - step: "add␣scene␣of␣sky␣island"
    scene_name: "BIG_STONE_ISLAND"
  - step: "gimbal"
    gimbal:
      <<: *BIG_STONE_POS
  - step: "␣gripper"
```

```
    gripper:
      <<: *OPEN_GRIPPER
  - step: "ready"
    arm:
      joints: [ 0.238, 0.185, 0.219, 0,1.6878, 0, 0 ]
      <<: *NORMALLY
      <<: *NORMAL_TOLERANCE
  - step: "ready"
    arm:
      joints: [ 0.238, 0.32, 0.3, 0,1.6878, 0, 0 ]
      <<: *NORMALLY
      <<: *BIGGER_TOLERANCE
  - step: "ready"
    arm:
      joints: [ 0.238, 0.28, 0.27, 0,1.6878, 0, 0 ]
      <<: *SLOWLY
      <<: *NORMAL_TOLERANCE
  - step: "ready"
    arm:
      joints: [ 0.238, 0.28, 0.27, 0,0.696, 0, 0 ]
      <<: *SLOWLY
      <<: *NORMAL_TOLERANCE

MID_BIG_ISLAND000:
  - step: "add␣scene␣of␣sky␣island"
    scene_name: "BIG_STONE_ISLAND"
  - step: "gimbal"
    gimbal:
      <<: *WALKING_POS
  - step: "move␣to␣mid␣ready"
    arm:
      <<: *NORMAL_HOME
```

7.2.6 动作序列示例

下图为动作序列示例，均为视频生成的等时间间隔照片。

图 7.19: 工程机器人吸取地面矿石

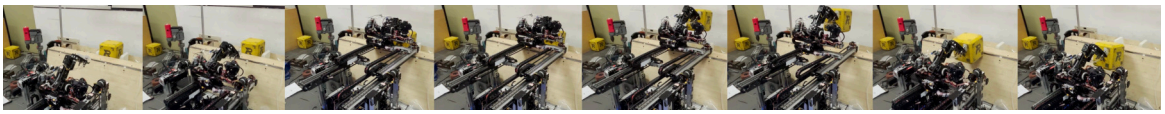


图 7.20: 工程机器人吸取资源岛矿石

图 7.21: 工程机器人空接资源岛矿石

图 7.22: 工程机器人存放与拿取矿仓中的矿石

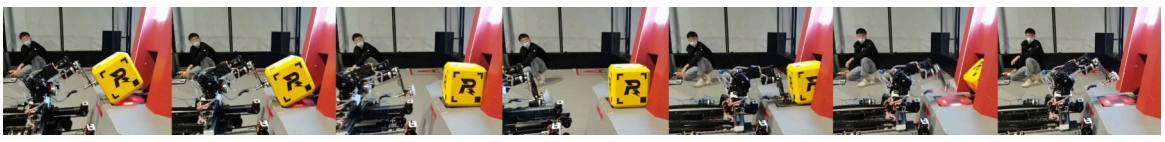


图 7.23: 工程机器人兑换矿石

第 8 章 视觉算法

8.1 算法相关的主要理论

8.1.1 数字图像处理

8.1.1.1 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[13]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：

$$H = \begin{cases} 0^\circ & ,\Delta = 0 \\ 60^\circ * (\frac{G'-B'}{\Delta} + 0) & ,C_{max} = R' \\ 60^\circ * (\frac{B'-R'}{\Delta} + 2) & ,C_{max} = G' \\ 60^\circ * (\frac{R'-G'}{\Delta} + 4) & ,C_{max} = B' \end{cases} \tag{8.1}$$

$$S = \begin{cases} 0 & ,C_{max} = 0 \\ \frac{\Delta}{C_{max}} & ,C_{max} \neq 0 \end{cases} \tag{8.2}$$

$$V = C_{max} \tag{8.3}$$

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。

8.1.1.2 形态学处理

形态学图像处理（简称形态学）[14]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。

8.1.1.2.1 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：

$$dst(x, y) = \max_{(x',y'):element(x',y')\neq 0} src(x + x', y + y') \tag{8.4}$$

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。

8.1.1.2.2 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：

$$dst(x,y) = \min_{(x',y'):element(x',y')\neq 0} src(x+x', y+y') \tag{8.5}$$

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。

8.1.1.2.3 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：$f \circ s = (f \ominus s) \oplus s$

8.1.1.2.4 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。

8.2 算法说明

8.2.1 矿石识别代码分析

矿石识别代码用于辅助工程机器人夹取矿石，但目前鲁棒性较差。

8.2.1.1 算法流程

8.2.1.1.1 阈值分割 色彩分割采用多通道阈值分割。在不同的灯光环境下，摄像头获取的图像的中矿石的颜色会不同，所以色彩分割的阈值需要根据当前环境调整阈值。

8.2.1.1.2 寻找矿石轮廓 筛选出有具有内轮廓的轮廓，求出这些轮廓的中点O，筛选出所有内轮廓最远离中点的点P，求出轮廓的中点O与各内轮廓最远点P相连所成的直线l，然后求出此内轮廓在直线l两侧距离直线的最远点M和N，接着根据得到的三个点判断判断直线PM和直线PN所成夹角是否接近直角并判断内轮廓是否为正方形，如果判断内轮廓存在直角或判定为正方形，则判定此轮廓为矿石轮廓，将所得到的点从最靠近左上角的点开始按逆时针排序作为内轮廓的特征点。

8.2.1.1.3 寻找特征点 优先使用正方形内轮廓的四个点否则使用带直角的内轮廓的构成正方形的四个点`rect`（第四个点与上一步所求得的三个点构成近似正方形的四个点）做 solvepnp 得到三个 ROI，判断轮廓的中点`0`是否在上述正方形`rect`中点和其他三个 ROI 的中点所围成的区域内，若在则顺序判断 ROI 里，是否包含上一步得到的内轮廓，若包含，则存储对应内轮廓的特征点，由于其他矿石特征内轮廓与正方形`rect`的相对位置有四种可能的情况，所以以上过程循环 4 次。

8.2.1.1.4 获得矿石位姿 用上一步过程中得到的特征点进行 solvepnp 得到矿石相对相机的位姿，如果是 4 份内轮廓的特征点，则通过仿射变换求出矿石中心图案的面积大小调整旋转向量，使求得的矿石中心为原点的右手坐标系的 z 轴朝向无图案的矿石表面，否则 z 轴朝向本次使用的内轮廓所在的矿石表面，最后广播距离摄像头最近的矿石的旋转向量和平移向量。

8.3 代码设计

矿石识别代码所在的 ROS 功能包名为`rm_stone`，整套程序使用 c++ 开发，功能包当中最主要的核心代码文件为：

1. `rm_stone.cpp`: 主函数，获取图片，创建并调用下面的三个函数的类对象用以处理图片、发布矿石位姿；
2. `stone_get_features.cpp`: 算法主体，获取用于最终 solvepnp 的特征点；
3. `stone_pose_solver.cpp`: 进行最终 solvepnp；
4. `stone_process.cpp`: 对图片进行预处理；

第 9 章 人机交互

9.1 自定义 UI

9.1.1 界面展示

工程机器人的 UI 示意图，如**图** 9.1所示。

图 9.1: 工程操作手自定义 UI 界面

9.1.2 功能说明

整个自定义 UI 分为四大部分，分别用于显示机械臂动作序列，操作状态，警告信息，以及抓取辅助线。

9.1.2.1 机械臂动作序列

位于操作手界面**左上方**，共显示 3 条信息，分别是 **STEP**，**QUEUE**，**PROGRESS**。

- **STEP** 用于显示机械臂当前序列正在执行的步骤名称。
  - 内容：当前序列正在执行的**步骤名称**；
  - 颜色：无特殊含义。
- **QUEUE** 用于显示机械臂当前动作序列的名称。
  - 内容：机械臂当前**动作序列名称**；
  - 颜色：无特殊含义。
- **PROGRESS** 用于显示机械臂当前序列完成度。
  - 内容：**序列完成度的百分比**；
  - 颜色：无特殊含义。

9.1.2.2 操作状态

位于操作手界面**右上方**，共显示 4 条信息，分别是 **CARD，DRAG，EFFORT，GRIPPER**。

- **CARD** 用于显示当前是否已经伸出救援卡。
  - 内容：救援卡状态（**ON/OFF**）；
  - 颜色：无特殊含义。
- **DRAG** 用于显示当前是否已经伸出爪子拖拽。
  - 内容：拖拽位置（**ON/OFF**）；
  - 颜色：无特殊含义。
- **EFFORT** 用于显示机器人上当前的最大电机力矩及对应电机名称。
  - 内容：最大电机力矩及对应电机名称。显示格式为 **“电机名称：力矩”**；
  - 颜色：当最大力矩低于 10N.m 时，为**绿色**；最大力矩大于 20N.m 时为**橙色**；其余情况为**黄色**
- **GRIPPER** 用于显示当前吸盘状态。
  - 内容：吸盘状态 **ON/OFF**；
  - 颜色：无特殊含义。

9.1.2.3 警告信息

位于操作手界面**右上方**，共显示 2 条信息，分别是 **CALIBRATION，CARD**。

- **CALIBRATION** 机器人重新上电后，以闪烁的形式提醒操作手进行机械臂校准。
  - 内容：**“PLEASE CALIBRATE!!"**；
  - 颜色：无特殊含义。
- **CARD** 救援卡卡住时，以闪烁的形式提醒操作手。
  - 内容：**“CARD STUCK!!"**；
  - 颜色：无特殊含义。

9.1.2.4 抓取辅助线

位于操作手界面**对应位置**，用于辅助操作手抓取、空接、矿石。

- 内容：线段；
- 颜色：无特殊含义。

9.1.3 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`，`\newline 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 9.1: engineer.yaml

```
rm_manual:
  ui:
    trigger_change:
      - name: "step"
        config: { start_position: [ 400, 750 ], size: 15, width: 2, title: "step: ", color: "white" }
      - name: "queue"
        config: { start_position: [ 400, 700 ], size: 15, width: 2, title: "queue: ", color: "yellow" }
      - name: "long_card"
        config: { start_position: [ 1300, 750 ], size: 15, width: 2, title: "long card: ", color: "
            green" }
      - name: "short_card"
        config: { start_position: [ 1300, 700 ], size: 15, width: 2, title: "short card: ", color: "
            green" }
      - name: "drag"
        config: { start_position: [ 1300, 650 ], size: 15, width: 2, title: "drag: ", color: "green" }
      - name: "gripper"
        config: { start_position: [ 1300, 600 ], size: 15, width: 2, title: "drag: ", color: "green" }
    fixed:
      - name: "island_lf_line"
        config: { type: "line", width: 2, color: "white",
                  start_position: [ 395, 4862 ], end_position: [ 455, 201 ] }
      - name: "island_rt_line"
        config: { type: "line", width: 2, color: "white",
                  start_position: [ 738, 186 ], end_position: [ 740, 491 ] }
      - name: "ground_lf_line"
        config: { type: "line", width: 2, color: "yellow",
                  start_position: [ 1206, 921 ], end_position: [ 1125, 476 ] }
      - name: "ground_rt_line"
        config: { type: "line", width: 2, color: "yellow",
                  start_position: [ 1510, 390 ], end_position: [ 1788, 810 ] }
    time_change:
      - name: "progress"
        config: { start_position: [ 400, 650 ], size: 15, width: 2, delay: 0.3, color: "green", title:
            "progress:" }
    flash:
      - name: "calibration"
        config: { start_position: [ 850, 700 ], size: 15, width: 2,
                  color: "yellow", content: "please calibrate!!", delay: 0.8 }
      - name: "card_warning"
        config: { start_position: [ 850, 750 ], size: 15, width: 2,
                  color: "green", content: "card stuck!!", delay: 0.8 }
```

9.1.3.0.1 trigger_change

该类 UI 只在触发机器人状态变化时刷新，在工程机器人中对应上文提到的**机械臂动作序列，操作状态** 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；

9.1.3.0.2 time_change

该类 UI 按固定频率刷新，在工程机器人中对应上文提到的**机械臂动作序列，操作状态** 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，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，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：线条颜色，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；
  - title：可选参数，代表字符串标题；

9.1.3.0.3 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，在工程机器人中对应上文提到的**抓取辅助线** 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；

9.1.3.0.4 flash

该类 UI 按指定的时间间隔以闪烁形式刷新，在工程机器人中对应上文提到的**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，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，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；

第 10 章 调试工具

10.1 rqt 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。本节与英雄机器人重复。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。

10.1.1 Home

本界面如 **图** 10.1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。

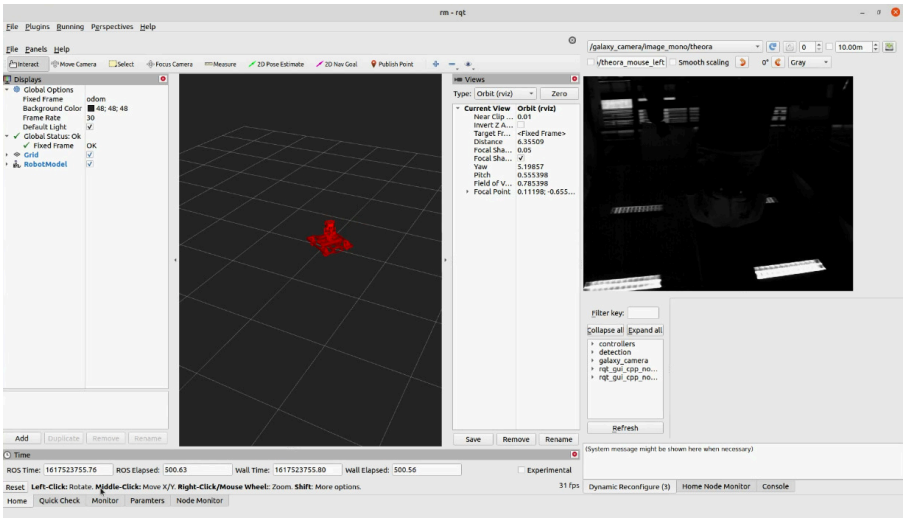


图 10.1: Home 界面

10.1.2 Quick Check

本界面用于快速检查代码运行状态，如 **图** 10.2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。

10.1.3 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**。

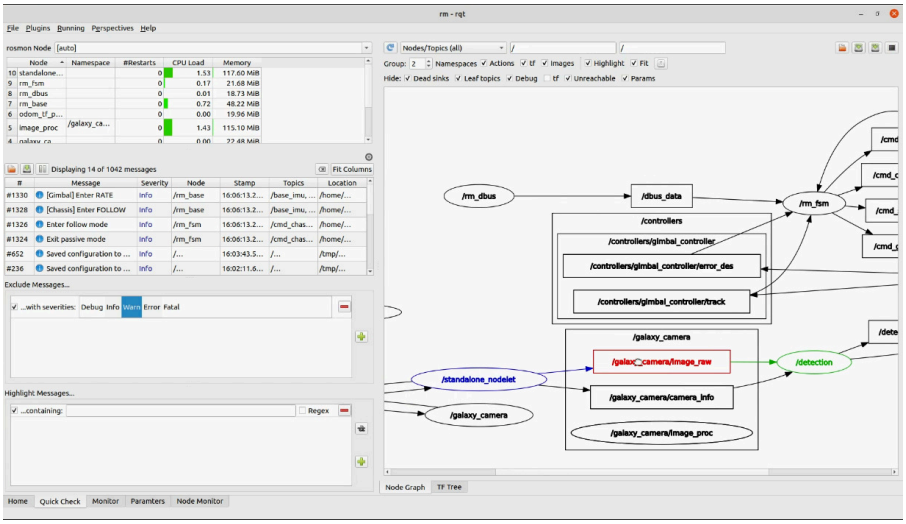


图 10.2: Quick Check 界面

10.1.3.1 Joint

本界面用于显示电机的速度和力矩，如 **图** 10.3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。

10.1.3.2 Imu

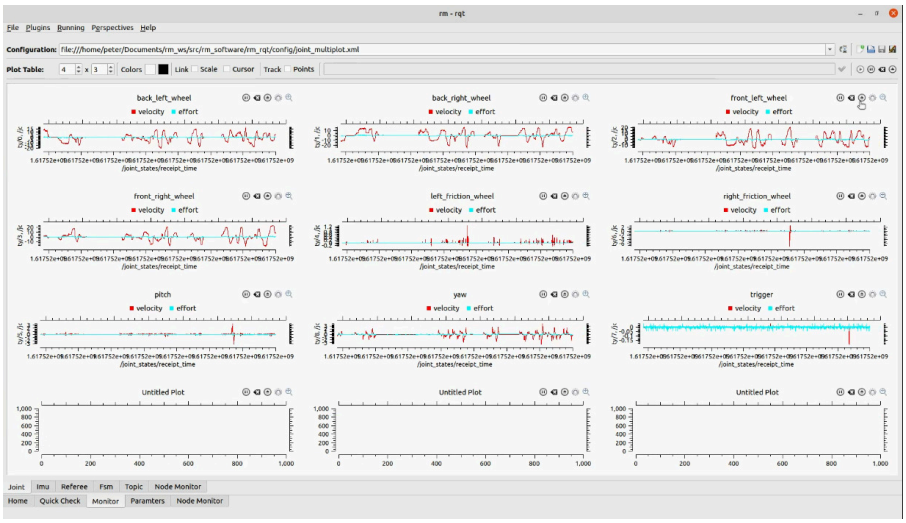
本界面用于显示 IMU 返回的数据，同样也是通过 rqt_multiplot 插件绘制出 IMU 返回的姿态信息。功能与 Joint 界面类似，此处不再重复说明。

10.1.3.3 Referee

本界面用于显示裁判系统的各项信息，如 **图** 10.4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。

10.1.3.4 Topic

本界面用于显示机器人运行时所有话题下的消息，如 **图** 10.5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。



(a) 多电机

(b) 单电机

图 10.3: Joint 界面

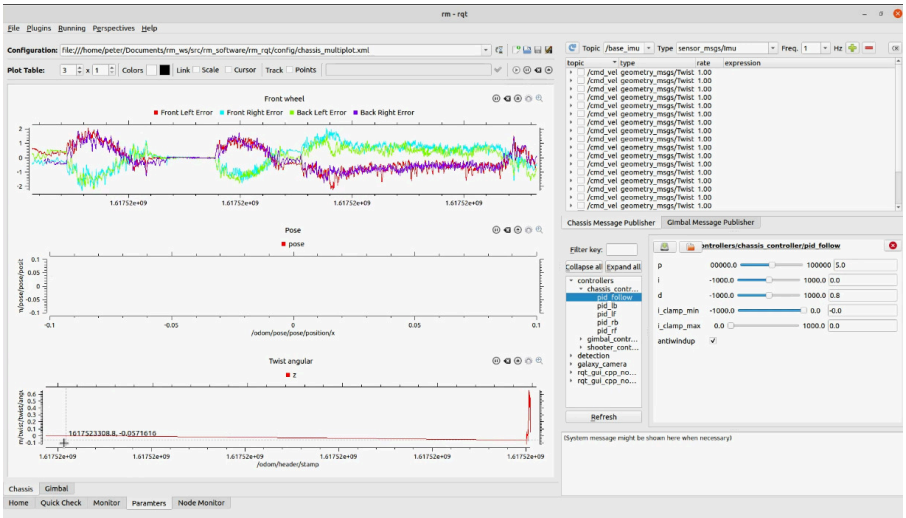


图 10.4: Referee 界面

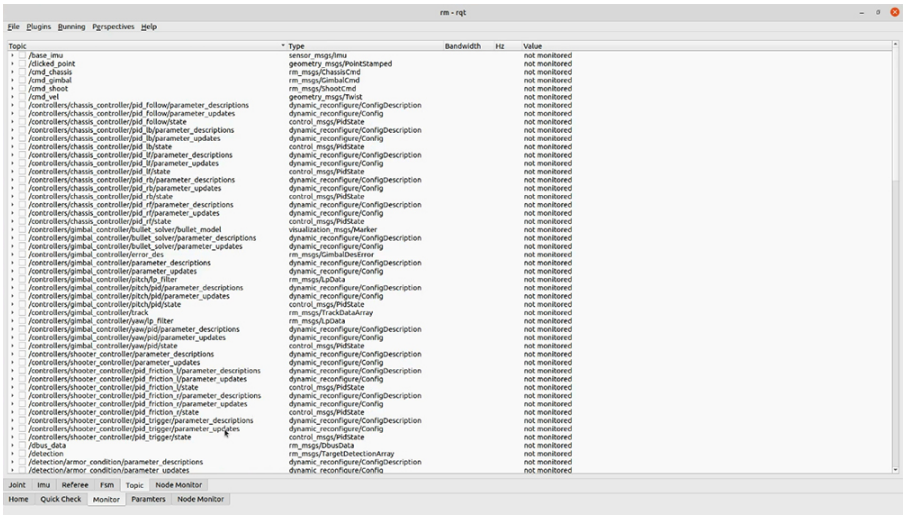


图 10.5: Topic 界面

10.1.4 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**。

10.1.4.1 Chassis

本页面用于调试底盘控制器的参数，如 **图** 10.6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。

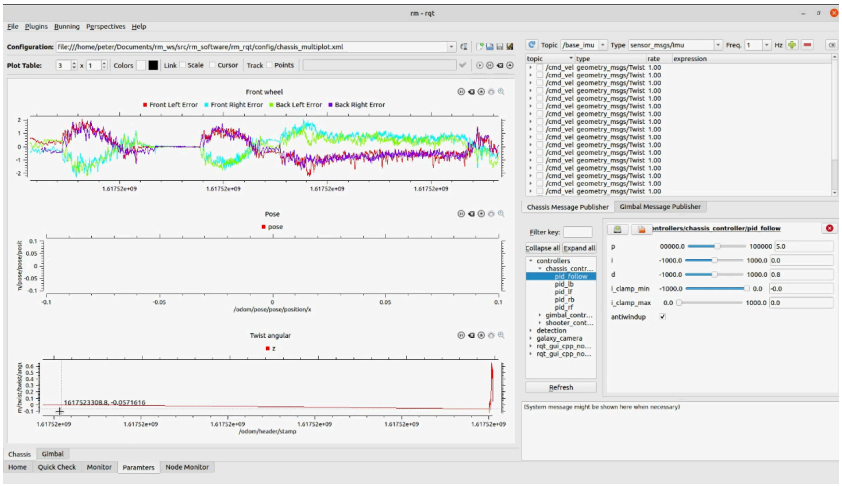


图 10.6: Chassis 界面

10.1.4.2 Gimbal

本页面用于调试云台控制器的参数，如 **图** 10.7所示。具体功能与 **Chassis** 界面类似，在此不再重复说明。

10.1.4.3 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。

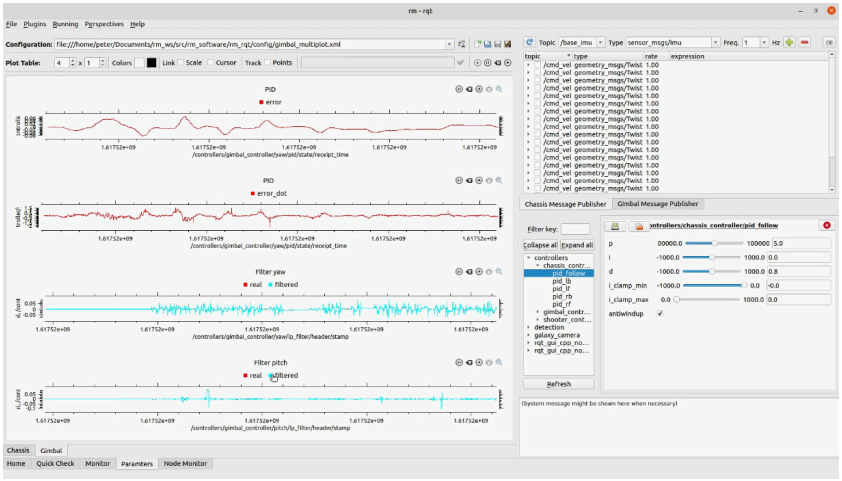


图 10.7: Gimbal 界面

第 11 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 3 组）
庾日熙	机械设计制造及其自动化、大二、机械组主力队员	整体机器人的底盘、机械臂结构设计及加工装配等	（机械）60%
梁伟聪	机械电子工程、大三、队长	机械臂结构设计	（机械）24%
向如成	机械设计制造及其自动化、大一、机械组梯度队员	整体机器人的底盘、机械臂加工装配等	（机械）4%
王一汀	机械设计制造及其自动化、大一、机械组梯度队员	整体机器人的底盘、机械臂加工装配等	（机械）4%
梁宇乐	机械设计制造及其自动化、大一、机械组梯度队员	整体机器人的底盘、机械臂加工装配等	（机械）4%
张展荣	机械设计制造及其自动化、大一、机械组梯度队员	整体机器人的底盘、机械臂加工装配等	（机械）4%
吕骏骐	自动化、大二、控制组主力队员	负责实现整体机器人的控制	（控制）45%
廖洽源	机械电子工程（创新班）、大三、技术顾问	机器人控制程序架构的搭建、指导机器人开发	（控制）45%
穆跃鑫	自动化（机器人学院）、大三、控制组主力队员	gpio_controller 的开发	（控制）10%
招尚霖	微电子科学与工程、大三、电路组主力队员	机器人整体电路框架的搭建及检修	（电路）50%
陈麒铵	自动化、大二、电路组主力队员	机器人整体电路框架的搭建及检修	（电路）50%

表 11.1: 团队成员贡献

第 12 章 研发迭代过程

12.1 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：拖拽救援结构测试、刷卡救援测试、1 轴重力补偿结构测试、舵轮轮系测试、改减速比末端 3 轴测试	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，最后选择使用舵轮井字型分布底盘，再结合独立悬挂、方形保护外框、拖拽救援、刷卡救援	2022.01.07
V1.5	第 1 代机械臂设计：1 轴升降、2 轴旋转副,3 轴左右平移，末端 3 轴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、自制舵轮的替换、拖拽救援机构和刷卡救援机构的替换	2022.02.25
V2.5	第 2 代机械臂设计：重新指定机械臂方案、替换了升降机构、增大 2 轴延伸臂的行程和刚度，将原来的旋转轴改为平移轴，提高了机械臂方案的可行性和稳定性	2022.03.20

表 12.1: 版本迭代过程记录

12.2 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
升降机构易发生倾斜刚性不足	因为尝试使用的碳管和塑料（本着减轻重量的目的）时表现出明显刚性不足	用铝管和滑块滑轨实现平移，刚度大大提高	V0.2	（机械）庾日熙
前后延伸臂用丝杆传动，齿轮与丝杆经常发生松动的情况	丝杆与齿轮连接不可靠	将丝杆传动换成链条传动，提高了衍生臂的稳定性。	V0.3	（机械）庾日熙
两边舵轮会有外八的情况	舵电机固定板与侧板只用 4 个板板连接件连接，导致刚度不足	侧板添加加强筋支撑舵电机固定板。	V0.4	（机械）庾日熙

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
刷卡救援用卷尺输出，在容易受起伏路段的干扰	卷尺的刚度不足，在伸出一定距离后会下垂	将其改成齿轮齿条传动，通过线轨输出。	V2.0	（机械）庾日熙
矿石被吸盘吸附住之后，会发生前倾，移动过程中摇晃剧烈	硅胶吸盘的刚度不足	通过增加两块碳板夹住吸盘，两块碳板上下之间通过铝柱连接，从而避免了吸盘前倾	V2.0	（机械）梁伟聪
2 轴延伸臂与 3 轴左右平移固定不稳	连接螺丝直接连接滑块上，滑块与导轨之间需要调节平行度，装的时候需要调节螺丝的松紧，导致连接不可靠	用一块碳板单独连接 3 轴与 2 轴上的滑块，改变了原有的装配顺序，先调节滑块的平行度再去连接 3 轴。	V2.5	（机械）庾日熙
拖拽救援刚度不足	使用两块不同厚度波纤板叠加起来	将 3 块波纤板改为一块 7mm 碳板增强爪子的刚度	V2.5	（机械）庾日熙
矿仓与前后延伸臂上的拖链干涉并且存矿时会卡矿石	设计经验不足	减小矿仓壁的之间距离，并在矿仓壁上添加轴承和在矿仓底部贴上特氟龙胶布	V2.5	（机械）庾日熙

表 12.2: 重点问题解决记录

参考文献

[1] RM2021-东北大学-T-DT 战队-工程机器人-机械结构开源[Z]. https://bbs.robomaster.com/forum.php?mod=viewthread&tid=12291. (Accessed on 08/10/2022).

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

[5] Transmission_interface: transmission_interface::DifferentialTransmission Class Reference[Z]. http://docs.ros.org/en/jade/api/transmission_interface/html/c++/classtransmission__interface_1_1DifferentialTransmission.html. (Accessed on 08/15/2021).

[6] Gazebo : Tutorial : ROS control[Z]. http://gazebosim.org/tutorials/?tut=ros_control. (Accessed on 08/15/2021).

[7] RTwiki[Z]. https://rt.wiki.kernel.org/index.php/Main_Page. (Accessed on 08/19/2021).

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

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

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

[11] COLEMAN D, SUCAN I, CHITTA S, et al. Reducing the barrier to entry of complex robotic software: a moveit! case study[J]. ArXiv preprint arXiv:1404.3785, 2014.

[12] DIANKOV R, KUFFNER J. Openrave: A planning architecture for autonomous robotics[J]. Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34, 2008, 79.

[13] Color Space Conversions[Z]. https://docs.opencv.org/4.2.0/d8/d01/group__imgproc__color__conversions.html. (Accessed on 08/23/2021).

[14] Image Filtering[Z]. https://docs.opencv.org/4.2.0/d8/d01/group__imgproc__filter.html. (Accessed on 08/23/2021).

[15] Ros-industrial/industrial_ci: Easy continuous integration repository for ROS repositories[Z]. https://github.com/ros-industrial/industrial_ci. (Accessed on 08/16/2021).