

前 言

第一章： 第一次亲密接触 **MT-AR**

第二章： MT-AR 机器人机械系统

第三章： MT-AR 机器人电子系统

第四章： MT-AR 机器人软件系统

第五章： MT-AR 智能型机器人开发指南

结 束 语



前 言

智能机器人为我们展现了未来美好生活的绚丽景象，同时，智能机器人也将具备与人一样的智能、一样甚至更多的劳动能力，在此过程中，智能机器人将吸取更多学科发展的最新成果，不断为我们带来新的技术和应用。

作为技术和学术研究的平台，我们立足提供更为开放的系统、模块化的系统、稳定的系统；作为大学生及高年级学生课程教学平台，我们立足提供更为多的项目实践机会，让学生学到更多、更为实用的技术和知识。

总之，我们愿意为智能机器人时代的来临，提供更多学习、交流、发展、共同升华的机会！公司的发展始终与大家的进步息息相关，我们共同期待着机器人时代的来临。

下面是研究型智能机器人标准配置清单：

项目	技术参数
本 体	
架构	三层模块化结构：信息传感与交互层；信息处理与规划层；信息执行与能源层
尺寸	49.5 cm ×48.0 cm×60.0 cm
重量	30Kg
电 源	
电池	24V, 20AH 动力锂电池（一组）
充电时 间	5 小时
运行时 间	4 小时
电 源 控 制 板	电源反接保护，过流保护、欠压保护、声光报警；可以实现电池与外接电源任意切换，切换时主机不会重启；具备电量显示功能（外部显示）。
外接电 源	DC 24V, 21A
充电器	
运动特性	
电 机	24V,70W MAXON 电机（2 个）
电 机 驱 动 器	驱动功率 240W，供电电压：18~36V，过流、过热保护，实时反馈电流值，模拟量/PWM 控制，控制频率>50KHz。
运 动 控 制 卡	专用高速数字信号处理器（DSP）+复杂可编程器件（CPLD）构成的两轴以上运动控制卡；既可以输出模拟量控制电机，也可以采用 PWM 信号控制电机；硬件采用工业级抗干扰设计；底层通讯总线采用抗干扰特性好、多主通讯模式的 CAN 总线，与主机采用 CAN、RS-232 串口通信方式。
减速比	33: 1
执行结 构	轮式结构
最大速 度	2.5m/s



推 力	10kg
运动地形:	室内地板、地砖、地毯
电子及传感器	
声纳	6 个 (20cm-7m)
PSD	6 个 (8mm-80mm)
编码器	500 线
计算机	工业级嵌入式主板, 工业级设计结构 (提供嵌入式工业级主板抗干扰、防静电测试报告), 包括散热系统 (4 个滚珠风扇)、静电保护、抗振设计等, 具备通用带反向保护的电气接口、信号接口以及机械固定装置。采用 INTEL PM1.8G 低功耗、高速处理器; SATA 80G 高速硬盘, DDR 512M 内存; 6 个串口 (包括 485、422 通信接口), 8 个独立 USB 接口, PCI 插槽; 双通道 RTL8110S 自适应以太网口
无线通讯	54M
A/D	可外接 32 路 A/D (0-5V) (16 路入/16 路出)
摄像机	130 万像素、30 帧/秒, USB 接口

技术支持:

需要技术支持, 请发 E-Mail: server@ingenious.cn。技术支持电话: 021-64858521。



第一章： 第一次亲密接触 MT-AR

1.1 安全事项

1.1.1 正确使用电池

电池充电一定要使用专用的 MT-AR 充电器；

机器人充电一定要有专人负责；

电池长时间不用时，为了保持电池的性能，每隔一周需要对电池重新充电；

电池充电时间在 6-7 个小时；

在对电池充电的同时，最好不要使用电池对 MT-AR 机器人供电。此时应该使用外接开关电源对 MT-AR 机器人供电。

1.1.2 正确使用机器人的急停开关

机器人开机和关机之前，为了保证安全，请将急停开关断开（即将急停开关按下）；

一旦机器人程序死机，请先将急停开关断开，然后重新启动机器人系统；

1.1.3 正确拆装机器人系统

用户应该按照培训要求正确拆装机器人；

为了保证机器人电子系统的安全，请不要将导电的东西掉入机器人内部，否则将导致系统的崩溃，甚至引起火灾；

1.1.4 安全调试机器人系统

用户在调试机器人时，请将机器人的速度设计在低速区；机器人运行时，请将非工作人员远离机器人；

1.1.5 安全使用外接电源

MT-AR 机器人带有外接电源接口，请使用上海英集斯自动化技术有限公司提供的专用外接电源；

外接电源和电池之间可以相互切换。当接上外接电源时，机器人的电源由外接电源提供，电池的供电线路被自动切断；当拔掉外接电源时，机器人的供电电源自动切换到由机器人的内部电池进行供电。



1.1.6 正确加装负载设备

MT-AR 机器人的负载能力在 30 公斤左右。加装超过此重量的负载，机器人的齿轮头和电机将可能被损坏；要避免负载对 MT-AR 机器人的冲击。

1.1.7 重视备份系统数据

电池的电压比较低时（低于 22.5V），电源控制板将自动切断电机的供电电路，此时计算机虽然能运行，但是机器人不会运动。只有电池电压恢复到 24.5V 以上时，机器人的电机驱动部分才能获得能源。如果电池电压不能恢复到 24.5V 以上，机器人的蜂鸣器将不断地报警，同时欠压指示等将被点亮。这时应该保护好系统数据，对机器人充电。

1.1.8 不要随意拔插机器人的接线

MT-AR 机器人的各种连线是有方向的，而且连线与接口是对应的。如果接错或者插反都可能导致灾难性的事故。因此用户（尤其是初级用户）在使用时，千万要小心。

当连接线不能轻松插进去时，请检查是否插错，千万不要硬插进去。

不要带电拔插各种连接线。

1.2 MT-AR 功能介绍

1.2.1 产品定位

1. 学术与技术研究平台——老师、研究生学术研究的好助手；
2. 大学生课程设计、毕业设计 & 创新平台；

1.2.2 产品功能

1. 图像、视频：MT-AR 可通过本体配置的 USB 摄像头采集视频图像，获取视频数据，用户可对该视频数据进行处理，包括图像的前处理、提取图像的各种特征、视频压缩与编码等研究；

2. 有线\无线网络（远程控制及监督）：可利用 MT-AR 本体的无线\有线网络的基础上研究多智能体系统以及基于网络技术的远程机器人控制技术研究；

3. 语音：建立了机器人的语音类库函数，包括语音的采集、编辑、识别，用户可对机器人进行语音控制以及做语音识别等方面的研究；

4. 运动控制：用户可通过选择各种运动模式，实时、准确的控制 MT-AR 机器人运动。同时，可针对非完整约束两轮驱动控制的特点，做运动闭环、运动学、



动力学等方面的研究;

5. 声纳测距: MT-AR 本体等间隔分布的 6 个声纳可以测定机器人周围障碍物的距离, 其检测范围为 20cm 到 7m, 用户可以根据传感器的采样数据建立机器人的导航图谱, 进行机器人自主避障、路径规划和多传感器融合技术等方面的研究;

6. 红外测距: MT-AR 本体等间隔分布的 6 个红外传感器可以测定机器人周围障碍物的距离, 其检测范围为 8cm 到 80cm;

7. 舵轮协调控制: MT-AR 可通过 SCI、CAN 总线伺服控制一轴舵轮, 采用绝对值编码器, 其分辨率为 0.17 度。同时, 左右驱动轮可与舵轮协调控制。

8. 扩展功能:

(1) 网络总线: 8 路 USB、6 路串口(5 路 232 接口, 1 路 232/485/422 接口)、1 路 CAN 总线、1 路 PCI 总线、1 路 PC104 总线、1 路 SATA 总线、2 路以太网;

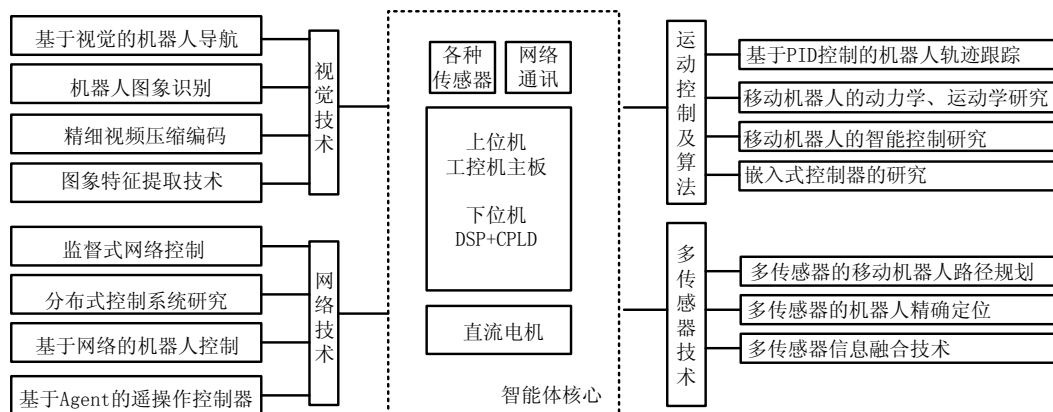
(2) 电机控制: 3 路 PWM 输出、8 路 D/A 输出(本体运动控制使用 2 路);

(3) 数字输入输出: 16 路带隔离的数字输入、16 路带隔离的数字输出;

(4) A/D 采集、处理: 16 路 10 位 A/D 采集(本体红外使用 6 路);

(5) 电源: 提供扩展专用的 5V(20A)、12V(5A)、24V(5A)的独立电源接口组, 便于传感器及其他配件的稳定工作。

1.2.3 产品可研究领域:



以研究型智能机器人 MT-AR 为主要平台, 结合其丰富的配置系统, 相关专业的老师和学生可以就机器人运动学、动力学、控制技术、人工智能、电机学、网络、传感器、数据融合等相关的学科进行综合的课题开发, 综合起来, 主要可以进行的研究课题描述如下:

一、智能控制相关课题:

- 1) 基于 PID 控制的机器人轨迹跟踪性能研究与比较
- 2) 移动机器人的滑模变结构控制研究
- 3) 移动机器人的动力学、运动学模型研究



- 4) 轮式移动操作机器人的鲁棒跟踪控制器设计及研究
- 5) 开放式机器人控制器的研究
- 6) 移动机器人神经网络控制研究
- 7) 移动机器人模糊控制研究
- 8) 移动机器人系统中嵌入式控制器研究
- 9) 基于遗传算法的移动机器人的智能控制
- 10) 移动机器人的智能控制相关课题研究

二、传感器（图像，超声，语音等）相关课题：

- 1) 基于声纳和 PSD 的环境建模；
- 2) 基于声纳和 PSD 的机器人路径规划
- 3) 基于多传感器的机器人精确定位研究；
- 4) 多传感器融合技术及其在机器人中的应用
- 5) 移动机器人系统中传感器扩展技术研究

三、机器人视觉技术

- 1) 景象匹配及其算法研究
- 2) 基于立体视觉的运动目标跟踪
- 3) 机器人视觉伺服控制系统设计
- 4) 主动视觉及其在机器人中的应用
- 5) 人脸识别技术及其在移动机器人中的应用
- 6) 视频的压缩编码及其在机器人系统中的应用
- 7) 小波方法在移动机器人系统中的应用
- 8) 基于视觉的运动目标跟踪；
- 9) 多视觉系统的机器人信息融合

四、机器人网络控制技术

- 1) 视频网络传输及其在移动机器人系统中的应用
- 2) 基于 Agent 的遥操作机器人控制器研究
- 3) 基于网络的移动机器人控制系统研究
- 4) 基于网络的移动机器人直接控制系统研究
- 5) 移动机器人中视觉临场感遥控系统的研究等等

五、人机交互技术

- 1) 语音识别技术及其在移动机器人系统中应用
- 2) 手势识别及其在移动机器人系统中应用
- 3) 多模态人机交互及其在移动机器人系统中
- 4) 虚拟现实技术在移动机器人系统中的应用



5) 自然语言识别和理解技术

六、多机器人技术

- 1) 多移动机器人系统合作与协调
- 2) 多机器人系统中实时通信研究
- 3) 多机器人路径规划与控制策略研究
- 4) 多智能体 (MAS) 技术及其在足球机器人竞赛中的应用

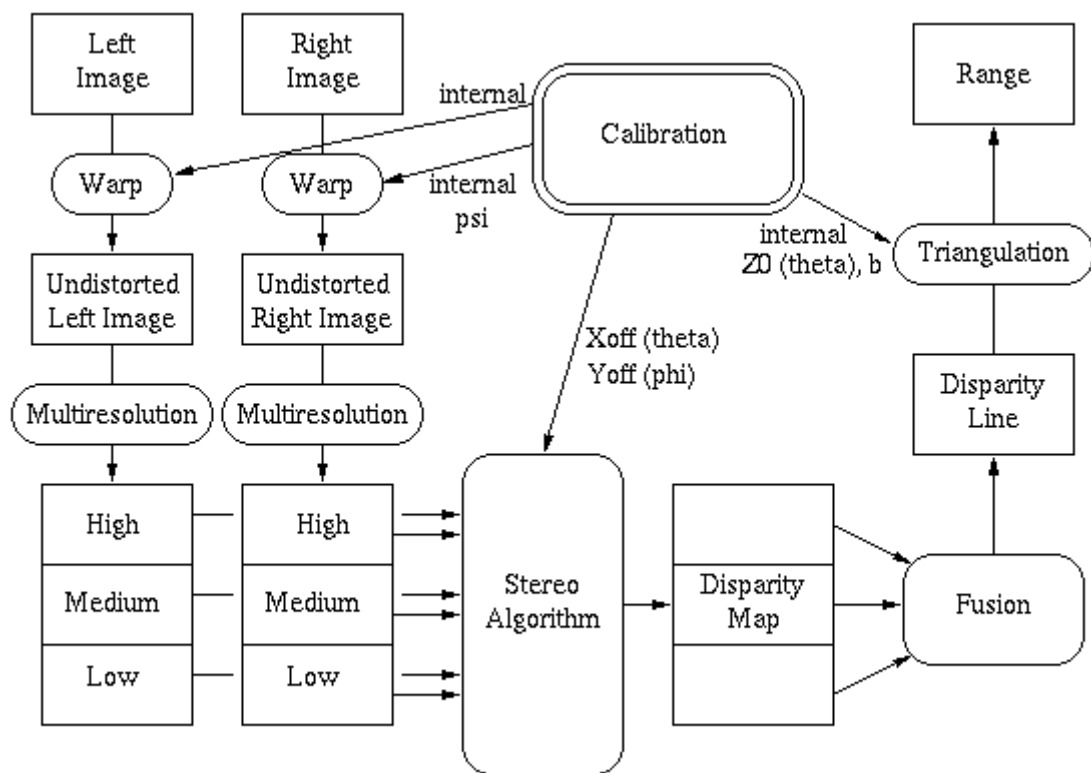
七、机器人学习

- 1) 移动机器人的再励学习研究
- 2) 基于视觉的移动机器人学习系统研究
- 3) 基于支持向量机的移动机器人学习
- 4) 动态环境下移动机器人的路径规划方法
- 5) 数据挖掘技术及其在移动机器人系统中的应用

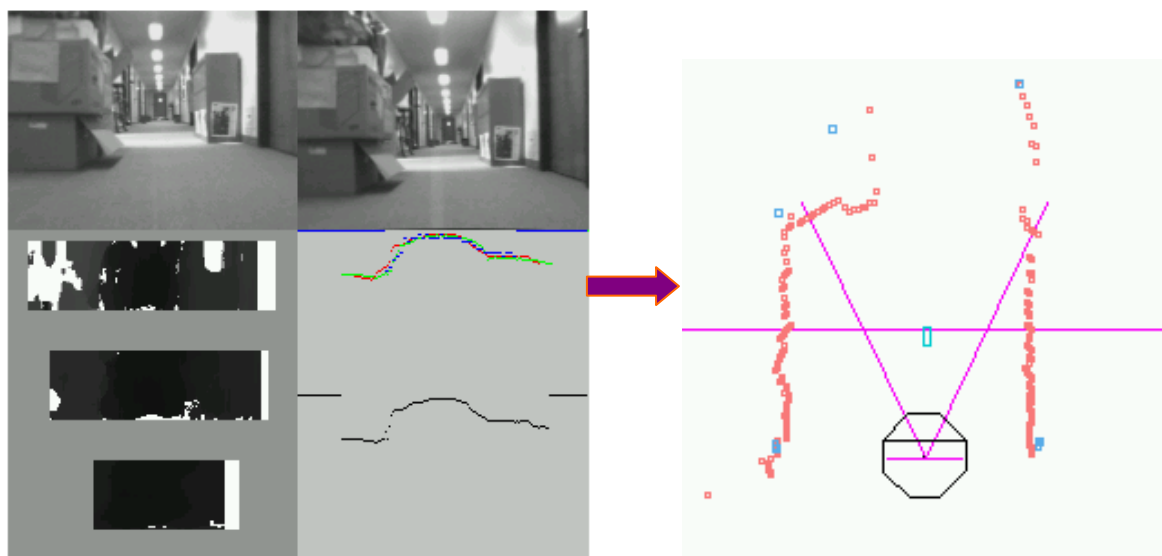
八、其他智能机器人相关课题

案例1: 基于立体视觉的目标识别与路径规划

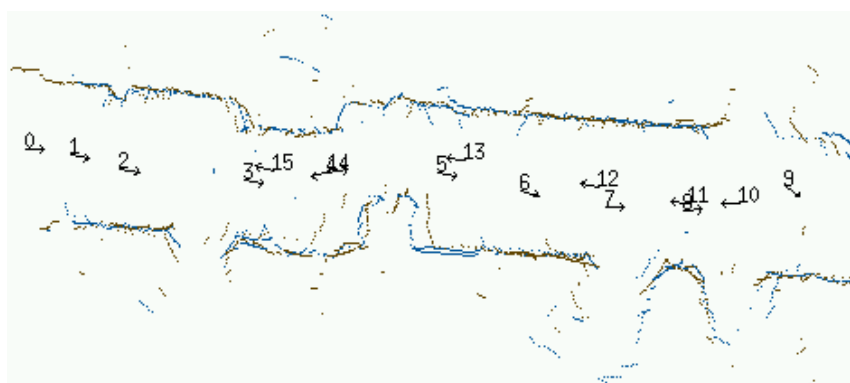
任务描述: 基于双目立体视觉的目标识别与路径规划涉及单目、双目图像信息的融合, 得出环境特征信息, 在特定算法的辅助下, 得到路径信息并实施路径规划。是融图像处理技术、人工智能, 模式识别、路径规划等多种前沿研究方向之一。



原理框图



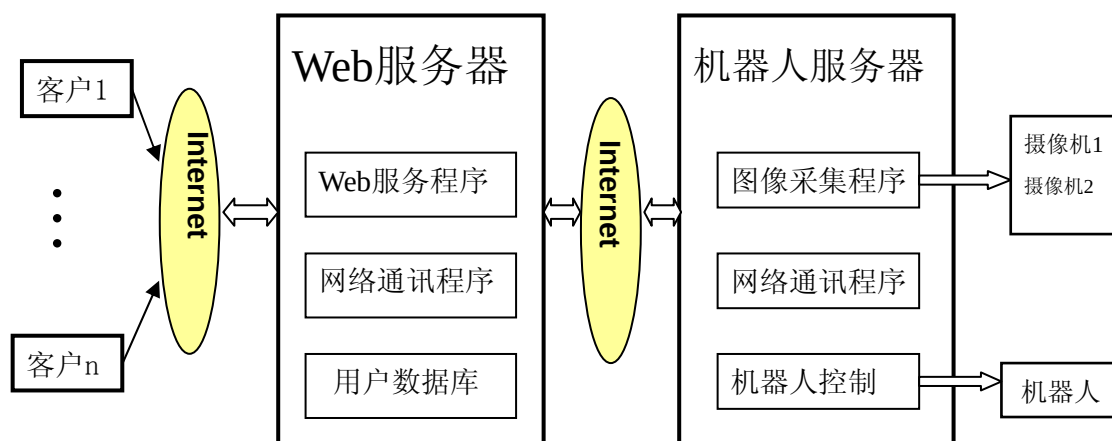
特征信息提取

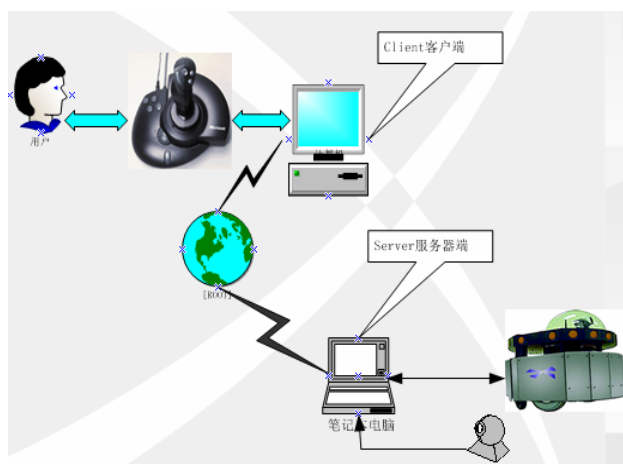


路径规划

案例2: 基于WED的遥操作机器人

任务描述: 遥操作机器人涉及传感器技术、图像技术、虚拟现实技术、面向对象技术、网络传输与控制技术、数据压缩技术等等, 是目前机器人技术、控制技术的前沿研究热点之一。





双向力反馈网络遥操作系统平台



第二章： MT-AR 机器人机械系统

2.1 MT-AR 的机械系统分析

分类	项目	性能	指标	测试结果
机械系统	1	外形尺寸	满足 Robocup 中型组比赛 (小于 50cm×50cm 或 Φ 63cm)	满足 Robocup 中型组比赛尺寸
	2	电机及减速器寿命	大于 1000 小时	大于 1000 小时
	3	电机堵转性能	电机短时堵转不损坏	电机短时堵转不损坏
	4	推力	推力大于 8kgf	推力大于()kgf
	5	负载能力	最大负载 30kg	达到负载(30)kg
	6	速度	最大速度 2.75m/s	最大速度(2.5)m/s
	7	转弯性能	零转弯半径	零转弯半径(需要通过闭环来实现)
	8	环境适应性	满足室内地板、地砖、地毯等运行环境	符合环境运行条件
	9	部件更换性能	各部件更换方便,可单独拆装	电池、驱动器、电源管理电路板能方便更换;
	10	扩展性能	机械扩展方便,满足盘球、踢球装置、云台、液晶显示器、全景摄像头、激光传感器、三维数字罗盘、双目摄像头等扩展安装	能方便扩展各种配件
	11	冲击振动性能	符合 GB2423 (振幅 7cm, 频率 20HZ)	通过上海市技术监督局振动测试;

2.2 MT-AR 的机械结构

2.2.1 MT-AR 的机械系统结构介绍

MTR 机器人为三层结构,分底盘行走机构总成、计算机控制系统总成、传感器及运动控制系统总成三层结构,模块间用四根定位支撑柱连接固定。但外观上分为两层,底盘和控制层被外壳包容在一起。整机三点支撑,前面两个驱动轮,后面一个万向轮平衡。机体宽度 485mm,总长 494mm,高度 495mm。

MT-AR 机器人主体外观及主要结构如图 1 所示。

MT-AR 机器人主体外形尺寸如图 2 所示。

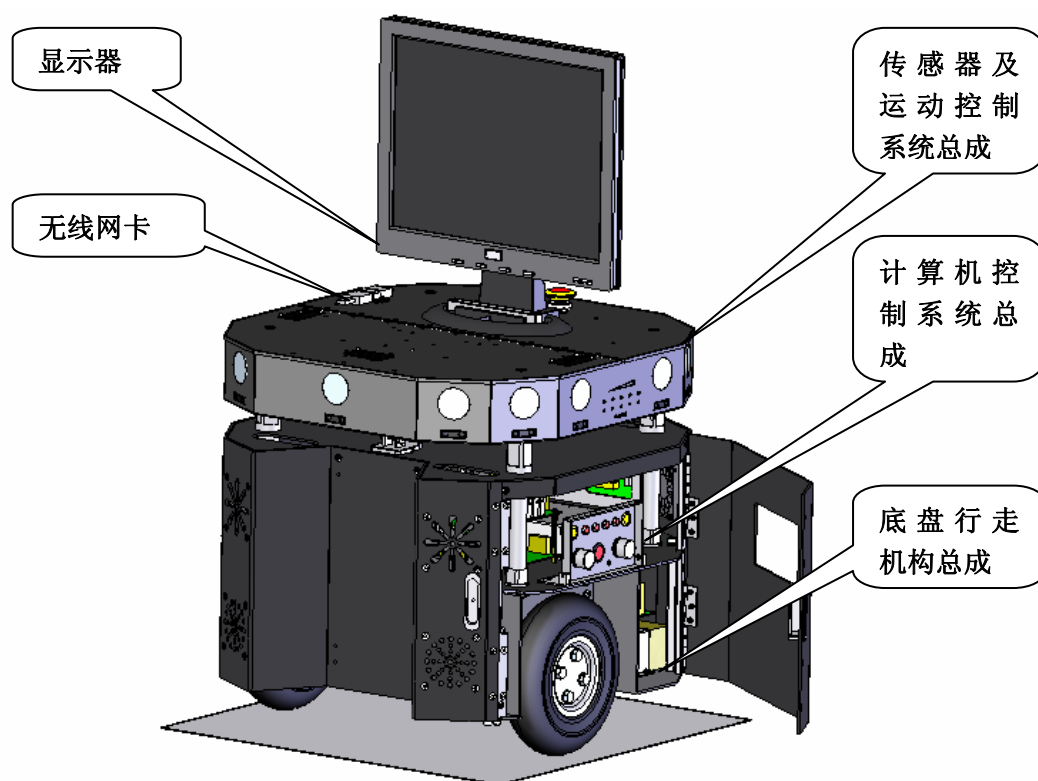


图1 MT-AR 机器人本体总体结构图

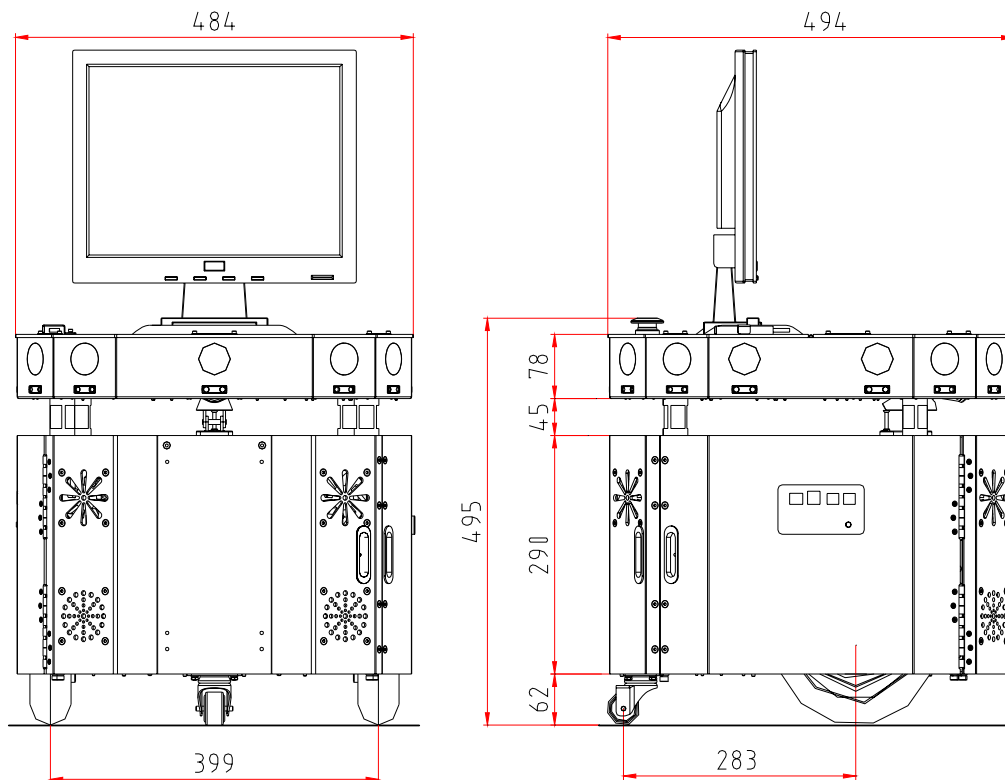


图2 MT-AR 机器人本体外形尺寸图

2.2.2 MT-AR 的机械系统结构组成

底盘行走机构总成包含动力系统，装有电机、驱动器和电池、并可以安装踢球装置，高度 150mm。底箱内后部安放电池,可安装 20Ah 锂电池。两个主动轮安装在前部，差速驱动，轮子采用聚胺脂轮免充气轮胎，主动轮直径 $\phi 194\text{mm}$ ，两轮间距 401mm,轮轴中心线距整机最前端 190mm，距盘球前端面 103.5mm。一个从动轮安装在箱底后部中间，为工业用高精度万向脚轮，耐磨聚氨酯轮体，轮径 $\phi 40\text{mm}$ ，从动轮轴线距主动轮轴线最大距离 283mm。安装轮子后，底盘离地高度 62mm。

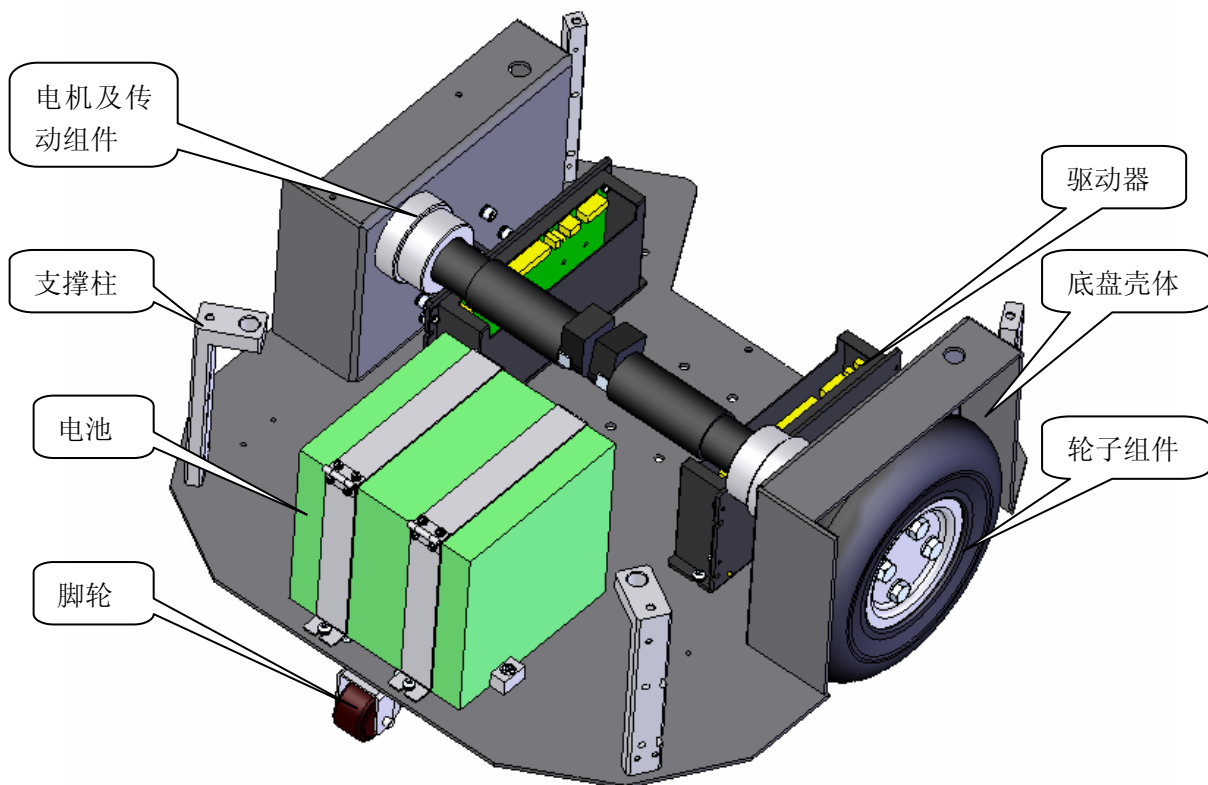


图 3 MT-AR 机器人底盘行走机构总成结构图

中间层为计算机控制系统总成，包括工业级主板、硬盘、ATX 电源、电源管理模块、转接板、接口面板及指示灯面板，层高 136mm。机体的底盘行走机构总成、计算机控制系统总成前后左右为可拆卸拉开的四块包络外壳，每块外壳都被做成门的形式一边通过合页固定到机体上，一侧通过螺钉紧固，维修时只需要拧下这几颗螺钉便可通过门上的拉手将门拉开，便于进行维修。其中前门板设计成向内凹陷，设计时其尺寸参考了 Robocup 中型组比赛规则，在规则允许的前提下最大限度地具备好的盘球功能。计算机系统的主板水平放置,ATX 电源放在主板侧边,硬盘架空放在主板后方。机体两侧面开孔，开关、按钮及接插口等引



出到机体侧面操作面板。机箱前后板斜面上开散热孔、扬声孔，机箱整体靠四个强力风扇通风和铝壳体传导散热。

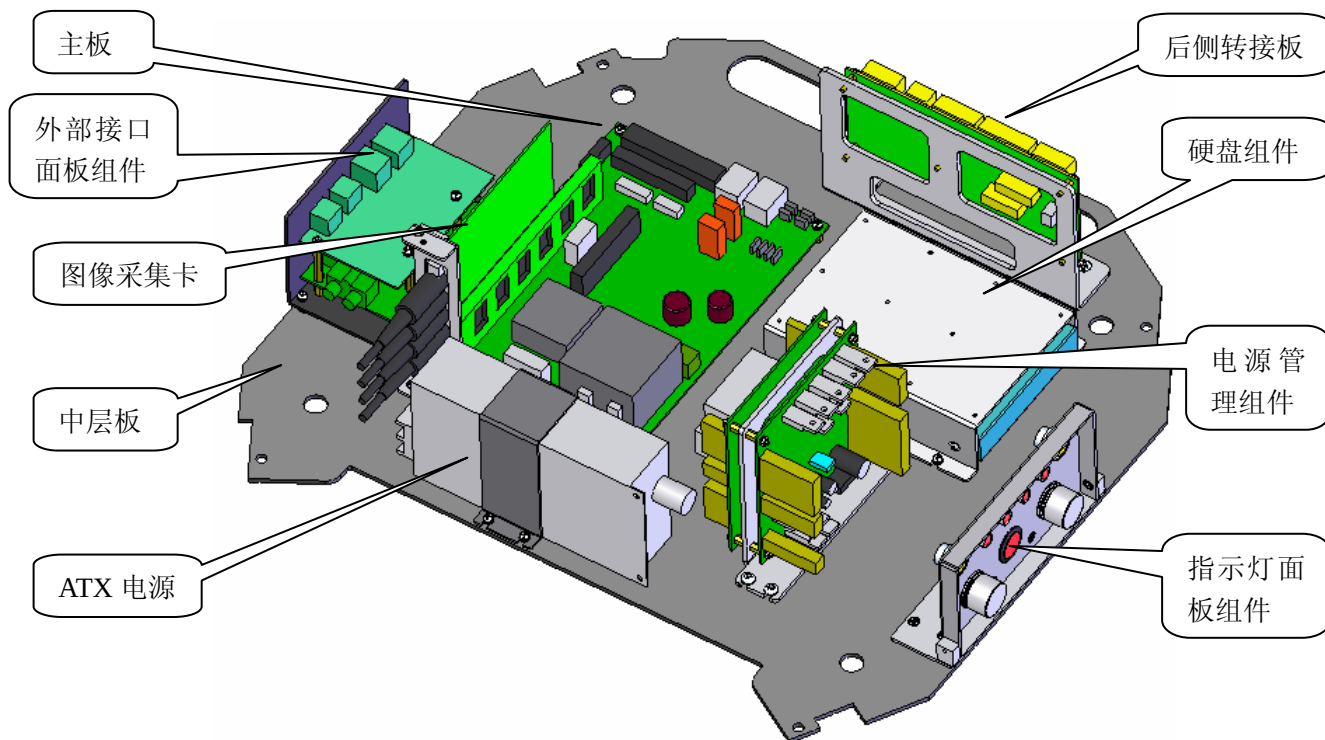


图4 MT-AR 机器人计算机控制系统总成结构图

传感器及运动控制系统总成，上下左右对称的12边形，长度490mm，宽度480mm，高度78mm。顶箱各侧面圆周开14个圆孔和方孔，方孔与圆孔中心上下共线。在前后面正中圆孔和左右相邻各两个圆孔共安装6个超声传感器,6个红外传感器，作为标准配置，当然我们还可以扩展至14路超声和14路PSD。顶箱面板上安装摄像头、无线网卡等，面板上预留M3、M4、M5的螺纹孔，可扩展安装全景摄像机、液晶显示器、双目摄像机、三维数字罗盘、GPS等配置。

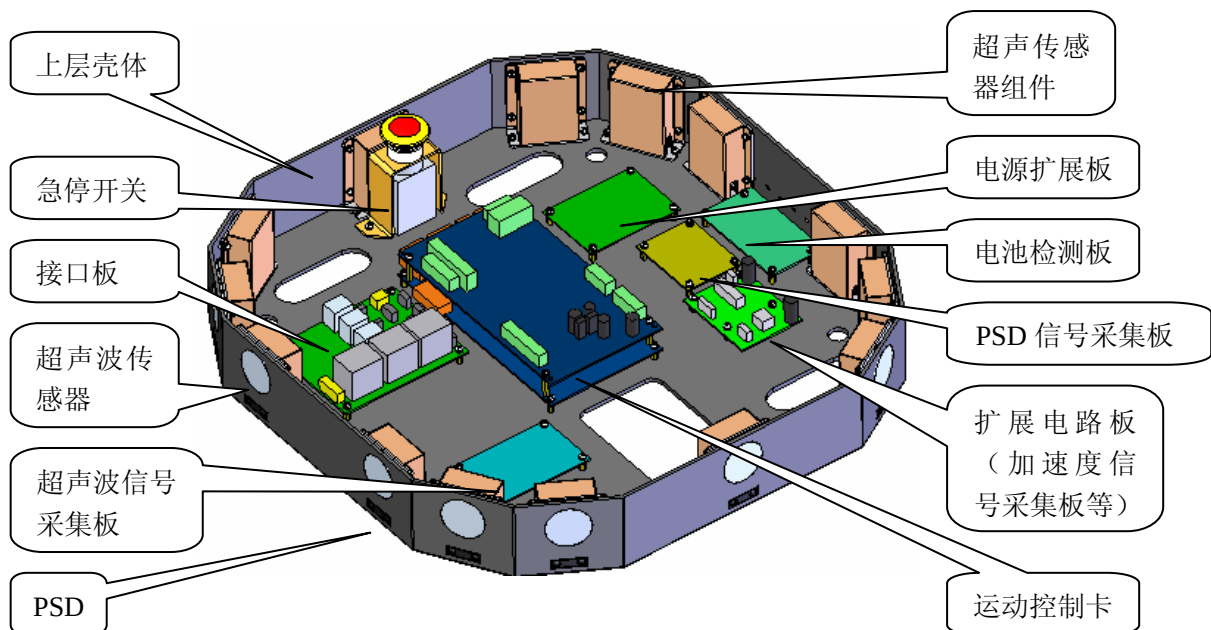


图 5 MT-AR 机器人传感器及运动控制系统总成结构图

MT-AR 机器人主体结构为各层底板及外壳，材料使用 3mm 厚铝合金板 AA5052-H32。机器人轮毂、轮盘、支撑柱等一些承力部件采用高强度硬铝合金 LY-12。底盘受力较大，采用不锈钢 SUS304。机械连接用的连接定位螺栓、连接柱及螺钉、螺母、垫圈等全采用不锈钢材料。箱体用铝合金板加工，工艺方法采用数控线切割裁剪、割孔，数控折弯钣金加工和焊接成型。承力结构零件采用数控机床精密加工。钣金件表面处理采用黑色小橘皮烤漆，机加工件采用黑色阳极氧化的工艺。

在传感层的左边侧壁，我们设计了电池电量显示区域，用户可以通过 5 个高亮 LED 灯点亮的个数来判断电池剩余电量，这样用户就可以时刻了解电池电量的信息。

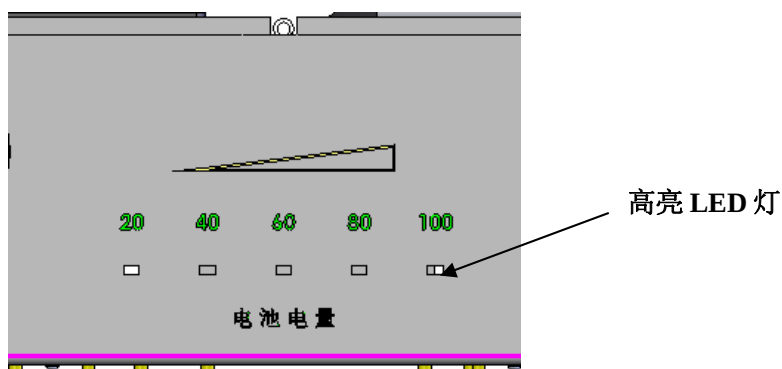


图 6 MT-AR 机器人电池电量显示图

在机器人本体左侧，我们设计了指示灯面板，上面排布了过流报警、欠压报警、电源、运行指示灯和开机、关机、复位操作按钮以及外接电源与充电插座等。

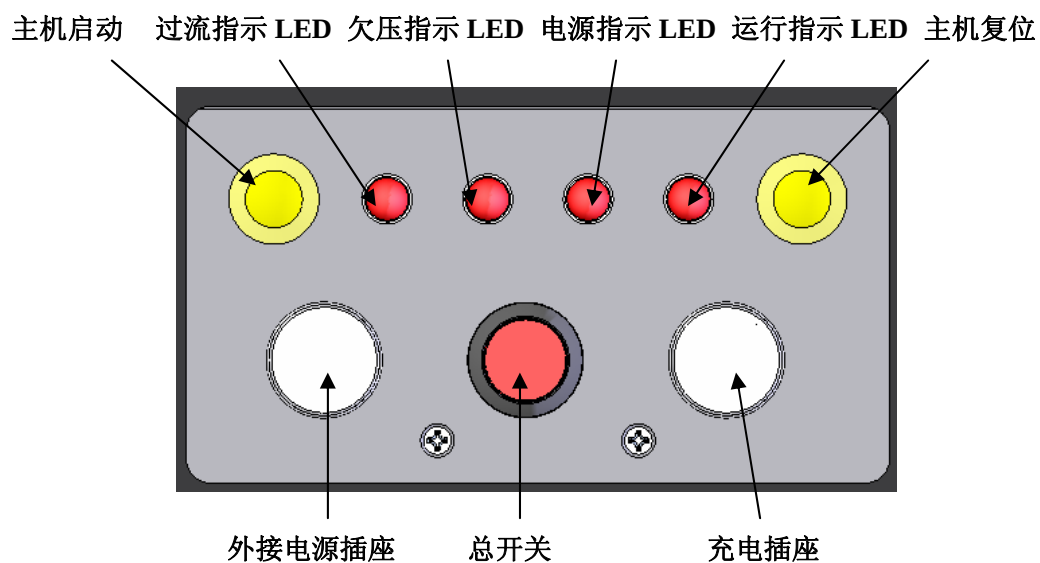


图 7 MT-AR 机器人指示灯面板图

在机器人本体左侧，设计了外部接口面板，可以通过此面板实现一些外部接口的扩展，如键盘、鼠标、麦克、网口及 USB 口：

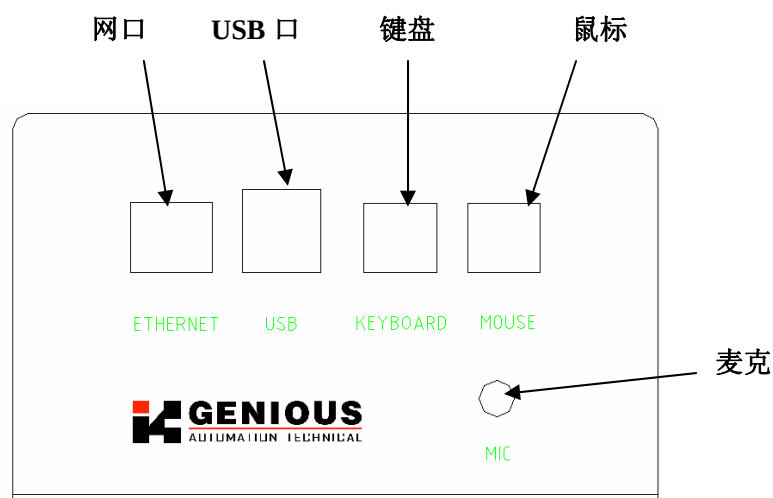


图 8 MT-AR 机器人外部接口面板图

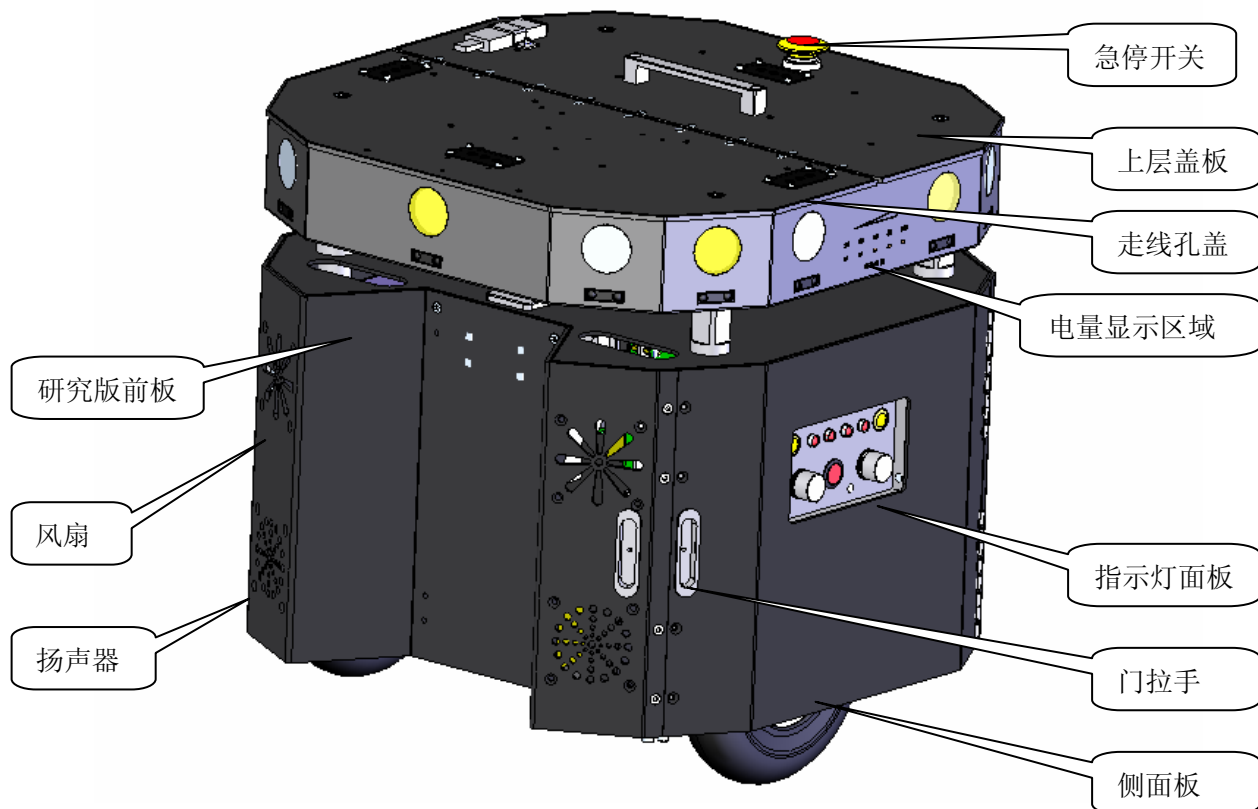


图9 MT-AR 机器人局部结构指示图

1.2.3 电机特性

MTR 机器人使用两只瑞士 maxon 公司生产的 RE36 型功率 70W 的空心杯转子直流电机。该型电机采用高磁能积的稀土钕铁硼磁钢、专利的空芯杯转子、石墨电刷、CLL（电容熄弧长寿命）技术，具有体积小、效率高、惯量低、寿命长的优势，并具有无齿槽效应、加速性能高、电磁干扰小、线性的电压/速度曲线、线性的负载/速度曲线、线性的负载/电流曲线、转矩波动小、可短时过载、结构紧凑的技术特点。

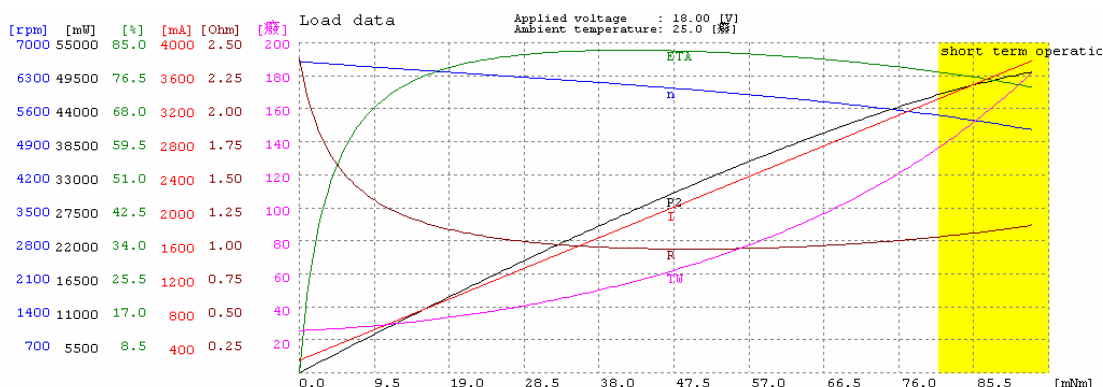
RE36(118797)电机性能参数：

电机数据			
1	标称功率	W	70
2	额定电压	V	24
3	空载转速	rpm	6610
4	堵转转矩	mNm	730
5	速度/转矩斜率	rpm/mNm	9.23



6	空载电流	mA	153
7	堵转电流	A	28.6
8	电机电阻	Ω	0.628
9	最大允许转速	rpm	8200
10	最大连续电流	A	3.18
11	最大连续转矩	mNm	81.0
12	最大输出功率	W	119
13	最大效率	%	84
14	转矩常数	mNm/A	25.5
15	速度常数	rpm/V	375
16	机械时间常数	ms	6
17	转子惯量	gcm^2	60.2
18	电机电感	mH	0.10
19	机壳到环境的热阻抗	K/W	6.4
20	转子到机壳的热阻抗	K/W	3.4
21	绕组的热时间常数	s	38

RE36 (118797) 电机的特性曲线:



2.2.4 齿轮头特性

为获得更大的转矩和相对较低的速度,MTR-1 机器人动力系统中使用了两只 maxon 公司生产的 GP32A 型行星齿轮减速箱与 RE36 型直流电机配套。该型减速箱为二级行星齿轮减速箱,具有传递转矩大、性能高、尺寸小、输入齿轮与输出齿轮同轴的特点。GP32A(166163)减速箱特性参数:



减速箱数据			
1	减速比		33:1
2	精确减速比		529:16
3	减速级数		2
4	输出的最大连续转矩	Nm	2.25
5	允许瞬间输出的转矩	Nm	3.4
6	输入输出旋转方向		相同
7	最大效率	%	75
8	重量	g	162
9	平均空回	°	0.8
10	惯量	gcm ²	0.8
11	减速箱长度	mm	36.3
12	输出轴材料		不锈钢
13	输出端轴承		滚珠轴承
14	径向间隙(法兰外 5mm 处)	mm	Max0.14
15	轴向间隙	mm	Max0.4
16	最大允许径向载荷	N	140
17	最大允许轴向载荷	N	120
18	最大允许安装力	N	120
19	推荐输入转速	rpm	<6000
20	推荐温度范围	℃	-15~+80

2.3 MT-AR 机器人维护拆装指导

在机器人的维护过程中涉及到拆卸和安装两个相反的过程，拆卸往往比安装顺利。所以用户请注意以下安装过程中，切忌孔位没有完全对准时强行拧进螺钉，否则将损坏螺纹，无法继续使用。孔位有偏离时可施加外力使钣金件产生微量变形或贴近螺纹孔，使孔位完全对准时方可拧入螺钉。并记住每次都从每一组孔里面对的最齐的那个位置开始拧入螺钉。

2.3.1 MT-AR 显示器拆装

显示器的拆卸，如下图所示，用内六角扳手拧下显示器固定座上的两颗螺钉，显示器即可以和上盖板脱离，取下显示器的电源线和数据线，即可以将显示器拆离机器人本体。

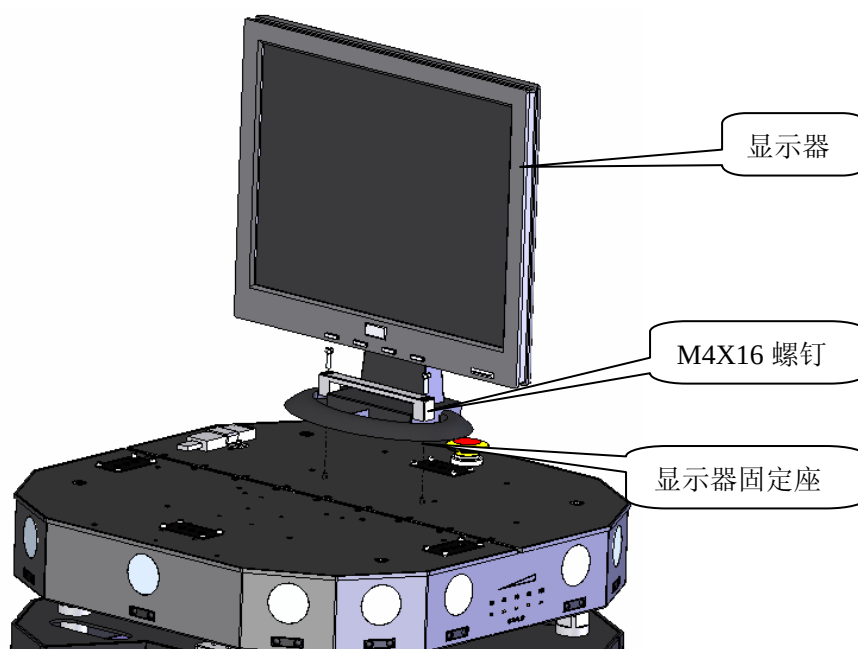


图 10 MT-AR 机器人显示器拆卸示意图

显示器安装时，可先将显示器摆在相应的位置上，压上固定座，用内六角扳手拧入螺钉，固定好显示器，然后插上电源线和信号线即可。注意安装好后显示器的底座不能盖住上盖板上的长铰链，否则会影响上盖板上任意一边的开启。

2.3.2 MT-AR 上层盖板拆装

在使用 MT-AR 机器人的过程当中，需要用户拆装最多的地方可能就要数上层盖板了。上层盖板上面可以扩展很多传感器，打开上层盖板，就是运动控制卡和各种信号采集卡以及各种扩展卡，部分电路板连接着外围的传感器。因此用户在扩展传感器和日常检修时会经常拆装上层盖板。上层盖板被设计成两半通过一个长铰链连接到一起，然后盖住整个传感层。这样的设计可使用户不必每次拆装都将整个上层盖板拆下来，只需要开启其中的一半即可完成操作。当然在必要时，也可以将整个上层盖板拆卸下来。在上层盖板上还设计有过线的椭圆孔，并且在通常的情况下有孔盖将这些孔盖住，一为美观，二为防止异物及灰尘进入机器人内部。上层盖板通常的拆卸过程如图 11 所示，整体拆卸过程如图 12 所示，走线孔盖的拆卸如图 13 所示。传感器的扩展我们后面再讲。

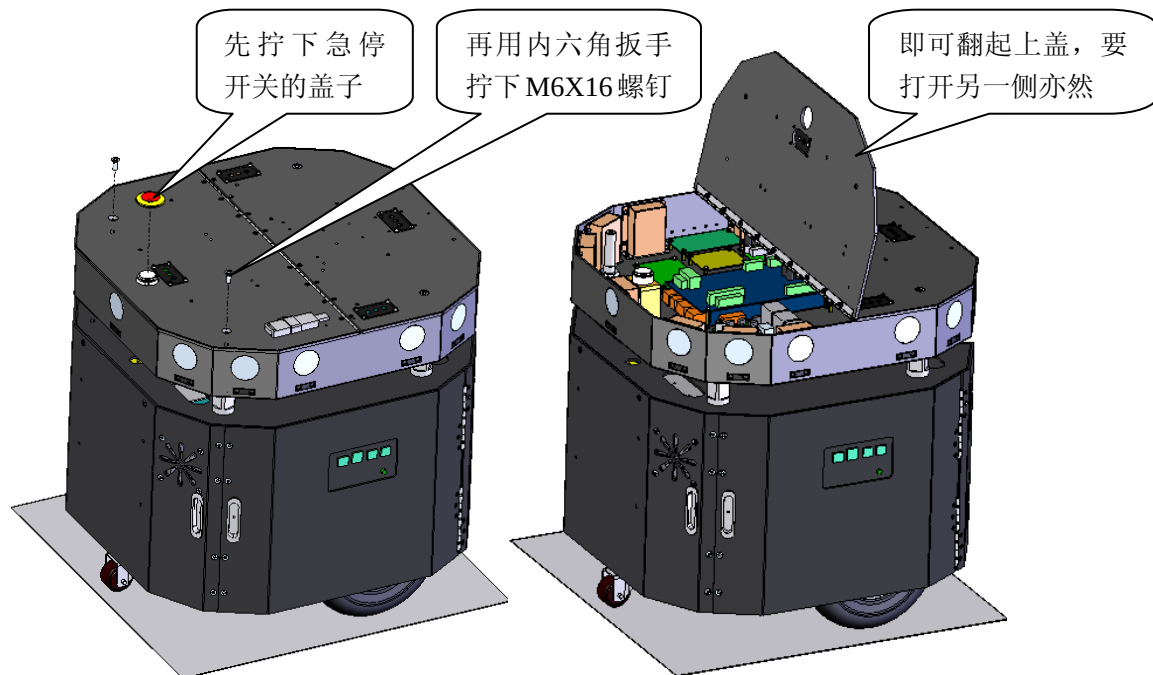


图 11 MT-AR 机器人上层盖板单面拆卸示意图

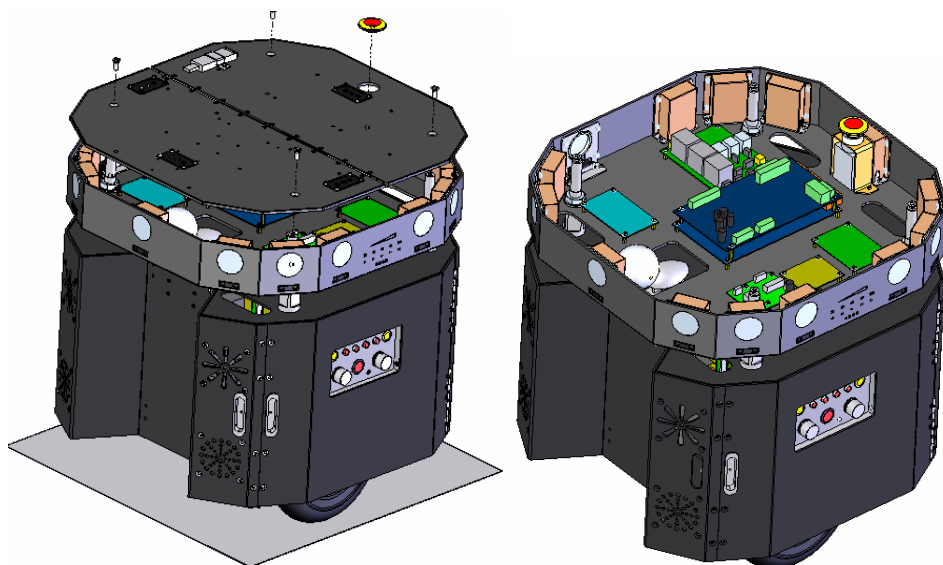


图 12 MT-AR 机器人上层盖板整体拆卸示意图

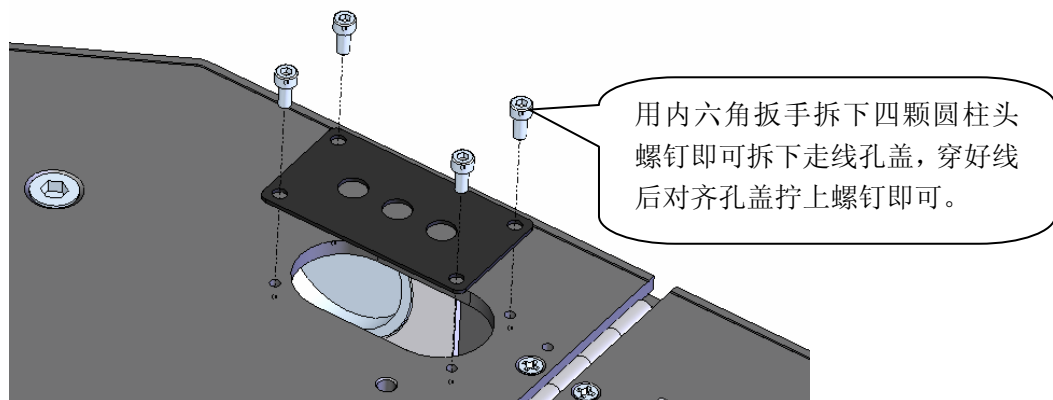


图 13 MT-AR 机器人上层盖板走线孔盖拆卸示意图



2.3.3 MT-AR 局部摄像头拆装

机器人本体所采用的局部摄像头，安装在传感层下方的中层盖板上的前端。通过一颗 M4X40 的沉头螺钉拧到中层盖板上，然后通过一个 M4 螺母锁紧，并利用长螺钉的头部顶起摄像头的底座弹簧夹的上半部分，通过长螺钉拧入深度的深浅还可以调节撑开弹簧夹的程度，以便摄像头可以调节到理想位置。拆卸时可以用小扳手拧松螺母，然后一只手掰开弹簧夹，用螺丝刀拧下螺钉，即可拆下摄像头。

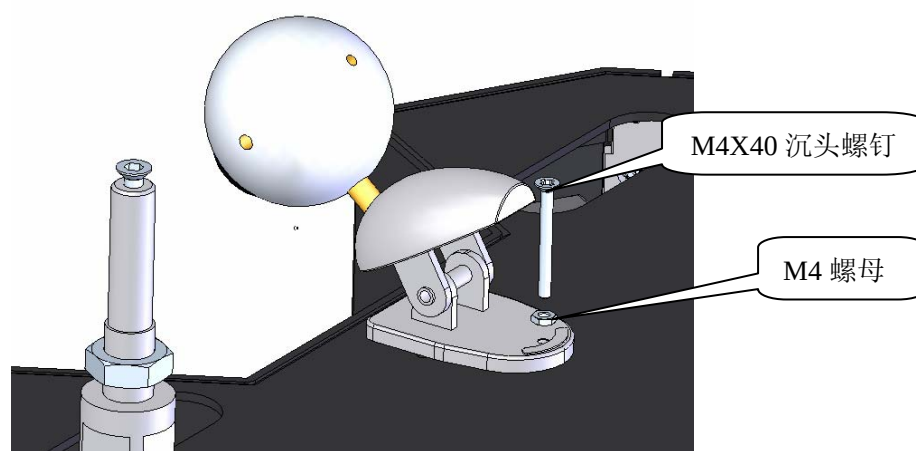


图 14 MT-AR 机器人局部摄像头拆卸图

摄像头安装时，可将上述过程反过来即可，先用手掰开弹簧夹，将螺钉带螺母拧入中层盖板至合适的深度，然后拧紧螺母锁紧即可。

2.3.4 MT-AR 各个围板拆装

为了方便检修和模块的更换我们将机器人本体外壳设计成可以拉开的“门”，通过铰链将围板连接到机器人的本体上，拧下固定“门”的螺钉即可以拉开“门”，然后就可以进行相关模块的更换或者维修。

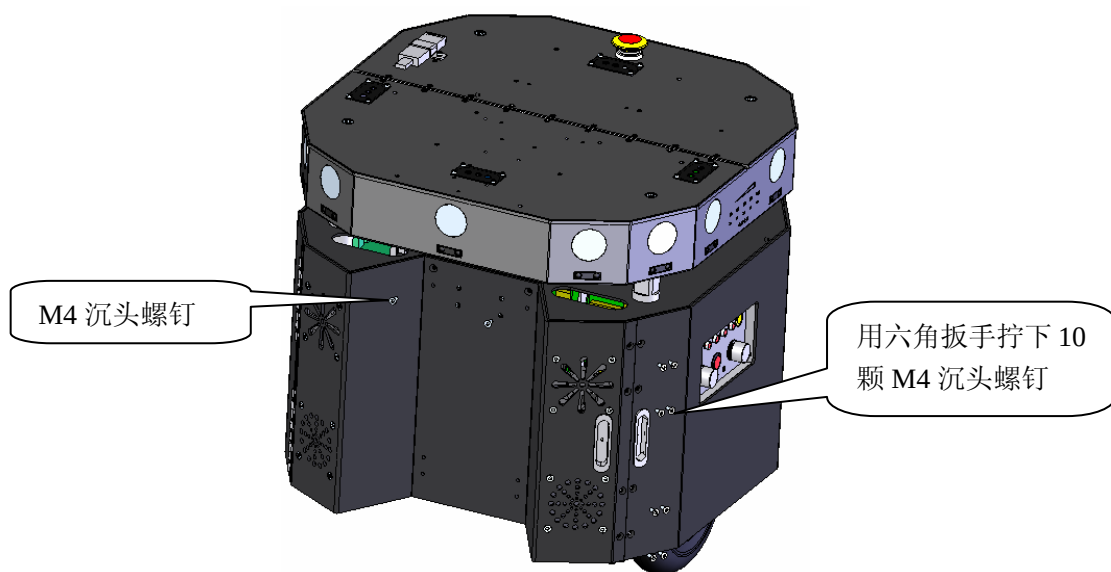


图 15 MT-AR 机器人前板和左侧板固定螺钉拆卸示意图

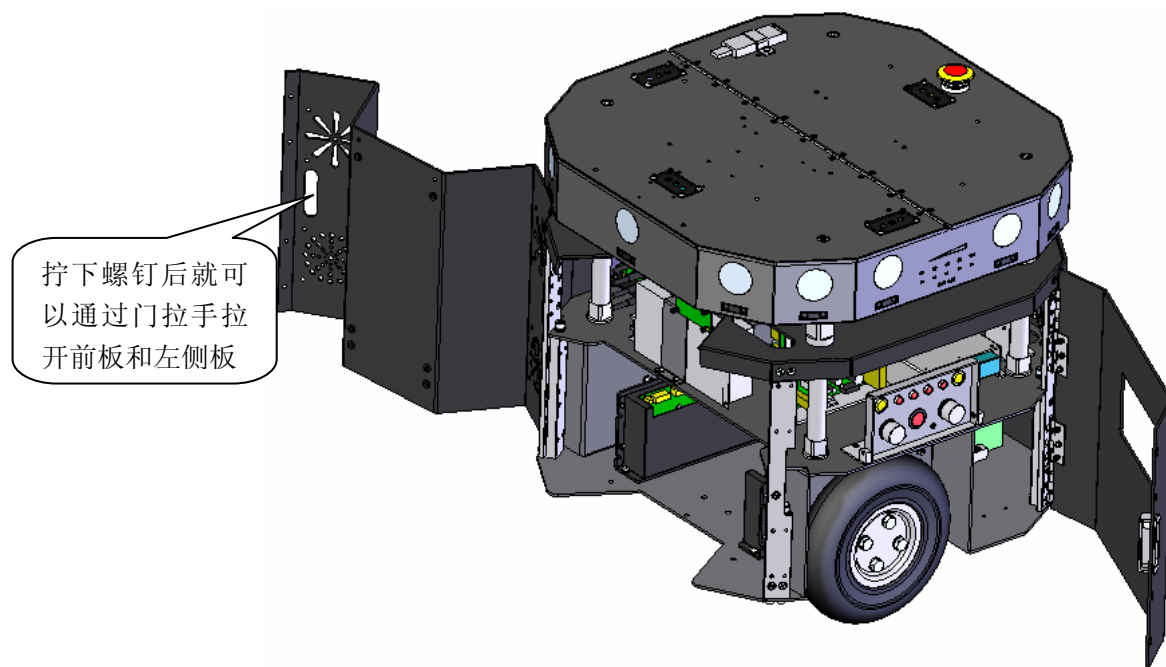


图 16 MT-AR 机器人前板和左侧板拉开示意图

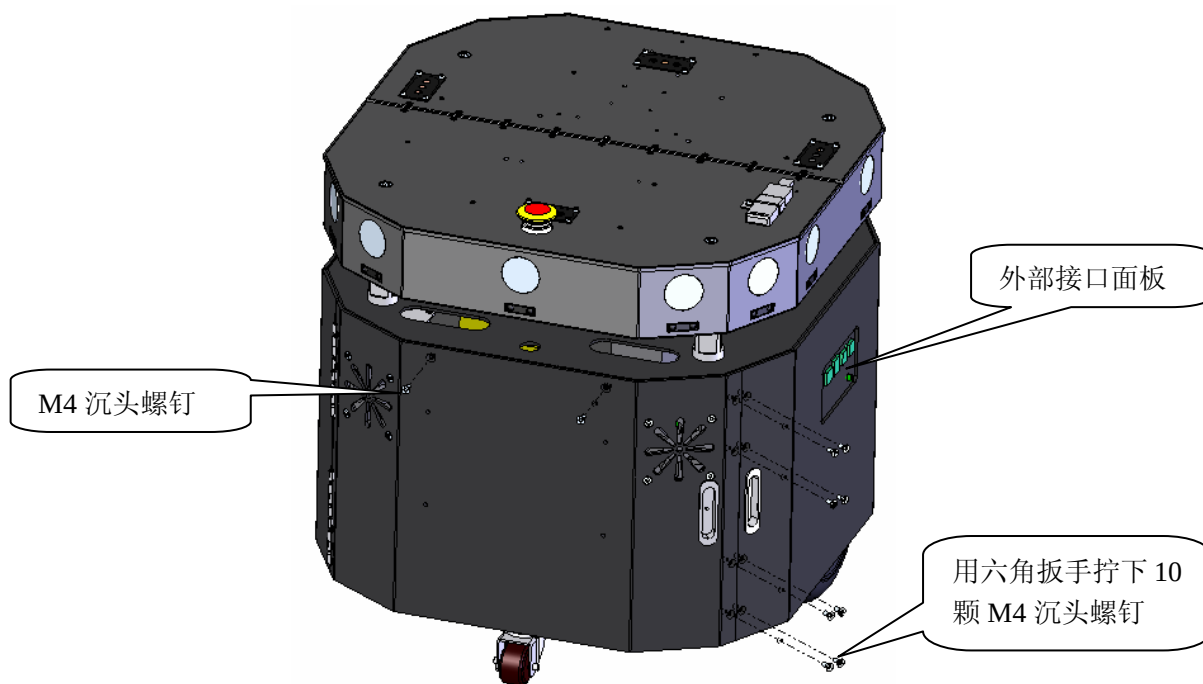


图 17 MT-AR 机器人后板和右侧板固定螺钉拆卸示意图

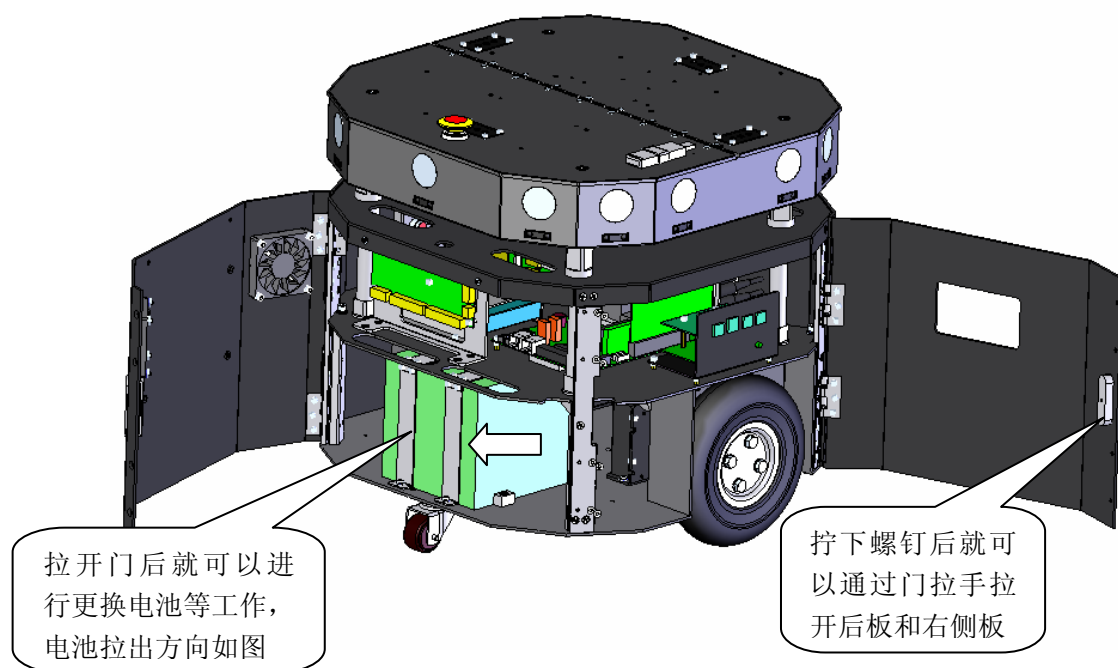


图 18 MT-AR 机器人后板和右侧板拉开示意图

如图 15、16、17、18 所示，拧下螺钉就可以通过门拉手拉开“门”。按压门拉手的上部，拉手下部就会自动弹出。“门”拉开后就可以进行中间层的电源管理板、接口板等各个模块的更换或者维修等工作，也可以更换下层的电池、驱动器，也可以安装踢球装置及踢球控制系统。拧下电池压板外侧的两颗螺钉即可以



抬起压板，这时就能向外抽出电池。完成维修或者更换工作后，可以关上各扇“门”，拧上螺钉。此时注意从孔位对的最准的位置开始拧入螺钉，并参考本节开头的提示，就能保证装配的顺利和正确。

2.4 MT-AR 机器人扩展功能介绍

MT-AR 机器人提供了强大的扩展功能，可以扩展数种传感器，也可以扩展踢球装置及其控制系统。机器人本体内部有较大的空间提供给用户，非常方便用户进行二次开发时进行相应的扩展。在上层盖板上设计有均匀的螺纹孔，提供给用户安装额外的传感器或者其他装置。

目前我们能提供的扩展传感器包括 14 个超声传感器、14 个 PSD 传感器、云台摄像机、全景摄像机、双目摄像机、三维数字罗盘、GPS 全球卫星定位系统、加速度传感器、室内激光测距传感器等。

我们的 MT-AR 机器人提供扩展满足 ROBOCUP 中型组比赛规则的盘球装置和踢球装置，并且完全将踢球装置及其控制系统包容在内部，形成一个完整有机结合的整体。踢球装置为自主开发的高冲击力电磁铁，结合巧妙的推销滑道摆板机构构成，盘球装置也为自主开发并拥有专利的独特机构。其功能强大，实现了主动盘球和挑射击球。

如何扩展踢球装置

踢球装置及其控制系统安装在底盘行走机构总成的内部。

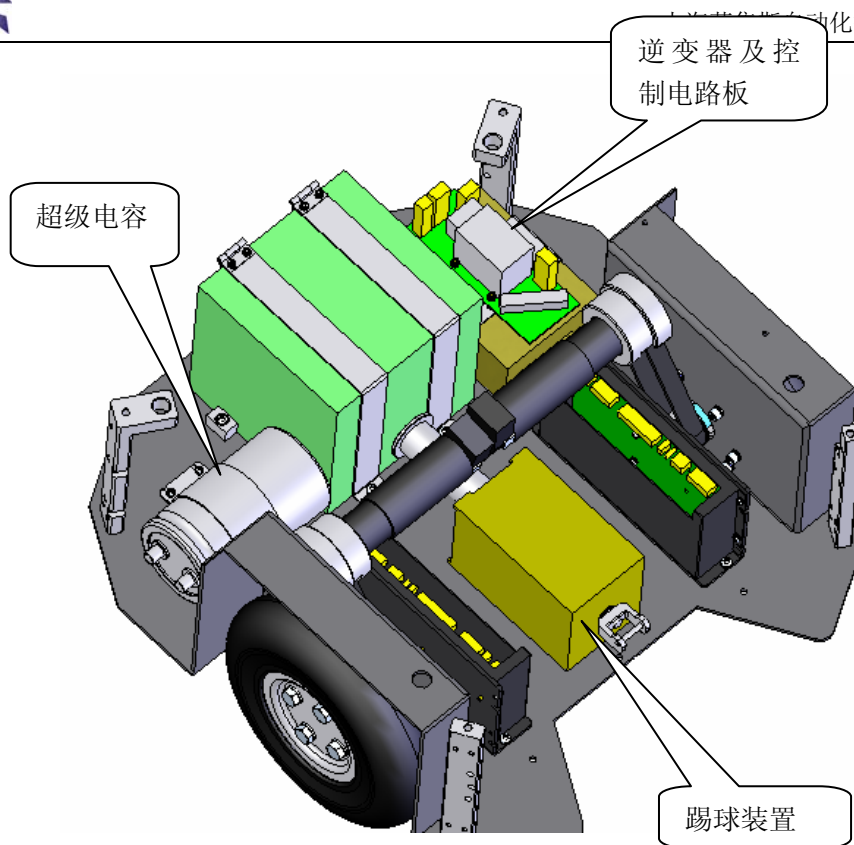


图 19 踢球装置及其控制系统安装示意图

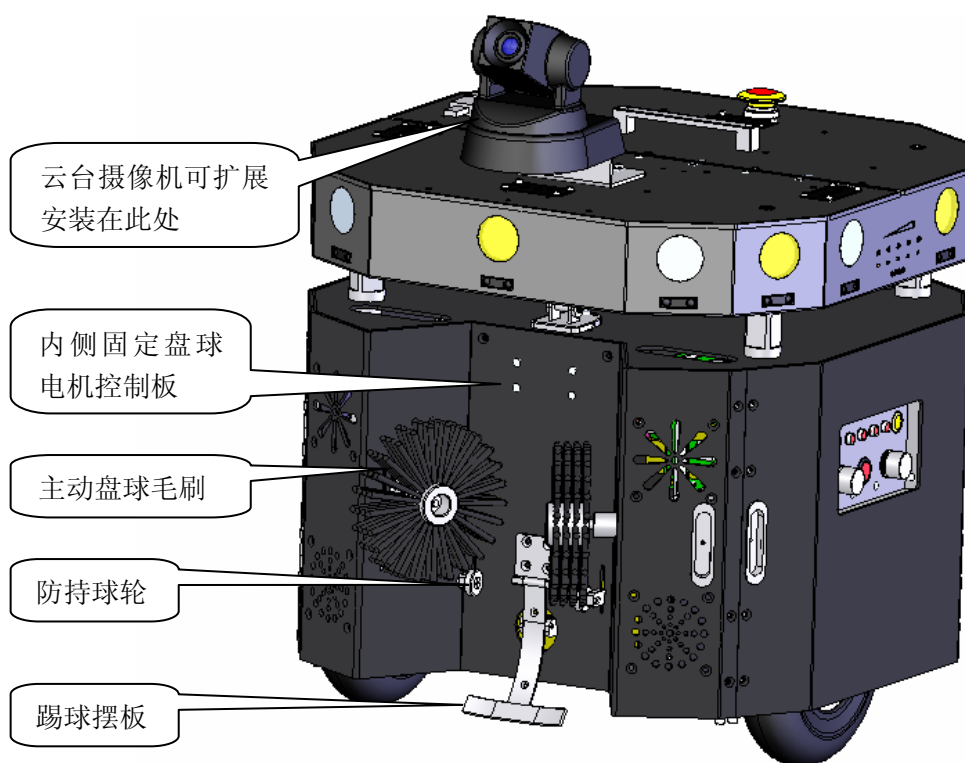


图 19 MT-AR 机器人盘球装置及踢球摆板安装示意图

如何扩展全景摄像机

全景摄像机被固定到一个可以升降的支架上，支架底座通过四颗 M4 螺钉固



定到上层盖板上。全景保护架由 4 颗 M5 圆柱头螺钉固定到上层盖板上，防止全景镜头及摄像机在运动中被碰伤，或者在足球比赛中被球砸坏。安装全景摄像机时，应该先打开上层盖板，穿好线缆，然后再拧螺钉固定。

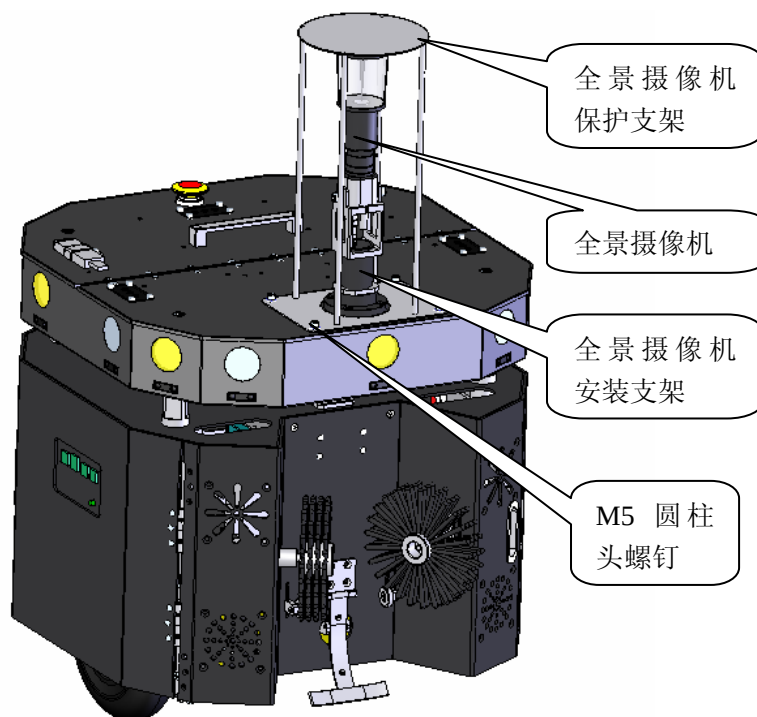


图 20 MT-AR 机器人全景摄像机安装示意图

如何扩展双目摄像机

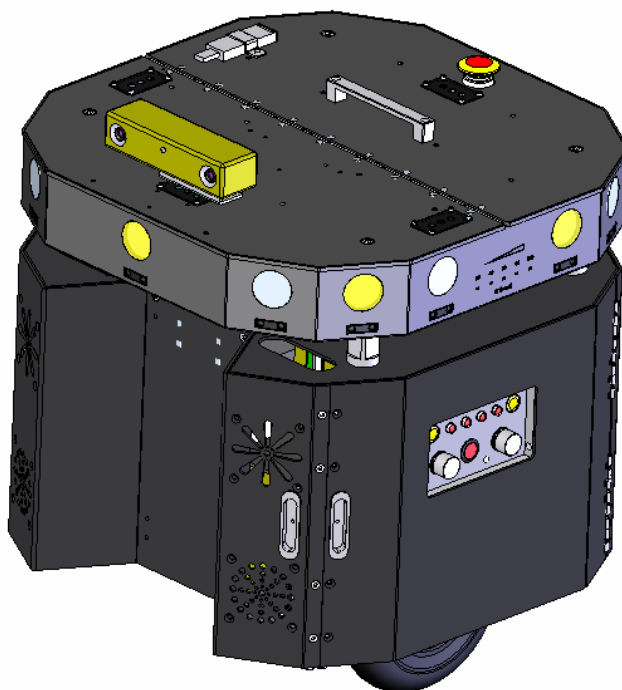


图 21 MT-AR 机器人双目摄像机安装示意图



双目摄像机可以通过双目安装支架安装到如图所示的位置。

如何扩展激光测距传感器

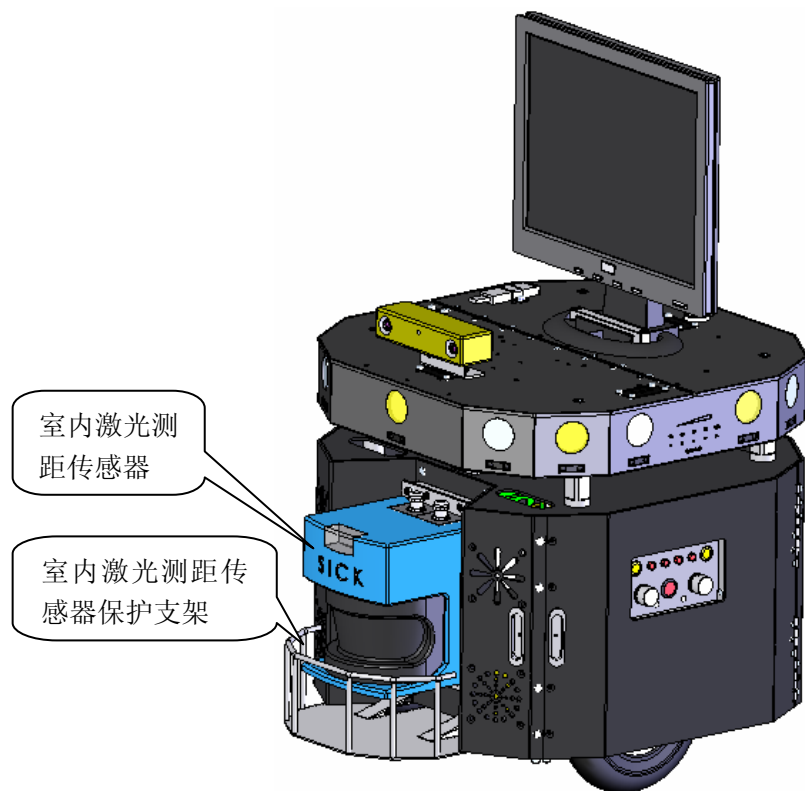


图 22 MT-AR 机器人激光测距传感器安装示意图

室内激光测距传感器安装在如图 22 所示的位置上,并有保护支架来防止其在机器人运动过程中碰伤。

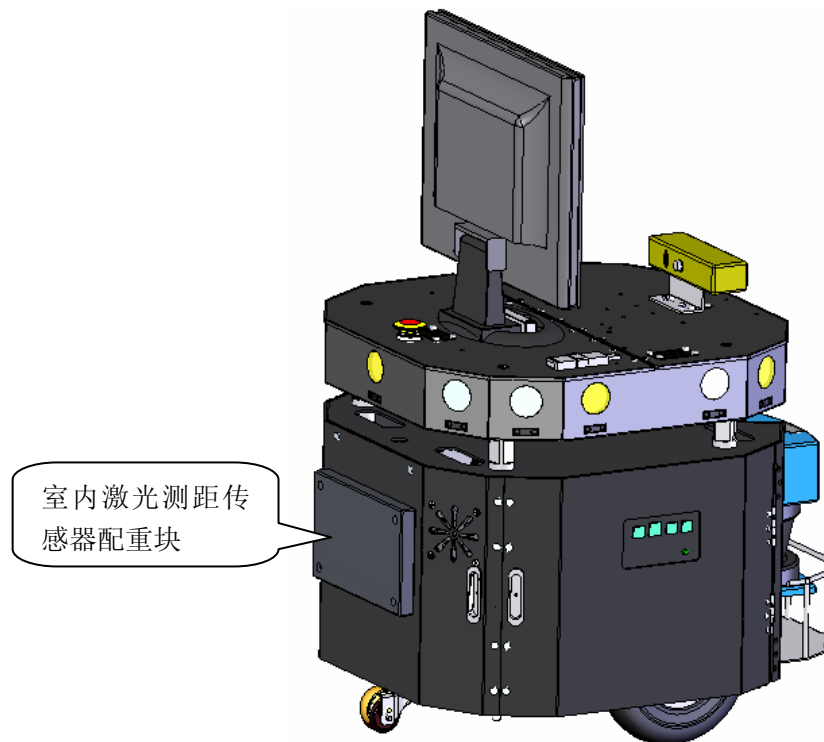


图 23 MT-AR 机器人激光测距传感器配重块安装示意图

室内激光测距传感器配重块可以保证机器人整机保持重心，不至于在减速时前倾。

如何扩展加速度传感器、GPS、三维数字罗盘

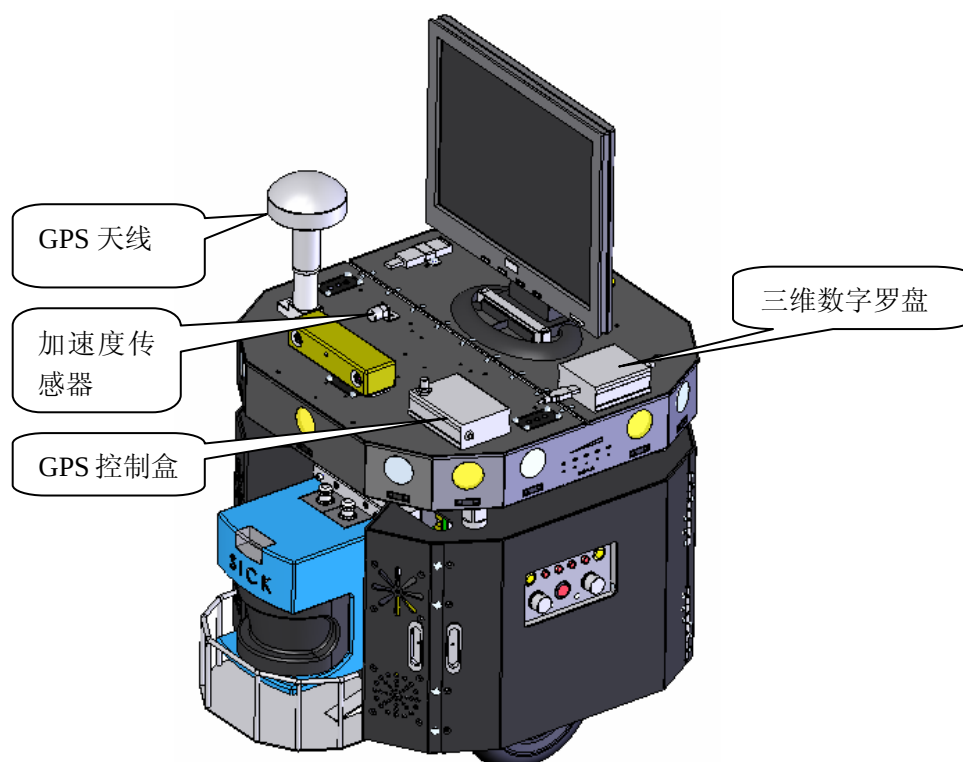




图 23 MT-AR 机器人加速度传感器、GPS、三维数字罗盘安装示意图

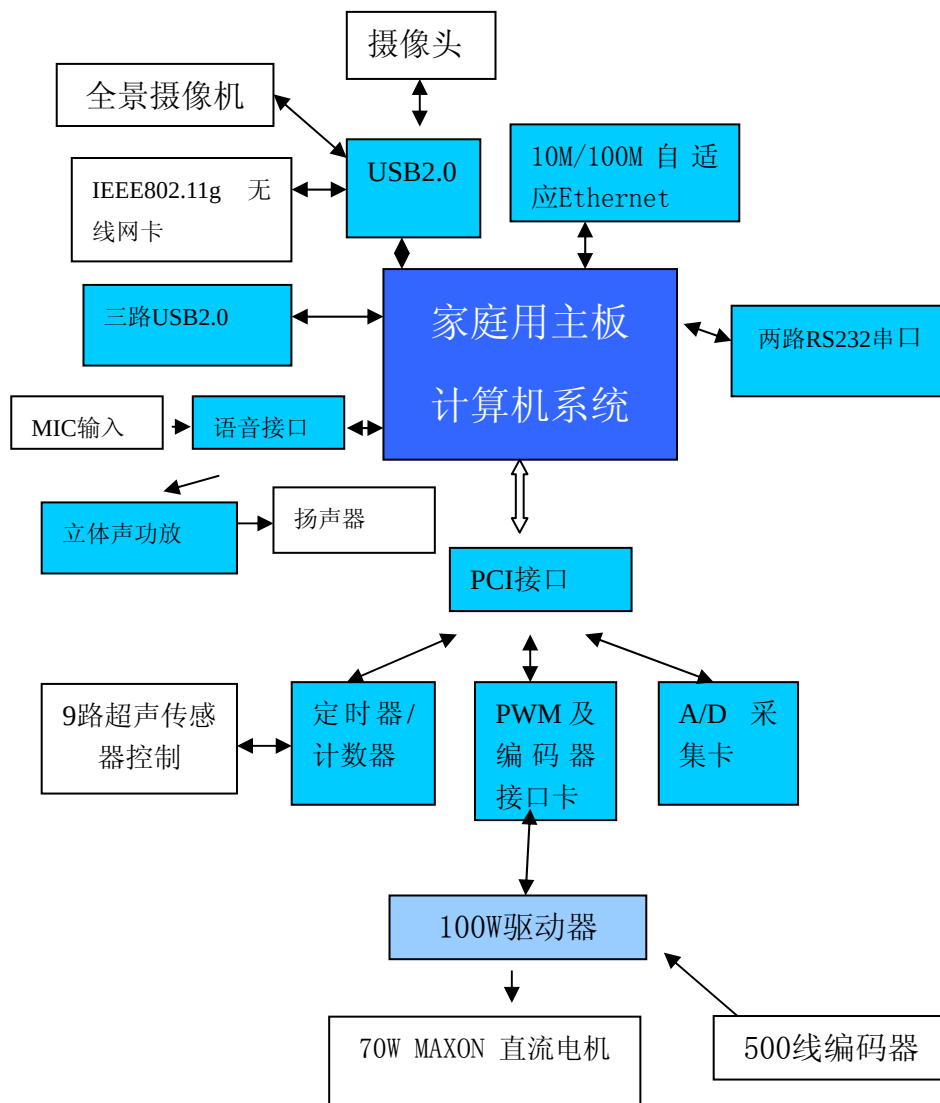
加速度传感器、GPS、三维数字罗盘均可安装在上层盖板上，如上图所示。
加速度传感器的控制电路板可以扩展安装到传感层总成内部。



第三章： MT-AR 机器人电子系统

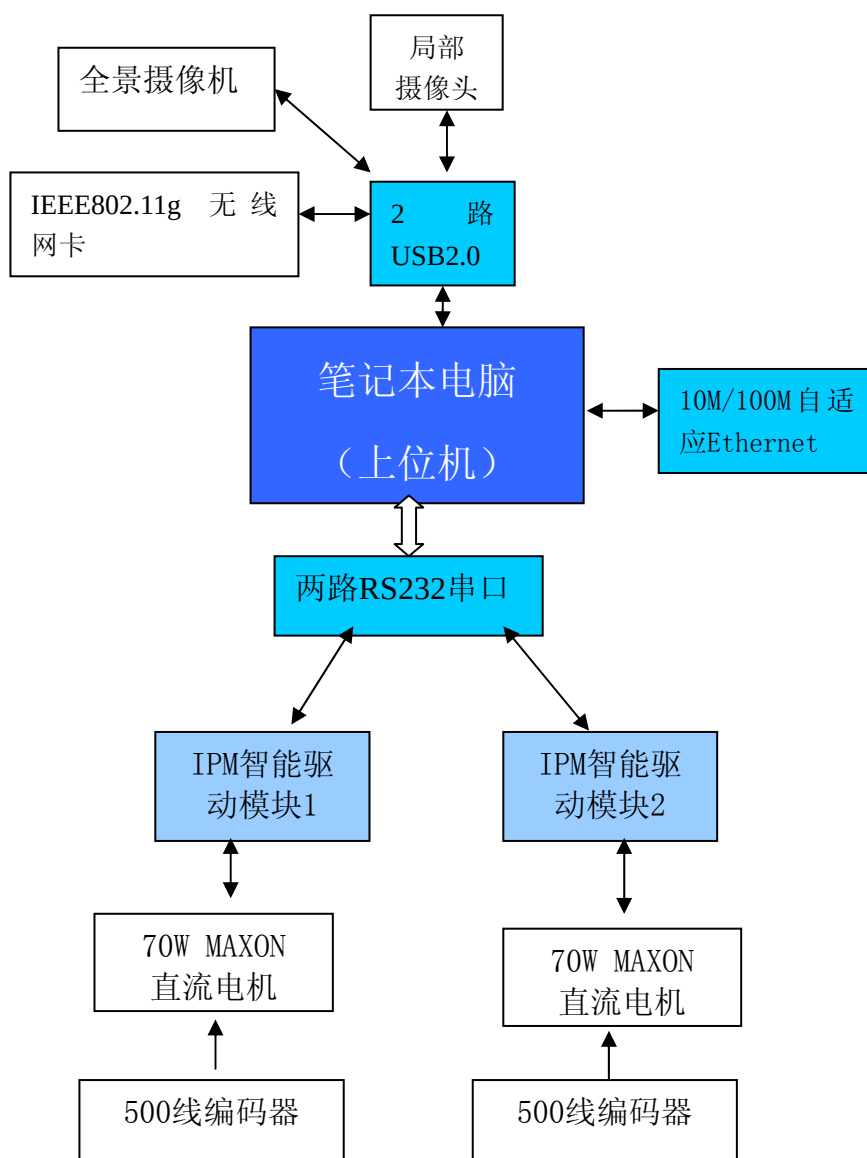
3.1 MT-AR 机器人系统架构设计及比较

3.1.1 一层架构：家用主板+PCI卡



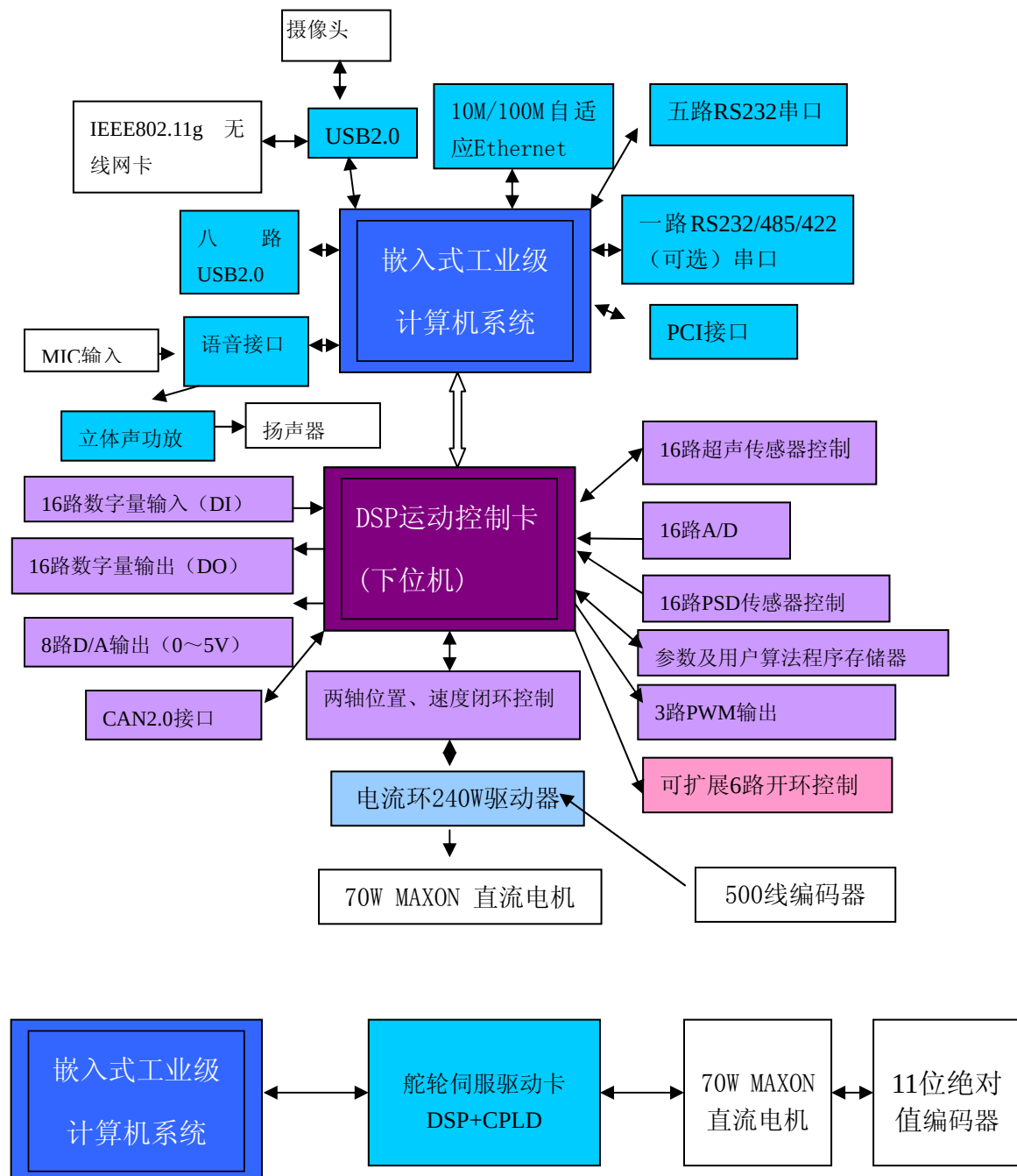


3.1.2 一层架构：笔记本+智能模块





3.1.3 MT-AR架构：二层架构：工业主板+运动控制卡





3.2 MT-AR 系统特点

1. 模块化设计：采用模块化设计，用户可以根据自己的需要采用不同的模块来实现自己制定的任务；
2. 智能体架构设计：两层架构，独立的运动规划与控制；
3. 独特的舵轮设计：建立运动学模型，左右驱动轮与舵轮协调控制；
4. 工业级系统设计：工业主板、工业产品架构、工业产品工艺；
5. 丰富的配件系统：图像、网络、超声、PSD、语音、GPS 等；
6. 扩展功能强：多路电子 DI/DO，AD/DA，网络通讯总线，机械扩展空间大，丰富的机械扩展配件；
7. 稳定性、可靠性好：采用工业级主板、工业级设置，各种保护(过流、过温、欠压)功能。

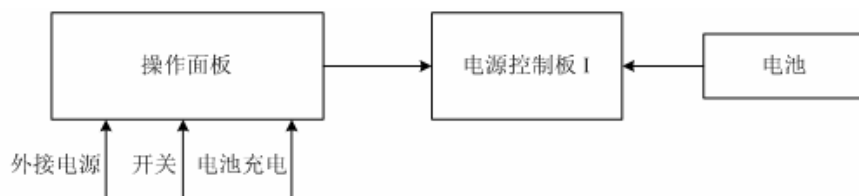
3.3 MT-AR 能源系统介绍

3.3.1 机器人能源系统功能介绍：

1. 输入能源：兼容锂电池、外接电源供电及供电切换、优先级选择；
2. 能源管理：具备锂电池电量显示、欠压报警及系统过流保护；
3. 输出能源：机器人系统整机供电，主要包含+24V、+12V、+5V；

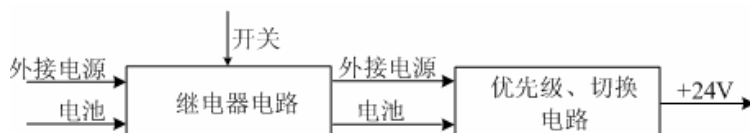
3.3.2 机器人能源系统实现：

1. 输入能源实现：



2. 锂电池、外接电源供电切换、优先级选择实现：

(1) 实现原理：



(2) 具体实现：



3. 电池电量显示实现：

(1) 实现原理



(2) 具体实现



LED指示灯状态显示

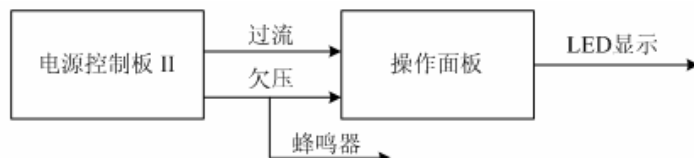
电池电压	LED 指示灯
>20%	D1 on
>40%	D1、D2 on
>60%	D1、D2 、D3 on
>80%	D1、D2 、D3、D4 on
100%	D1、D2 、D3、D4 、D5 on

4. 电池欠压、系统过流保护实现:

(1) 实现原理



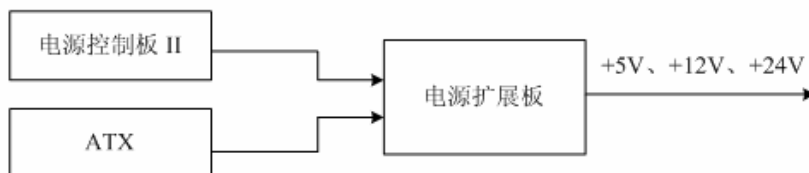
(2) 具体实现



说明：检测到电池电压低于22.5V，蜂鸣器及欠压指示灯报警，提示用户充电，电池电压高于24.5V，将退出报警；当机器人系统运行电流大于35A，系统断电保护，过流指示灯亮，此时用户需要关闭开关按钮，再重新打开机器人开关按钮，系统退出保护。

5. 电源扩展实现:

5. 电源扩展实现:



3.4 机器人控制系统

3.4.1 上位机控制系统功能介绍

机器人控制系统作为机器人系统的核心，它包括对机器人语音、图像、网络、运动及其它传感设备的控制，本机器人控制系统主要由上位机(嵌入式工业主板)



和下位机(运动控制卡)组成。

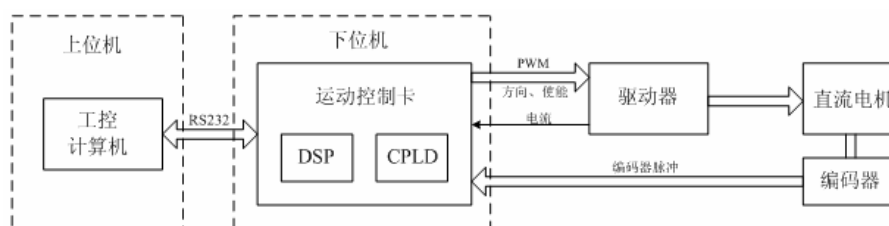
(1) 上位机(嵌入式工业主板)系统需求分析

作为研究版智能机器人，需要集图像、语音、网络、运动及传感于一体，概括起来，我们的上位机控制系统的核心需求或功能如下：

- 1) 控制系统的稳定性、可靠性、开放性、通用性、可扩展性；
- 2) 良好的人机界面、易上手及可操作性；
- 3) 能兼容语音、图像、网络及各种标准总线。

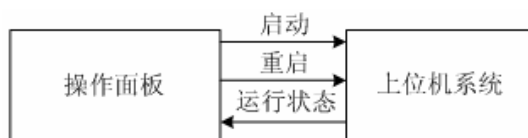
综合以上分析，我们选择嵌入式工业级主板作为上位机控制系统，从稳定性、可靠性及功能需求各方面都满足了以上需求分析。

3.4.2 实现原理及具体实现：

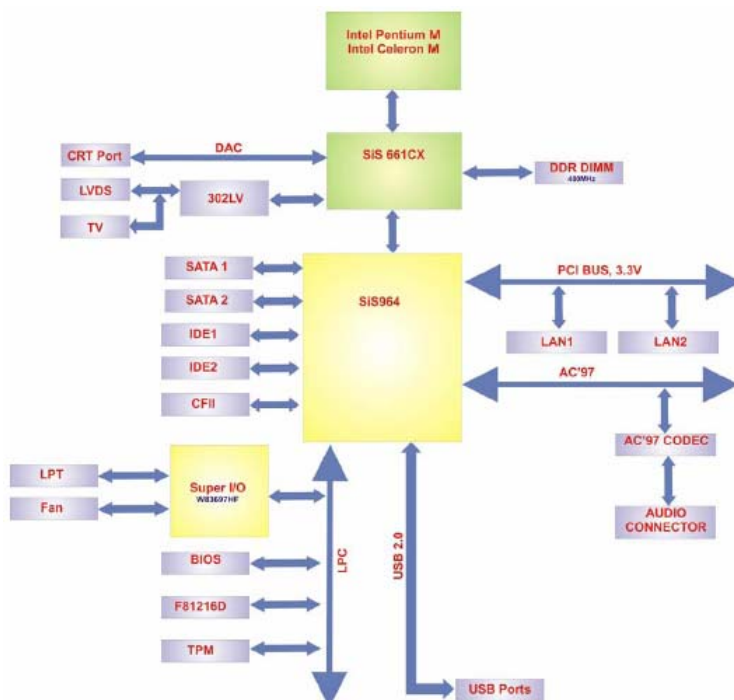


(1) 上位机(嵌入式工业主板)系统实现

1) 上位机启动\停止实现



2) 上位机系统架构

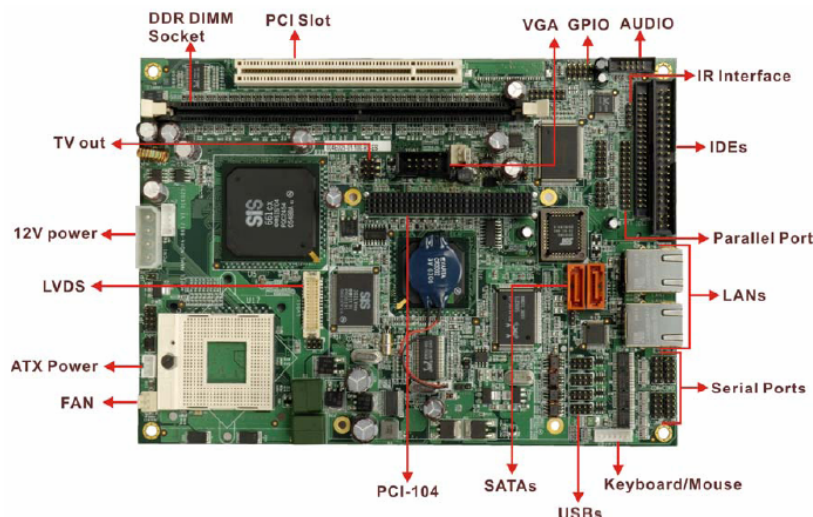




3) 上位机主要资源介绍

Detailed Specifications		
CPU	Socket 479 Intel Pentium M/Celeron M support 400/533 MHz to 2.26GHz	Pentium M 1.8GHz
System Chipset	SiS 661CX + SiS 964	default
BIOS	AMI BIOS	default
Graphics Support	SiS Mirage™ Graphic Engine	default
System Memory	1*184-pin DDR400/333MHz up to 1GB	512MHz DDR400
Ethernet	2*RTL8110SC GbE	extended
Display	CRT; LVDS; TV-out	VGA
PCI Bus Interface	33MHz, Revision 2.3	extended
Serial ATA (SATA)	2*SATA channels with 150MB/s transfer rates	1*SATA for 80G hard disk
HDD Interface	2*IDE channels supports four Ultra ATA/100 devices	extended
USB Interfaces	8*USB 2.0 connectors supported	extended
Serial Ports	6*COM ports	extended
Super I/O	W83697HG	extended
IrDA	By super I/O	extended
Digital I/O	4 input / 4 output by super I/O	extended
Audio Interfaces	One Audio Codec'97 (AC'97) version 2.3 connector	extended
Power	AT/ATX ,+5V, +12Vsupported	ATX
Operating Temperature Minimum: 0° C (32° F);Maximum: 60° C (140° F)		0° C to 60° C

4) 上位机结构介绍





3.4.3 下位机(运动控制卡)控制系统需求分析

就机器人运动控制系统而言,有多种实现方式,而其核心需求不只是带动电机运转,实现简单的闭环,概括起来,我们的运动控制系统的核心需求或者功能如下:

1) 作为学习运动控制系统的一个平台,我们要求运动控制系统能够把运动控制系统的基本实现方式、控制方式、运动过程、参数调节、典型系统、典型算法等体现出来,能够让学生有一个实践场所。

2) 运动控制系统必须综合当前最为先进的技术,是学生进行技术学习、算法验证、的最佳平台,这样的技术能够开放地展现在学生的面前,模块化地呈现这些关键技术的实现方式、工作方式、相关关联性等等。

3) 运动控制的二次开发能力要求,平台是学生进行算法实践、系统实践的重要工具,在通信、电子设计、软件设计、电源设计、接口设计等方面都需要具备开放性和模块化特点。

为此,我们在得到运动控制系统的详细设计前,分析了目前几种通用的控制实现方式:

1) 计算机+转接板+驱动器方式:这种方式对于专用系统来说是非常好的,速度快,处理能力强,但作为开放式系统、学习平台,同时又在计算机担负很多功能(如人机交互、图像处理等)时,这种方式就会被实时性、资源利用率等问题影响,所以不是一种好的选择。

2) 上位计算机+DSP 多轴控制卡+驱动器方式:这种方式将运动控制系统的规划+伺服闭环+放大器有层次、模块化地体现出来,同时伺服闭环的方式可选、参数可调节都为学生进行自己的运动控制系统算法实现提供便利。

3) 智能伺服模块:目前这是一个趋势,很多厂家在这方面进行了尝试,但概括起来,这种实现方式是前一种运动控制实现方式的缩减版本,在专用性上有独特之处,但是在方便学生学习内部结构这方面来说,还是比较欠缺的。

通过以上深入分析,我们选择 DSP 多轴控制卡作为机器人下位机的运动控



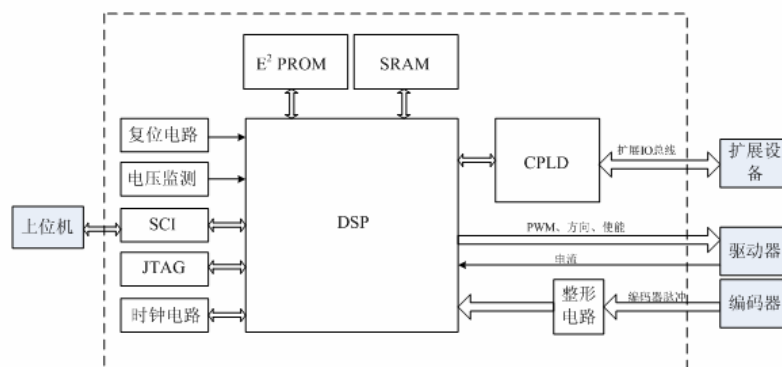
制系统，其优势主要体现在以下几个方面：

1) 选用 TI 公司 2000 系列 DSP，它是针对电机伺服控制开发的 DSP 芯片，自身集成了电机控制所需的各种功能，如 PWM 控制、编码器信息采集等，同时，还集成了 SCI、SPI、CAN 总线通讯，16 通道 10 位 AD 采集以及数字量输入、输出信号；

2) 采用 40MHz 工作频率，四级流水线，满足了机器人运动控制实时性的要求，同时，高精度的定时周期，还满足了超声等对实时性要求较高设备的要求；

3) 独立的运动控制系统，多种总线通讯，使系统更具模块化，易于维护及二次开发。

3.4.4 下位机(运动控制卡)控制系统实现原理



运动控制卡的核心处理器采用 TI 公司的 DSP 芯片 TMS320LF2407，它是一种专门针对控制领域而开发的高性能 16 位定点 DSP 芯片。其主要特点为：采用高性能静态 CMOS 技术，供电电压仅 3.3 V，降低了控制器功耗；两个事件管理器模块 EVA 和 EVB，分别包括两个 16 位通用定时器、8 个 16 位脉冲调制(PWM)通道、3 个捕获单元、1 个正交编码脉冲单元 (QEP)，正交编码脉冲单元可对光电编码器输出的相位相差 90° 的 A、B 两路脉冲信号进行鉴相和四倍频；16 通道模数转换模块 (ADC)；串行外设接口模块 (SPI)；串行通讯接口模块 (SCI)；CAN 控制器模块；40 个可单独编程或复用的通用 I/O。为了满足机器人大量扩展传感器的需要和减少处理器外围数字逻辑器件的使用数量，该运动控制卡采用 Altera 公司的复杂可编程逻辑器件 (CPLD) 来扩展 DSP 的 I/O 端口和实现外围数字逻辑电路设计。采用 CPLD 不仅增强了运动控制卡的扩展性，还增加了电路板的可靠性和设计灵活性，其在线可编程特性使得数字逻辑电路设计如同软件设计一样简便。

除 DSP 和 CPLD 之外，运动控制卡还包括：复位电路、时钟电路、电压监测电路、JTAG 接口电路、RS-232 总线驱动电路、编码器脉冲整形电路、外部扩展存储器 E2PROM、外部扩展存储器 SRAM。下面分别介绍各单元及其在运动控制卡中的作用。



1) 复位电路和时钟电路, 系统上电后必须将 DSP 的 RS 引脚拉为低电平对 DSP 的寄存器和 I/O 端口进行复位, 复位电路采用 RC 延时电路并联轻触开关, 可实现上电复位和手动复位; 时钟电路采用 20 M 晶体振荡器, 为 DSP 芯片提供时钟信号。

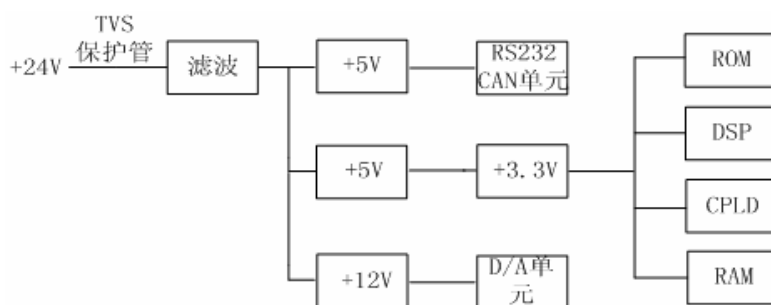
2) RS-232 电平转换电路, 运动控制卡和工控计算机之间采用 RS-232 总线通讯方式, 但由于 DSP 无法直接驱动 RS-232 总线, 因而需要总线驱动电路, 这里采用符合 RS-232 标准的驱动芯片 MAX232。

3) 编码器脉冲滤波整形电路, 由于编码器反馈的脉冲可能收到一些外部干扰, 而出现一定程度的变形和产生尖脉冲, 造成 DSP 脉冲计数错误, 因而需要滤波整形电路。

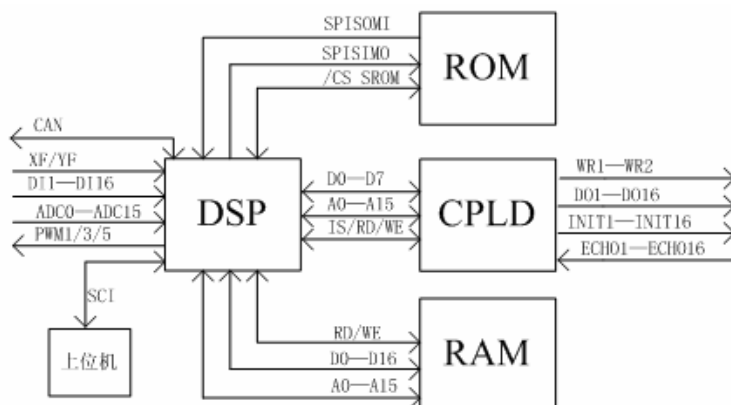
4) JTAG 接口电路和 SRAM, DSP 芯片通过 JTAG 接口电路和仿真器连接, 同时为了方便在线仿真调试需要扩展外部 RAM, 调试时通过 JTAG 口将程序下载到外部 RAM 中执行, 调试通过后再将程序烧写到 DSP 的内部 FLASH 中。这里 RAM 采用 64K 静态随机存储器。

5) 电压检测电路和 E2PROM, 系统掉电时, 电压检测电路检测到系统电压低于一个设定的域值时, 就会给 DSP 一个信号, DSP 立刻将一些需要保存的数据保存到外部扩展的 E2PROM 中, 下次系统启动时可以从 E2PROM 中读取数据, 也可以将一些运动控制的参数比如 PID 参数, 保存在 E2PROM 中, 编程实现系统上电自动加载这些参数。E2PROM 采用 SPI 接口的 8K×8bit 存储器。

电源单元:

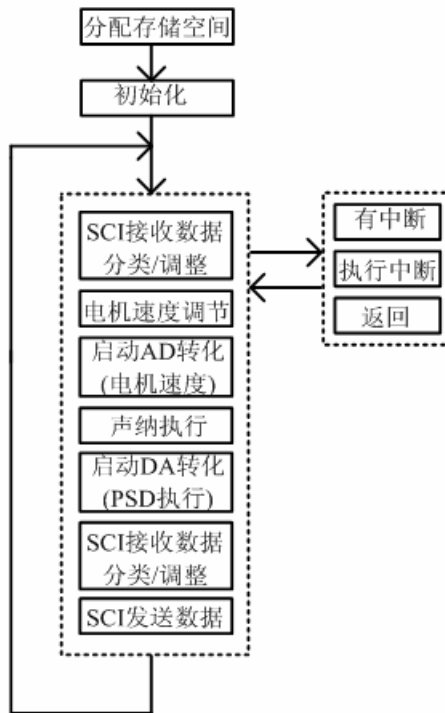


控制单元:

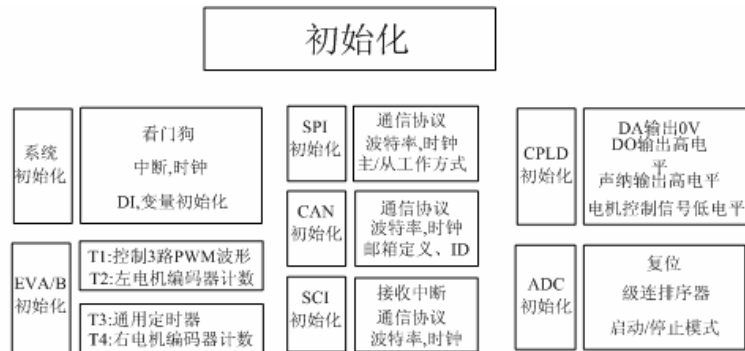




DSP 软件设计:



其中初始化主要指以下单元:

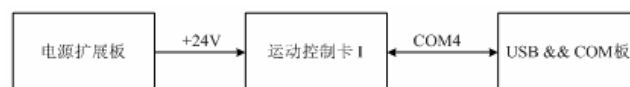


分配存储空间和中断主要指以下单元:

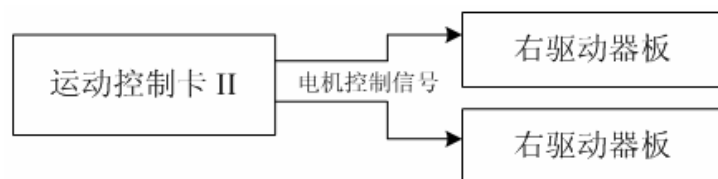


3.4.5 下位机(运动控制卡)控制系统实现

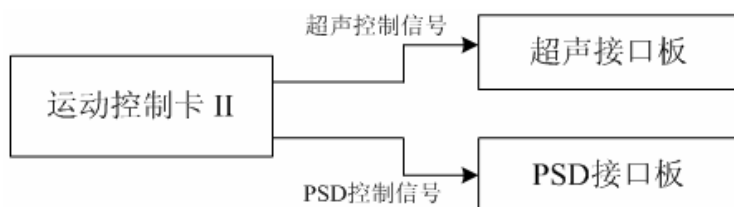
(1) 电源、通讯:



(2) 机器人左、右电机控制:



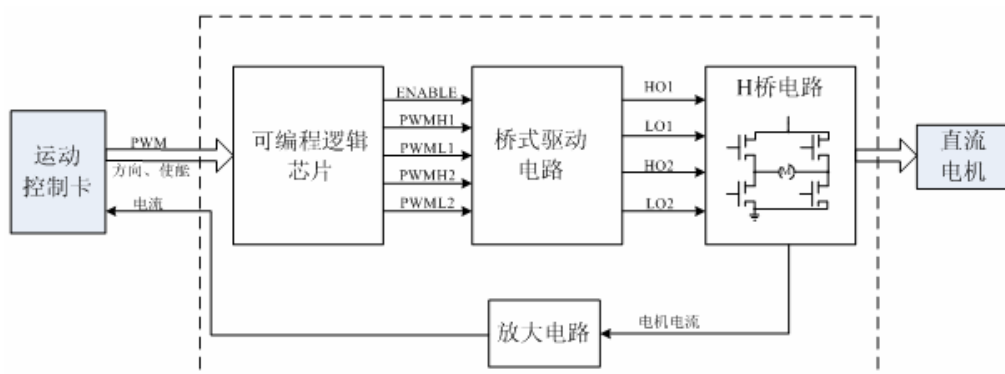
(3) 机器人超声、红外控制:



说明: 系统上电后, 运动控制卡 I 五个电源 (+12V、+5V、+3.3V、-12V、+5V) 指示灯点亮, DSP 状态指示灯闪烁, 说明运动控制卡工作正常; 运动控制卡 II 电源指示灯点亮。

3.5 机器人驱动系统

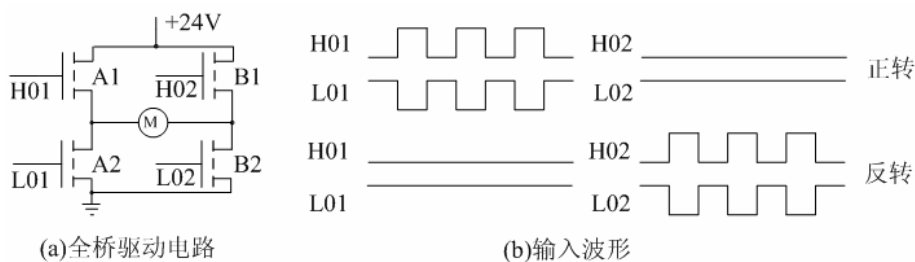
3.5.1 实现原理:



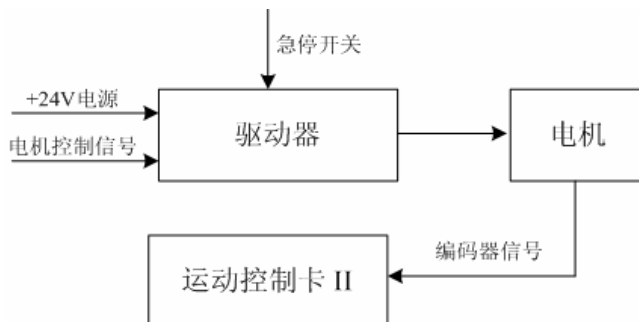
(1) 单极性H桥驱动原理

移动机器人的前后左右移动, 要求电机能够实现正反转, 这就需要使用可逆 PWM 系统。可逆 PWM 系统分为单极性驱动和双极性驱动。单极性驱动是指在一个 PWM 周期里, 电动机电枢的电压极性呈单一性 (或者正、或者负) 变化。双极性驱动是指在一个 PWM 周期里, 电动机电枢电压极性呈征服变化。由于单极性驱动较易于实现, 该驱动器采用单极性 H 桥式方式来驱动电机。

单极性 H 桥驱动电路原理及输入波形如图 3 所示, H 桥电路主要由 4 个 VMOS 管组成, 当电机正转时, 要求 B2 导通, B1 截止, 调节 A1 的输入信号 HO1 的占空比实现对电机的调速, A2 的输入波形则要求与 A1 的输入波形反向; 同理, 当电机反转时, 要求 A2 导通, A1 截止, 调节 B1 的输入信号 HO1 的占空比实现对电机的调速, B2 的输入波形则要求与 B1 的输入波形反向。



3.5.2 具体实现



说明：系统上电后，驱动器电源指示灯为绿色，当驱动器过流或者过温，指示灯变为红色，驱动器停止工作，5 秒后，指示灯再次恢复为绿色，驱动器恢复正常工作。

3.6 传感系统

3.6.1 距离传感器

(1) 声纳传感器

MT-AR 研究版智能机器人配有声纳传感器，有效探测距离为 20cm 到 7m。MT-AR 能安装 14 个声纳传感器，缺省配置是 6 个声纳传感器。

1) 声纳传感器工作原理

声纳传感器的测距工作原理：声纳激发出一束很窄的超声波在空气中传播，当遇到障碍物时，超声波返回。根据超声波的传递时间就能准确的计算出障碍物的相对的距离。

MT-AR 研究版智能机器人采用的是集成声纳测距传感器，其特点是集成度高、使用简单，可靠性好。首先将声纳的 INIT 信号拉到高电平，此时声纳换能器将激发出超声波。当超声波遇到障碍物时，超声波将被返回，声纳传感器检测到回波信号后，将 ECHO 置为高电平。因此可以计算障碍物的距离：障碍物的距离=（INIT 高电平时间—ECHO 高电平的时间）×声纳速度/2。

2) 声纳测距模块参数

声纳测距传感器的测量范围：20cm—7m；

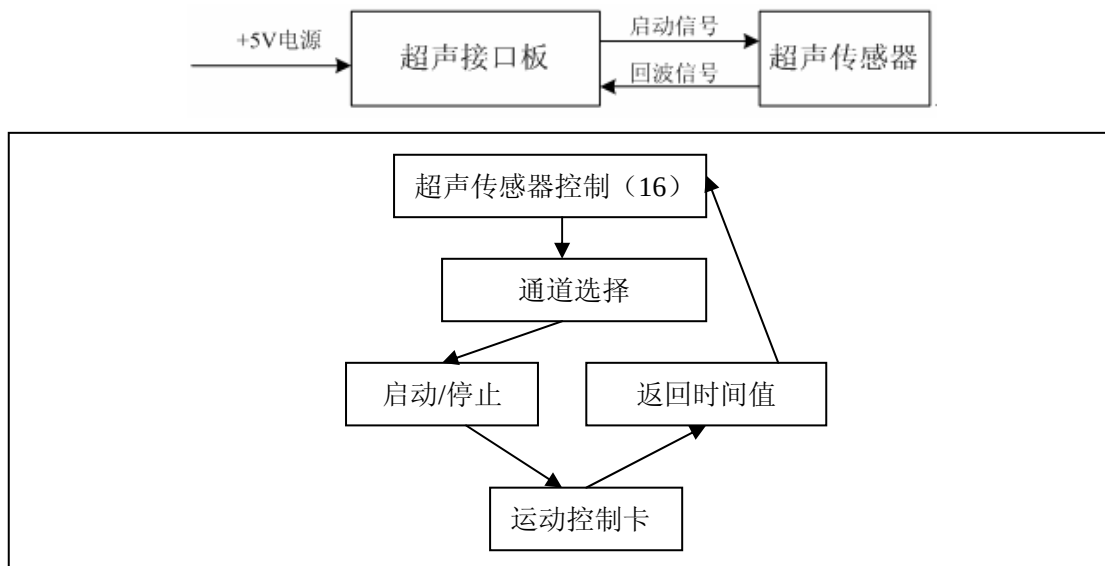
声纳测距传感器的测量精度：±5%；



声纳测距传感器的测量频率：20Hz；

声纳测距可以由用户选择不同的激发方式。可以选择单个声纳轮询测量，也可以选择任何几个声纳同时测量。

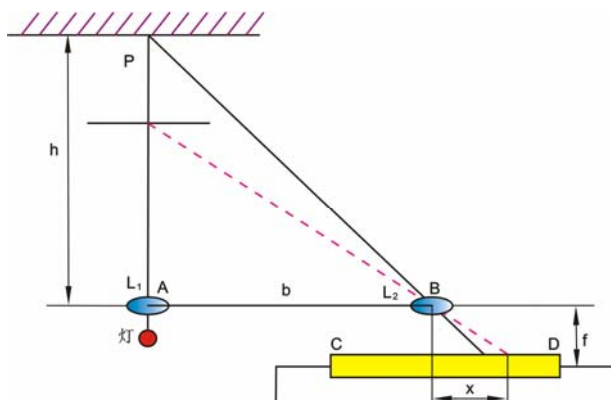
3) 具体实现：



(2) PSD 传感器

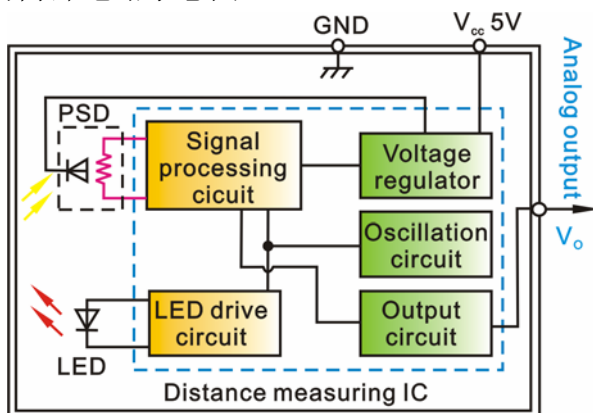
PSD(Position Sensitive Detector)传感器是一种对入射光位置敏感的光电器件。即当入射光照在器件感光面的不同位置时，PSD 将输出不同的电信号。通过对此输出电信号的处理，即可确定入射光点在 PSD 器件感光面上的位置。PSD 可分为一维 PSD 和二维 PSD。一维 PSD 可以测定光点的一维位置坐标，而二维 PSD 则可以检测出光点的平面位置坐标。

1) PSD 传感器工作原理



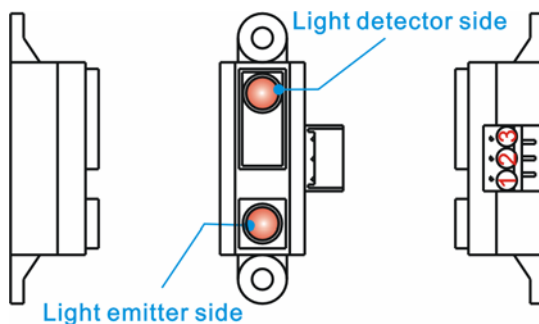
应用 PSD 进行距离的测量是利用了光学三角测距的原理。如上图所示，光源发出的光经透镜 L1 聚焦后投向待测体的表面。反射光由透镜 L2 聚焦到一维 PSD 上,形成一个光点。若透镜 L1 与 L2 间的中心距离为 b ，透镜 L2 到 PSD 表面之间的距离(即透镜 L2 的焦距)为 f ，聚焦在 PSD 表面的光点距离透镜 L2 中心的距离为 x ，则根据相似三角形的性质，可得出待测距离 h 为：

因此，只要测出 PSD 的光点位置坐标 x 的值，即可测出待测体的距离。下图为 PSD 传感器的内部电路示意图：



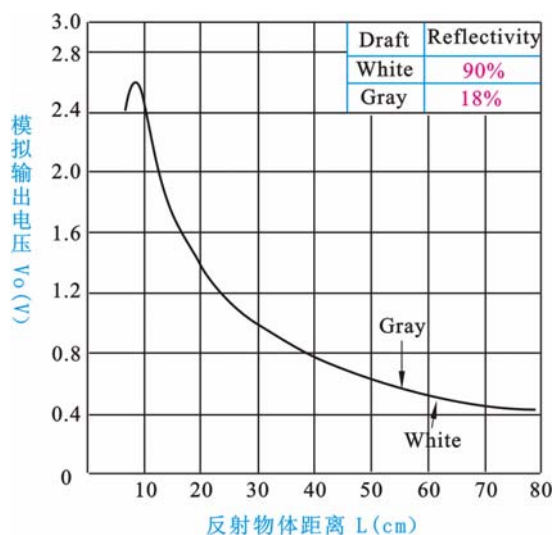
同许多传统的光电位置敏感器件如象限光电池、CCD 等相比, PSD 位置分辨率高、响应快、性能价格比较高, 特别适合于对位置、位移、角度等的实时测量。由 PSD 构成的距离测量系统具有非接触、测量范围大、响应速度快等优点。

外部特征如下:

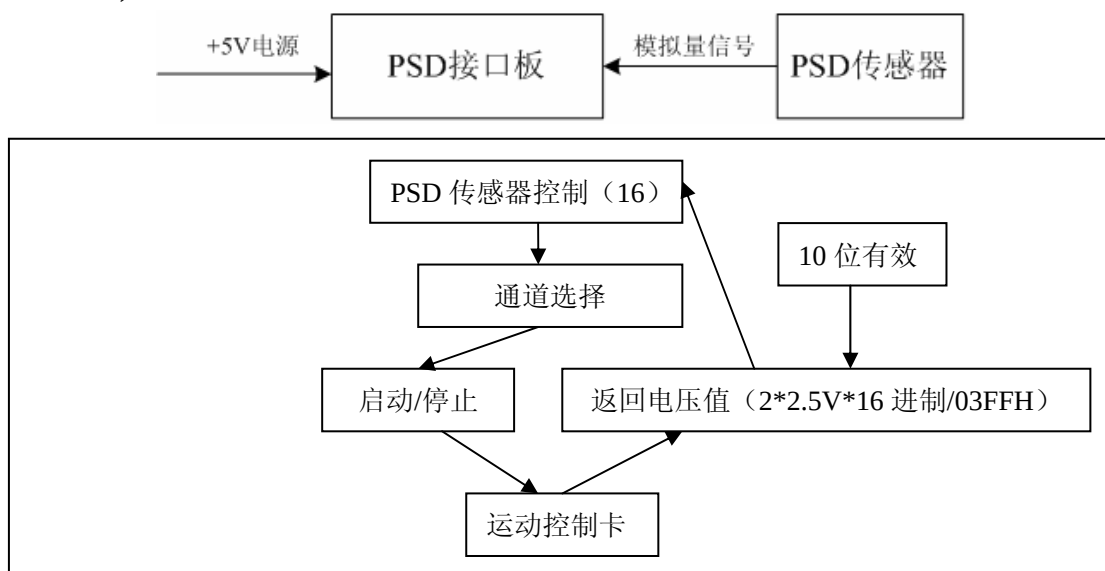


① V_o ② GND ③ V_{cc}

PSD 传感器输出电压与距离关系如下图:



2) 具体实现

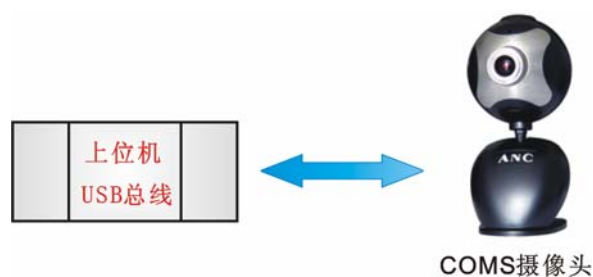


3.6.2 语音传感模块

MT-AR 语音传感器直接利用控制主板本身集成的音频输入和输出来进行的。音频的输入通过 5W 的立体声功放接到的 MT-AR 下部的喇叭，MT-AR 配带有语音识别及合成软件。可以利用 MT-AR 缺省的麦克风接口和扬声器来进行此方面的研究及演示。

3.6.3 视觉传感模块

MT-AR 研究版智能机器人缺省配置的视觉系统如图：





MT-AR 研究版智能机器人还可以通过 PCI 接口视频采集卡扩展各种视频设备，如立体视觉、全景摄像头、云台摄像机等。

3.7 网络通讯系统

MT-AR 研究版智能机器人主要涉及到以太网、PCI、PC104、USB、串口、并口、SATA、IDE、SPI、CAN 总线。其中，MT-AR 本体已使用 USB、串口、SATA、SPI 总线，其它总线可供用户二次开发。

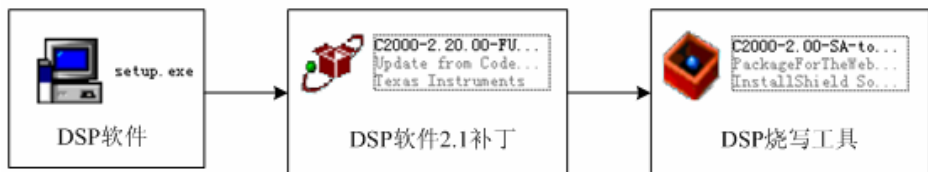
USB、串口、PCI、以太网接口实现：



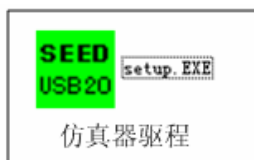
3.8 MT-AR 二次开发

3.8.1 DSP开发环境 CCS使用介绍

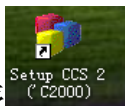
(1). DSP软件安装



(2). 仿真器驱动程序安装



(3). 首次启动DSP开发软件CCS需配置CPU

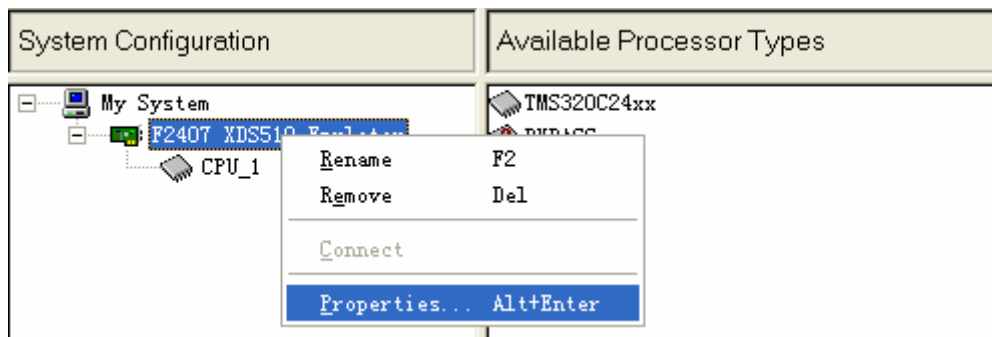


- 1). 打开CCS配置程序
- 2) 配置CPU
 - A) 删除默认的 CPU，选择 F2407



选择F2407
XDS510→Import

B) 配置 CPU



C) 箭头处为需要改变的设置



D) 点击Next, 启动CCS

如出现下图提示, 则证明仿真器与DSP没有成功连接。出现该种情况可能的原因:

- 1: 机器没有上电

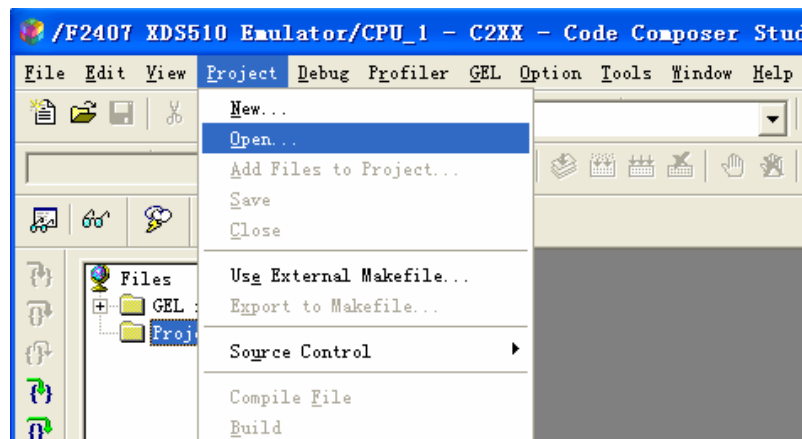


- 2: 仿真器驱程没有装
- 3: CPU配置有误



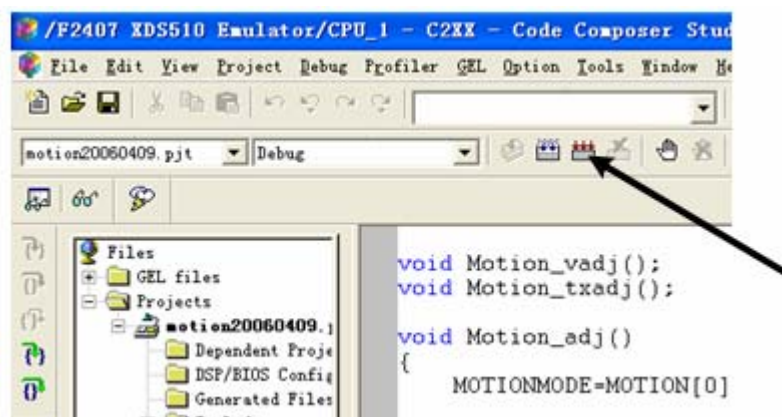
(4). 启动CCS

A) 打开工程

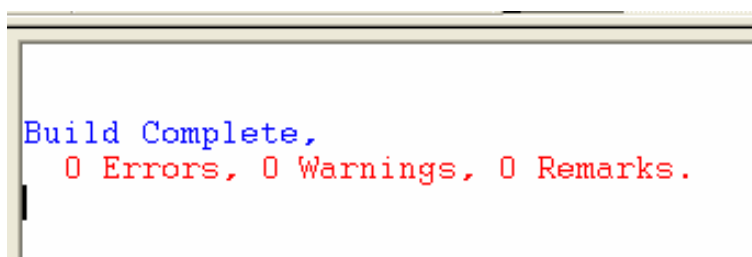


B) 编写代码

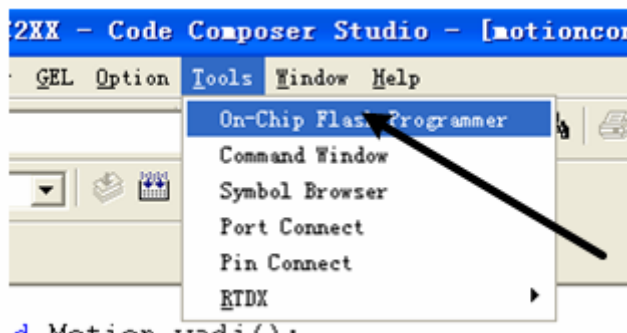
C) 编译工程



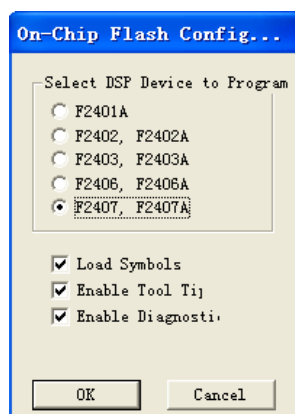
D) 编译成功，出现以下提示



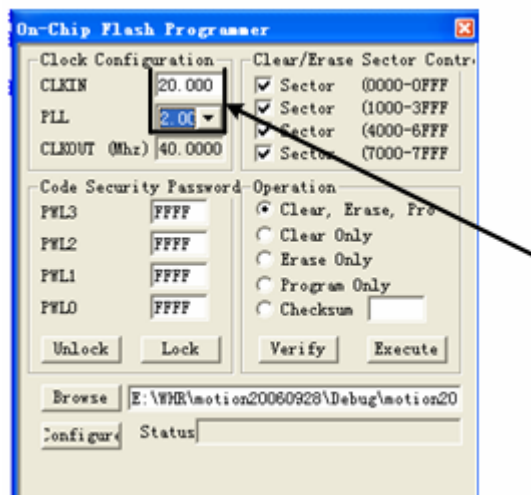
- (5). 烧写代码：选择箭头处（注：跳线需设置为 MC 模式）
A)



- B) 选择 OK



- C) 箭头处为需修改的参数

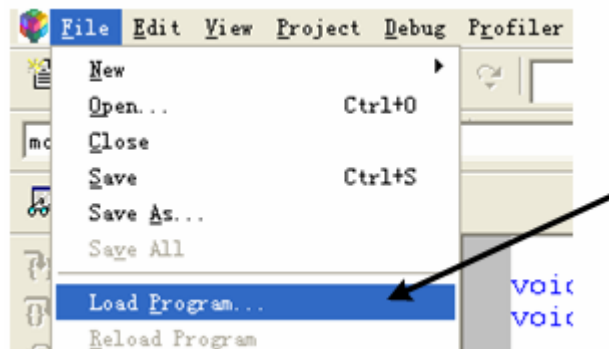




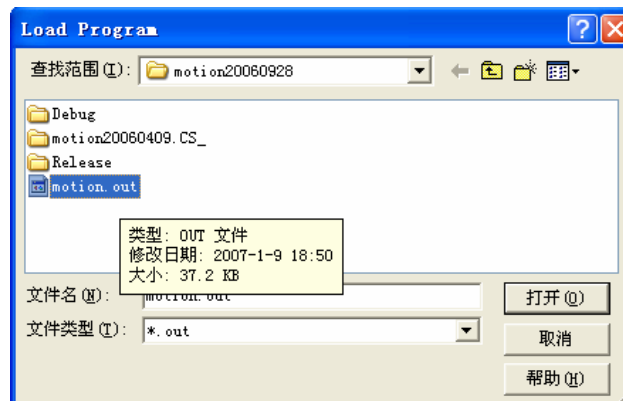
- D) 单击 Execute，开始烧写代码（注：生成的.out 文件路径应尽量浅）
E) 程序烧完，断电，去掉仿真器，即可运行程序。

(6). 仿真代码（注：跳线需设置为 MP 模式）

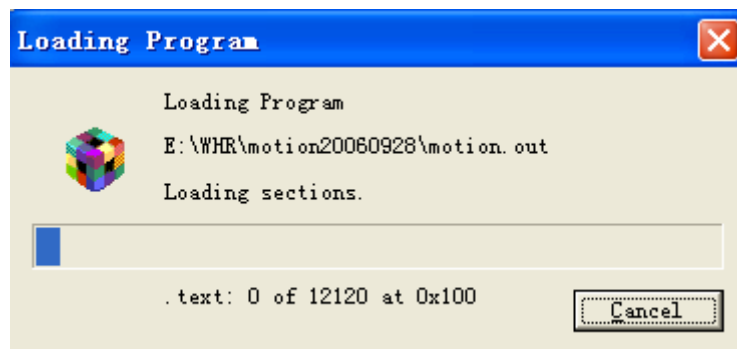
- A) 启动 CCS，打开项目，编译项目，同上。
B) 加载代码。选择箭头处



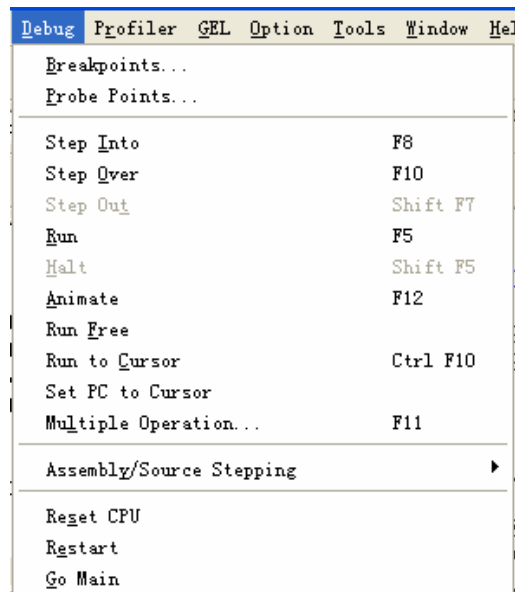
C) 选择.out 文件



D) 加载代码



E) 调试程序



Step Into: 程序单步运行

Run: 程序全速运行

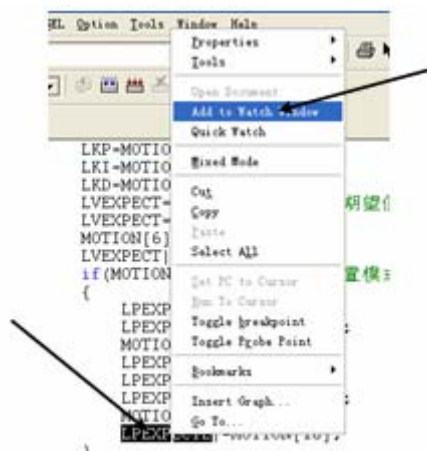
Halt: 停止程序

Run to Cursor: 程序运行到光标处

Reset CPU: 复位 CPU

Restart: 重启程序

F) 查看变量值。单击箭头处



G) Value 即为变量的值

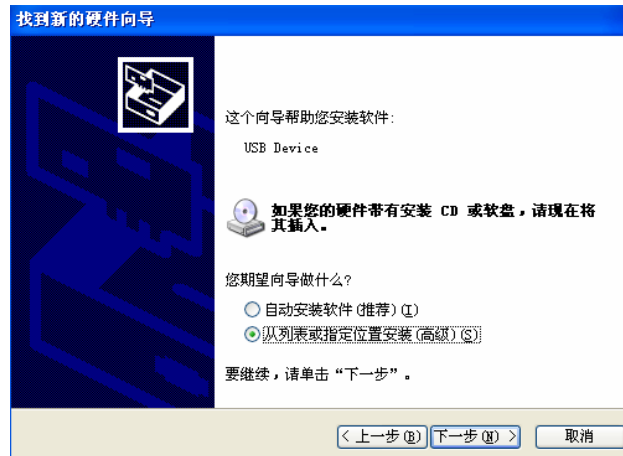
Name	Value	Type	R...
LPEXPECTL	4983	u...	u...

Watch Locals Watch 1

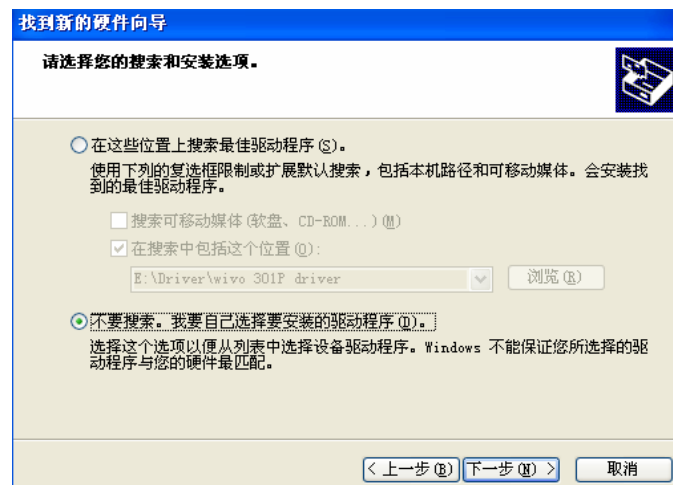


(7). 仿真器驱动安装说明

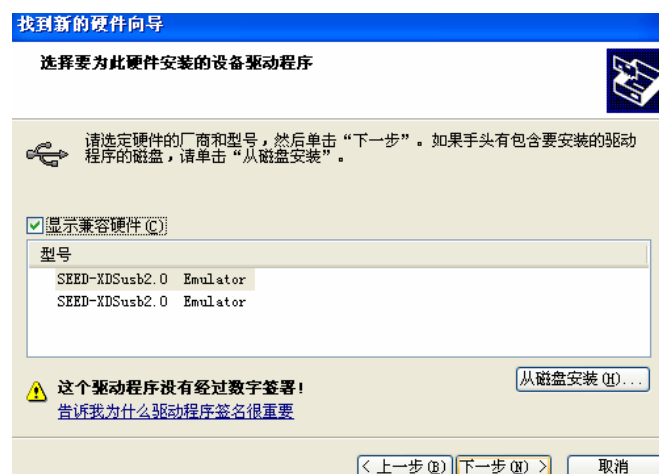
A)



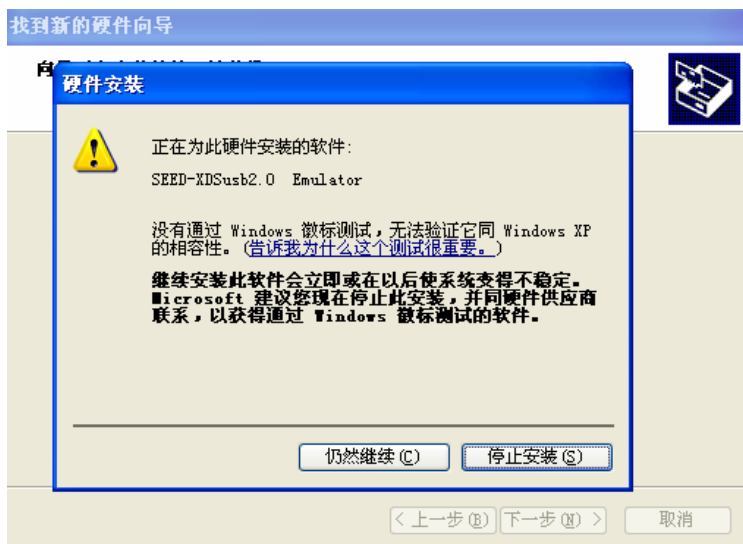
B)



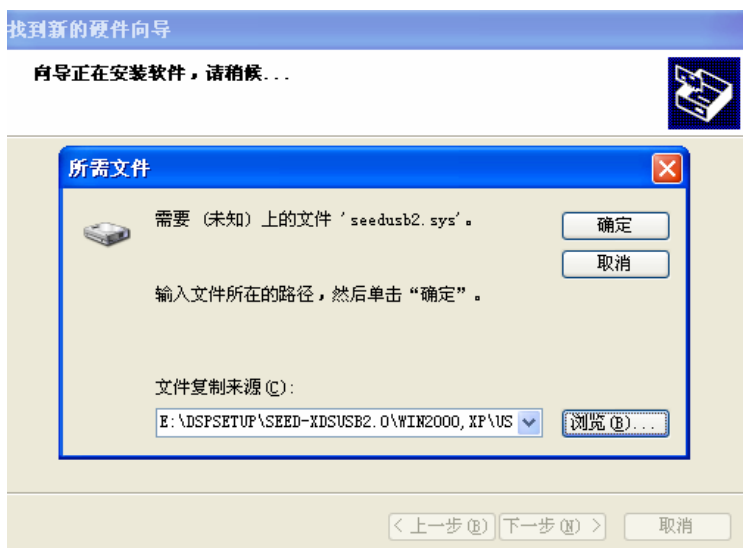
C)



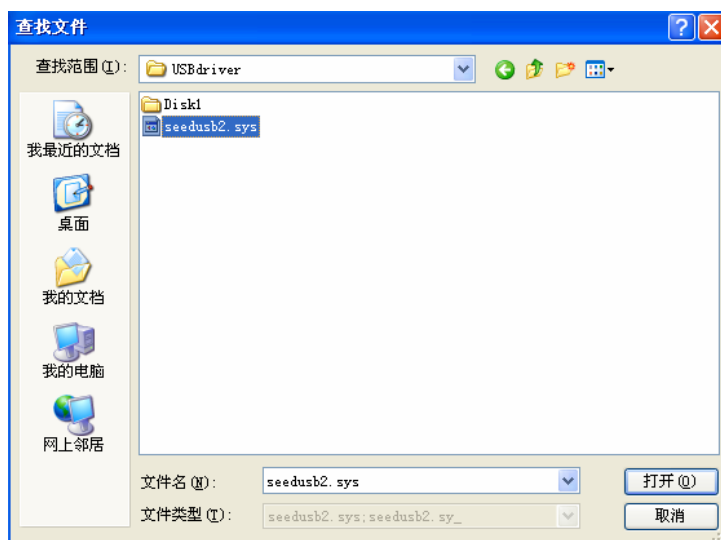
D)



E)

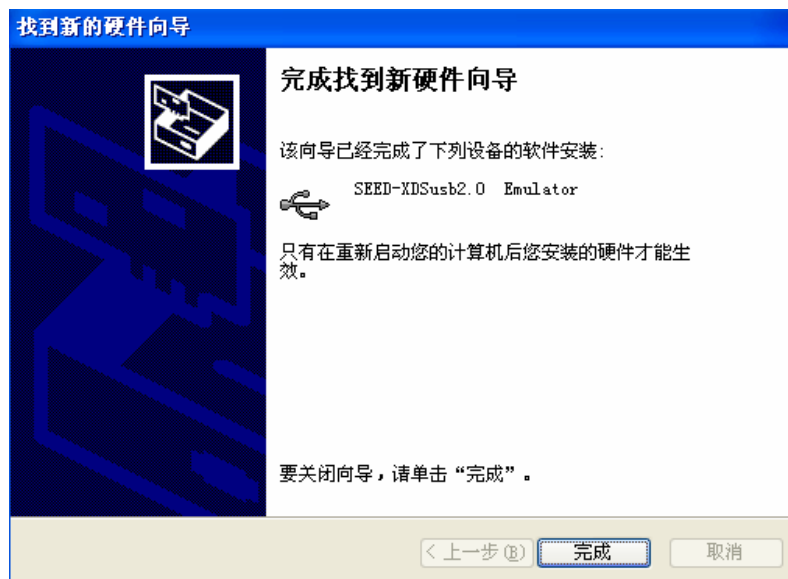


F)





G)

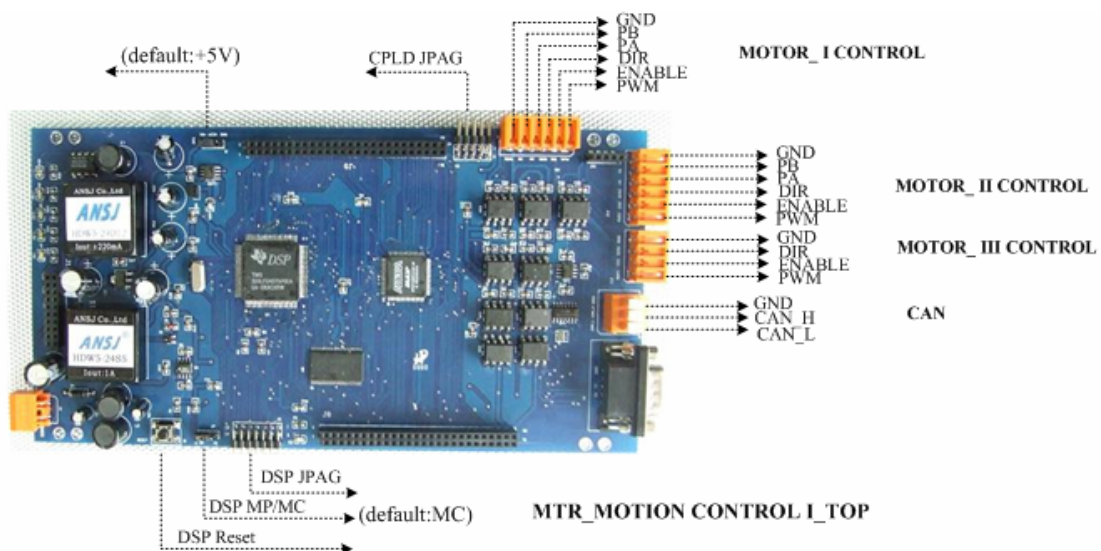


3.8.2 运动控制卡扩展功能介绍

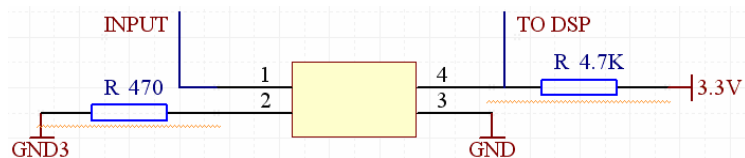
本节所介绍的MT-AR二次开发主要围绕运动控制卡展开，用户需要对MT-AR有较深的认识。

运动控制卡可扩展资源：

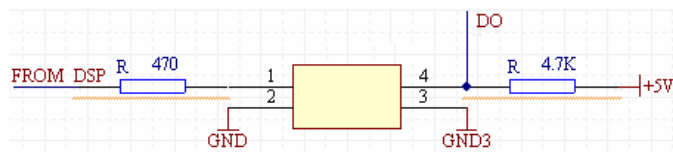
- (1). 数字量输入、输出；
- (2). D/A、A/D控制；
- (3). 电机运动控制；
- (4). CAN总线。



1. 数字量输入、输出信号采集\控制：
- (1) 电路原理图：

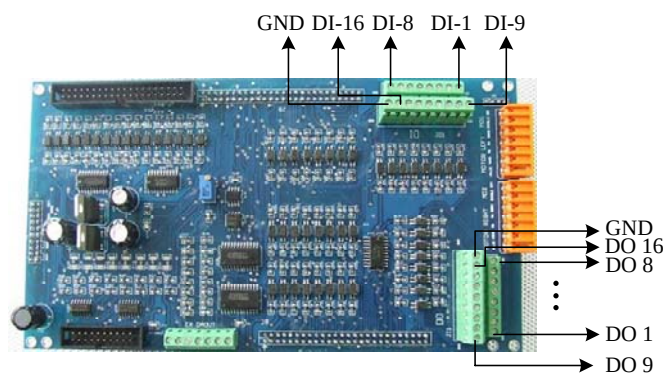


DI 原理图



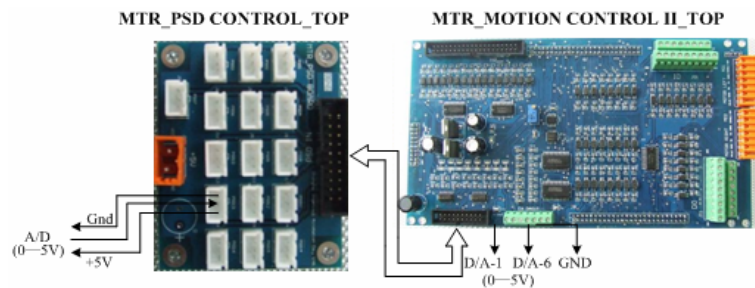
DO 原理图

(2) 具体实现:

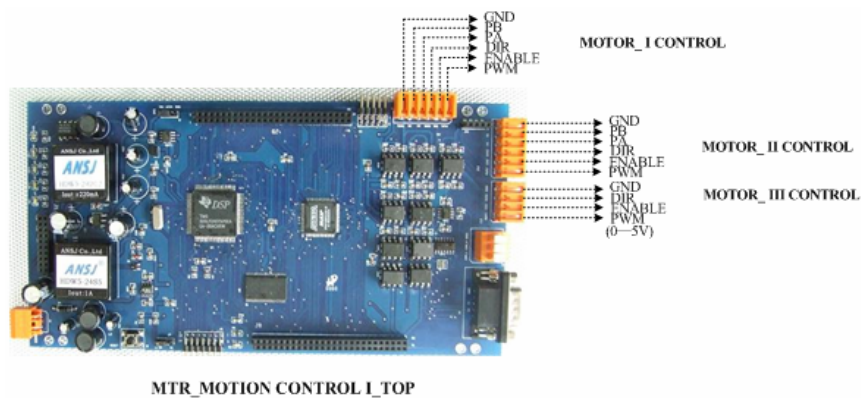


MTR_MOTION CONTROL II_TOP

2. A/D、D/A采集控制:



3. 电机运动控制:





第四章 MT-AR 机器人软件系统

4.1 MT-AR 软件结构介绍

MTR 机器人是一种通过各种传感器系统感知外界环境实现在复杂的未知环境中自主移动并完成一定任务的机器人系统。其实现需要有多个功能特性各异的模块，如：感知、任务分析、规划、推理、决策和动作执行等，并且各个功能模块以不同方式在不同层次协调工作。体系结构作为机器人的设计框架，从整体上联系着智能机器人的各功能模块，并在软硬件的实现上起着统一调配和决定性作用。因此智能移动机器人的体系结构决定了机器人本身的功能特点，也是首先要考虑的问题。该结构除了具有自主性、智能性等基本特点外，还需要满足以下要求：1) 整个结构既具有较强的应急反应能力，也具有远期规划能力；2) 具有较强的扩展性，可以不断增强完善自身的各项功能。

考虑到智能机器人体系结构的基本要求，软件设计采用了基于多智能体的结构。多智能体系统是一种分布式智能系统，其结构方式类似网络形式，其中基本节点为智能体，如图 1 所示，每个智能体是独立的，应用程序通过创建和销毁来管理每个智能体的生命周期，而智能体之间则是通过消息传递来互相通讯，如果智能体需要与下位机通讯，则通过串口发送命令，如果需要得到传感器的测量值，也可以通过串口实时返回到上位机。这样保证了程序的良好开放性。如果需要添加新的虚拟设备，只需要创建新的虚拟设备接口就可以了，不会影响其他设备，也不会破坏原来的软件结构。

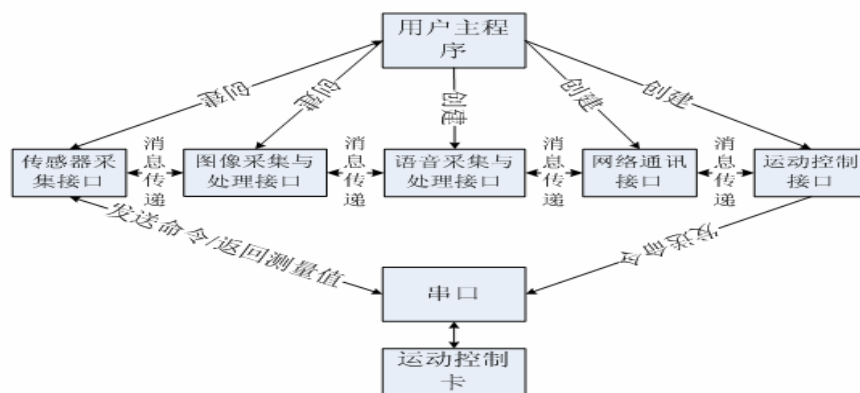


图 1 软件结构图

运动控制接口：通过串口发送运动控制指令，控制电机正转反转及停止；



传感器采集接口：发送传感器启动命令，采集超声，PSD，码盘等到障碍物之间的距离值；

图像采集与处理接口：采集图像，图像处理如图像的色彩体系转换、图像的各种处理算法等；

网络通讯接口：实现对网络的操作，包括网络数据的传输、网络图像传输等；

语音采集与处理接口：声控控制机器人运动；

MTR 软件除了能做这些基本的运动控制、整机检测、图像采集、网络传输、语音识别、超声检测、PSD 检测、DI 检测、DO 检测、DA 检测、码盘检测外，它还具有很多其他的功能：

人脸识别：能够在复杂的环境下，检测到人脸，并可进行人机对话。

三维数字罗盘：能够测得机器人此时的水平角度、俯仰角度和翻转角度值，根据这些值，机器人在野外行走也不怕了。

平面罗盘：能够测得机器人与初始运动方向的角度值，根据这个值加入 PID 算法，机器人就按着你的要求行走了。

GPS：能够得到当前的时间、东经、北纬、距离海平面的高度等值，把机器人放在野外，就可以轻松让机器人走到目标点了。

加速度计：能够测得机器人在运行过程中的加速度值。

陀螺仪：能够测得机器人的角速度值和角度值。

激光：能够构造周围的环境地图，机器人就能在未知环境下安全行走了。

云台摄像机：能够采集 S 端子和复合端子的图像，并能控制云台水平旋转，俯仰旋转，设置白平衡，改变转动速度等很多的功能。

双目摄像机：能够获得深度图，用它可以做环境构图、路径规划等研究。

全景摄像机：能够获得 360 度图像信息，用它可以做 Robocup 足球比赛，也可以做全景展开等研究。

带有力反馈的方向盘：能够控制机器人运动，如同开真正的车一样，并且具有力觉临场感能使你有亲临其境的感受。

除上述功能外还可以在机器人上添加很多的传感器，让机器更具有人的智能。



4.2 MT-AR 编程知识准备

MTR 机器人软件开发环境为 Windows XP/2000, 编程语言使用 Visual Studio C++.NET 2003, 应用程序框架采用 Microsoft 基本类库 7.0 版 (简称 MFC 库)。

4.1 MFC 概述

MFC (Microsoft Foundation Class Library) 中的各种类结合起来构成了一个应用程序框架, 它的目的就是让程序员在此基础上来建立 Windows 下的应用程序, 这是一种相对 SDK 来说更为简单的方法。因为总体上, MFC 框架定义了应用程序的轮廓, 并提供了用户接口的标准实现方法, 程序员所要做的就是通过预定义的接口把具体应用程序特有的东西填入这个轮廓。Microsoft Visual C++.NET 提供了相应的工具来完成这个工作: AppWizard 可以用来生成初步的框架文件 (代码和资源等); 资源编辑器用于帮助直观地设计用户接口; 属性用来协助添加代码到框架文件; 最后, 编译, 则通过类库实现了应用程序特定的逻辑。

4.2 MFC 应用程序的构成

以 MDI 应用程序为例

用 AppWizard 产生一个 MDI 工程 MyDemo (无 OLE 等支持), AppWizard 创建了一系列文件, 构成了一个应用程序框架。这些文件分四类: 头文件 (.h), 实现文件 (.cpp), 资源文件 (.rc), 模块定义文件 (.def), 等。

1、构成应用程序的对象

图 2 解释了该应用程序的结构, 箭头表示信息流向。

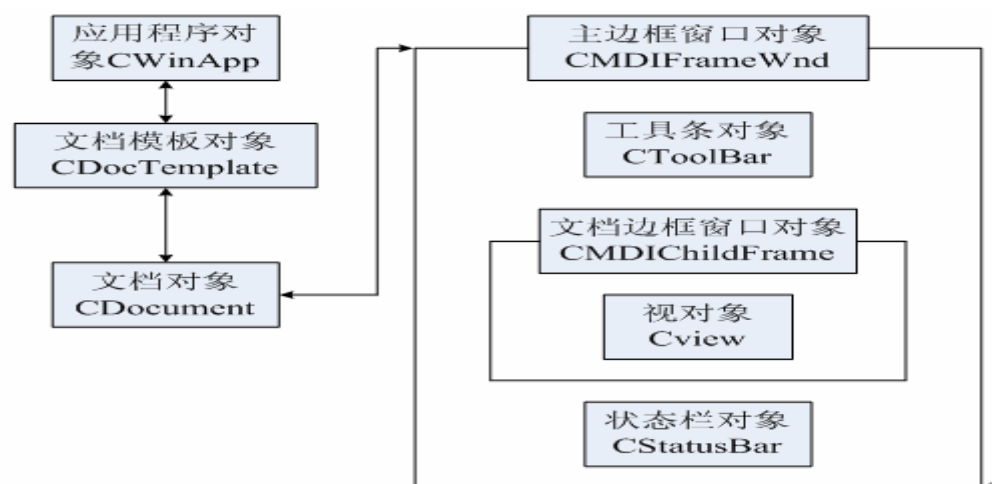


图 2 一个 MDI 应用程序的构成

从 CWinApp、CDocument、CView、CMDIFrameWnd、CMDIChildWnd 类对应地派生出 CMyDemoApp、CMyDemoDoc、CMyDemoView、CMainFrame、CChildFrame 五个类,



这五个类的实例分别是应用程序对象、文档对象、视对象、主框架窗口对象和文档边框窗口对象。主框架窗口包含了视窗口、工具条和状态栏。对这些类或者对象解释如下。

1) 应用程序

应用程序类派生于 CWinApp。基于框架的应用程序必须有且只有一个应用程序对象，它负责应用程序的初始化、运行和结束。

2) 边框窗口

如果是 SDI 应用程序，从 CFrameWnd 类派生边框窗口类，边框窗口的客户子窗口 (MDIClient) 直接包含视窗口；如果是 MDI 应用程序，从 CMDIFrameWnd 类派生边框窗口类，边框窗口的客户子窗口 (MDIClient) 直接包含文档边框窗口。

如果要支持工具条、状态栏，则派生的边框窗口类还要添加 CToolBar 和 CStatusBar 类型的成员变量，以及在一个 OnCreate 消息处理函数中初始化这两个控制窗口。

边框窗口用来管理文档边框窗口、视窗口、工具条、菜单、加速键等，协调半模式状态（如上下文的帮助 (SHIFT+F1 模式) 和打印预览）。

3) 文档边框窗口

文档边框窗口类从 CMDIChildWnd 类派生，MDI 应用程序使用文档边框窗口来包含视窗口。

4) 文档

文档类从 CDocument 类派生，用来管理数据，数据的变化、存取都是通过文档实现的。视窗口通过文档对象来访问和更新数据。

5) 视

视类从 CView 或它的派生类派生。视和文档联系在一起，在文档和用户之间起中介作用，即视在屏幕上显示文档的内容，并把用户输入转换成对文档的操作。

6) 文档模板

文档模板类一般不需要派生。MDI 应用程序使用多文档模板类 CMultiDocTemplate；SDI 应用程序使用单文档模板类 CSingleDocTemplate。

应用程序通过文档模板类对象来管理上述对象（应用程序对象、文档对象、主边框窗口对象、文档边框窗口对象、视对象）的创建。

2、构成应用程序的对象之间的关系



这里，用图的形式可直观地表示所涉及的 MFC 类的继承或者派生关系，如图 3 所示意。

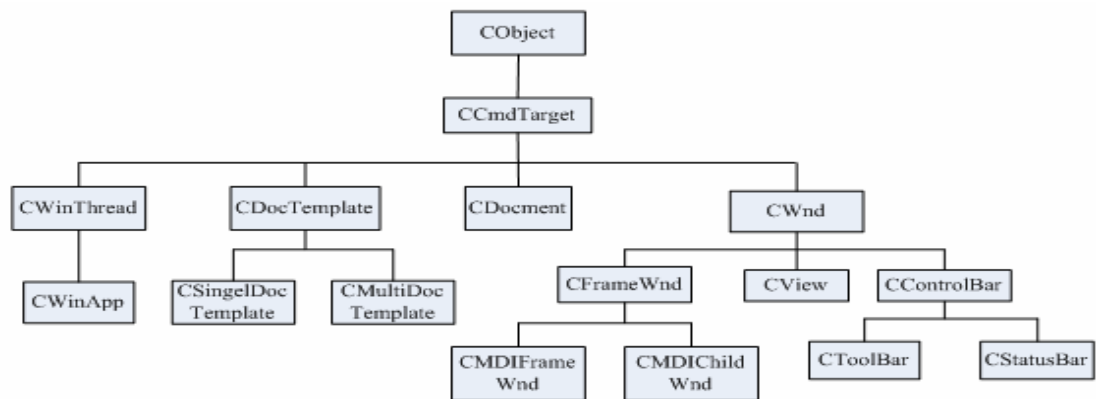


图 3 一些 MFC 类的层次

图 3 所示的类都是从 CObject 类派生出来的；所有处理消息的类都是从 CCmdTarget 类派生的。如果是多文档应用程序，文档模板使用 CMultiDocTemplae，主框架窗口从 CMdiFarmeWnd 派生，它包含工具条、状态栏和文档框架窗口。文档框架窗口从 CMdiChildWnd 派生，文档框架窗口包含视，视从 CView 或其派生类派生。

3、构成应用程序的文件

通过上述分析，可知 AppWizard 产生的 MDI 框架程序的内容，所定义和实现的类。下面，从文件的角度来考察 AppWizard 生成了哪些源码文件，这些文件的作用是什么。下面为 AppWizard 所生成的头文件：

stdafx.h：标准 AFX 头文件

Resource.h：定义了各种资源 ID

MyDemo.h: `#include "resource.h"` 定义了从 CWinApp 派生的应用程序对象 CMyDemoApp

ChildFrm.h：定义了从 CMDIChildWnd 派生的文档框架窗口对象 CMyDemoChildFrame

MainFrm.h：定义了从 CMDIChildWnd 派生的框架窗口对象 CMainFrame

MyDemoDoc.h：定义了从 CDocument 派生的文档对象 CMyDemoDoc

MyDemoView.h：定义了从 CView 派生的视图对象 CMyDemoView

下面为 AppWizard 所生成的实现文件及其对头文件的包含关系。

stdafx.cpp 包含 `#include "stdafx.h"` 用来产生预编译的类型信息。

MyDemo.cpp 包含

```
#include "stdafx.h"
```



```
#include "MyDemo.h"

#include "MainFrm.h"

#include "ChildFrm.h"

#include "MyDemoDoc.h"

#include "MyDemoView.h"
```

定义 CMyDemoApp 的实现，并定义 CMyDemoApp 类型的全局变量 theApp。

ChildFrm.cpp 包含

```
#include "stdafx.h"

#include "MyDemo.h"

#include "ChildFrm.h"
```

实现了类 CChildFrame

MyDemoDoc.cpp 包含

```
#include "stdafx.h"

#include "MyDemo.h"

#include "MyDemoDoc.h"
```

实现了类 CMyDemoDoc

MyDemoView.cpp 包含

```
#include "stdafx.h"

#include "MyDemo.h"

#include "MyDemoDoc.h"

#include "MyDemoView.h"
```

实现了类 CMyDemoView 类

对预编译头文件说明如下：

所谓头文件预编译，就是把一个工程(Project)中使用的一些 MFC 标准头文件(如 Windows.H、Afxwin.H)预先编译，以后该工程编译时，不再编译这部分头文件，仅仅使用预编译的结果。这样可以加快编译速度，节省时间。

预编译头文件通过编译 stdafx.cpp 生成，以工程名命名，由于预编译的头文件的后缀是“pch”，所以编译结果文件是 projectname.pch。

编译器通过一个头文件 stdafx.h 来使用预编译头文件。stdafx.h 这个头文件名



是可以在 project 的编译设置里指定的。编译器认为，所有在指令#include "stdafx.h"前的代码都是预编译的，它跳过#include "stdafx.h"指令，使用projectname.pch 编译这条指令之后的所有代码。

因此，所有的 CPP 实现文件第一条语句都是：#include "stdafx.h"。

另外，每一个实现文件 CPP 都包含了如下语句：

```
#ifdef _DEBUG  
#define new DEBUG_NEW  
#endif
```

这是表示，如果定义过调试_DEBUG，则定义一个新的 DEBUG_NEW，防止重定义。

4.3 如何让机器人动起来



双击桌面上的 MTCheck.exe 图标，这是机器人自检程序，它分为三大块：基本功能，扩展功能。基本功能包括：整机检测、运动控制、图像采集、网络传输、语音处理；扩展功能包括：超声测试、PSD 检测、DI 检测、DO 检测、DA 检测、码盘检测；下面分别介绍他们的功能和操作步骤。

4.3.1 整机检测

请首先将机器人架起，以防发生意外。

功能：

- (1) 检测左电机是否运行正常
- (2) 检测右电机是否运行正常
- (3) 检测超声各通道是否运行正常
- (4) 检测 PSD 个通道是否运行正常

操作步骤：

第一步：基本功能->整机检测->开始，结果如下图 4：



图 4 整机检测界面

可能出现的情况及解决办法:

点击“开始”按钮时出现错误, 请检查“任务管理器”-“进程”内是否有其他 MTR 软件程序在运行, 如果有请将其结束。

第二步: 点击“启动自检”按钮, 将机器人上的外围设备放好如键盘鼠标等, 打开急停开关, 请操作人员和其他观看人员站在离机器人一米外。

此时该软件就会逐项进行检查, 这个过程需要几分钟的时间, 请耐心等待。

在检测过程中, 在界面左侧会看到此时哪个设备正在被检测。

1、检测左电机:

左电机正在检测时, 图标为 ;

左电机检测异常时, 图标为 ;

左电机运行正常时, 图标为 .

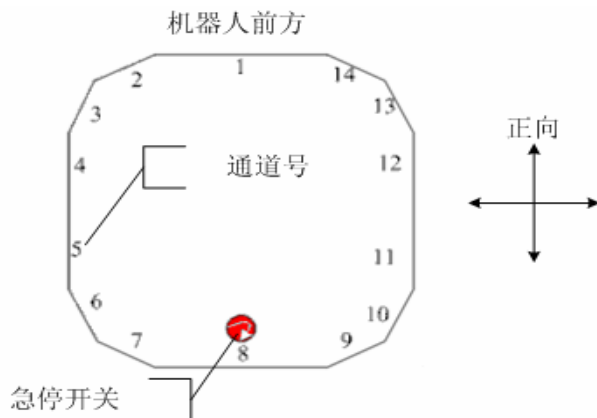
可能发生的异常情况及解决方法:

电机不转或乱转, 请关闭程序后关机检查硬件设备。

2、检测右电机:

与左电机同。

3、检测超声设备:



机器人本体上超声的位置放在 1、3、5、8、11、13 六个通道上，对应的位置请看上图，先将障碍物放在超声 1 通道的前面 1m 内，大约几秒钟后，再将障碍物放在 3 通道上，依次类推。

可能出现的情况及解决方法：

正在检测的超声通道没有声音，请关闭程序后，检查超声板是否上电，超声通道是否插对。

4、PSD 检测

与超声检测方法类似，不同之处在 PSD 没有声音。

全部检测完后，如果图标中有，说明该设备运行异常，程序会继续进行检测，

如果全部运行成功，所有图标为。

第三步：按下急停开关后，退出程序

4.3.2 运动控制

功能：

- 1) 速度闭环+默认 PID 或在线调试 PID 或 LoadPID
- 2) 位置闭环+默认 PID 或在线调试 PID 或 LoadPID
- 3) 速度闭环+协调控制+默认 PID 或在线调试 PID 或 LoadPID
- 4) 位置闭环+协调控制+默认 PID 或在线调试 PID 或 LoadPID
- 5) 保存 PID 参数
- 6) 查看上次保存的 PID 参数值
- 7) 得到码盘信息
- 8) 画出左右轮速度曲线，左右轮位移曲线，左右轮加速度曲线

操作步骤：

第一步：基本功能->运动控制->启动，界面如图 5 所示

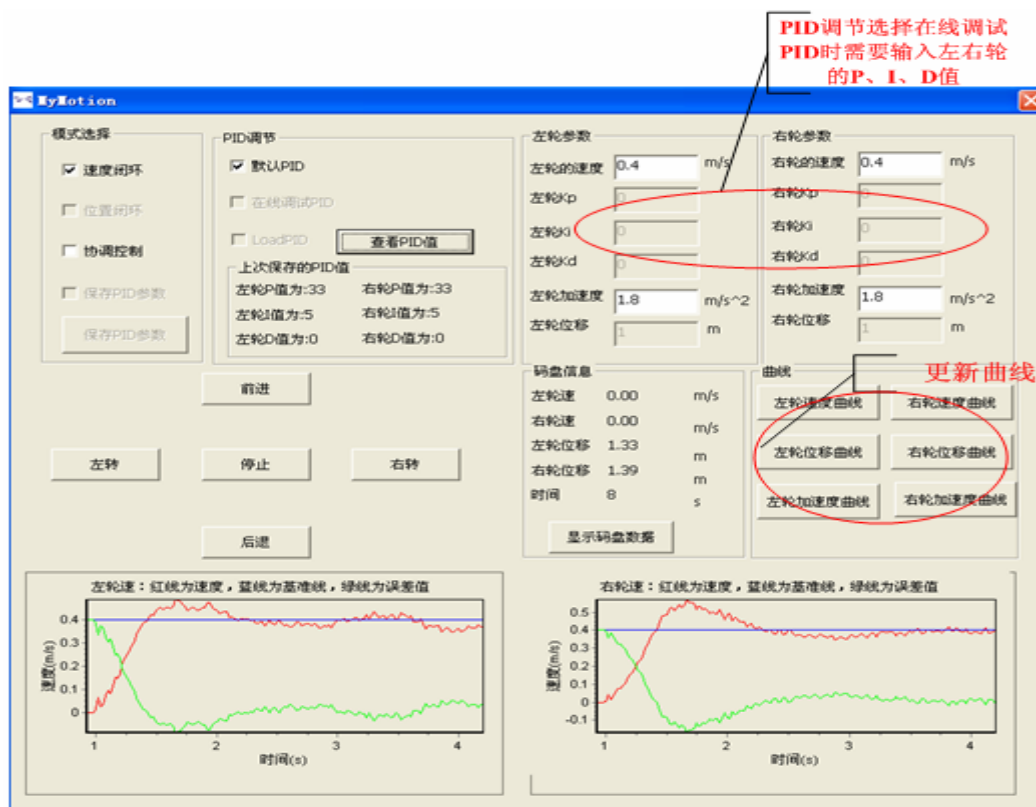


图 5 基本运动界面

模式选择:

1) 选择速度闭环+默认 PID

输入左轮的速度和右轮的速度（范围：-2.5m/s-2.5m/s），输入左右轮加速度（0-2.5m/s²）

单击“显示码盘数据”按钮，“前进”按钮，打开急停开关，想进行其它操作前先按下急停开关，单击“停止”按钮，然后单击“左轮速度曲线”，或“右轮速度曲线”，或“左轮位移曲线”，或“右轮位移曲线”，或“左轮加速度曲线”，或“右轮加速度曲线”，在界面下方会看到相关实际曲线、基准曲线、误差曲线。

2) 选择速度闭环+在线调试 PID

输入“速度”，“加速度”，“Kp”，“Ki”，“Kd”（默认值为左右轮 Kp: 33, Ki: 5, Kd: 0）后，单击“前进”或“后退”或“左转”或“右转”按钮，测试得到一组较好的 PID 参数时，可把该参数保存下来，选择“保存 PID 参数”，点击“保存 PID 参数”按钮。

3) 选择速度闭环+LoadPID

先点击查看 PID 值看该值是否合理，然后点击“前进”等按钮。

4) 选择速度闭环+协调控制+默认 PID 或在线调试 PID 或 LoadPID

操作步骤分别与选择速度闭环相同。

5) 选择位置闭环+默认 PID

输入左轮的速度和右轮的速度（范围：-2.5m/s-2.5m/s），输入左右轮加速度（0-2.5m/s²），位移，点击“前进”等按钮。

6) 选择位置闭环+ 在线调试 PID

输入“速度”，“加速度”，“Kp”，“Ki”，“Kd”后，点击“前进”等按钮，当觉得此时 PID 参数很好时，可把该参数保存下来，选择“保存 PID 参数”，点击



“保存 PID 参数”按钮。

7) 选择位置闭环+LoadPID

先点击查看 PID 值，然后点击“前进”等按钮。

8) 选择位置闭环+协调控制+默认 PID 或在线调试 PID 或 LoadPID

操作步骤分别与选择位置闭环相同。

第二步：按下急停开关，关闭程序

4.3.3 图像采集

功能：

1) 实时采集（160*120，320*240，640*480 大小 10，20，30 帧率的）彩色图像

2) 单帧显示

操作步骤：

第一步：基本功能->图像采集->开始采集

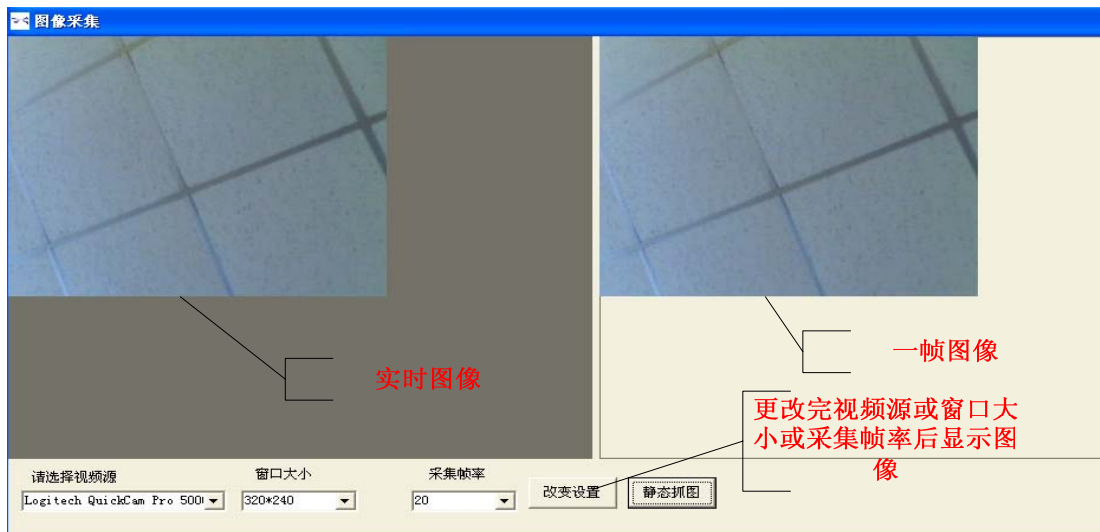


图 6 图像采集界面

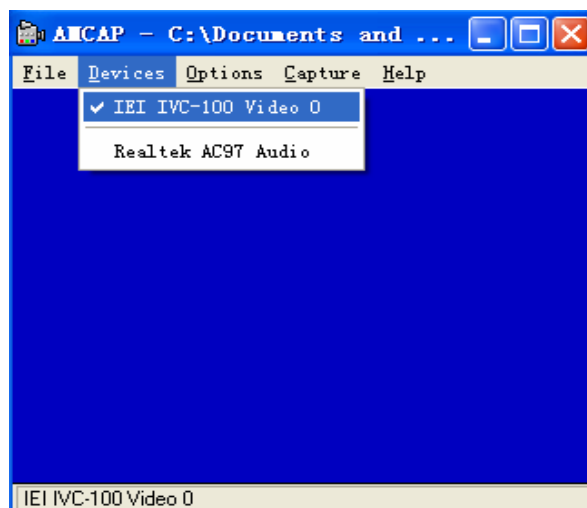
可能出现的情况及解决问题：

1) 程序出现异常时，可能摄像头的驱动掉了，重新安装驱动即可。

2) 图像为纯蓝或纯黑时，需要更改图像采集制式，更改方法如下：



打开桌面上的  图标，选择图像为纯蓝的摄像头设备，如：



如果摄像头制式为 PAL，则选择菜单栏内 Options->Video Capture Filter...->影像解码器->S 视频标准(->PAL_B。如果摄像头制式为 NTSC。

当摄像头接在图像采集卡 S 端子上时，选择菜单栏内 Options->Video Crossbar...->Input->Video SVideo In，而接在复合端子上时选择 Video Composite In。

第二步：实时显示

选择视频源，窗口大小，采集帧率后单击“改变设置”按钮，想换其他视频源时只需选择需要显示的视频源，单击“改变设置”按钮。

第三步：单帧显示

点击“静态抓图”按钮。

第四步：关闭程序

4.3.4 网络传输

功能：

- 1) 网络图像传输
- 2) 网络运动控制

操作步骤：

第一步：基本功能->网络传输->启动服务器端

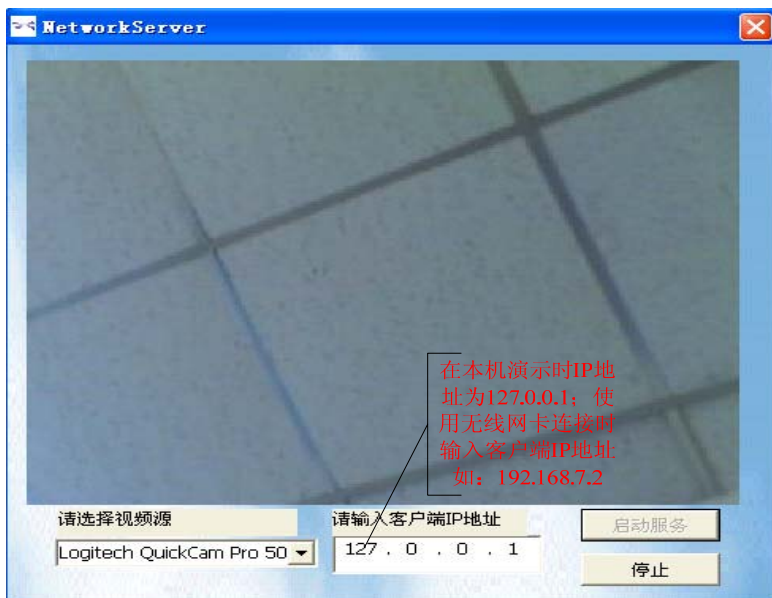


图 7 网络传输服务器端界面

第二步：选择视频源，输入客户端 IP 地址，启动服务

第三步：基本功能->网络控制->启动客户端

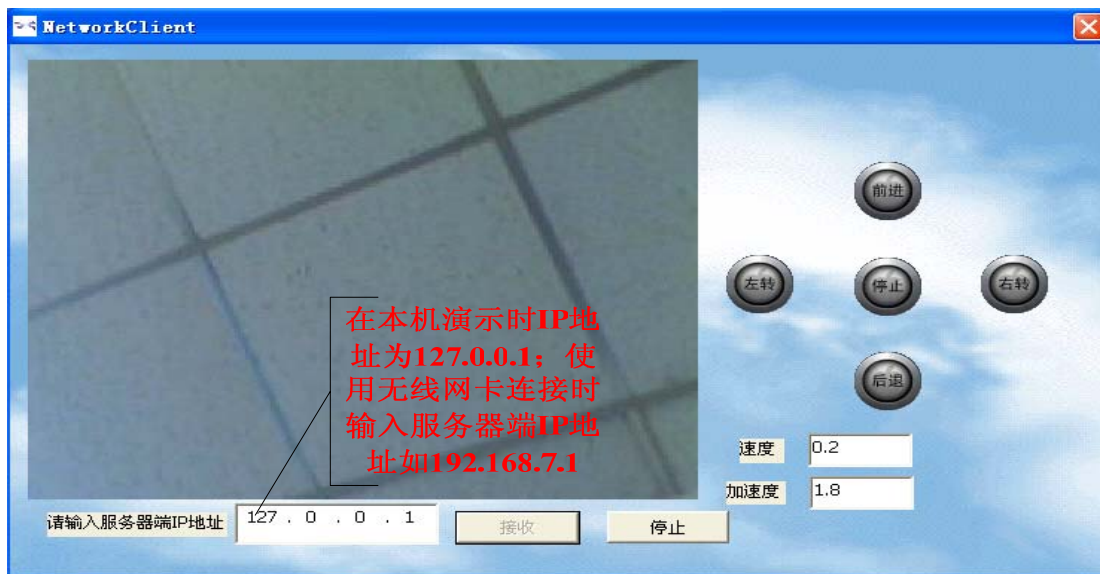


图 8 网络传输客户端界面

第四步：启动服务

连接成功后，通过右边的按钮可控制机器人运动。

第五步：退出程序

先点击客户端“停止服务”，关闭服务器程序；再点击服务器端“停止服务”，关闭程序。

可能出现的情况及解决办法：

1) 启动服务失败

请检查图像设备或看是否有其他 MTR 软件进程在运行。

2) 网络连接失败

有两种情况：一、两个程序都在机器人上运行时，IP 地址都为 127.0.0.1。



二、当服务器端程序在机器人上运行而客户端程序在 PC 机或笔记本上运行时，首先请参阅 3.10 节，看完后按上述步骤运行。

4.3.5 语音处理

功能：

- 1) 测试麦克
- 2) 测试音箱

操作步骤：

第一步：基本功能->语音处理->启动语音处理程序

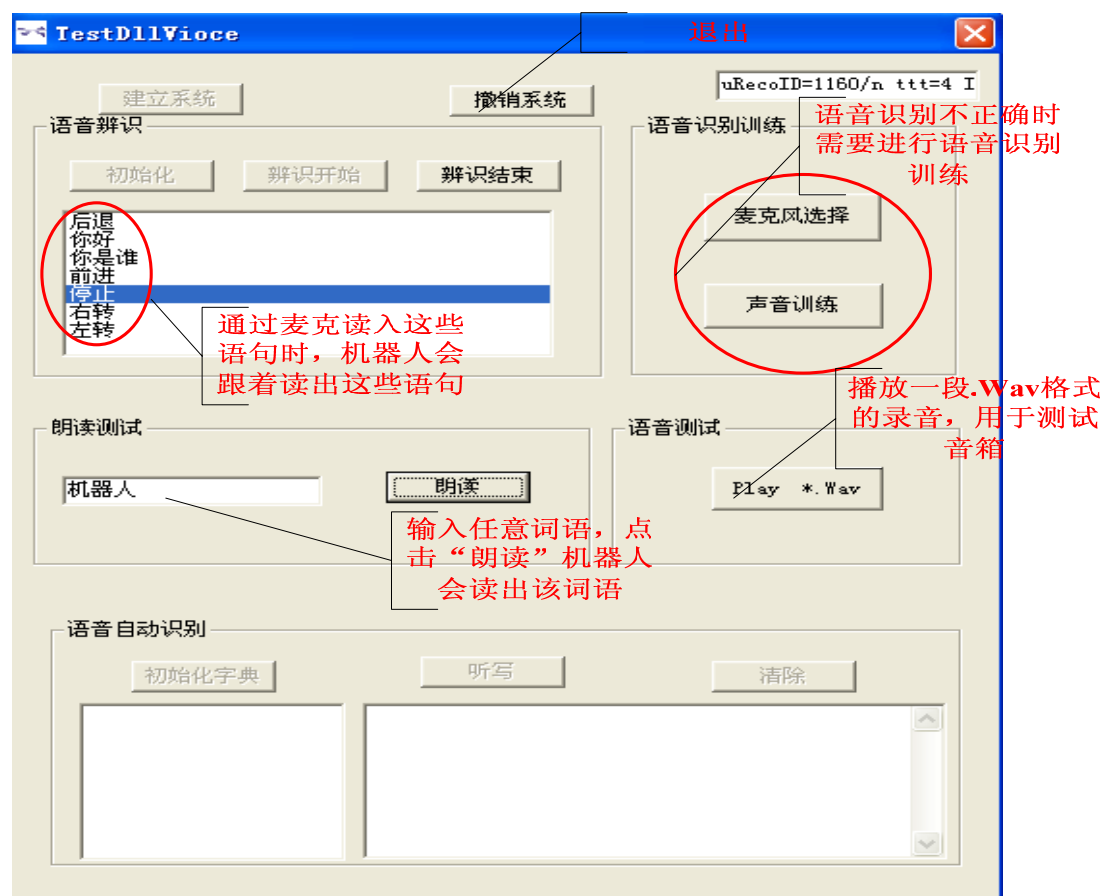


图 9 语音处理界面

第二步：建立系统->初始化

- 1) 语音识别训练，选择麦克风后进入声音训练。
- 2) 在朗读测试内输入汉字，点击朗读。
- 3) 语音测试，打开 .wav 格式的文件，在 C:\MTSoft\Program\Demo\TestDllVoice\Debug 内有例子。
- 4) 点击辨识开始，朗读里面的文字，如果你读“你是谁”，机器人也会跟你说“你是谁”，成功。

可能出现的情况及解决办法：

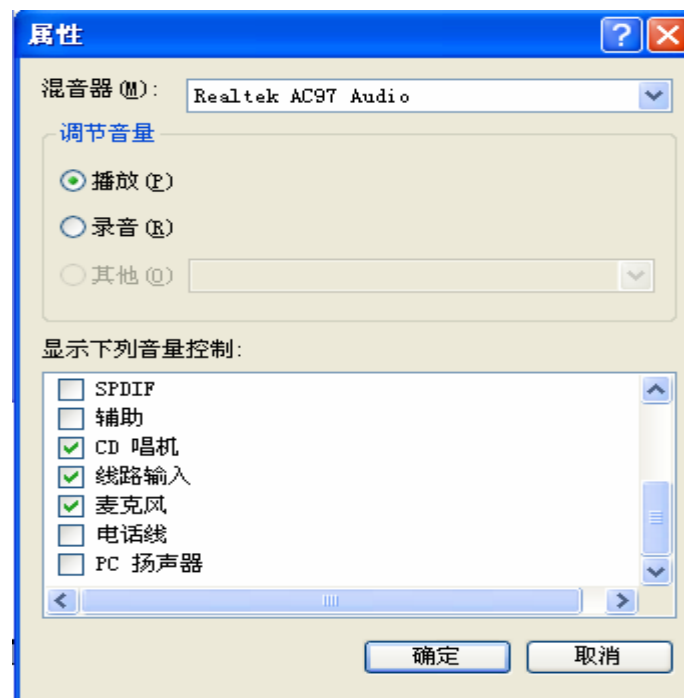
- 1) 机器人没有发出声音，检查音频设备
- 2) 机器人不能进行语音识别



双击桌面右下角声音小喇叭



选择菜单栏内选项->属性，将其中的麦克风选项钩上，如下图



重新打开程序进行测试。

4.3.6 超声测试

功能：

测量超声到障碍物之间的距离。

运行结果，如下图 10：

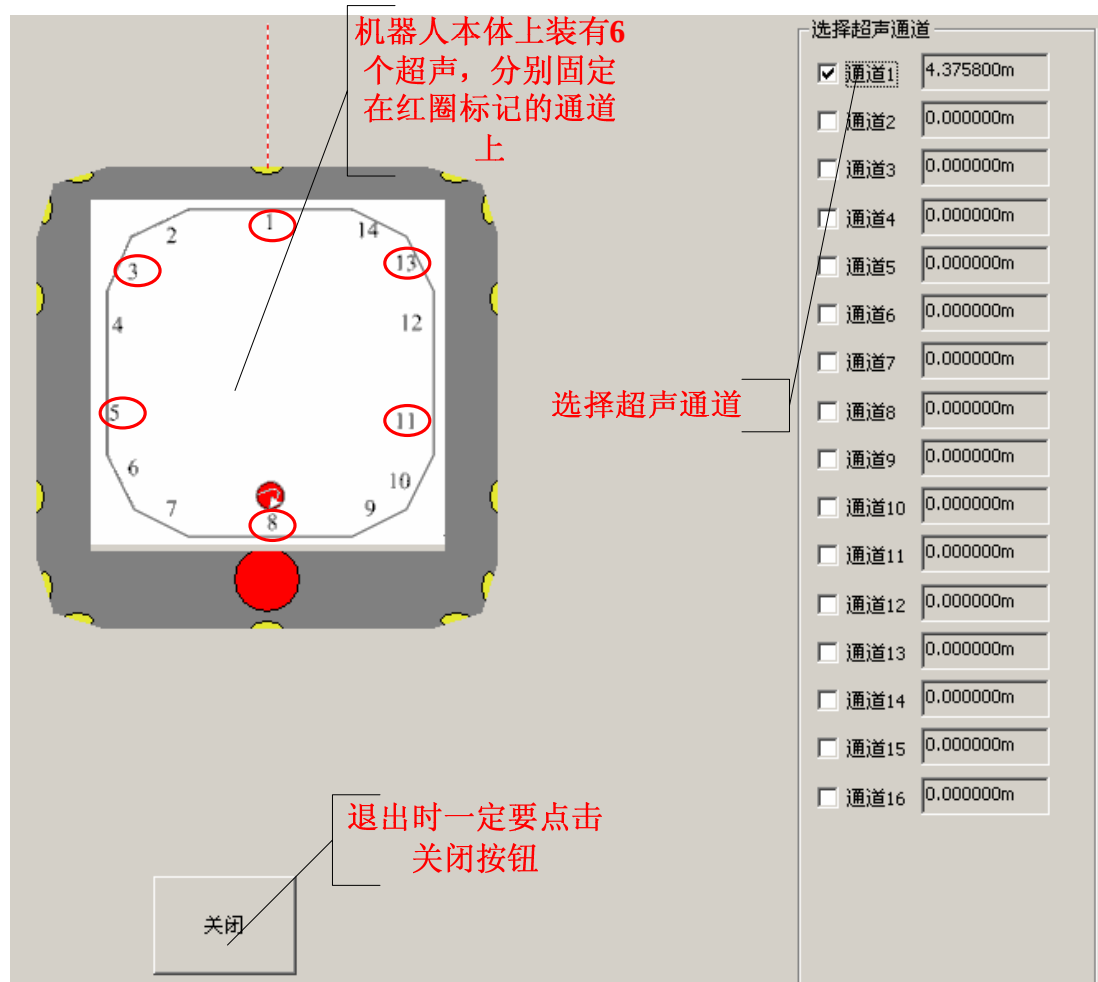


图 10 超声采集界面

操作步骤:

第一步: 扩展功能->超声测试->启动

第二步: 选择超声通道

参照上图机器人模型打开对应位置的超声, 如果超声运行正常时会发出响声, 并在对应的位置发出一条射线, 射线随着到障碍物的距离变化而变化。

注意: 不要同时选择所有通道。

第三步: 关闭程序

点击“关闭”按钮。

可能出现的情况及解决方法:

1、启动不起来: 检查串口是否被占用, 打开桌面上的串口调试助手, 串口选择 4, 波特率选择 19200, 打开串口, 如果能够不能打开, 检查是否有 MTR 程序在运行或重起电脑。

4.3.7 PSD 检测

功能: 测量 PSD 传感器到障碍物之间的距离

运行结果如图 11:



The image shows a software window titled "PSD检测" (PSD Detection). Inside, there is a section titled "选择PSD通道" (Select PSD Channel). It contains two columns of controls. The left column has checkboxes for PSD1 through PSD8, with PSD1 checked. Each checkbox is followed by a text box showing a distance value: PSD1 is 15.1cm, and PSD2 through PSD8 are 0.0cm. The right column has checkboxes for PSD9 through PSD16, all of which are unchecked, followed by text boxes showing 0.0cm. At the bottom of the window is a "关闭" (Close) button.

图 11 PSD 采集界面

操作步骤:

第一步: 扩展功能->PSD 检测->启动

第二步: 选择 PSD 通道

打开 PSD 通道, 如上图, 在右侧列表框控件内可以看到 PSD 测得的距离值。

第三步: 关闭程序

点击“关闭”按钮。

4.3.8 DI 检测

若机器人上没有 DI 输入设备, 不需要检测该项, DI 口输入高电平则对应通道被选择时通道值为 1, 勾掉选择时为 0。

操作步骤:

第一步: 扩展功能->DI 检测->启动

第二步: 选择通道

如下图, 假设通道 1 上装有按钮传感器, 选择通道 1, 当按钮按下和弹起时, 观察右面 Edit 控件内值的变化。



DI检测

DI口输入高电平则对应通道被选择时通道值为1，勾掉选择时为0

通道号	通道值	通道号	通道值
☒ DI 1	1	☐ DI 9	0
☐ DI 2	0	☐ DI 10	0
☐ DI 3	0	☐ DI 11	0
☐ DI 4	0	☐ DI 12	0
☐ DI 5	0	☐ DI 13	0
☐ DI 6	0	☐ DI 14	0
☐ DI 7	0	☐ DI 15	0
☐ DI 8	0	☐ DI 16	0

停止

图 12 DI 检测界面

第三步：退出程序
点击“停止”按钮

4.3.9 DO 检测

若机器人上没有 DO 输出设备，不需要检测该项，选择通道时 DO 口输出 0.5V 左右，勾掉选择时对应通道输出 5V。

操作步骤：

第一步：扩展功能->DO 检测->启动

第二步：选择通道

如下图，假设通道 2 上装有踢球装置，先给踢球装置上电，一定不要空踢，放好球后，选择通道 2，踢完后，勾掉选择。

DO检测

选择通道时DO口输出0.5V左右，勾掉选择时对应通道输出5V

DO通道

☐ DO 1	☐ DO 9
☒ DO 2	☐ DO 10
☐ DO 3	☐ DO 11
☐ DO 4	☐ DO 12
☐ DO 5	☐ DO 13
☐ DO 6	☐ DO 14
☐ DO 7	☐ DO 15
☐ DO 8	☐ DO 16

关闭

图 13 DO 检测界面

第三步：退出程序



点击“关闭”按钮。

4.3.10 DA 检测

若机器人上没有 DA 设备，不需要检测该项，0~255 对应 0~5V 电压
操作步骤：

第一步：扩展功能->DA 检测->启动

第二步：选择通道

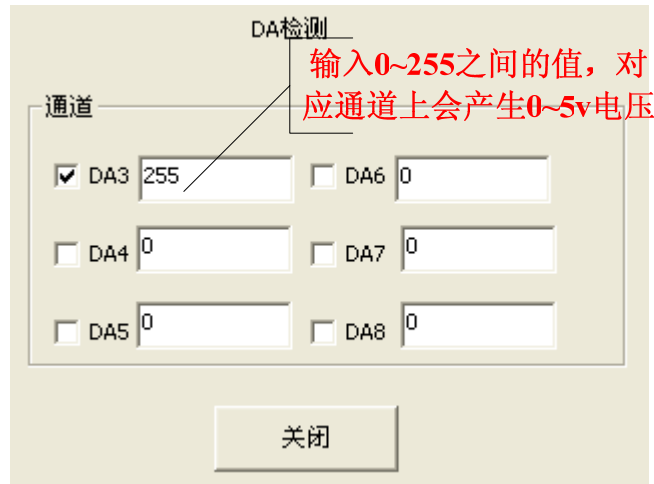


图 14 DA 检测界面

第三步：关闭程序

4.3.11 客户端软件安装说明

客户端必备软件包括：

名称	版本
1、Microsoft Visual Studio .NET 2003	7.0
2、Microsoft DirectX	9.0 及以上
3、Microsoft SpeechSDK	5.11 及以上
4、Microsoft SpeechSDK Langpack（将该文件夹内容拷贝到 3 的安装目录下即可）	5.11 及以上
5、Morgan M-JPEG codec	V3 及以上
6、Microsoft Server 2003 Platform SDK	R2 及以上
7、Microsoft .Net Framework	1.1 及以上
8、Microsoft .Net Framework Chinese(Simplified) Language Pack	1.1 及以上
9、无线网卡驱动	
10、MTSoft 安装包	1.1 版本

若只在客户端演示只需安装 3、4、5、7、8、9、10 即可。



部分安装程序说明：

一、无线网卡配置

以 TL-WN321G 客户端应用程序为例，注意两个无线网卡型号要一致：

1) 配制文件

双击屏幕右下角的图标，会弹出对话框如图 15 所示



图 15 初始界面

点击图 15 中“新增”按钮，如图 16 所示



图 16 新增配置文件

按图 16 中配置，“确定”后如图 17



图 17 客户端应用程序

选中新增的文件，点击“激活”按钮，可以看到新增的文件前面出现，如图 18 所示



图 18 激活界面

2) 连接无线网络



点击“无线网络”，如图 19



图 19 无线网络连接

选择 TP-LINK，点击“连接”按钮，如果机器人上的无线网卡连接并已经启用，那么点击连接后，就会在网络名称前出现，如图 20



图 20 连通状态

这表明两个无线网卡之间已经连通了，但并不表明客户端软件与服务器(机器人)之间能够通讯了，还要设置客户端的 IP 地址，一般机器人在出厂前已经对机器人的 IP 地址配置好了，都是 192.168.7.1，那么就需要对客户机(笔记本或者台式机)设置 IP，使它的 IP 地址与机器人的 IP 地址在同一个域中，一般设置为 192.186.7.2，如图 21 中

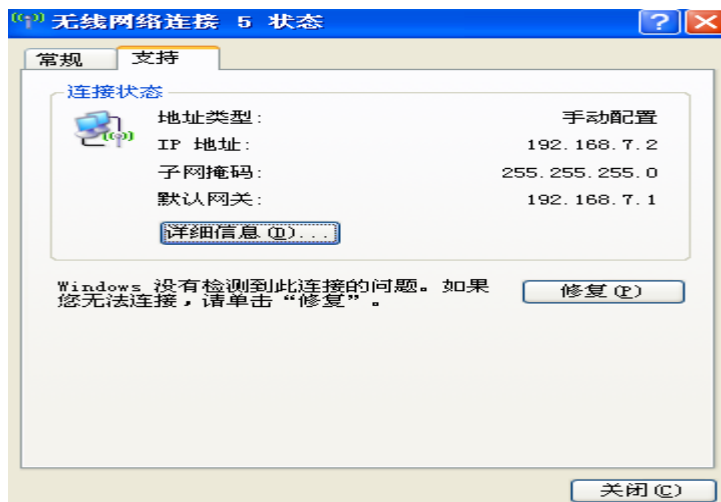


图 21 IP 地址

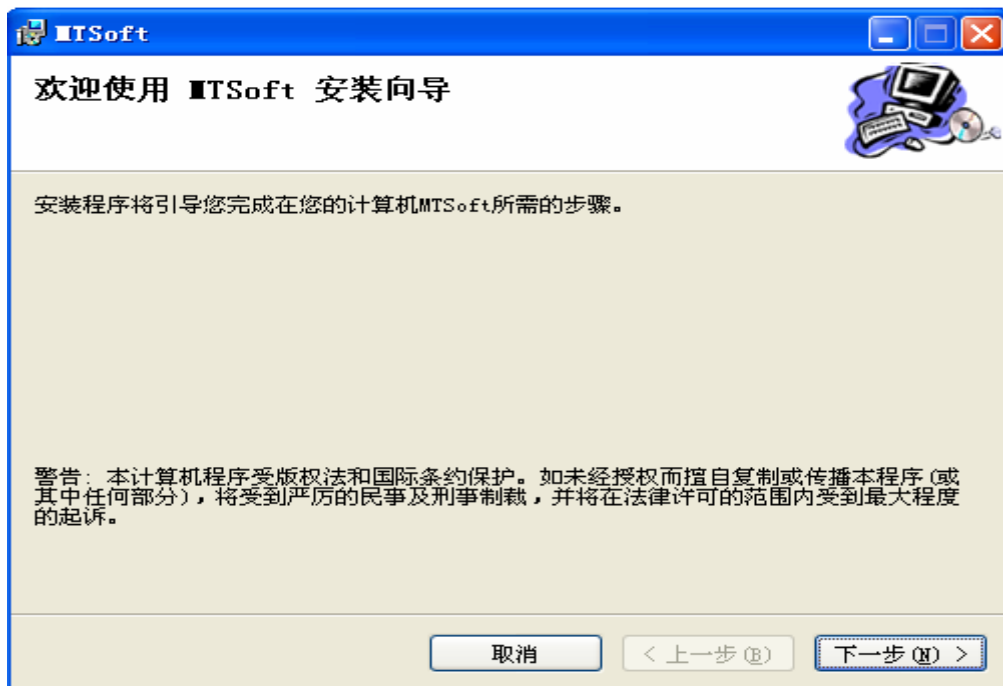
这时你可以在客户机上 ping 机器人服务器，看是否两台机器间是连通的，你也可以通过上位机软件进行测试，这是在你的客户机驱动程序都已经装好的情况下才可以进行测试。

二、mogan 设置

双击电脑右下角图标 打开 mjpeg 主界面，有个 enable codec 复选框，将它打钩，并将 Forced quality 选项钩上，然后点击 about 按钮，弹出一个关于对话框，在该对话框里有个 register 按钮，点击一下，输入注册名和序列号就可以了。

三、MTSoft 安装

点击光盘内 MTSoft 安装包。
按下列步骤进行

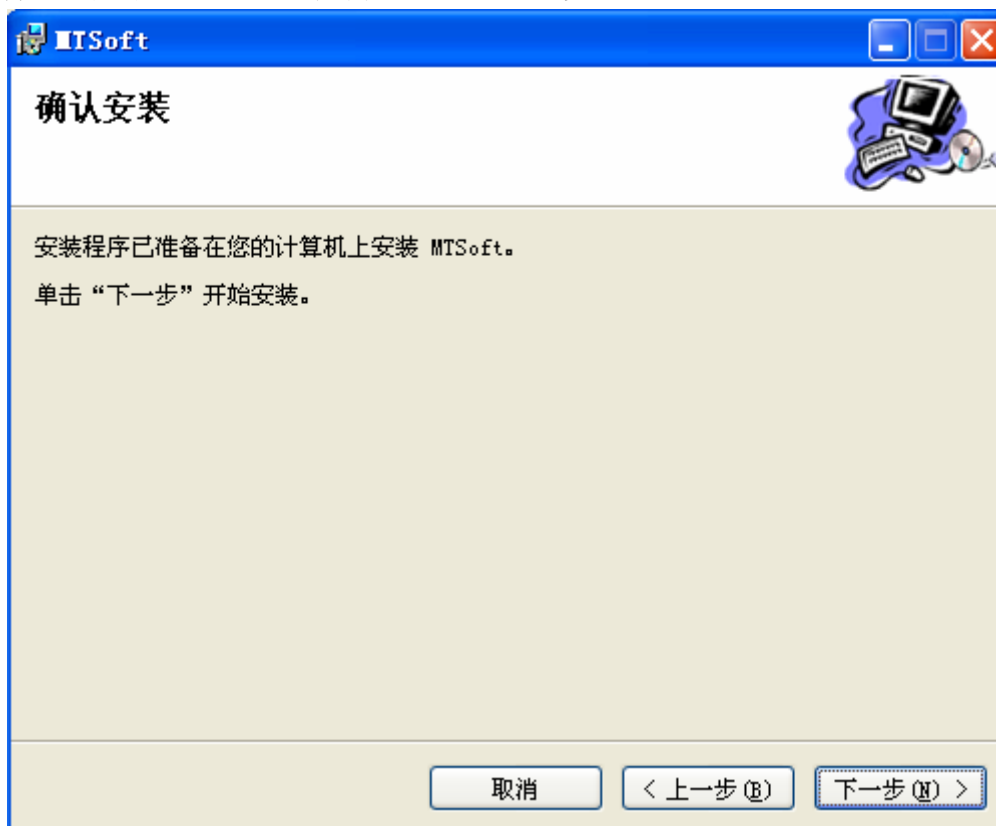


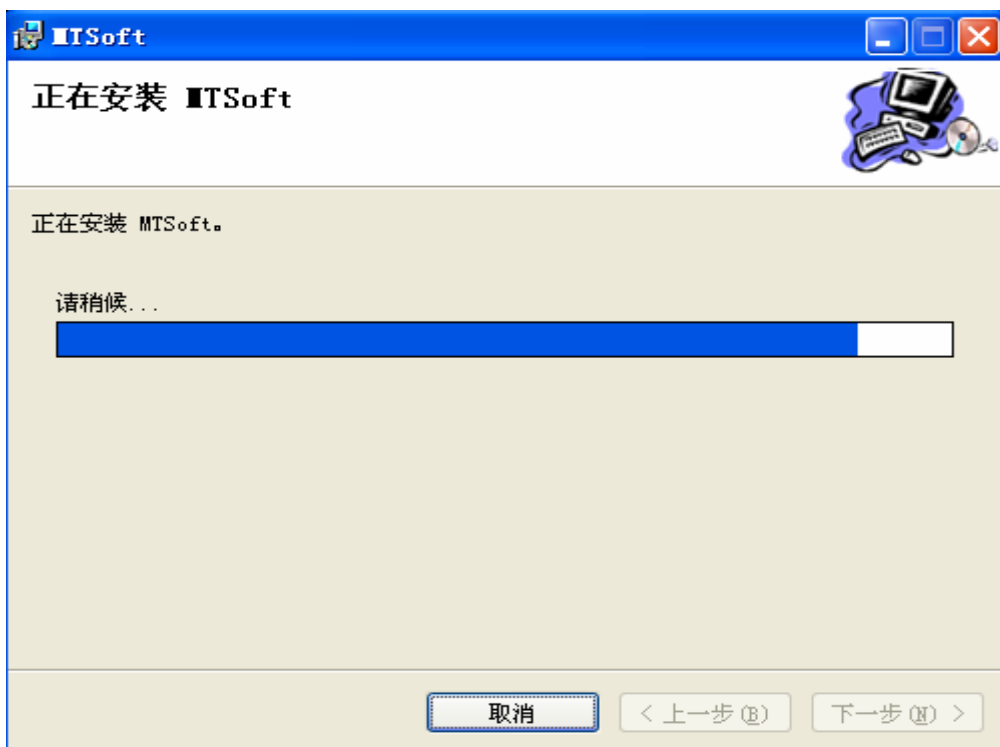


“下一步”



文件夹必须为“C:\” -> “任何人” -> “下一步”





“关闭”后，找到双击 C:\MTSoft\Tools 文件夹，双击“regist.exe”可执行文件进行系统注册。

以网络控制为例：

操作步骤：

第一步：基本功能->网络传输->启动服务器端

第二步：选择视频源，输入客户端 IP 地址，启动服务



第三步：基本功能->网络控制->启动客户端



第四步：输入服务器端 IP 地址，启动服务
连接成功后，通过右边的按钮可控制机器人运动。

第五步：退出程序

先点击客户端“停止服务”，关闭服务器程序；再点击服务器端“停止服务”，关闭程序。

4.4 初级编程

以基本运动为例，实现机器人前进、后退、左转、右转、停止功能。

4.4.1 建立一个基于对话框的 MFC 应用程序

打开桌面上的 Microsoft Visual Studio .NET 2003，新建->项目弹出如图 22：

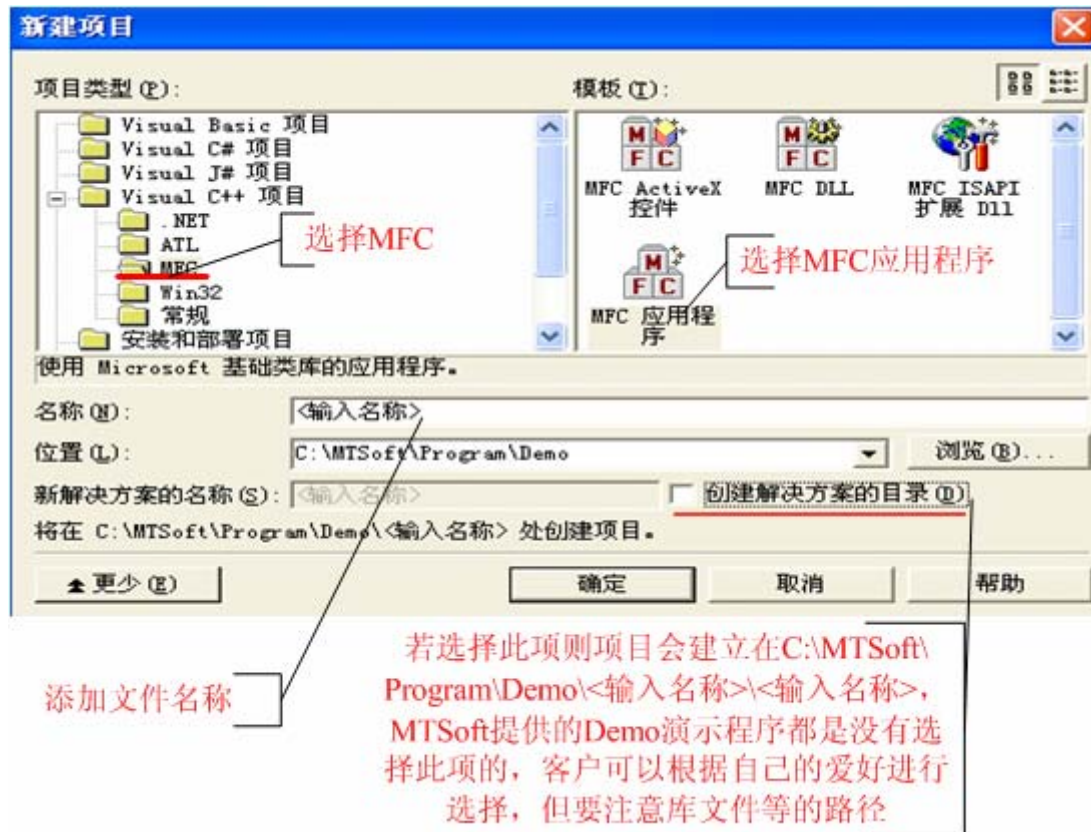
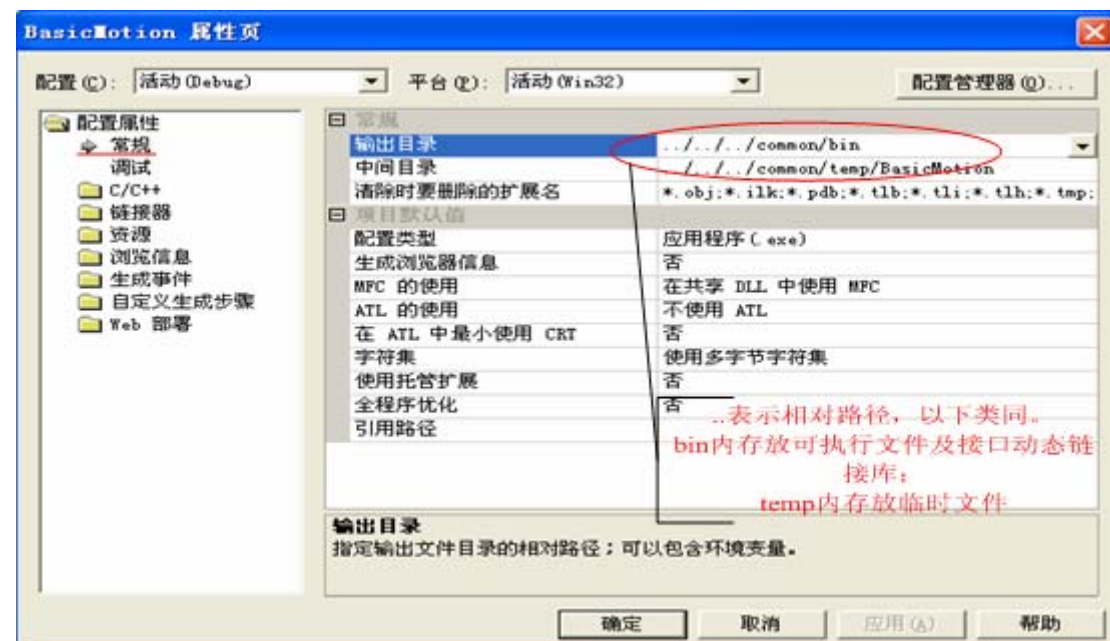


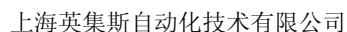
图 22 新建项目

完成这步后单击确定进入 MFC 应用程序向导-BasicMotion, 应用程序类型选择基于对话框, 单击完成, 此时就建立了一个基于对话框的 MFC 应用程序, 文件名为 BasicMotion。

4.4.2 配置属性页

第一步:





BasicMotion 属性页

配置 (C): 活动 (Debug) 平台 (P): 活动 (Win32) 配置管理器 (O)...

配置属性

- 常规
- 调试
- C/C++
 - 常规**
 - 优化
 - 预处理器
 - 代码生成
 - 语言
 - 预编译头
 - 输出文件
 - 浏览信息
 - 高级
 - 命令行
- 链接器
- 资源
- 浏览信息
- 生成事件
- 自定义生成步骤
- Web 部署

附加包含目录

解析 #using 引用	用于“编辑并继续”的程序数据库 (/ZI)
调试信息格式	是 (/nologo)
取消显示启动版权标志	3 级 (/V3)
警告等级	是 (/Wp64)
检测 64 位可移植性问题	否
将警告视为错误	

附加包含目录

指定一个或多个要添加到包含路径中的目录; 当目录不止一个时, 请用分号分隔。 (/[path])

确定 取消 应用 (A) 帮助

BasicMotion 属性页

配置 (C): 活动 (Debug) 平台 (P): 活动 (Win32) 配置管理器 (O)...

配置属性

- 常规
- 调试
- C/C++
- 链接器
 - 常规
 - 输入
 - 调试
 - System
 - 优化
 - 嵌入的 IDL
 - 高级
 - 命令行
- 资源
- 浏览信息
- 生成事件
- 自定义生成步骤
- Web 部署

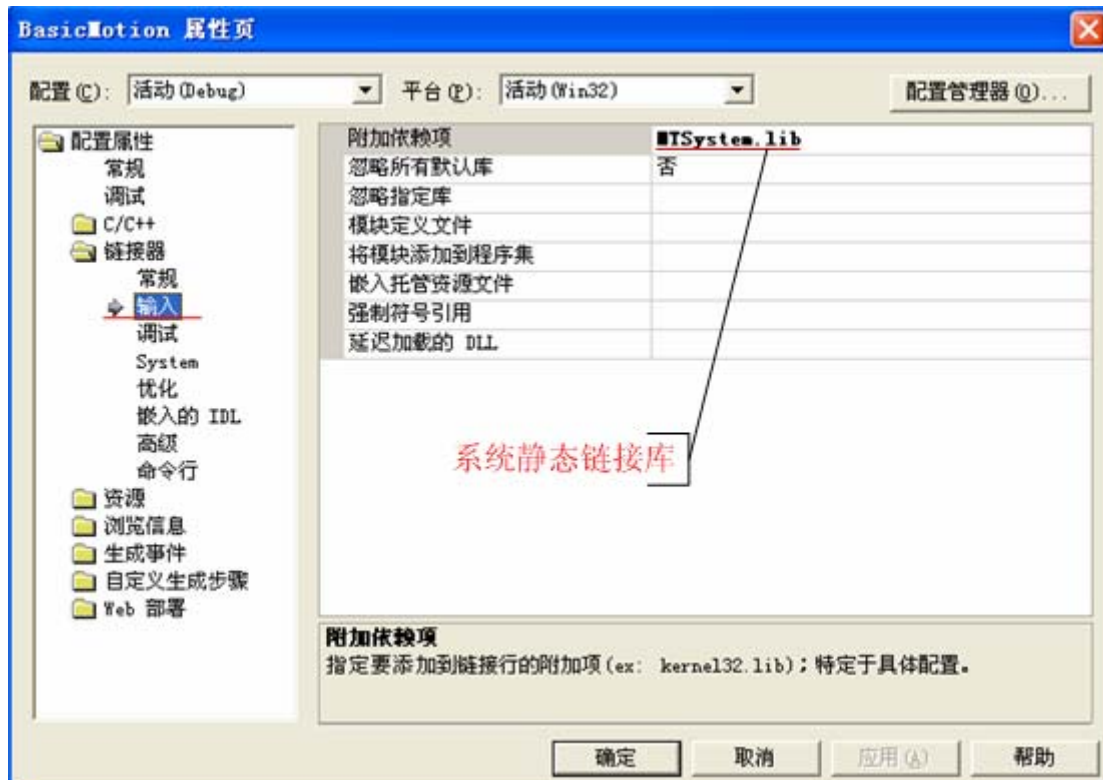
输出文件	\$(OutDir)/\$(ProjectName).exe
显示进度	未设置
版本	
启用增量链接	否 (/INCREMENTAL:NO)
取消显示启动版权标志	否
忽略导入库	否
注册输出	否
附加库目录	../../../../common/lib

存放静态链接库

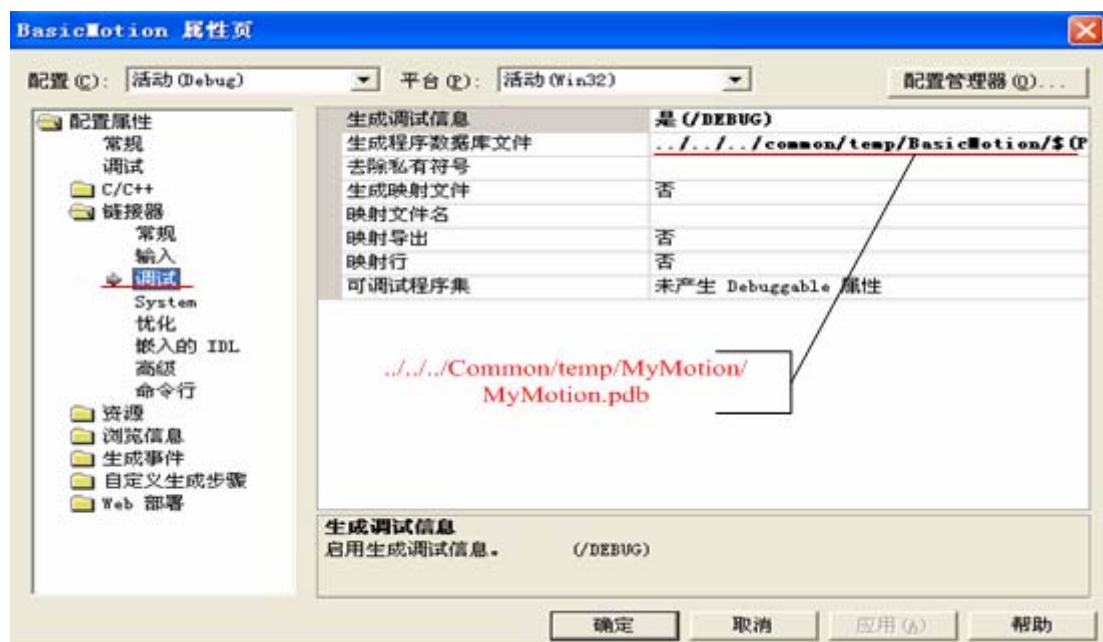
输出文件
重写默认的输出文件名。 (/OUT:[file])

确定 取消 应用 (A) 帮助

<http://www.ingenious.cn>



第五步:



单击确定退出属性页。

4.4.3 添加接口头文件

打开解决方案资源管理器->头文件->stdafx.h, 将下列头文件加入:

```
//MTSOFT Include
#include "globalDef.h" //全局定义头文件
#include "MTSystem.h" //系统头文件
```



```
#include "MTDevice.h" //设备头文件
#include "MTConfig.h" //系统配置头文件
#include "IMTMotion.h" //运动接口头文件
#include "Ingenious.h" //参数类型定义
#include "IngeniousException.h" //异常处理
```

```
class CMTSystem; //系统类
interface IMTMotion; //运动接口
```

4.4.4 添加成员变量、成员函数或消息处理函数

1、选择类视图，点击鼠标右键->添加->添加变量，如定义运动接口设备，如图 23

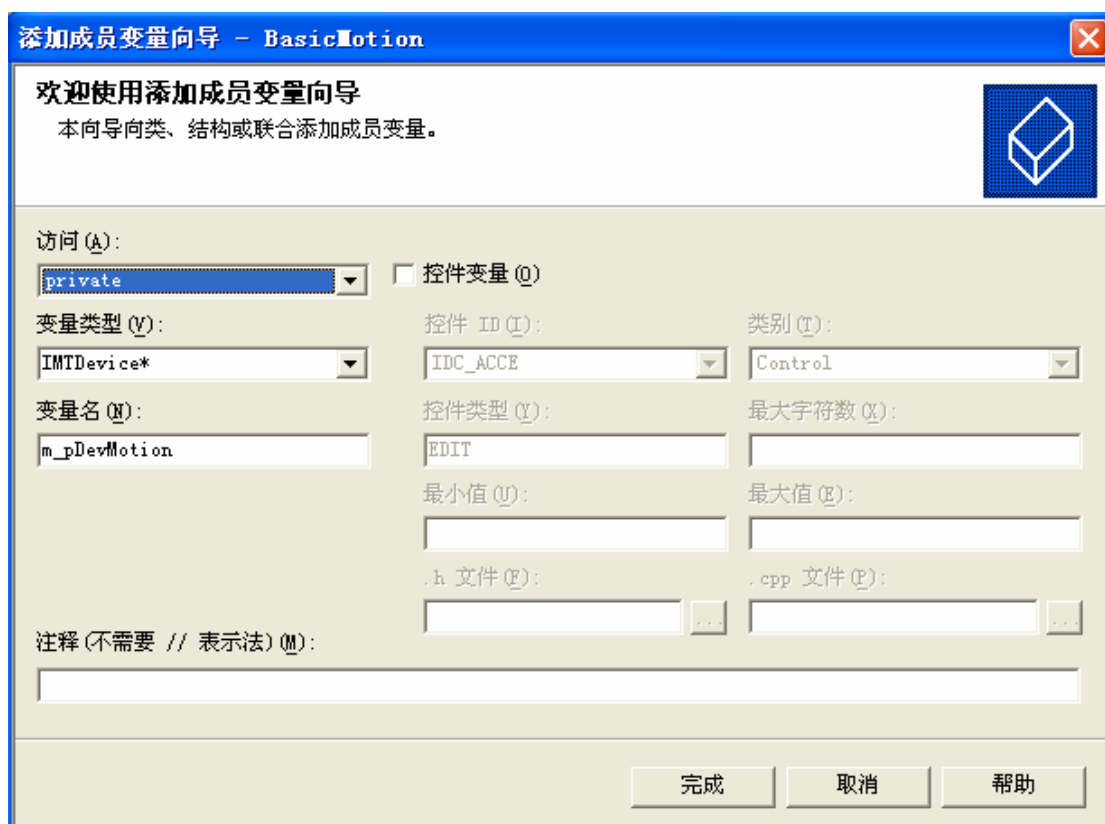


图 23 添加成员变量向导

系统会自动在 `BasicMotionDlg.h` 内生成

private:

```
IMTDevice *m_pDevMotion;
```

在源文件 `BasicMotionDlg.cpp` 自动增加

```
CBasicMotionDlg::CBasicMotionDlg(CWnd* pParent /*=NULL*/)
: CDialog(CBasicMotionDlg::IDD, pParent)
, m_pDevMotion(NULL)
{
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}
```



其他类似。

2、添加成员函数

添加创建系统函数 BuildSystem(), 打开类视图, 鼠标右键点击 CBasicMotionDlg, 添加->成员函数, 如图 24

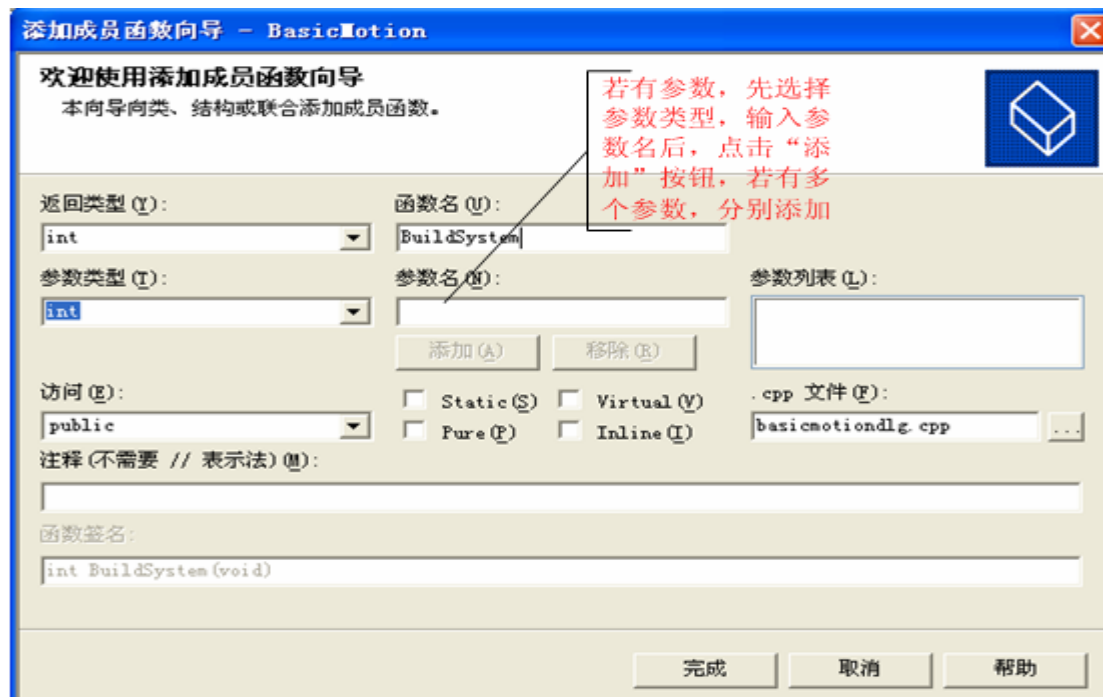


图 24 添加成员函数向导

其他类似。

2、添加消息处理函数

以添加 OnDestroy 函数为例, 类视图->选择 CBasicMotionDlg->属性->消息



->WM_DESTROY-><添加>OnDestroy。



4.4.5 设计界面



下表为按钮控件、单选按钮控件及编辑框控件的属性值、ID 值和类别

类别	Caption	ID 号
ButtonControl	前进	IDC_FRONT
ButtonControl	后退	IDC_BACK
ButtonControl	左转	IDC_LEFT
ButtonControl	右转	IDC_RIGHT
ButtonControl	停止	IDC_STOP
EditControl		IDC_SPEED
EditControl		IDC_ACCE

下面添加成员变量，如为度添加成员变量，点击鼠标右键，如图 25

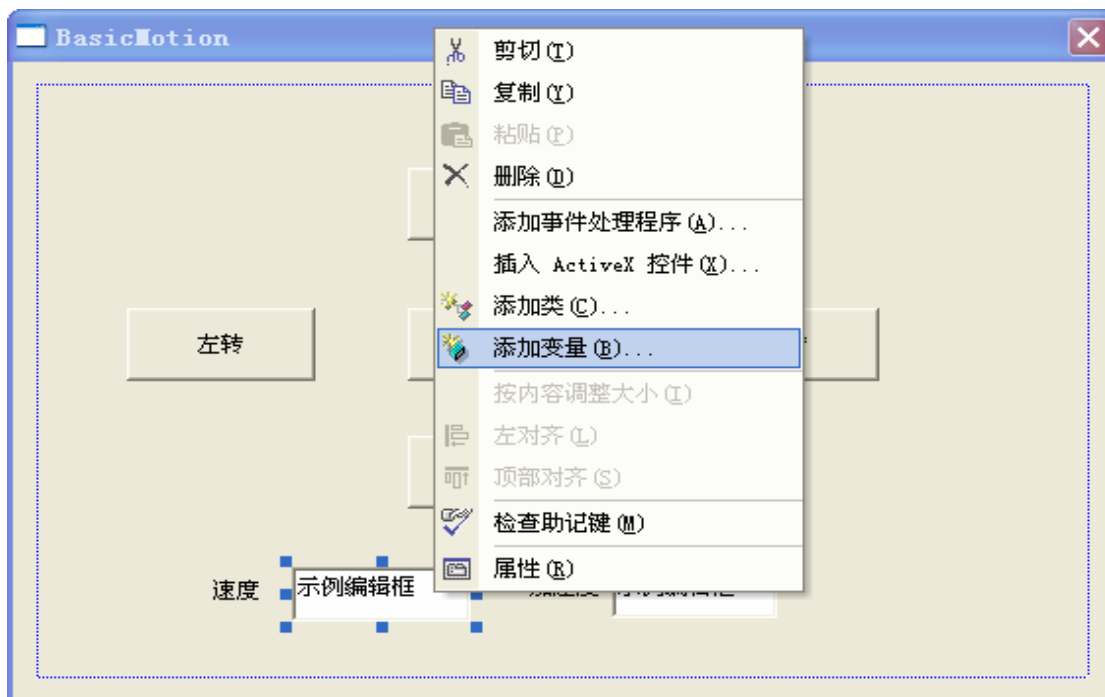
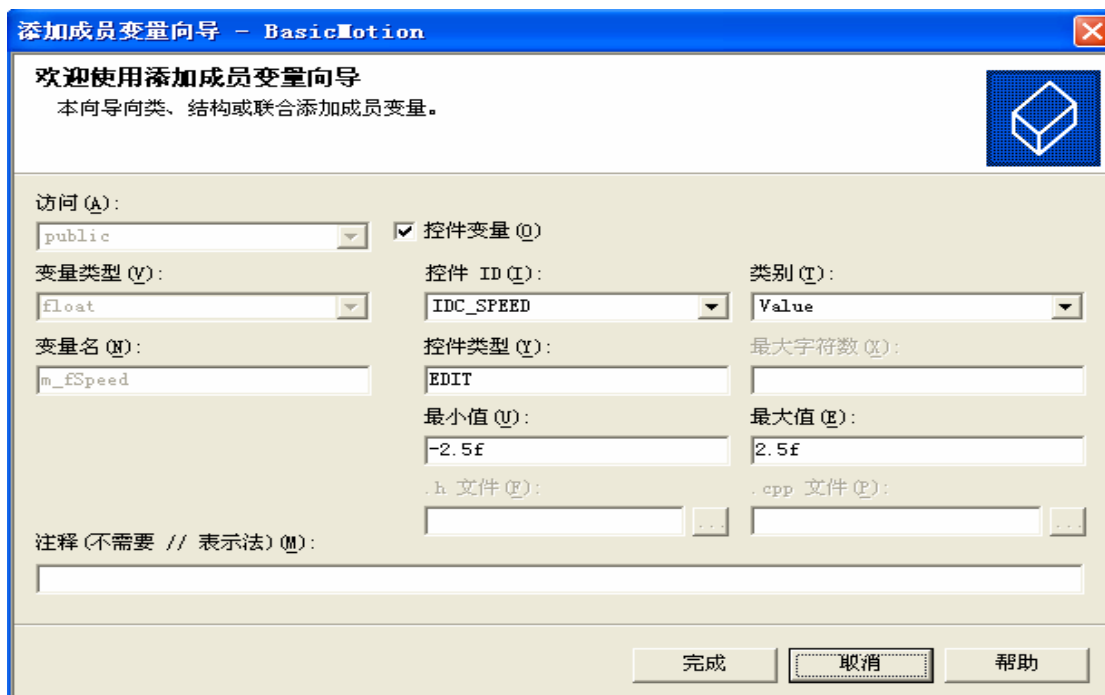


图 25 添加变量



为控件添加事件处理程序，若默认函数名称，只需用鼠标双击即可；若需要更改函数名称，鼠标右键点击相应的控件，选择“添加事件处理程序”，按提示步骤执行。

编写程序：

第一步：初始化

在 OnInitDialog() 内添加 BuildSystem()；



第二步: BuildSystem()

```
int CBasicMotionDlg::BuildSystem(void)
{
    try
    {
        CreateDevice("MOTION", m_pDevMotion); // 根据给定的设备名称创建对应
        设备, 若创建成功, 给传入的设备接口赋值。第一个参数: 需要创建的
        设备名称, 如运动设备名称为: MOTION; 第二个参数: 创建的设备接口,
        详见下一节接口函数库
        m_pDevMotion->Init(); // 初始化部件
        AddMsgReceiver(m_pDevMotion); // 将当前设备加入用户程序的
        设备列表;
        m_pDevMotion->QueryInterface(InterfaceID_MOTION,
        (LPVOID*)&m_pMotion);
        } // 查询部件实现的接口。第一个参数: 对应接口标识; 第二个参数:
        对应设备对象地址, 若设备创建失败, 返回 NULL;
    catch (IngeniousException& e) // 异常处理
    {
        Cleanup(); // 当程序出现异常时, 停止并销毁创建的指针
        String16 str16;
        char strTchar[256];
        str16 = e.GetErrorMessage();
        setlocale(LC_CTYPE, "");
        wcstombs(strTchar, str16.c_str(), 256);
        MessageBox(strTchar, "ERROR", MB_OK);
        return 1;
    }
    return 0;
}
```

第三步: 运动函数

```
void CBasicMotionDlg::Move(float lSpeed, float rSpeed, float Acce)
{
    ASSERT(m_pMotion);
    if (m_pMotion)
    {
        m_pMotion->drive(1, 1, lSpeed, 0, 0, 0, Acce, 0, rSpeed, 0, 0, 0, Acce,
        0); // 参数1: 1为速度闭环; 参数2: 1为默认PID; 参数3: 左轮速度; 参数4: 左轮P值;
        参数5: 左轮I值; 参数6: 左轮D值; 参数7: 左轮加速度; 参数8: 左轮位移; 参数9:
```



右轮速度；参数10：右轮P值；参数11：右轮I值；参数12：右轮D值；参数13：右轮加速度；参数14：右轮位移；

```
    }  
}
```

第四步：为按钮添加程序

```
void CBasicMotionDlg::OnBnClickedButtonFront()//前进  
{  
    UpdateData(TRUE); //更新数据  
    Move(m_fSpeed, m_fSpeed, m_Acce); //第一个参数：左轮速；第二个参数：右轮速；  
    //第三个参数：加速度  
}  
  
void CBasicMotionDlg::OnBnClickedButtonBack()//后退  
{  
    UpdateData(TRUE);  
    Move(-m_fSpeed, -m_fSpeed, m_Acce);  
}  
  
void CBasicMotionDlg::OnBnClickedButtonLeft()//左转  
{  
    UpdateData(TRUE);  
    Move(-m_fSpeed, m_fSpeed, m_Acce);  
}  
  
void CBasicMotionDlg::OnBnClickedButtonRight()//右转  
{  
    UpdateData(TRUE);  
    Move(m_fSpeed, -m_fSpeed, m_Acce);  
}  
  
void CBasicMotionDlg::OnBnClickedButtonStop()//停止  
{  
    UpdateData(TRUE);  
    Move(0.0f, 0.0f, m_Acce);  
}
```

第五步：销毁函数

```
int CBasicMotionDlg::DestroySystem(void)  
{  
    if(m_pMotion)  
    {  
        m_pMotion->End(); //停止使用运动设备  
        m_pMotion->Release(); //释放基本运动控制接口  
        m_pMotion = NULL;  
    }  
}
```



```
}  
    return 0;  
}  
  
void CBasicMotionDlg::OnDestroy()  
{  
    CDialog::OnDestroy();  
    DestroySystem();  
    // TODO: 在此处添加消息处理程序代码  
}  
  
int CBasicMotionDlg::Cleanup(void)  
{  
    if(m_pMotion)  
    {  
        m_pMotion->End();           //停止使用运动设备  
        m_pMotion->Release();       //释放基本运动控制接口  
        m_pMotion = NULL;  
    }  
    return 0;  
}
```

菜单栏->生成->生成 **BasicMotion**，若编译通过，在输出栏会提示成功，若不成功，请通过提示错误更改，若成功了，调试->启动，若编译通过了，但运行不成功，则按下面的单步调试步骤调试：

1、设置断点

一般先将断点设置在 **BuildSystem()**函数内

2、按 F5 运行

如果程序没有进入调试状态，请看**OnInitDialog()**函数内是否添加了

BuildSystem();

这是容易遗漏的地方，若进入调试状态，进入下一步。

3、按 F10 单步运行

若运行完 **CreateDevice("MOTION",m_pDevMotion);**后，直接进入 **Catch** 块内，请检查 **MTSoft** 文件夹是否在 **C: **下，若不在，请将 **MTSoft** 通过添加删除程序删掉，从新安装，并按照安装帮助一步步安装 **MTSoft**。

4、按 F11 可进入想要进入的函数。

5、若有多个断点，可按 F5 从一个断点跳到下一个断点。



第五章：MT-AR 智能型机器人开发指南

MTSoft 源程序采用 COM 接口，COM 是一项功能强大的集成技术，它允许您在运行时将各种异构的软件部分混合起来。COM 使得开发人员可以编写出能够混合起来运行的软件，而不用担心诸如线程特性以及所用开发语言等问题。COM 协议将一个软件模块同另一个软件模块连接起来，然后其自身不再参与两者之间的通信过程。当这种连接建立起来之后，两个模块之间就可以通过称为“接口”的机制来进行通信。这些接口既不需要静态的或动态的链接入口点，也不需要硬编码的地址，而且不同于那些用于启动通信过程的一般 COM 函数。

下面详细介绍各种接口函数：

5.1 系统函数接口描述

◆ CreateDevice

➤ 原型：

```
MT_EXP HRESULT __stdcall CreateDevice(LPSTR  
pModule, IMTDevice*& prDevice);
```

➤ 参数说明：

参数1为需要创建的设备名称；

如运动设备名称为：MOTION，TCP/IP 网络通讯客户端：NETWORK_CLIENT、服务器端：NETWORK_SERVER，UDP 网络通讯客户端：UDPVIDEO_CLIENT、服务器端：UDPVIDEO_SERVER，超声：MTULTRASONIC，PSD：MTPSD，图像采集：IMAGE_CAP_DS，图像处理：MT_IMAGE_PROCESS，DIDO 设备：MTDIDO，码盘：MTENCODER，语音：MTVOICE，硬件自检：MTCHECK，DA 设备：MTDA
参数2为创建的设备接口；

➤ 功能：

根据给定的设备名称创建对应设备，若创建成功，给传入的设备接口赋值。

◆ AddMsgReceiver

➤ 原型：

```
MT_EXP int AddMsgReceiver(IMTDevice * inReceiver);
```



➤ 参数说明:

参数1为当前设备指针;

返回值为当前设备列表中的设备数;

➤ 功能:

将当前设备加入用户程序的设备列表;

◆ Broadcast

➤ 原型:

```
MT_EXP bool Broadcast(MessageT inMessage, void * ioParam = 0, void  
* ioParam2 = 0);
```

➤ 参数说明:

参数1为预定消息, 在此一般为NTY_MSG_DEV_PTR;

参数2为消息发送的目标设备;

参数3为用户程序的设备列表指针;

➤ 功能:

将设备列表指针发送给指定目标设备, 这样各设备之间可以实现互访。

◆ IMTObject

接口原形:

```
interface IMTObject
```

```
{  
    virtual HRESULT __stdcall QueryInterface(InterfaceID, LPVOID*) = 0;  
    virtual void __stdcall AddRef() = 0;  
    virtual void __stdcall Release() = 0;  
};
```

所有部件基接口, 客户可以使用该接口查询需要的目标接口, 部件自己维护自身的生命周期, 所有具体的底层部件必须实现该接口

接口描述:

➤ QueryInterface

查询部件实现的接口。

参数 1 对应接口标识;



如运动接口标识为: InterfaceID_MOTION, 网络: InterfaceID_NETWORK, 超声: InterfaceID_ULTRASONIC, PSD: InterfaceID_PSD, 图像采集: InterfaceID_IMAGECAPTURE, 图像处理: InterfaceID_IMAGE, DDO 设备: InterfaceID_DDO, 码盘: InterfaceID_ENCODER, 语音: InterfaceID_VOICE, 硬件自检: InterfaceID_CHECK, DA 检测: InterfaceID_DA。

参数 2 对应设备对象地址, 若设备创建失败, 返回 NULL;

如:

```
定义: IMTMotion      *m_pMotion;  
      m_pMotion = NULL;
```

```
m_pDevMotion->QueryInterface(InterfaceID_MOTION, (LPVOID*)&m_pMotion);
```

➤ AddRef

用户成功查询到一个, 引用计数增 1

➤ Release

用户不再需要使用该接口, 引用计数减 1

COM 术语中有“类对象”(class object)的概念, 有时也被称为“类厂”(class factory)。精确地来讲, 它实际上应该被称为“对象厂”(object factory)。COM 中的一个类对象代表了一个特定 COM 类的全局静态域。其意义与 MFC 中的 CRuntimeClass 是等价的。类对象之所以被称为类厂, 其原因在于它实现了一个名为 IClassFactory 的特殊的 COM 接口。因为我们无法直接调用目标类的构造函数, 所以只能让组建模块来决定如何构造对象。为此组建提供了类厂, 从而将具体的创建步骤封装了起来。

◆ IMTFactory

接口原形

```
interface IMTFactory  
{  
    virtual HRESULT __stdcall CreateObject(IMTObject*& ) = 0;  
    virtual void __stdcall Release() = 0;  
};
```



接口描述:

➤ CreateObject

工厂创建部件的标准方法

参数 1 为部件的接口指针的引用。成功时指向刚创建的部件，否则返回

NULL

➤ Release

用户不再需要使用该工厂接口，释放该工厂

◆ IMTDEVFunc

接口原形

```
interface IMTDEVFunc
{
    virtual HRESULT __stdcall DevFunc(LPVOID pParam) = 0;
};
```

接口描述:

➤ DevFunc

执行用户设定的命令，与特定的部件关联

参数 1 为与命令本体关联的设备

◆ IMTDevice

接口原形

```
interface IMTDevice:public IMTObject
{
    virtual HRESULT __stdcall Init() = 0;
    virtual HRESULT __stdcall Pause() = 0;
    virtual HRESULT __stdcall Start() = 0;
    virtual HRESULT __stdcall End() = 0;
    virtual bool ReceiveMessage(long , void * ,void * ) = 0;
    virtual bool Broadcast(MessageT inMessage, void * , void * ) = 0;
    virtual HRESULT __stdcall SetDevFunc(IMTDEVFunc* pDevFunc) = 0;
    virtual HRESULT __stdcall ReadSettings() = 0;
```



```
virtual HRESULT __stdcall WriteSettings() = 0;  
};
```

接口描述:

- Init
初始化部件;
- Pause
将部件转换到暂时停止转态;
- Start
启动设备;
- End
将部件转换到停止转态;
- ReceiveMessage
处理接收到的消息及相应参数指针;
参数 1 为预定义的消息值;
参数 2 和参数 3 为需要传递的参数指针;
- Broadcast
向当前设备列表发送消息及相应参数指针;
参数 1 为预定义的消息值;
参数 2 和参数 3 为需要传递的参数指针;
- SetDevFunc
把指定命令对象绑定到当前设备;
- ReadSettings
获取设备上次的配置参数;
- WriteSettings
保存设备上次的配置参数;

5.2 设备接口参考说明

5.2.1 PSD 模块接口

◆ IMTPSD接口



函数列表:

➤ SetPSDProc

- 原型:

`virtual void SetPSDProc(DeviceProc proc, void* pParam) = 0;`

- 功能:

设定事件处理过程（接收到数据）

- 参数说明:

DeviceProc: 传入一个函数指针，该函数接收到事件后，交由该指针指向的函数处理

函数原形为: `typedef void ( __stdcall *DeviceProc)( void* pParam, float32* speed, UInt16 num );`

speed 为返回的 PSD 测得到障碍物的距离值; num 为通道号 (0-15)

pParam: DeviceProc 指向的函数所需的参数，一般为一个类或结构的指针

➤ OpenChannel

- 原型:

`virtual void OpenChannel(UInt16 channel) = 0;`

- 功能:

打开通道

- 参数说明:

channel: 通道号 (0-15)

➤ CloseChannel

- 原型:

`virtual void CloseChannel(UInt16 channel) = 0;`

- 功能: 关闭通道

- 参数说明:

channel: 通道号 (0-15)



5.2.2 超声模块接口

◆ IMTUltraSonic接口

函数列表

➤ SetSonarProc

- 原型:

```
virtual void SetSonarProc (DeviceProc proc, void* pParam) = 0;
```

- 功能:

设定事件处理过程（接收到数据）

- 参数说明:

DeviceProc: 传入一个函数指针，该函数接收到事件后，交由该指针指向的函数处理

函数原形为: `typedef void ( __stdcall *DeviceProc)( void* pParam, float32* speed, UInt16 num );`

speed 为返回的超声测得到障碍物的距离值; num 为通道号 (0-15)

pParam: DeviceProc 指向的函数所需的参数，一般为一个类或结构的指针

➤ OpenChannel

- 原型:

```
virtual void OpenChannel(UInt16 channel) = 0;
```

- 功能:

打开通道

- 参数说明:

channel: 通道号 (0-15)

➤ CloseChannel

- 原型:

```
virtual void CloseChannel(UInt16 channel) = 0;
```

- 功能: 关闭通道

- 参数说明:

channel: 通道号 (0-15)



5.2.3 网络模块接口

◆ IMTNetwork接口

使用前，需先使用 SetParam 函数配置网络工作参数，然后调用 Init 初始化网络环境，再用 Start 函数启动。可以调用 Send 函数，发送数据到指定的 IP 地址。

函数列表

➤ GetParam: 获得当前该设备的工作参数

- 原型: NET_DESC __stdcall GetParam()
- 返回类型: NET_DESC
- 备注: 获得当前该设备的工作参数。这些参数包括 IP 地址、端口号、缓冲区大小等参数

➤ SetParam

- 原型
`virtual void __stdcall SetParam(const NET_STRU&)`
- 参数说明
NET_STRU &: 设备工作参数，包括 IP 地址、端口和缓冲区长度。
- 备注
设置特定的网络的参数，必须在 Init 前调用。

➤ Init

- 备注
初始化网络工作环境，并改变设备的状态为 DEVICE_INIT。在调用该函数前，必须调用 SetParam 来配置网络参数。然后，可以调用 Start 函数来启动网络模型。如果成功，返回 0。

➤ Start

- 备注



设备状态切换到 DEVICE_RUNNING。在设备启动后，就可以用 Send 函数发送数据，同时，自动接收客户端发送的数据。之后，调用 Pause 可以暂停设备运行，也可调用 End 终止设备运行。

➤ Pause

- 备注

暂停网络设备运行，并将设备状态切换到 DEVICE_PAUSE。该函数必须在设备初始化或运行时调用，否则将返回错误编码。在该函数调用后，可调用 Start 到启动设备继续运行，也可调用 End 函数终止设备运行。

➤ End

- 备注

结束网络设备，并将设备状态切换到 DEVICE_RAW。调用该函数后，终止网络设备的运行。如果想要再次启动设备，必须先调用 Init 对设备初始化后，才能调用 Start 函数来启动设备。

➤ Send

- 原型

```
HERROR __stdcall Send(LPSTR, LPSTR, LONG)
```

- 参数说明

- a) LPSTR: 指定对方的 IP 地址
- b) LPSTR: 用户需要传送的字符数据
- c) LONG: 用户需要传送的字符数据长度

- 备注

该函数向指定的地址发送数据。如果发送成功，返回 0，否则，返回错误的编码值。该函数只有在网络启动后，才会有效，也就是必须在 Start 函数后，才可调用。

➤ ResponseRecvEvent: 设定事件处理过程（接收到数据）

- 原 型

```
void __stdcall ResponseRecvEvent(PEVENTPROC, LONG)
```



- 参数说明:

a) PEVENTPROC: 传入一个函数指针。该函数接收到事件后, 交由该指针指向的函数处理。PEVENTPROC 原型定义如下:

```
typedef bool (*PEVENTPROC) (LONG nOwner, void* lpReturn)
```

b) LONG: PEVENTPROC 指向的函数所需的参数, 一般为一个类或者结构的指针。

- 备 注:

设定事件处理过程, 接收到数据后, 调用指定的函数处理。

PEVENTPROC 为一个函数指针, 指向处理事件的函数。LONG 为指向一个结构的指针。该类或结构为 PEVENTPROC 指向的函数所需的参数。

结构参数

➤ NET_STRU

- 原型:

```
typedef struct
{
    char szAddrIP[32];
    int nPort;
    int nBufLen;
    bool bSource;
} NET_DESC;
```

- 说明:

szAddrIP: 指定 IP 地址

nPort: 指定端口号

nBufLen: 指定缓冲区大小

➤ NETVIDEODATAFORMAT

- 原型:

```
typedef struct
```



```
{  
    unsigned short nWidth;  
    unsigned short nHeight;  
}NETVIDEODATAFORMAT;
```

- 说明:

nWidth: 像素宽

nHeight: 像素高

5.2.4 图像采集接口

◆ IMTImageCapture

利用 DirectShow 来驱动 WDM 摄像设备，完成对视频的采集。

函数列表

➤ BindCapWnd

- 原型

```
HERROR __stdcall BindCapWnd(HWND hWnd)
```

- 参数说明

hWnd: 要绑定的窗口的句柄

- 备注

将视频输出绑定到指定的窗口上。

➤ BindStillWnd

- 原型

```
HERROR __stdcall BindStillWnd(HWND hWnd)
```

- 参数说明

hWnd: 要绑定的窗口的句柄

- 备注

将静态图的抓拍输出绑定到指定的窗口。

➤ CapStill

- 原型

```
HERROR __stdcall CapStill(int lWidth, int lHeight)
```



- 参数说明

lWidth: 窗口宽

lHeight: 窗口高

- 备注

抓拍当前图像帧，并在绑定窗口中显示。

➤ GetDeviceName

- 原型

short __stdcall GetDeviceName(char** pDevList)

- 参数说明

摄像头设备名称列表指针

- 备注

获取当前主机连接的摄像头设备名称列表指针

➤ BindDevName

- 原型

HERROR __stdcall BindDevName(char* strName, int m_nIndex)

参数说明

strName: 需要绑定的摄像头名称及通道。

m_nIndex: 通道

- 备注

将当前设备绑定到指定摄像头。

➤ BindDevIndex

- 原型

HERROR __stdcall BindDevIndex(int m_nIndex)

参数说明

m_nIndex: 通道号，通过 GetDeviceName 获得设备名称按顺序添加到 Combo 控件内，通过选择 Combo 控件内的名称，对应通道号从 0 开始增加。

- 备注

将当前设备绑定到指定摄像头。

➤ CaptureImage



- 原型

```
virtual void __stdcall CaptureImage(char  
*ImageBuffer)=0;
```

- 参数说明

参数图像缓存，通过这条语句可获得当前这帧图像的数据

- 备注

捕获一帧图像

➤ AdjustVideoQuality

- 原型

```
virtual HRESULT __stdcall AdjustVideoQuality() = 0;
```

- 参数说明

- 备注

调节视频质量，包括图像亮度，色调等等

➤ CrossBar

- 原型

```
virtual HRESULT __stdcall CrossBar() = 0;
```

- 参数说明

- 备注

改变摄像头的制式

➤ Init

- 原型

```
HRESULT __stdcall Init()
```

- 参数说明

- 备注

➤ Start

- 原型

- 参数说明

- 备注

该函数将启动设备进行视频采集，并输出到由 BindCapWnd 函数指定



的窗口中，可以调用 End 函数终止采集。在调用该函数前，确保设备已经初始化，即已经调用过 Init 函数，否则将返回一个错误值。

➤ End

- 备注

该函数终止视频的采集。但并不释放设备环境，之后，如果调用 Start 函数可以继续视频的采集。

➤ SetVideoSettings

- 原型

```
HERROR __stdcall SetVideoSettings(short nWid, short nHei, short nFrameRate)
```

- 参数说明

参数 1 为图像宽度像素大小；

参数 2 为图像高度像素大小；

参数 3 为图像采集帧率；

- 备注

设置图像参数

结构参数

➤ CAP_STRU

- 原型

```
typedef struct  
{  
    WORD wWidth;  
    WORD wHeight;  
    int nFramePerSec;  
}CAP_STRU;
```

- 备注

wWidth: 图像的宽度(字节为单位)

wHeight: 图像的高度(字节为单位)

nFramePerSec: 每秒的帧数



5.2.5 图像处理接口

◆ Image接口

操作图像文件，如打开、关闭文件等操作。

函数列表

➤ SetParam

- 原型

```
HERROR __stdcall SetParam(IMAGEPROC_DESC&)
```

- 参数说明

IMAGEPROC_DESC&: 图像处理的参数设置

- 备注

IMAGEPROC_DESC 结构里，存储的是设备信息。如设备长度、宽度。
使用该函数来格式化输出图像。如果，设置的长、宽与原图像不相同，
则会把图像缩放。

➤ BackupDIB

- 原型

```
HERROR __stdcall BackupDIB ();
```

- 参数说明

- 备注

复制当前图像数据到内存当中，在需要时，使用 RestoreDIB 把这些数据
还原到原图像。一般情况下，在图像打开后，把图像备份。图像经过
一些处理，如果处理失误，就可以把备份的图像数据还原。

➤ RestoreDIB

- 原型

```
HERROR RestoreDIB __stdcall RestoreDIB(BOOL bClear);
```

- 参数说明

bClear : TRUE 表示清除备份的数据，FALSE 表示保留备份数据

- 备注

该函数将备份的图像还原，如果没有备份数据，则不做出反映。通常，
在图像打开后，调用 BackupDIB 把图像备份。图像经过一些处理，如果



处理失误，就可以把备份的图像数据还原。

➤ CopyDIBObject

- 原型

```
LPSTR __stdcall CopyDIBObject ();
```

- 参数说明

bClear : TRUE 表示清除备份的数据, FALSE 表示保留备份数据

- 备注

复制图像数据到一块内存区，并返回指向该内存的句柄。该函数同 BackupDIB 类似。但比 BackupDIB 更加灵活。BackupDIB 复制的数据的句柄是隐藏的，不能够对其进行操作。而 CopyDIBObject 则是得到当前图像数据的一个副本，并可对它进行处理。

➤ LoadDIBFile

- 原型

```
LPSTR __stdcall LoadDIBFile(char* lpFilePath);
```

- 参数说明

lpFilePath : 需要处理的图像文件文件名

- 备注

加载图像文件，并初始化 IMTImage 对象。只有当调用了该函数后，才可以进行后续处理，否则后续的调用将会返回错误信息。

➤ SaveDIBToFile

- 原型

```
HERROR __stdcall SaveDIBToFile (char* lpFilePath);
```

- 参数说明

lpFilePath : 需要处理的图像文件文件名

- 备注

将图像数据保存到参数中指定的文件，如果文件为空，将创建一个新文件，如果文件已经存在，将覆盖现有文件。

➤ GetDIBPtr

- 原型



LPSTR __stdcall GetDIBPtr();

- 参数说明

返回图像数据的指针。

- 备注

得到图像的指针，该指针指向 BITMAPINFOHEADER 结构。得到该指针后，就可以用它来进行自定义的操作。如可以把它绘制在窗口上，或保存为文件。

➤ GetWidth

- 原型

LONG __stdcall GetWidth();

- 参数说明

返回图像的宽度。

- 备注

获取图像的宽度值，该值是以实际的字节数计算的，并且必须是 4 的整数倍。与图像的 BITMAPINFOHEADER 结构中保存的宽度值可能会不一样。BITMAPINFOHEADER 中保存的值是以像素数计算的，而一个像素可能会占用多个字节。如 8 位图像，一个像素占用一个字节，24 位图像，一个像素占用三个字节（B, G, R），即蓝、绿、红。32 位图像一个像素占用四个字节（B, G, R, A），即蓝、绿、红、透明度。所以，计算图像的实际宽度时，须用 BITMAPINFOHEADER 中的宽度乘以每像素的字节数，然后取其离 4 的整数次倍最近一个数。并且它必须是大于或等于 bi.biWidth。

➤ GetHeight

- 原型

LONG __stdcall GetHeight();

- 参数说明

返回图像的高度。

- 备注

获取图像的高度值。以字节数计算。该值总是信任图像的 BITMAPINFOHEADER 结构中提供的高度值。

➤ GetDIBSize



- 原型

```
LONG __stdcall GetDIBSize ();
```

- 参数说明

返回图像数据的大小。

- 备注

获取图像数据部分的大小。不包括 BITMAPINFOHEADER 结构和调色板数据。采用公式： $biSize = biWidth' \times biHeight$ 。其中 $biWidth'$ 必须是 4 的整倍数，表示大于或等于 $biWidth$ ，离 4 最近的整倍数。请参见 `GetWidth` 函数。

➤ PaintDib

- 原型

```
HERROR __stdcall PaintDib(HWND hWnd);
```

- 参数说明

`hWnd`: 窗体对象的指针。

- 备注

该函数将图像绘制在指定的窗体上。并根据窗体大小，将图像进行伸缩后绘制。通常可以将该函数放在 `Dialog` 的 `OnPaint` 事件中调用或在 `CView` 的 `OnDraw` 中调用。

➤ RGBToGray256

- 原型

```
HERROR __stdcall RGBToGray256();
```

- 参数说明

- 备注

将 24 位真彩图转化为 256 色灰度图。常用的图像处理算法，只能够处理 0—255 之间颜色的灰度图，所以，打开图像文件后，就首先要将其转换为灰度图，以便后续算法进行处理。转换的公式为： $y = (0.299R + 0.587G + 0.114B)$ 。其中， y 为亮度值， R 、 G 、 B 为每个像素的红、绿、蓝份量值。

结构参数



- 原型

```
typedef struct
{
    int nWidth;
    int nHeight;
} IMAGEPROC_DESC;
```

- 备注

nWidth: 图像的宽度; nHeight: 图像的高度。使用该结构来存储设备信息。

◆ ImageProcessGray接口

用于处理灰度图像。包含了大多数常用的图形图像处理算法。如行进图像点运算、几何运算、正交变换、边缘检测、形态学变换、图像增强等操作。

➤ IntensityChart: 直方图分析, 统计每一种颜色使用的次数。

- 原型

```
HERROR __stdcall IntensityChart(LONG[], int = 256) < BR >
```

- 参数说明

a) LONG[]: 长度为 N 的数组, 用来存放直方图的数据。

b) int : 为 LONG[] 数组的长度

- 返回值: 返回 HERROR, 错误编码

- 备注: LONG[] 一般为长度为 256 的数组, int 默认为 256。

➤ LinerTransfer: 线形变换

- 原型

```
HERROR __stdcall LinerTransfer(float fa, float fb)
```

- 参数说明

a) fa: 线性函数的斜率;

b) fb: 线形函数的截距;

- 返回值: 返回 HERROR, 错误编码

- 备注: 灰度的线性变换就是将图像中所有的点的灰度按照线性灰度变换函数进行变换。该函数 $f(x)$ 是一个一维线性函数: $f(x) = fA * x + fB$ 。fA 为



线性函数的斜率， fB 为线性函数在 Y 轴的截距。当 $fA > 1$ 时，输出图像对比度增大，当 $fA < 1$ 时，输出图像对比度将减小。当 $fA = 1$ && $fB \neq 0$ 时，操作仅使所有像素的灰度值上移或下移，整个图像变亮或变暗。当 $fA < 0$ 时，暗区域将变亮，亮区域将变暗。当 $fA = 1$ && $fB = 0$ 时，输入、输出相同。当 $fA = -1$ && $fB = 255$ 时，输出图像灰度刚好翻转。

- **ThresholdTransfer: 阈值分割。** 将一幅灰度图像转换为黑白二值图像
 - 原型: `HERROR __stdcall ThresholdTransfer(BYTE ThresholdValue)`
 - 参数说明:
 - ThresholdValue: 阈值, 缺省为 128
 - 返回值: 返回 HERROR, 错误编码
 - 备注:

阈值变换将一幅灰度图转换成黑白二值图。首先由用户指定一个阈值, 如果图像中的像素的灰度值小于该阈值, 则将图像的灰度值设置为 0, 否则, 设置为 255。当然, 在许多时候, 都不能够由人工指定一个固定阈值, 而只能根据当前图像, 求取一个最佳阈值。这时, 可以调用 [ThresholdAuto](#) 函数, 它会分析当前图像, 当求出一个最佳阈值, 然后再进行阈值分割。
- **WindowThresholdTransfer: 窗口变换。** 它的操作和阈值变换有点类似
 - 原型: `HERROR __stdcall WindowThresholdTransfer(BYTE nLow, BYTE nHigh)`
 - 参数说明
 - a)nLow: 窗口下限
 - b)nHigh: 窗口上限
 - 返回值: 返回 HERROR, 错误编码
 - 备注:

它设定一个窗口的范围, 该窗口中的灰度值保持不变, 小于窗口下限的设置为 0, 大于窗口上限的设置为 255。
- **GrayStretch: 灰度拉伸。** 类似于线性变换。但不同之处它不是完全的线性变换, 而是分段进行线性变换。



- 原型: `HERROR __stdcall GrayStretch(BYTE x1, BYTE y1, BYTE x2, BYTE y2)` ;
- 参数说明
 - a) x1: 需要拉伸区域的下限;
 - b) x2: 需要拉伸区域的上限;
 - c) y1: 拉伸后区域的下限;
 - d) y2: 拉伸后区域的上限。
- 返回值: 返回 `HERROR`, 错误编码
- 备注: 将 0-x1 之间的区域拉伸到 0-y1; 将 x1-x2 之间的区域, 拉伸到 y1-y2; 将 x2-255 之间的区域, 拉伸到 y2-255。

➤ **IntensityEqualize:** 灰度补偿。也叫直方图均衡

- 原型: `HERROR __stdcall IntensityEqualize()` ;
- 返回值: 返回 `HERROR`, 错误编码
- 备注:

使图像在每一灰度级上都有相同的像素点数, 即直方图输出是平的。

直方图均衡的公式:

$$f(Da) = \frac{D_{\max}}{Ao} \sum_{i=0}^{i=Da} Hi$$

其中 $D_{\max}$ 为最大灰度值 (255), Ao 为图像面积, Hi 为第 i 级灰度的像素个数。

5.2.6 运动控制模块接口

◆ IMTMotion接口

该接口用来控制对应电机运动。

函数列表

➤ **drive:** 运动指令

- 原型:

```
virtual HERROR __stdcall drive(int ModeSelect, int PidSelect, float lv, int lKp, int lKi, int lKd, float la, float ls, float rv, int rKp, int rKi, int rKd, float ra, float rs) = 0;
```



- 参数说明:

ModeSelect: 模式选择, 速度闭环为 1, 位置闭环为 2, 协调控制为 4, 保存 PID 参数为 8, 速度闭环+协调控制为 5, 位置闭环+协调控制为 6。

PidSelect: PID 调节, 默认 PID 为 1, 在线调试 PID 为 2, LoadPID 为 4。

lv: 左轮速

lKp: 左轮 PID 调节中的比例

lKi: 左轮 PID 调节中的积分

lKd: 左轮 PID 调节中的微分

la: 左轮加速度值

ls: 左轮位移值

rv: 右轮速

rKp: 右轮 PID 调节中的比例

rKi: 右轮 PID 调节中的积分

rKd: 右轮 PID 调节中的微分

ra: 右轮加速度值

rs: 右轮位移值

➤ EncoderDis: 获得相对位移

- 原型: `virtual HRESULT __stdcall EncoderDis() = 0;`

➤ MotionPlan: 运动规划

- 原型: `virtual HRESULT __stdcall MotionPlan(TYPESELECT TypeSelect, float velocity, float distance)=0;`

- 参数类型:

TypeSelect: 0 时为直线; 2 时为圆

velocity: 速度

distance: 类型选择直线时为位移值; 圆时为半径。

➤ EnableMotion

- 原型: `virtual HRESULT __stdcall EnableMotion(int mode, bool enable) = 0;`

- 参数类型:



mode: 为 0 时发送码盘命令, 为 1 时查看 PID 参数。

Enable 为使能信号, 为 1 时表示开使能, 为 0 时表示关使能

➤ SetMotionProc: 设置查看 PID 参数处理过程

- 原型: `virtual void SetMotionProc(DeviceMotionProc proc, void* pParam) = 0;`

- 参数类型:

DeviceMotionProc: 传入一个函数指针, 该函数接收到事件后, 交由该指针指向的函数处理

函数原形为: `typedef void (__stdcall *DeviceMotionProc)(void* pParam, UInt8 LP, UInt8 LI, UInt8 LD, UInt8 RP, UInt8 RI, UInt8 RD);`

pParam: DeviceMotionProc 指向的函数所需的参数, 一般为一个类或结构的指针

LP, LI, LD, RP, RI, RD 分别为左轮 P, 左轮 I, 左轮 D, 右轮 P, 右轮 I, 右轮 D 值。

➤ SetEncoderProc: 设置码盘处理过程

- 原型: `virtual void SetEncoderProc(DeviceEncoderProc proc, void* pParam)=0;`

- 参数类型:

DeviceEncoderProc : 传入一个函数指针, 该函数接收到事件后, 交由该指针指向的函数处理

函数原形为: `typedef void ( __stdcall *DeviceEncoderProc)(void* pParam, float32* lvelocity, float32* rvelocity, float32* ldistance, float32* rdistance, float32 timer);`

pParam: DeviceEncoderProc 指向的函数所需的参数, 一般为一个类或结构的指针

lvelocity: 左轮速度返回值

rvelocity: 右轮速度返回值

ldistance: 左轮位移返回值

rdistance: 右轮位移返回值



timer: 上次返回值与这次返回值之间的时间间隔

➤ SetEncoderDisProc: 设置获得码盘相对位移处理过程

● 原型: virtual HRESULT __stdcall SetEncoderDisProc(DeviceProc proc, void* pParam) = 0;

● 参数类型:

DeviceProc : 传入一个函数指针, 该函数接收到事件后, 交由该指针指向的函数处理

函数原形为: typedef void (__stdcall *DeviceProc)(void* pParam, float32* speed, UInt16 num);

pParam: DeviceProc 指向的函数所需的参数, 一般为一个类或结构的指针

speed[0]: 左轮相对位移

speed[1]: 右轮相对位移

speed[2]: x 中相对位移

speed[3]: y 轴相对位移

5.2.7 语音接口

IMTVoice接口

语音采集、识别、播放、训练。

a) Init()

i. 原型

HRESULT __stdcall Init ()

ii. 说明

语音设备初始化

b) End()

i. 原型

HRESULT __stdcall End ()

ii. 说明

结束语音设备运行



c) Start()

i. 原型

HERROR __stdcall Start ()

ii. 说明

启动语音设备运行

d) Pause()

i. 原型

HERROR __stdcall Start ()

ii. 说明

暂停语音设备运行

e) AddKeyWord()

i. 原型

HERROR __stdcall AddKeyWord (char*)

ii. 说明

向用户词典添加关键词

f) Speak()

HERROR __stdcall Speak (char*)

i. 原型

朗读指定内容

ii. 说明

g) SpeakSetRate()

i. 原型

HERROR __stdcall SpeakSetRate (long)

ii. 说明

设定朗读语速

h) SpeakSetVolume()

i. 原型

HERROR __stdcall SpeakSetVolume (long)

ii. 说明



设定朗读音量

i) PlayWavSound()

i. 原型

`HERROR __stdcall PlayWavSound (char*)`

ii. 说明

播放指定路径的音频文件

j) GetText()

i. 原型

`HERROR __stdcall GetText (char*)`

ii. 说明

获取识别出的内容

k) VoiceTraining()

i. 原型

`HERROR __stdcall VoiceTraining ()`

ii. 说明

执行语音训练

l) MicrophoneSetup()

i. 原型

`HERROR __stdcall MicrophoneSetup ()`

ii. 说明

运行麦克风配置向导

m) IsDictationOn()

i. 原型

`BOOL __stdcall IsDictationOn ()`

ii. 说明

返回值为当前是否处于听写状态的布尔变量。

n) InitVoiceCombo()

i. 原型

`virtual HRESULT InitVoiceCombo(HWND hWnd){return false;;}`

ii. 说明

初始化语音列表



参数：获得 Combo 控件句柄

o) voiceChange()

i. 原型

virtual HRESULT VoiceChange(HWND hWnd){return false;;}

ii. 说明

改变语音发音

参数：获得 Combo 控件句柄

o) GetReadRate()

i. 原型

virtual long GetReadRate(){return 0;;}

ii. 说明

获得朗读速度

返回值为返回的速度值，从-10 到 10 变化，0 表示正常速度，-10 表示速度慢，10 表示速度快

p) RateChange()

i. 原型

virtual HRESULT RateChange(long m_Rate){return false;;}

ii. 说明

改变朗读速度

参数值为设定的速度值，从-10 到 10 变化，0 表示正常速度，-10 表示速度慢，10 表示速度快

q) GetVoiceVolume()

i. 原型

virtual USHORT GetVoiceVolume(){return 0;;}

ii. 说明

获得声音音量

返回值为音量值，从 0 到 100 变化

r) VolumeChange()

i. 原型

virtual HRESULT VolumeChange(USHORT m_Volume){return false;;}

ii. 说明

改变声音音量

设定值为音量值，从 0 到 100 变化

s) InitAudioInCombo()

i. 原型

virtual HRESULT InitAudioInCombo(HWND hWnd){return false;;}

ii. 说明

初始化音频输入设备列表

参数值为获得 Combo 控件句柄

t) AudioControl()

i. 原型

virtual HRESULT AudioControl(){return false;;}

ii. 说明

录音控制，可以调整 Mic 音量及音量控制



u) AudioProperty()

i. 原型

```
virtual HRESULT AudioProperty(){return false;;}
```

ii. 说明

音频属性，可以选择音频输入设备

v) LanguageChange()

i. 原型

```
virtual HRESULT LanguageChange(int LangeuageID){return false;;}
```

ii. 说明

语言选择

参数说明：LangeuageID 为 0 时语言为汉语，为 1 时语言为英语

5.2.9 硬件自检

IMTCheck 接口：

接口原型：

```
interface IMTCheck:public IMTDevice
{
virtual void SetCheckProc(DeviceCheckProc proc, void* pParam) = 0;
virtual void Check(bool enable) = 0;
};
```

函数说明：

➤ SetCheckProc

功能：

设定事件处理过程（接收到数据）

参数说明：

DeviceCheckProc：传入一个函数指针，该函数接收到事件后，交由该指针指向的函数处理

函数原形为：`void __stdcall CheckProcFun(void* pParam, UInt16 DeviceID, UInt16 channel, UInt16 status);`

pParam：DeviceCheckProc 指向的函数所需的参数，一般为一个类或结构的指针

DeviceID 为设备 ID 号，运动 ID 号为 1，超声为 3，PSD 为 5

channel 为通道号，运动分为左电机和右电机；超声和 PSD 有 16 个通道
status 为状态位，为 1 时表示正在检测，为 2 时表示异常，为 4 时表示正常

➤ Check

功能：

启动或暂停自检

参数说明：



enable 为 true 时启动自检，为 false 时暂停自检

5.2.10 DIDO 设备

IMTDIDO 接口：

接口原型：

```
interface IMTDIDO:public IMTDevice
{
    virtual void SetDIProc(DeviceProc proc, void* pParam) = 0;
    virtual void OpenChannel(UInt16 channel)=0;
    virtual void CloseChannel(UInt16 channel)=0;
    virtual void D0(int channel,bool enable) = 0;
    virtual void KickBall(int channel) = 0;
};
```

函数说明：

➤ SetDIProc

功能：

设定事件处理过程（接收到数据）

参数说明：

DeviceProc：传入一个函数指针，该函数接收到事件后，交由该指针指向的函数处理

函数原形为：`typedef void ( __stdcall *DeviceProc)( void* pParam, float32* speed, UInt16 num );`

speed 为返回的 DI 值；num 为通道号（0-15）

pParam：DeviceProc 指向的函数所需的参数，一般为一个类或结构的指针

➤ OpenChannel

功能：打开通道

参数说明：

channel：通道号 0~15

➤ CloseChannel

功能：关闭通道

参数说明：

channel：通道号 0~15

➤ D0

功能：打开或关闭 D0 通道

参数说明：

channel：通道号 0~15



enable: 使能信号

➤ KickBall

功能: 踢球

参数说明:

channel: 踢球接在 D02 上, channel=2

5.2.11 DA 设备

IMTDA 接口:

接口原型:

```
interface IMTDA:public IMTDevice
{
    virtual void DA(int channel,int value,bool enable) = 0;
};
```

函数说明:

➤ DA

功能: 发送 DA 指令

参数说明:

channel: 通道值, DA 有 8 路通道, 第一路和第二路被占用, 还有其他六路提供给用户开发使用, channel 值从 2-7 对应第三路到第八路通道。

value: 为 0~255 之间的值, 对应 0~5V 电压

enable: 为使能信号, 1 为开使能, 0 为关使能。



结 束 语

仓促之间，准备了这本说明书，其实更多的知识是在跟大家交流过程中体验到的，欢迎您多与我们联系！我们共同学习，共同成长！

随时欢迎您访问我们的网站：www.ingenious.cn

光临我们的论坛：bbs.ingenious.cn 与我们的工程师及朋友们共商机器人！

也欢迎您多提意见和建议，因为您的建议和意见对我们的成长非常重要！