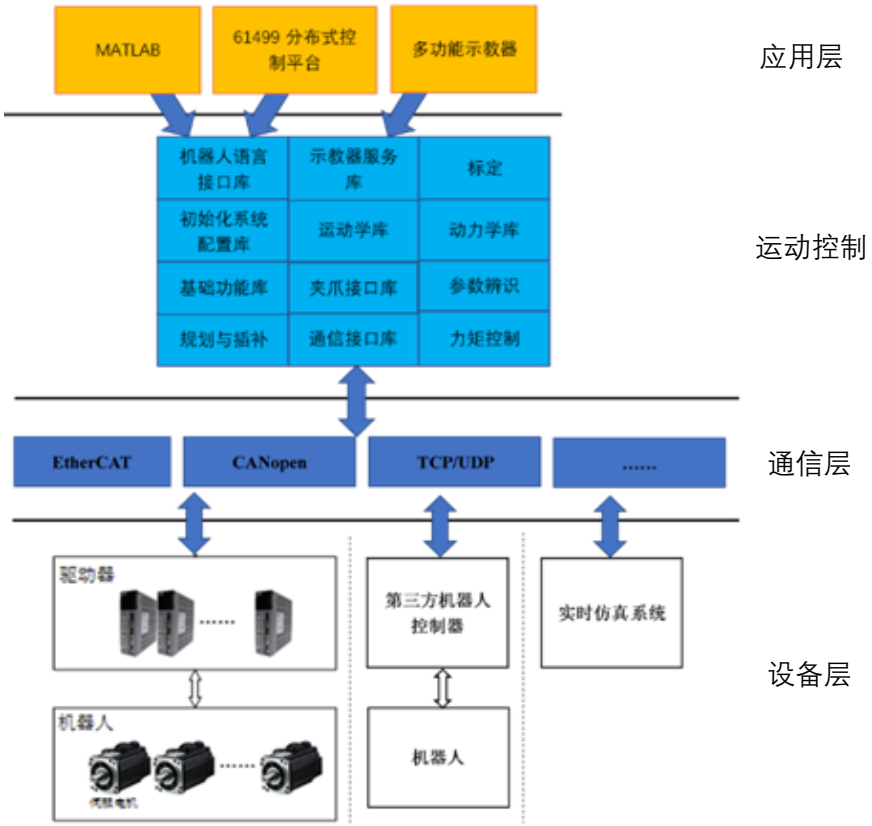
机器人控制系统手册

目录

1 机器人系统架构

1.1 控制系统框架

控制系统分为四层，最上层为应用层，第二层为运动控制层、第三层为通信层、第四层为设备层。运动规划层与控制层提供了大量的二次开发接口。

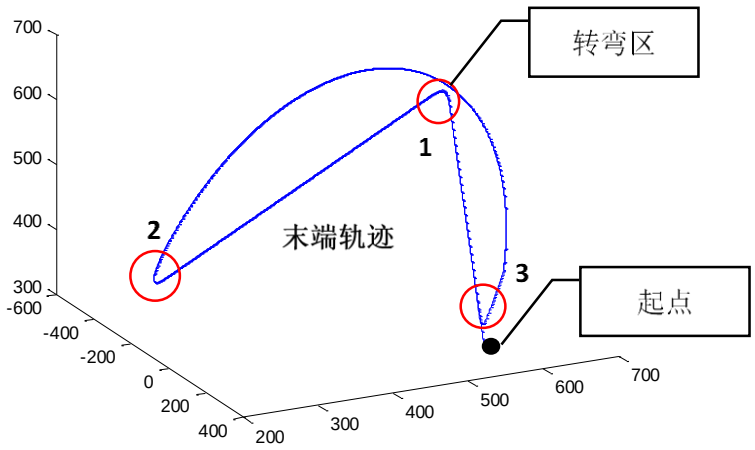


1.2 控制系统功能介绍

1.2.1运动控制功能

支持关节空间和笛卡尔空间的运动控制。支持笛卡尔空间的直线、圆弧运动，及直线与直线、直线与圆弧，圆弧与直线、圆弧与圆弧的平滑过渡运动。支持过多必经点的高阶 B 样条曲线运动。

通过调用相应接口，实现快速搭建机器人应用功能。同时可与相机等外部设备进行通信，完成多系统协作工作。



1.2.2工具坐标系、工件坐标系及用户坐标系的标定

提供了工具、工件、用户坐标系的标定，便于用户应用开发。

工具是指安装在机器人末端法兰上用来完成特定加工工序的器具，常见的工具有气动/电动手抓、焊枪、 喷头等。机器人出厂时没有附带任何工具，您需要根据实际情况选择外购或者自行设计合适的工具并完成安装和设置，才可使用机器人进行工作。使用任何一个工具之前都必须先进行标定，以获取 TCP（什么是工具中心点？）的数据。当使用外部工具功能时，工具安装在机器人工作范围内的固定位置而不是机器人上。

工件是指机器人使用工具进行加工或者处理的物品，引入工件的概念是为了简化编程步骤，提高效率。机器人的运动轨迹都是在工件坐标系下定义的，这样主要有两个好处：

①当工件发生移动或者加工多个相同工件时，只需要重新标定工件坐标系，程序中的所有路径即可随之更新，而不需要重新编写程序。

②允许加工被外部轴(如导轨，变位机等)移动的工件。

每一个工件实际上使用包含两个坐标系，一个是工件相对的用户坐标系，可以理解为摆放工件的工作台/桌子，在处理多个相同工件时非常有用；另一个是固连在工件上的工件坐标系，所有的程序路径都是在工件坐标系下描述的。

1.2.3运动学标定

机器人在制造装配过程中容易产生一些误差，这些误差对机器人的运动精度会有较大的影响，所以减小误差对提高机器人运动精度至关重要。对机器人进行标定的目的是了解末端位姿与运动学几何参数误差之间的准确关系。在机器人加工制造和装配过程中会使连杆长度参数和连杆偏置参数产生误差。相邻直线之间的平行度和垂直度会产生连杆扭角误差。在机器人装配过程中，由于电动机编码器的零点与机器人三维模型中关节的旋转零位不在同一轴线上，会产生关节角误差。为了提高机器人运动精度，需要通过机器人标定对这些参数进行补偿。

1.2.4 动力学参数辨识

机器人技术正在向高速、高精度和智能化方向发展，因此，对机器人的控制精度提出了更高的要求。相比于传统仅基于误差反馈的控制方案，基于模型的控制由于加入了机器人的动力学模型，因而可提高机器人的动态性能及对轨迹的跟踪精度。构造基于模型的控制方案离不开精确的动力学模型，然而实际机器人存在诸多影响动力学的因素，必须对其进行补偿。采用对整体机器人进行动力学参数辨识的方法，既不增加动力学模型的复杂性，又可体现各种动力学影响因素的作用，获取最接近机器人实际动态特性的动力学模型。

1.2.5JOG操作

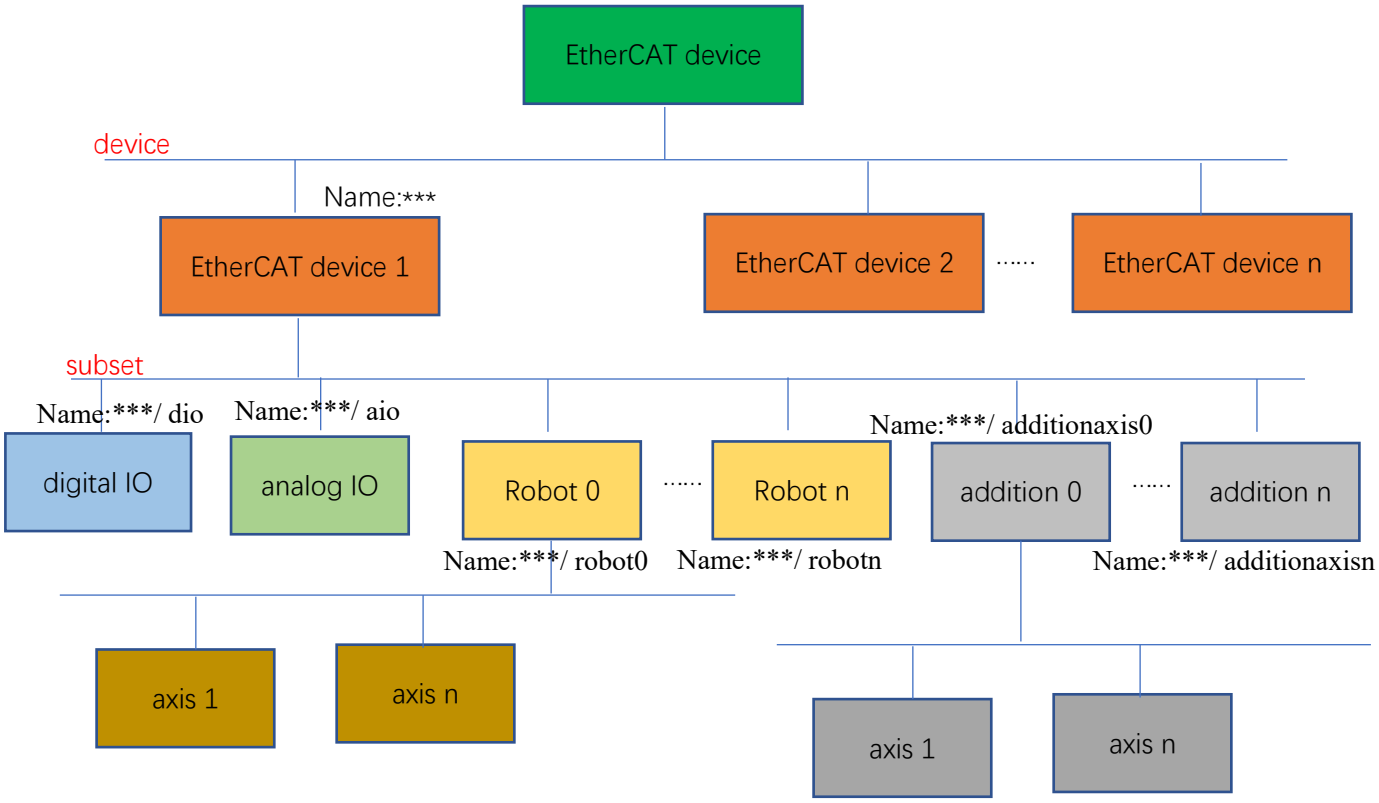
JOG 操作是指在手动模式下通过示教器手动控制机器人的关节或者外部导轨运动。可实现关节空间、世界坐标系、工具坐标系、工件坐标系下的 JOG 操作。

2 机器人系统二次开发接口说明

2.1 EtherCAT 总线通信接口说明

EtherCAT总线接口支持多条总线同时使用，每条总线又支持多种设备，如：数字io,模拟io,附加轴，机器人。可利用一个总线管理带有数字io,模拟io,多组附加轴的双臂机器人系统，甚至有更多机器人的系统。EtherCAT通信接口约100个。

2.1.1设备管理拓扑结构



本系统实现的EtherCAT接口将被管理的EtherCAT设备分为四个层次。并通过字符名称对不同层级的设备进行管理。利用相应接口并指定设备名称来创建一个EtherCAT设备。系统会自动完成对io、机器人等字设备进行分类并命名。从而实现底层对EtherCAT从站设备进行数据操作。

2.1.2 主要接口介绍

接口可分为两类，管理接口和数据操作接口。管理接口用于对 EtherCAT 设备及其子设备进行创建等管理操作。数据操作接口用于对 EtherCAT 设备及其子设备进行数据读写等操作。

用法概述：

利用函数函数 createEC_device("******") 创建一个 EtherCAT 设备，其中"******"为设备名称。创建的 EtherCAT 设备属于全局数据，使用设备名称即可获取设备数据。

可用设备名称操作其中的子设备，如：robot_getposition_c ("****** /robot0", pos)获取子设备"****** /robot0"的编码器数据。"****** /robot0"为 EtherCAT 设备的第一个机器人子设备。子设备的命名规则见上图。

也可使用接口 get_name_robotEC_deviceHandle_c("******", x) 获取第 x+1 个机器人子设备名称，x=0,1,2,3,…

即利用 robot_getposition_c(get_name_robotEC_deviceHandle_c("******", 0), pos)可实现同样的效果。

其他即可使用与 void robot_getposition_c(char* ecHName,int* pos)类似。

主要管理接口：

管理接口主要用于管理EtherCAT设备及其子设备，如：创建、销毁、设备名称获取等。是操作EtherCAT设备及其子设备的必要接口。管理接口可管理多个EtherCAT设备，通过不同的名称即可实现对不同EtherCAT设备的操作。创建的EtherCAT设备全局有效。

名称	输入	输出
createEC_device	EtherCAT设备名称	无
destroyEC_device	EtherCAT设备名称	无
getEC_deviceName	设备索引	EtherCAT设备名称
get_name_robotEC_deviceHandle_c	EtherCAT设备名称及子设备索引	EtherCAT子设备名称

该类接口EtherCAT接口中最主要的接口之一，用于创建EtherCAT设备。创建时需要指定设备名称，用于后期设备操作。接口会自动完成io、机器人等子设备的分类及命名。命名规则如：
EtherCAT设备名/robot0 表示第一个机器人子设备，EtherCAT设备名/dio 表示数字io设备。
同时科里利用destroyEC_device 释放创建的设备资源，需要指定要设备的EtherCAT设备名称。释放后如还需继续使用EtherCAT设备需要重新创建。

设备及其子设备的操作需要通过设备及其子设备的名称来完成，所以提供了设备及其子设备的名称获取接口函数。分别用于获取EtherCAT设备及不同种类的子设备名称，同类一类子设备中可能包括多个子设备。每个子设备名称不同，通过索引来进行区分。

主要数据操作接口：

数据操作接口用于操作EtherCAT设备及其子设备，如：读写操作。是EtherCAT设备数据操作的必要接口，用于获取子设备数据及设置子设备数据。如：机器人上电、下电、编码器获取等。每个子设备是由多个EtherCAT从站组成。操作接口即支持子设备中单个从站设备又支持子设备中所有从站的统一操作。

名称	输入	输出
axis_power_c	EtherCAT子设备名称，从站id	无
robot_power_c	EtherCAT子设备名称	无
axis_getposition_c	EtherCAT子设备名称，从站id	EtherCAT子设备其中一个从站的编码器位置，用于附加轴及机器人子设备
robot_getposition_c	EtherCAT子设备名称及获取数据的地址	EtherCAT子设备中所有从站的编码器位置，用于附加轴及机器人子设备
setdo_c	EtherCAT子设备名称，io 索引及设置值	无
getdi_c	EtherCAT子设备名称，io 索引及获取数据的地址	EtherCAT数字io 子设备的状态数据。

数据操作接口包括附加轴与机器人操作类，数字io与模拟io操作类。其中附加轴与机器人操作使用相同接口，通过名称可区别所要操作的子设备。如：robot_power_c，robot_getposition_c等接口。附加轴与机器人子设备的从站是伺服电机。所以操作包括上电、下电、编码器期望位置的设置与实际位置的获取。数字io与模拟io设备不同于附加轴与机器人操作，有单独的操作接口。如：setdo_c 接口。其中io 操作的索引是从0开始的。

2.2 机器人模型接口

2.2.1 模型接口概述

机器人模型接口包括管理接口、运动学模型接口及动力学模型接口。

正运动学是利用机器人各个关节变量的信息求取机器人末端的位置与姿态。逆运动学则与正运动学相反，利用机器人末端点的位置与姿态求取机器人各个关节变量的值。

D-H 表示法是推导机器人运动学模型的常用方法。D-H 表示法是通过对机器人的连杆及其关节进行建模的一种简单方法，适用于多数机器人的构型，无论机器人具有何种结构顺序及复杂程度如何。多数机器人是由一系列连杆及关节组成的。这些可以是关节滑动的也可以是旋转的。关节和连杆的顺序可以是任意的，也可以处于不同的平面。连杆的长度也可以是任意的，连杆也可以是弯曲。
因此，需要在每个关节处建立一个参考坐标系。然后，确定从一个关节坐标系到下一个关节坐标系的变换。从基座坐标系到关节一坐标系，再从关节一坐标系到关节二坐标系直至到最后一个关节坐标系的所有变换组合起来，就可以得到机器人的总变换矩阵了。该变化矩阵就反应了机器人的正运动学特性。

运动学逆解有多种方法求解。如对偶四元素、反变换法、对偶矩阵、旋量代数、迭代法和几何法等。从运动学方程可以看出机器人的各个关节的角度之间存在较强的耦合关系。因此，无法从变换矩阵中得到足够的元素以求取单个正弦和余弦项来计算关节角度。为了使关节角度解耦，可用单个矩阵左乘逆矩阵，使得方程右边不再包括计算角度。再利用包含所求关节角度的正弦值或余弦值的元素求导关节角度。利用这种方法便可求得运动学的逆解。

采用的正逆运动学模型即支持较为通用的数值解法又支持解析法。并可方便的通过配置文件进行设置。共 9 个模型接口。

2.2.2 主要接口介绍

管理接口主要用于创建及释放模型资源。正运动学模型接口用于由机器人关节角度求取机器人末端位置及姿态。逆运动学模型接口由于由机器人末端位置于姿态求解机器人关节角度。正动力学模型接口由关节驱动力矩求解机器人角度、角速度及角加速度。该接口可作为被控对象的数字模型。逆动力学模型接口由机器人角度、角速度及角加速度求解关节力矩。该接口可用于控制器设计。

用法概述：

利用接口int CreateSerialLink(char* serialLinkName) 创建机器人模型，包括运动学与动力学，其中serialLinkName为机器人名称，利用该名称可以获取机器人数据。利用接口int destroySerialLink(char* serialLinkName)释放数据内存区。

使用接口int Kine_Forward(char* serialLinkName, R7_KINE* r7kine) 可获取机器人对应关节角度的末端位置与姿态。int Kine_Inverse(char* serialLinkName, R7_KINE* r7kine) 作用相反。

数据结构：

```
typedef struct{
    double      R[3][3];//末端姿态矩阵
    double      X[3];//末端位置
    double      joint[10];//关节角度
    double      kps[3];//末端姿态的欧拉角描述
    int         dof;//机器人自由度
    double  redundancy;//冗余角
    TOOL    tool;//工具坐标系
    WOBJ    wobj;//工件坐标系
}R7_KINE;
```

该数据结构用于存放运动学模型的输入与输出数据。

```
typedef struct {
```

```
    double q[R_DYNAMICS_MEMBER_NUM];
    double qd[R_DYNAMICS_MEMBER_NUM];
    double qdd[R_DYNAMICS_MEMBER_NUM];
    double fext[6];
    int fcv_flag;
    int g_flag;
    double torque[R_DYNAMICS_MEMBER_NUM];
    double torque_1[R_DYNAMICS_MEMBER_NUM];
    int dof;
}R_DYNAMICS;
```

该数据结构用于存放动力学模型的输入与输出数据。

```
typedef struct {
    double q[R_DYNAMICS_MEMBER_NUM];
    Dyn_inertia inertia;
    int dof;
}R_DYNAMICS_INERTIA;
```

该数据结构用于存放动力学模型的惯性矩阵。

```
typedef struct {
    double q[R_DYNAMICS_MEMBER_NUM];
    double qd[R_DYNAMICS_MEMBER_NUM];
    Dyn_coriolis coriolis;
    int dof;
}R_DYNAMICS_CORIOLIS;
```

该数据结构用于存放动力学的奥利科里矩阵。

```
typedef struct {
    double q[R_DYNAMICS_MEMBER_NUM];
    Dyn_gravity gravity;
    int dof;
}R_DYNAMICS_GRAVITY;
```

该数据结构用于存放动力学模型的重力矩。

```
typedef struct {
    double qd[R_DYNAMICS_MEMBER_NUM];
    Dyn_friction friction;
    int dof;
}R_DYNAMICS_FRICTION;
```

该数据结构用于存放动力学模型的摩擦力。

```
typedef struct {
    double q[R_DYNAMICS_MEMBER_NUM];
    double fext[6];
    int dof;
    Dyn_externalTorque externalTorque;
}R_DYNAMICS_EXTERNALTORQUE;
```

该数据结构用于存放动力模型的外部附加力矩。

主要管理接口：

管理接口主要用于管理模型数据，如：创建、销毁等，是模型操作的必要接口。

名称	输入	输出
CreateSerialLink	模型名称	无
destroySerialLink	模型名称	无

为了模型的通用和使用方便，模型数据是以配置文件的形式给出的。管理接口主要完成配置文件数据的解析和模型创建工作。即利用配置文件中配置的模型数据创建模型，可用于运动学与动力学模型求解。管理接口可管理多个模型数据，通过不同的名称即可实现对不同模型的操作。创建的模型数据全局有效。

运动学模型接口：

运动学模型接口包括正运动学接口和逆运动学接口。分别用于机器人正运动学和逆运动学的求解。

名称	输入	输出
Kine_Forward	模型名称及输入输出数据地址	机器人末端位置与姿态
Kine_Inverse	模型名称及输入输出数据地址	机器人关节角度

用于完成机器人正运动学和逆运动学的求解。完成关节角度与末端位置和姿态的相互转换。模型可以再用通用的数据解法或解析解法。可以通过配置文件进行设置。无论那种解法，也可用法一致。机器人的运动学模型在机器人运动控制中起这至关重要的作用。对于不同的机器人模型可以通过机器人名称进行区别。

动力学模型接口：

动力学模型接口包括正动力学接口和逆动力学接口。分别用于机器人正动力学和逆动力学的求解。正动力学模型可用作被控对象的数字模型，可以用于算法的仿真验证。而逆动力学模型一般可用于控制算法中力矩的前馈。在基于模型的控制算法中一般需要获取机器人的惯性矩阵、奥利科利矩阵，重力矩、摩擦力矩、外部力矩。所以动力学接口中还包括这部分数据的获取。

名称	输入	输出
Dyn_Inverse	模型名称及输入输出数据地址	关节力矩
Dyn_Forward	模型名称及输入输出数据地址	关节角度、角速度、角加速
Dyn_Inertia	模型名称及输入输出数据地址	惯性矩阵
Dyn_Coriolis	模型名称及输入输出数据地址	奥利科利矩阵
Dyn_Gravity	模型名称及输入输出数据地址	重力矩
Dyn_Friction	模型名称及输入输出数据地址	摩擦力
Dyn_ExternalTorque	模型名称及输入输出数据地址	外部力矩

在机器人控制过程中机器人动力学模型是必不可稍等。正动力学模型常用做为算法仿真的被控对象。逆动力学可用于算法的前馈控制，对于提升控制效果较为显著。为了控制算法的设计，动力学模型可写为：

$$\tau = M(q)\ddot{q} + C(q,\dot{q})\dot{q} + D(q) + F_{cv} + F_n$$

的形式。其中：$M(q)$为惯性矩阵；$C(q,\dot{q})$为奥利科利矩阵；$D(q)$为重力矩；F_{cv}为摩擦力；F_n为外部力矩。所以动力学接口中提供力上述相关数据的获取接口，便于基于模型的控制算法实现。

2.3 机器人基础规划插补接口

由于机器人的驱动装置提供的功率仅能保证关节的速度、加/减速度在一定的范围。因此，机器人

某个关节从一个位置到另一个位置并不是简单的给定目标为位置就可以的，而是需要在当前位置与目标位置之间分割若干小段，以保证机器人关节的运动速度、加/减速度不超过最大限制。
机器人的规划一般分为笛卡尔空间规划和关节空间规划。这里主要主要介绍机器人关节空间的规划。关节空间规划方法可以获得各个中间点的期望位姿。尽管各个中间点之间的路径在关节空间综的描述非常简单，但在笛达尔坐标空间中的描述却很复杂。一般情况下，关节空间的规划方法便于计算，并且由于关节空间与笛达尔坐标空间之间并不存在连续的对应关系，因而不会发生机构的奇异性问题。机器人关节由起点到终点，要经历加速、匀速、减速的过程。把整个过程中的速度随时间的变化关系画出来就是速度曲线或者速度轮廓。常见的速度曲线包括：梯形规划(Trapezoidal Profile)、S 型规划(S-Curve/Profile)、多项式规划(Polynomial Profile)。

2.3.1 插补接口概述

该部分接口能够实现基本的规划与插补。利用接口 int InterpolateBase_project(Inter_in* inter_in,Inter_mid* inter_mid) 可实现 S 型速度曲线的规划。周期调用接口 int InterpolateBase_computer(Inter_in* inter_in,Inter_mid* inter_mid,Inter_out* inter_out) 来获取每个规划时刻的规划位置、速度、加速度、加加速度。接口 int InterpolateBase_computer_online(Inter_in* inter_in,Inter_mid* inter_mid,Inter_out* inter_out, Inter_state* inter_state) 与 int InterpolateBase_computer(Inter_in* inter_in,Inter_mid* inter_mid,Inter_out* inter_out) 类似，但多了在线停止功能。

2.3.2 主要接口介绍

该接口函数为关节空间多轴同步插补及笛卡尔空间直线、圆弧等规划的基础接口函数。决定这规划，插补的优劣。

用法概述：

先利用 int InterpolateBase_project(Inter_in* inter_in,Inter_mid* inter_mid) 进行规划，再周期调用 int InterpolateBase_computer_online(Inter_in* inter_in,Inter_mid* inter_mid,Inter_out* inter_out, Inter_state* inter_state) 与 int InterpolateBase_computer(Inter_in* inter_in,Inter_mid* inter_mid,Inter_out* inter_out)获取规划结果。

数据结构：

```
typedef struct{
    double dt;//采样周期(s)
    double S;//位移(mm)或(rad)
    double vs;//起始速度(mm/s)或(rad/s)
    double ve;//终止速度(mm/s)或(rad/s)
    double as;//起始加速度(mm/s^2)或(rad/s^2)
    double ae;//终止加速度(mm/s^2)或(rad/s^2)
    double Vmax;//最大速度(mm/s)或(rad/s)
    double Amax;//最大加速度(mm/s^2)或(rad/s^2)
    double Jmax;//最大加加速度(mm/s^3)或(rad/s^3)
}Inter_in;
```

插补接口输入数据，用于存放存放规划输入数据

```
typedef struct{
    double T;//总规划时间(s)
    double Taa;//加加速时间(s)
    double Tca;//匀加速时间(s)
    double Tda;//减加速时间(s)
    double Tv;//匀速时间(s)
    double Tad;//加减速速时间(s)
    double Tcd;//匀减速时间(s)
```

```
    double Tdd;//减减速时间(s)
    double ac_Vmax;//实际最大速度(mm/s)或(rad/s)
    double ac_Amaxa;//实际最大加速度(mm/s^2)或(rad/s^2)
    double ac_Amaxd;//实际最大减速度(mm/s^2)或(rad/s^2)
    double ac_Jmax;//实际最大加加速度(mm/s^3)或(rad/s^3)
}Inter_mid;
```

插补规划的输出，用于存放规划结果。

```
typedef struct{
    double t;//插补时间(s)
    double s;//插补位置(mm)或(rad)
    double v;//插补速度(mm/s)或(rad/s)
    double a;//插补加速度(mm/s^2)或(rad/s^2)
    double j;//插补加加速度(mm/s^3)或(rad/s^3)
}Inter_out;
```

插补的输出数据，存放每一时刻的规划结果。

```
typedef struct{
     int stop;//停止命令   0:停止；1：运行
     int status;//停止状态  0：停止；1：运动
     int flag;//停止命令初始化标志 0：已初始化；1:未初始化
     double _t;//切换停止状态时到中间变量
     double _s;//切换停止状态时到中间变量
}Inter_state;
```

插补的状态数据

规划接口主要用于实现注入S型、梯形速度曲线，保证在机器人允许的能力范围内进行平滑的运动。是关节空间多轴及笛卡尔空间直线、圆弧等运动的基础。

名称	输入	输出
InterpolateBase_project	规划数据	规划结果
InterpolateBase_computer	规划结果	插补结果
InterpolateBase_computer_online	规划结果及运动状态设置标志	插补结果

上述接口主要是运动曲线规划及规划数据的获取，分别调用InterpolateBase_project 和InterpolateBase_computer实现，且在InterpolateBase_computer 之前必须使用InterpolateBase_project进行规划。InterpolateBase_computer_online与InterpolateBase_computer类似，但具有在线停止功能，保证运动曲线的平滑，没有位置、速度及加速度突变。接口InterpolateBase_project仅需调用一次即可完成规划，而接口InterpolateBase_computer和InterpolateBase_computer_online需要被循环调用来获取每一时刻的插补结果。关节空间的多轴及笛卡尔空间直线、圆弧等运动都是建立在上述接口的基础上完成的。

2.4 机器人运动控制接口

下列接口用于实现机器人运动控制，如void moveL(robpose* rpose, speed* rspeed, zone* rzone, tool* rtool, wobj* rwobj)实现笛卡尔空间的直线运动。支持双臂运动控制，及多个机器人系统中某个机器人的运动控制。共18个运动控制接口。

2.4.1 运动接口概述

运动接口包括单臂接口，双臂接口及多机器人系统中某个臂运动接口。运动形式包括：关节空

间多轴同步运动，笛卡尔空间的直线、圆弧等运动。该不部分接口可快速实现机器人应用。通过多种运动形式的有效组合可实现复杂的运动轨迹。

2.4.1 主要运动接口介绍

运动接口的输入相对复杂，因此提过了相关的数据结构来设置运动接口的输入数据。

数据结构：

```
typedef struct robjoint{
    double angle[10];
}robjoint;
```

目标关节角度

```
typedef struct robpose{
    double xyz[3];
    double kps[3];
}robpose;
```

目标位置与姿态

```
typedef struct speed{
    double per[10];
    int per_flag;
    double tcp;
    int tcp_flag;
    double orl;
    int orl_flag;
}speed;
```

运动速度

```
typedef struct zone{
    int zone_flag;//0:不采用转弯区平滑过度，1：百分比，2：距离或圆周角
    double zone_size;
}zone;
```

转弯区

```
typedef struct tool{
    int robhold;
    robpose tframe;
}tool;
```

工具坐标系

```
typedef struct wobj{
    int robhold;
    int ufprog;
    int ufmec;
    robpose uframe;
    robpose oframe;
}wobj;
```

工件坐标系

单臂运动接口：

单臂接口用于单臂机器人系统，给定运动目标机器人其相应参数即可实现机器人的运动控制。通过不同接口直接的组合可实现机器人复杂运动。

名称	输入	输出
moveA	关节期望位置，运动速度，转弯区，使用的工件，采用的工件坐标系	无
moveJ	末端期望位置与姿态，运动速度，转弯区，使用的工件，采用的工件坐标系	无
moveL	末端期望位置与姿态，运动速度，转弯区，使用的工件，采用的工件坐标系	无
moveC	末端期望位置与姿态，运动速度，转弯区，使用的工件，采用的工件坐标系	无
moveAJBS	关节或末端期望位置与姿态必经数据，运动速度，使用的工件，采用的工件坐标系	无

上述接口之间可以通过转弯区参数进行关联，转弯区在当前指令与下一条指令之间起作用。保证机器人高效的运动。上述接口种即包括关机空间运动又包括笛卡尔空间运动。二至之间的配合使用可克服关节空间末端轨迹不可控和笛卡尔空间奇异问题。为了能够实现更为复杂的运动形式，还提供了过多个必经点的B样条运动。

双臂运动接口：

双臂接口用于双臂机器人系统，给定运动目标机器人其相应参数即可实现双臂机器人的运动控制。通过不同接口直接的组合可实现机器人复杂运动。

名称	输入	输出
dual_moveA	关节期望位置，运动速度，转弯区，使用的工件，采用的工件坐标系	无
dual_moveJ	末端期望位置与姿态，运动速度，转弯区，使用的工件，采用的工件坐标系	无
dual_moveL	末端期望位置与姿态，运动速度，转弯区，使用的工件，采用的工件坐标系	无
dual_moveC	末端期望位置与姿态，运动速度，转弯区，使用的工件，采用的工件坐标系	无
dual_moveAJBS	关节或末端期望位置与姿态必经数据，运动速度，使用的工件，采用的工件坐标系	无

双臂接口可实现双臂的同时运动控制，允许双臂构型不同。运动形式与单臂接口类似，可完成双臂之间的协同工作。

多机器人系统运动接口：

在多机器人系统中为了实现单独某个机器人的运动控，给供了相应的操作接口。可以通过索引实现其中一个机器人的运动控制。

名称	输入	输出
multi_moveA	关节期望位置，运动速度，转弯区，使用的工件，采用的工件坐标系，机器人索引	无
multi_moveJ	末端期望位置与姿态，运动速度，转弯区，使用的工件，采用的工件坐标系，机器人索引	无
multi_moveL	末端期望位置与姿态，运动速度，转弯区，使用的工件，采用的工件坐标系，机器人索引	无
multi_moveC	末端期望位置与姿态，运动速度，转弯区，使用的工件，采用的工件坐标系，机器人索引	无
multi_moveAJBS	关节或末端期望位置与姿态必经数据，运动速度，使用的工件，采用的工件坐标系，机器人索引	无

与单臂机器人类似，但可以应用于多机器人系统，通过索引来区分所要控制的机器人。

2.5 机器人通信接口

在机器人系统中常常需要机器人与相机等外部设备进行通信，完成诸如视觉抓起等工作。简化的通信接口有助于一个用的快速搭建。共34个通信几口

2.5.1 通信接口概述

通信接口分为两类：tcp 通信接口和 udp 接口。

TCP 的优点： 可靠，稳定 TCP 的可靠体现在 TCP 在传递数据之前，会有三次握手来建立连接，而且在数据传递时，有确认、窗口、重传、拥塞控制机制，在数据传完后，还会断开连接用来节约系统资源。 TCP 的缺点： 慢，效率低，占用系统资源高，易被攻击 TCP 在传递数据之前，要先建连接，这会消耗时间，而且在数据传递时，确认机制、重传机制、拥塞控制机制等都会消耗大量的时间，而且要在每台设备上维护所有的传输连接，事实上，每个连接都会占用系统的 CPU、内存等硬件资源。而且，因为 TCP 有确认机制、三次握手机制，这些也导致 TCP 容易被人利用，实现 DOS、DDOS、CC 等攻击。

UDP 的优点： 快，比 TCP 稍安全 UDP 没有 TCP 的握手、确认、窗口、重传、拥塞控制等机制，UDP 是一个无状态的传输协议，所以它在传递数据时非常快。没有 TCP 的这些机制，UDP 较 TCP 被攻击者利用的漏洞就要少一些。但 UDP 也是无法避免攻击的，比如：UDP Flood 攻击…… UDP 的缺点： 不可靠，不稳定 因为 UDP 没有 TCP 那些可靠的机制，在数据传递时，如果网络质量不好，就会很容易丢包。 基于上面的优缺点，那么： 什么时候应该使用 TCP： 当对网络通讯质量有要求的时候，比如：整个数据要准确无误的传递给对方，这往往用于一些要求可靠的应用，比如 HTTP、HTTPS、FTP 等传输文件的协议，POP、SMTP 等邮件传输的协议。

本系统同时支持 tcp 通信和 udp 通信，用户可针对不同需求进行选择使用。在接口的使用上二者类似。操作简单方便。

2.5.2 主要通信接口介绍

首先利用接口 int SocketCreate(char* ip, int port, char* sName) 或 int ClientCreate(char* ip, int port, char* sName)建立 tcp 服务端或客户端并建立连接。再利用 int SocketSendString(char* data, char* sName)等接口实现数据的收发。udp 通信类似。利用接口 int UDPServerCreate(char* ip, int port, char* sName)或 int UDPClientCreate(char* ip, int port, char* sName)建立 udp 服务端或客户端。再利用 int UDPSendString(char* data, char* sName)等接口实现数据的收发。

tcp 通信接口：

tcp 通信接口包括服务端与客户端的创建及数据传输接口。该接口支持多个服务端和客户端同时存在。通过创建时使用的名字对对不同的服务端和客户端进行区分。省去了传统 tcp 繁琐的设置操作，大大简化了通信的设置工作。

名称	输入	输出
SocketCreate	创建服务端的ip、端口号及名字	无
ClientCreate	创建客户端的ip、端口号及名字	无
SocketClose	服务端或客户端的名字	无
SocketSendByte	服务端或客户端的名字，发送的数据	无
SocketRecvByte	服务端或客户端的名字	接收数据
SocketSendString	服务端或客户端的名字，发送的数据	无
SocketRecvString	服务端或客户端的名字	接收数据
SocketSendDouble	服务端或客户端的名字，发送的数据	无
SocketRecvDouble	服务端或客户端的名字	接收数据
SocketSendByteArray	服务端或客户端的名字，发送的数据	无
SocketRecvByteArray	服务端或客户端的名字	接收数据
SocketSendIntArray	服务端或客户端的名字，发送的数据	无
SocketRecvIntArray	服务端或客户端的名字	接收数据

上述通信接口支持多种数据类型的数据通信，同时支持数组数据的传输。数组数据在机器人通信中是经常被使用的。接口的使用方便简洁。

udp 通信接口：

与 tcp 通信接口类似包括服务端与客户端的创建及数据传输接口。该接口支持多个服务端和客户端同时存在。通过创建时使用的名字对对不同的服务端和客户端进行区分。省去了传统 udp 繁琐的设置操作，大大简化了通信的设置工作。

名称	输入	输出
UDPServerCreate	创建服务端的ip、端口号及名字	无
UDPClientCreate	创建客户端的ip、端口号及名字	无
UDPClose	服务端或客户端的名字	无
UDPSendByte	服务端或客户端的名字，发送的数据	无
UDPRecvByte	服务端或客户端的名字	接收数据
UDPSendString	服务端或客户端的名字，发送的数据	无
UDPRecvString	服务端或客户端的名字	接收数据
UDPSendDouble	服务端或客户端的名字，发送的数据	无
UDPRecvDouble	服务端或客户端的名字	接收数据
UDPSendByteArray	服务端或客户端的名字，发送的数据	无
UDPRecvByteArray	服务端或客户端的名字	接收数据
UDPSendIntArray	服务端或客户端的名字，发送的数据	无
UDPRecvIntArray	服务端或客户端的名字	接收数据

上述通信接口支持多种数据类型的数据通信，同时支持数组数据的传输。udp 通信接口扩展了机器人的通信能力。

3 机器人系统参数配置

本机器人系统采用通用模型机制，通过修改配置文件可实现，单臂、双臂等多机器人系统的快速配置。以及不同厂家机器人模型的快速配置。配置文件包括总线配置文件、机器人模型配置文件、附加轴配置文件。其中总线配置文件用于配置机器人系统，如包括附加轴、io、及多机器人的系统。模型配置文件用于配置系统中机器人的模型数据。附加轴配置文件用于配置系统中附加轴数据。

3.1 总线配置

①数据名称配置
总线周期：　　（ns）
状态字（state）：
控制字（control）：
模式字（mode）：

实际位置（ac_position）：
实际电流（ac_current）：
实际力矩（ac_torque）：
实际速度（ac_velocity）：

期望力矩（torque）：
期望位置（position）：
期望速度（velocity）：

电机实际位置（ac_motor_position）：
电机实际速度（ac_motor_velocity）：
力矩传感器（ac_sensor_torque）：

数字输入 Di（ec_di）：
数字输出 Do (ec_do)：
模拟输入 Ai (ec_ai)：
模拟输出 Ao (ec_ao)：
数字逻辑输入 Di (ec_li)：
数字逻辑输出 Do (ec_lo)：

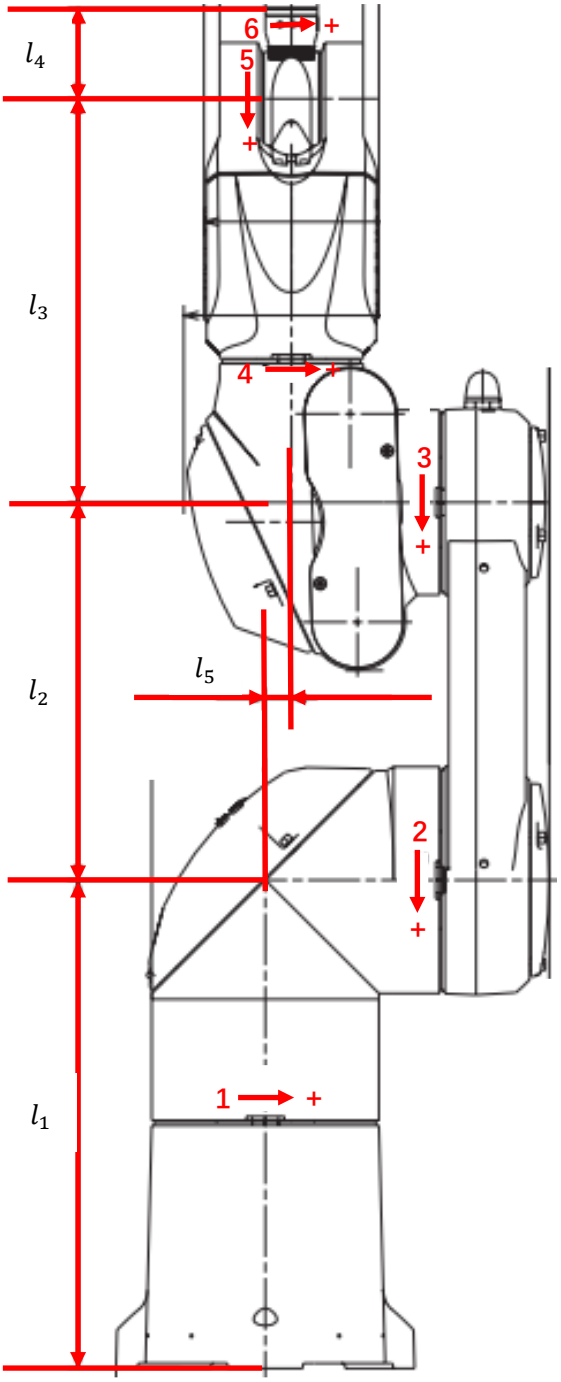
②附加轴配置
附加轴组数：
第一组附加轴轴数目：
第一组附加轴轴索引：
1：
2：
……
（个数由第一组附加轴数目数目决定）

第二组附加轴轴数目：
第二组附加轴轴索引：
1：
2：
……
（个数由第二组附加轴数目数目决定）
……
（由附加轴组数决定）
③机器人配置
机器人数：
第一个机器人轴数目：
第一个机器人轴索引：
1：
2：
……
（个数由第一个机器人轴数目决定）
第二个机器人轴数目：
第二个机器人轴索引：
1：
2：
……
（个数由第二个机器人轴数目决定）
……

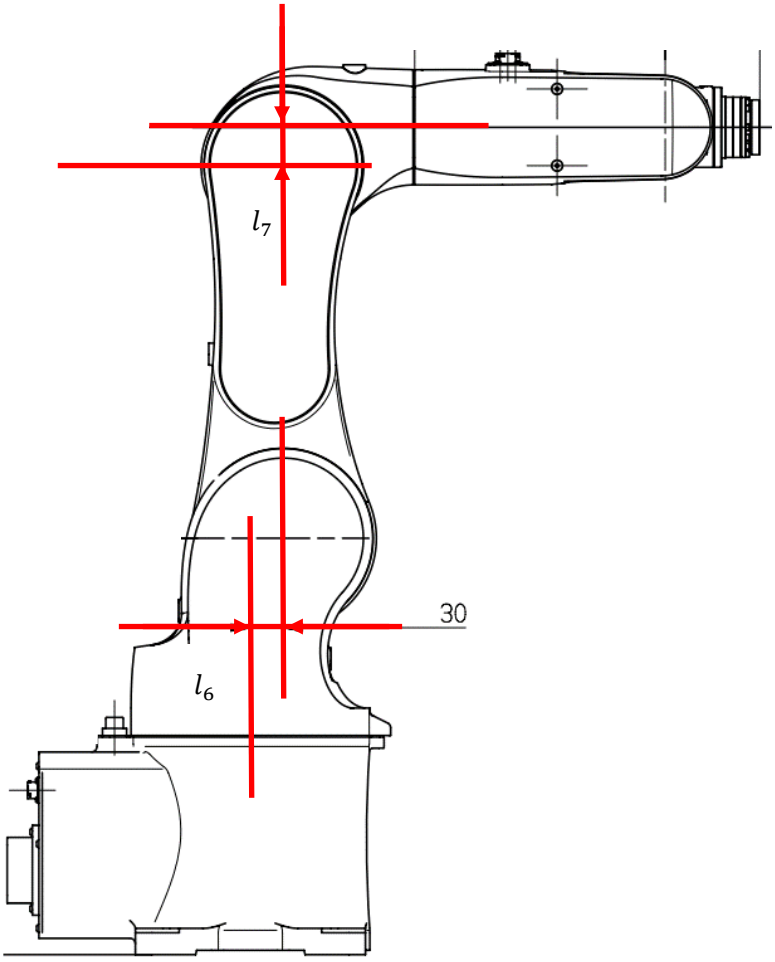
（由机器人数决定）

3.2 模型配置

①机器人运动学模型（六自由度）



前侧



右侧 （默认零位）

杆长：(mm)

l_1：

l_2：

l_3：

l_4：

l_5：

l_6：

l_7：

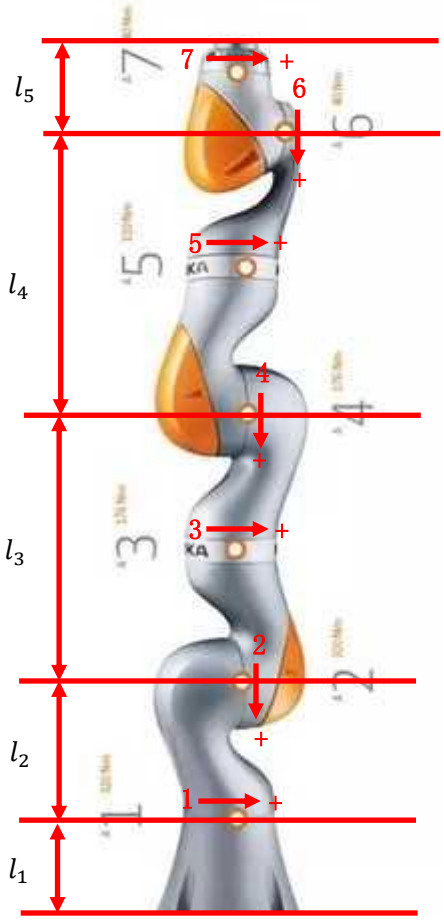
零位：（实际零位与默认零位夹角，度）

$offset_1$：

$offset_2$：
$offset_3$：
$offset_4$：
$offset_5$：
$offset_6$：
转向：(实际转向是否与标注一致，0：不一致；1：一致)
$dire_1$：　　0 or 1
$dire_2$：　　0 or 1
$dire_3$：　　0 or 1
$dire_4$：　　0 or 1
$dire_5$：　　0 or 1
$dire_6$：　　0 or 1
关节位置限制：（度）
q_1：　　—
q_2：　　—
q_3：　　—
q_4：　　—
q_5：　　—
q_6：　　—
关节速度限制：（度/s）
v_1：　　—
v_2：　　—
v_3：　　—
v_4：　　—
v_5：　　—
v_6：　　—
关节加速度限制：（度/s^2）
a_1：　　—
a_2：　　—
a_3：　　—
a_4：　　—
a_5：　　—
a_6：　　—
关节加加速度限制：（度/s^3）
j_1：　　—
j_2：　　—
j_3：　　—
j_4：　　—
j_5：　　—
j_6：　　—
单圈编码值：
1：
2：
3：
4：
5：
6：
减速比：
1：

2：
3：
4：
5：
6：
最大允许速度：
平动：
转动：
最大允许加速度：
平动：
转动：
最大允许加加速度：
平动：
转动：

②机器人运动学模型（七自由度）



前侧 （默认零位）

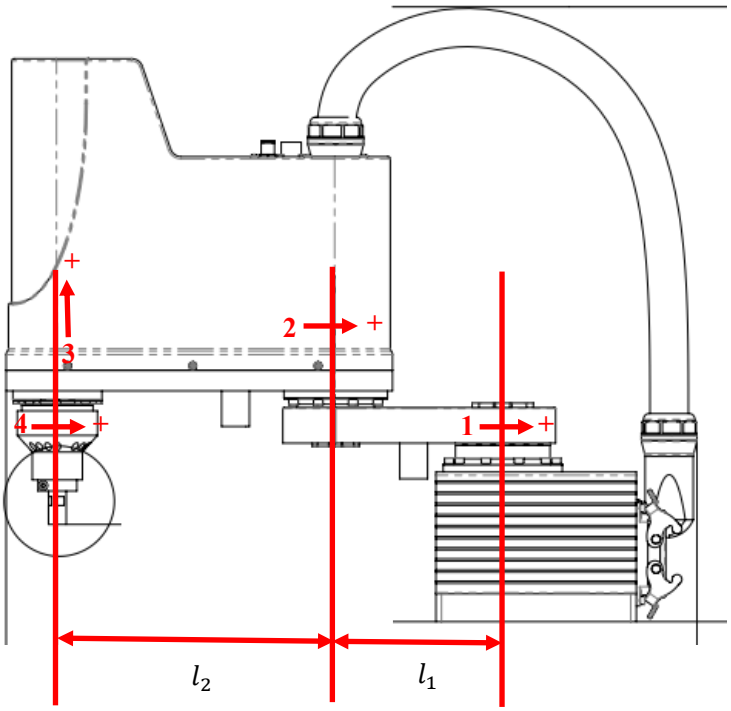
杆长：(mm)
l_1：
l_2：
l_3：
l_4：

l_5：
零位：（实际零位与默认零位夹角，rad）
$offset_1$：
$offset_2$：
$offset_3$：
$offset_4$：
$offset_5$：
$offset_6$：
$offset_7$：

转向：(实际转向是否与标注一致，0：不一致；1：一致)
$dire_1$：　　0 or 1
$dire_2$：　　0 or 1
$dire_3$：　　0 or 1
$dire_4$：　　0 or 1
$dire_5$：　　0 or 1
$dire_6$：　　0 or 1
$dire_7$：　　0 or 1
关节位置限制：（度）
q_1：　　—
q_2：　　—
q_3：　　—
q_4：　　—
q_5：　　—
q_6：　　—
q_7：　　—
关节速度限制：（度/s）
v_1：　　—
v_2：　　—
v_3：　　—
v_4：　　—
v_5：　　—
v_6：　　—
v_7：　　—
关节加速度限制：（度/s^2）
a_1：　　—
a_2：　　—
a_3：　　—
a_4：　　—
a_5：　　—
a_6：　　—
a_7：　　—
关节加加速度限制：（度/s^3）
j_1：　　—
j_2：　　—
j_3：　　—
j_4：　　—
j_5：　　—
j_6：　　—

j_7：　　—
单圈编码值：
1：
2：
3：
4：
5：
6：
7：
减速比：
1：
2：
3：
4：
5：
6：
7：
最大允许速度：
平动：
转动：
最大允许加速度：
平动：
转动：
最大允许加加速度：
平动：
转动：

③机器人运动学模型（四自由度）



左侧（默认零位）

杆长：(mm)

l_1：

l_2：

零位：（实际零位与默认零位夹角，rad）

$offset_1$：

$offset_2$：

$offset_3$：　　（mm）

$offset_4$：

转向：(实际转向是否与标注一致，0：不一致；1：一致)

$dire_1$：　0 or 1

$dire_2$：　0 or 1

$dire_3$：　0 or 1

$dire_4$：　0 or 1

关节位置限制：（度，mm）

q_1：　—

q_2：　—

q_3：　—

q_4：　—

关节速度限制：（度/s, mm/s）

v_1：　—

v_2：　—

v_3：　—

v_4：　—

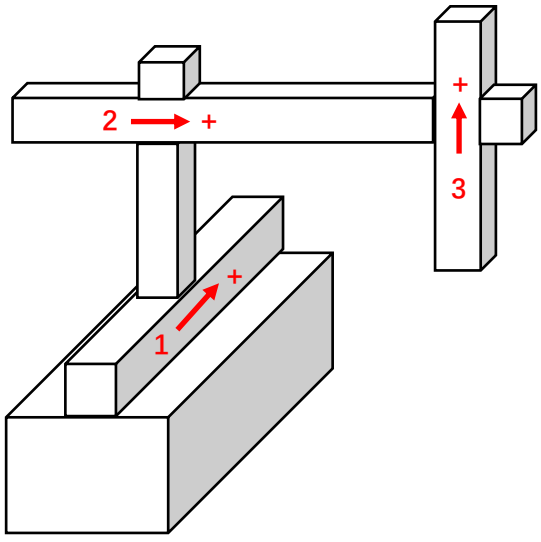
关节加速度限制：（度/s^2, mm/s^2）

a_1：　—

a_2：　—

a_3：　　—
a_4：　　—
关节加加速度限制：（度/s^3, mm/s^3）
j_1：　　—
j_2：　　—
j_3：　　—
j_4：　　—
单圈编码值：
1：
2：
3：
4：
减速比：
1：
2：
3：
4：
最大允许速度：
平动：
转动：
最大允许加速度：
平动：
转动：
最大允许加加速度：
平动：
转动：

④机器人运动学模型（三自由度）



转向：(实际转向是否与标注一致，0：不一致；1：一致)

$dire_1$：　　0 or 1
$dire_2$：　　0 or 1
$dire_3$：　　0 or 1
关节位置限制：（mm）
q_1：　　—
q_2：　　—
q_3：　　—
关节速度限制：（mm/s）
v_1：　　—
v_2：　　—
v_3：　　—
关节加速度限制：（mm/s^2）
a_1：　　—
a_2：　　—
a_3：　　—
关节加加速度限制：（mm/s^3）
j_1：　　—
j_2：　　—
j_3：　　—
单圈编码值：
1：
2：
3：
减速比：
1：
2：
3：
最大允许速度：
平动：
转动：
最大允许加速度：
平动：
转动：
最大允许加加速度：
平动：
转动：

2、附加轴配置
附加轴轴数：
关节位置限制：（mm）
q_1：　　—
q_2：　　—
q_3：　　—
关节速度限制：（mm/s）
v_1：　　—
v_2：　　—
v_3：　　—
关节加速度限制：（mm/s^2）

a_1：　　—
a_2：　　—
a_3：　　—
关节加加速度限制：（mm/s^3）
j_1：　　—
j_2：　　—
j_3：　　—
单圈编码值：
1：
2：
3：
减速比：
1：
2：
3：
转向：
1：　　0 or 1
2：　　0 or 1
3：　　0 or 1
转动或滑动：
1：　　0 or 1
2：　　0 or 1
3：　　0 or 1
(个数由附加轴轴数决定)

3.3 附加轴配置

附加轴轴数：
关节位置限制：（mm）
q_1：　　—
q_2：　　—
q_3：　　—
关节速度限制：（mm/s）
v_1：　　—
v_2：　　—
v_3：　　—
关节加速度限制：（mm/s^2）
a_1：　　—
a_2：　　—
a_3：　　—
关节加加速度限制：（mm/s^3）
j_1：　　—
j_2：　　—
j_3：　　—
单圈编码值：
1：
2：
3：
减速比：

1：
2：
3：
转向：
1：　　0 or 1
2：　　0 or 1
3：　　0 or 1
转动或滑动：
1：　　0 or 1
2：　　0 or 1
3：　　0 or 1
(个数由附加轴轴数决定)