目录

一、 灵巧手组成

UB 手（UBH）是一种拟人机器人手，他得益于一个可以多自由度运动的灵巧手，他的控制方式和成人的手运动学控制相似，他能够模仿人类丰富地操纵能力。UB 手（UBH）被设计成一个灵活的工业机器人末端执行器，它设想为初级操作员能够处理简单的工具和对象。

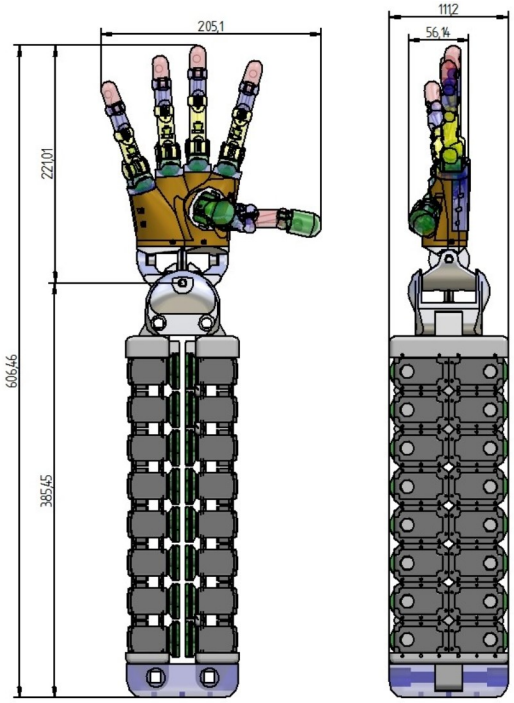
UB 手（UBH）设有五个手指和手腕。每个手指由四个关节和三个旋转关节组成，拇指有五个关节和四个旋转关节。其中两个是独立可控的。其他四个手指采用相同的设计及运动学，在连接部分采取了不同方式为了增加拟人化和敏捷性。拇指有三自由度但不同运动与上手指，提高对抗。手腕有两个独立的旋转关节，可以独立地控制。一共有 17 个控制驱动。

设计是模块化的，因此要求任何变化的原始版本是可能的，包括改变尺寸、数量的手指，自由度独立控制，手腕的存在，最大的力量在指尖，右手或左手。

人造前臂由伺服电机控制。手指需要两电机独立控制的自由度，而手腕需要二自由度，由两电机驱动。

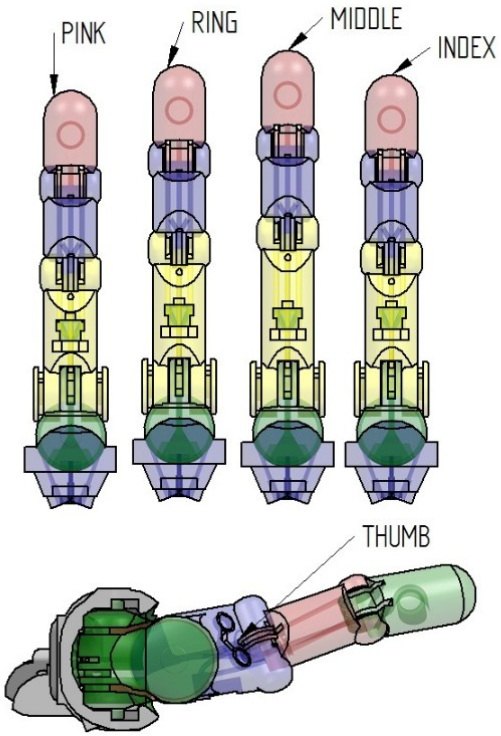
UBH 可以看作是一个完全集成的系统由于驱动、传感和控制系统是嵌入在手和前臂。它提供有线和无线接口来与外部设备通信。

UB 手（UBH）被设计为具有相似的形状和大小的一个典型的对成年男性的手。下图显示了设备的正面和侧面轮廓。



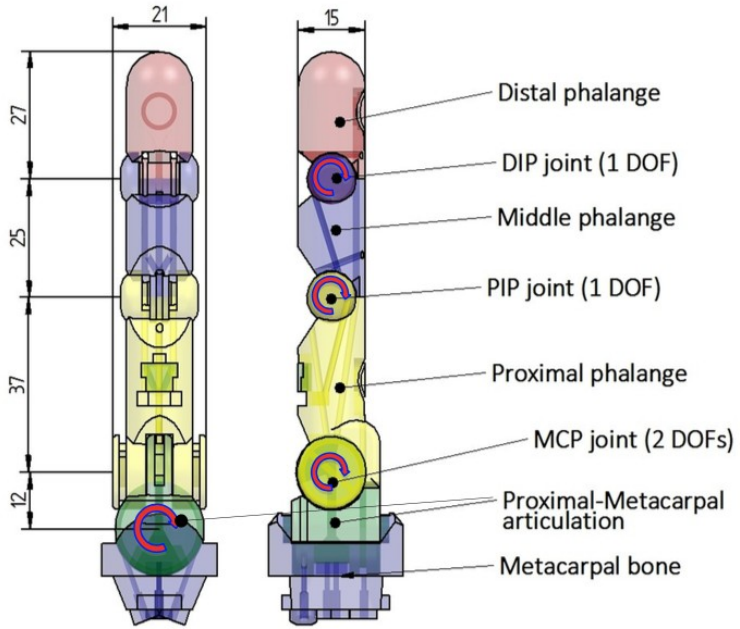
机器人手可以分为三个主要子组件：

- 手由五个手指和手掌；
- 手腕与手掌的连接，中间的关节和前臂的连接
- 前臂的伺服电机；



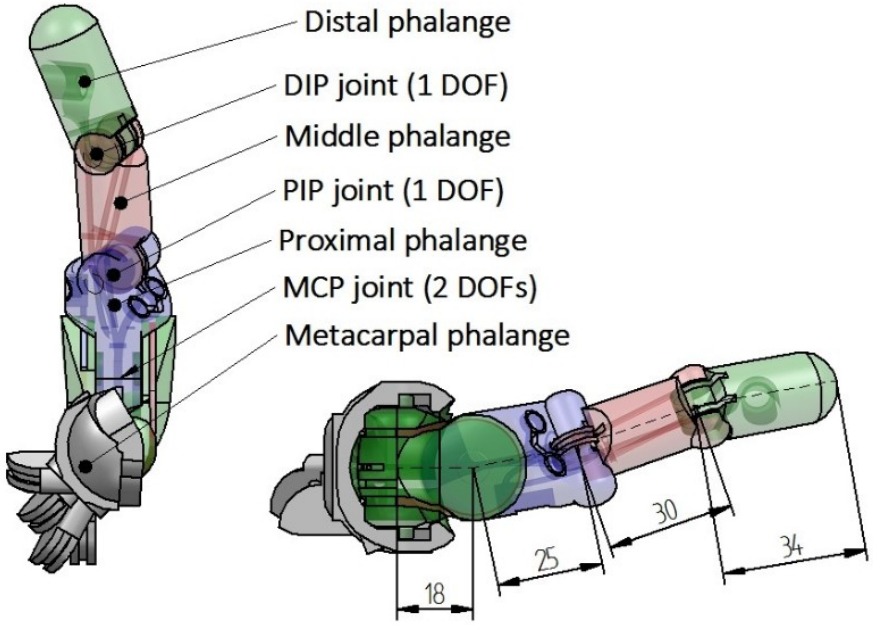
手指由四个环节组成：

- 掌骨；
- 掌近关节；
- 近节指骨；
- 远端指骨。



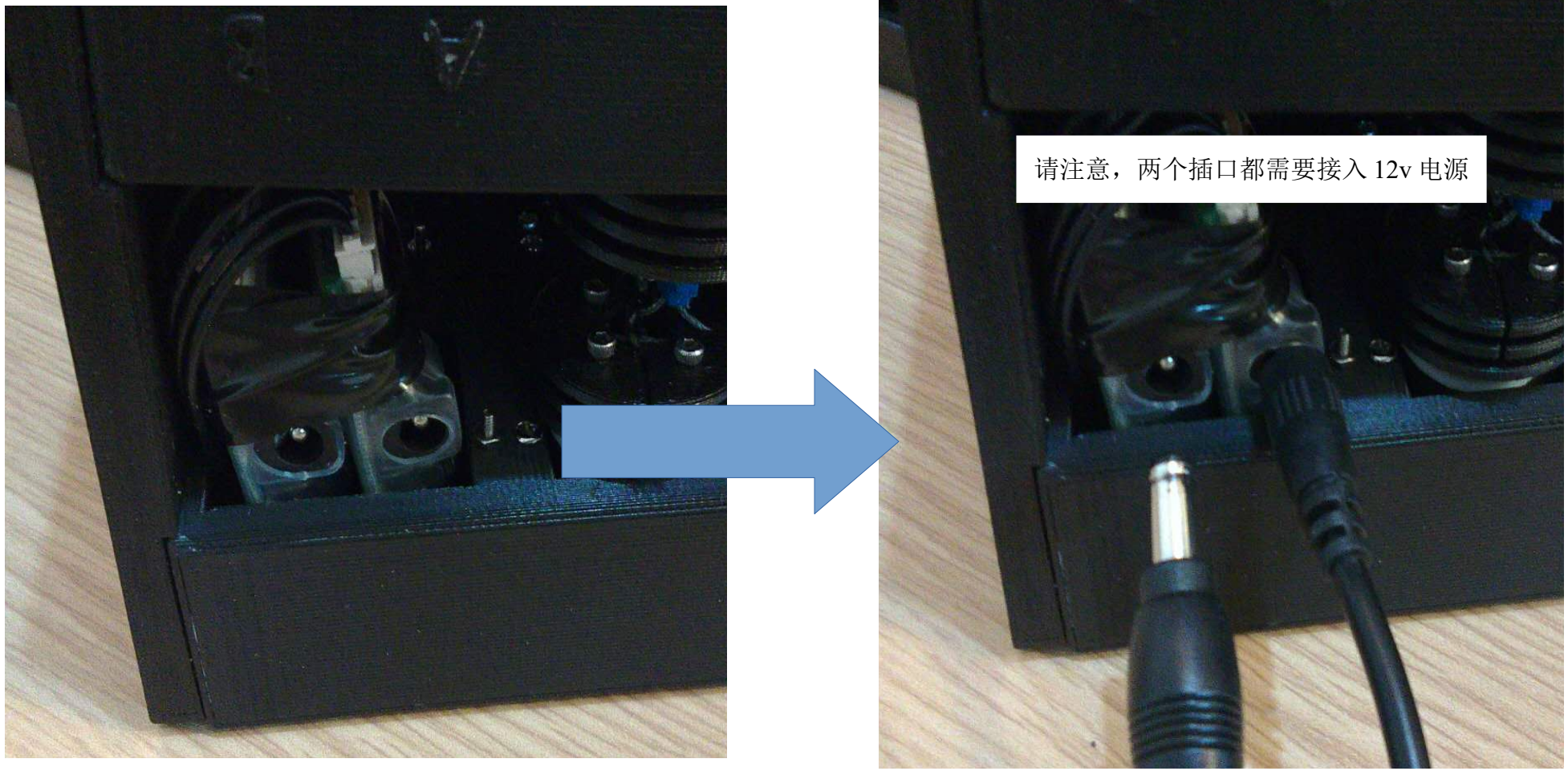
旋转接头连接：

- 掌指关节（正交）-（2 自由度）；
- 近端指间关节（1 自由度）

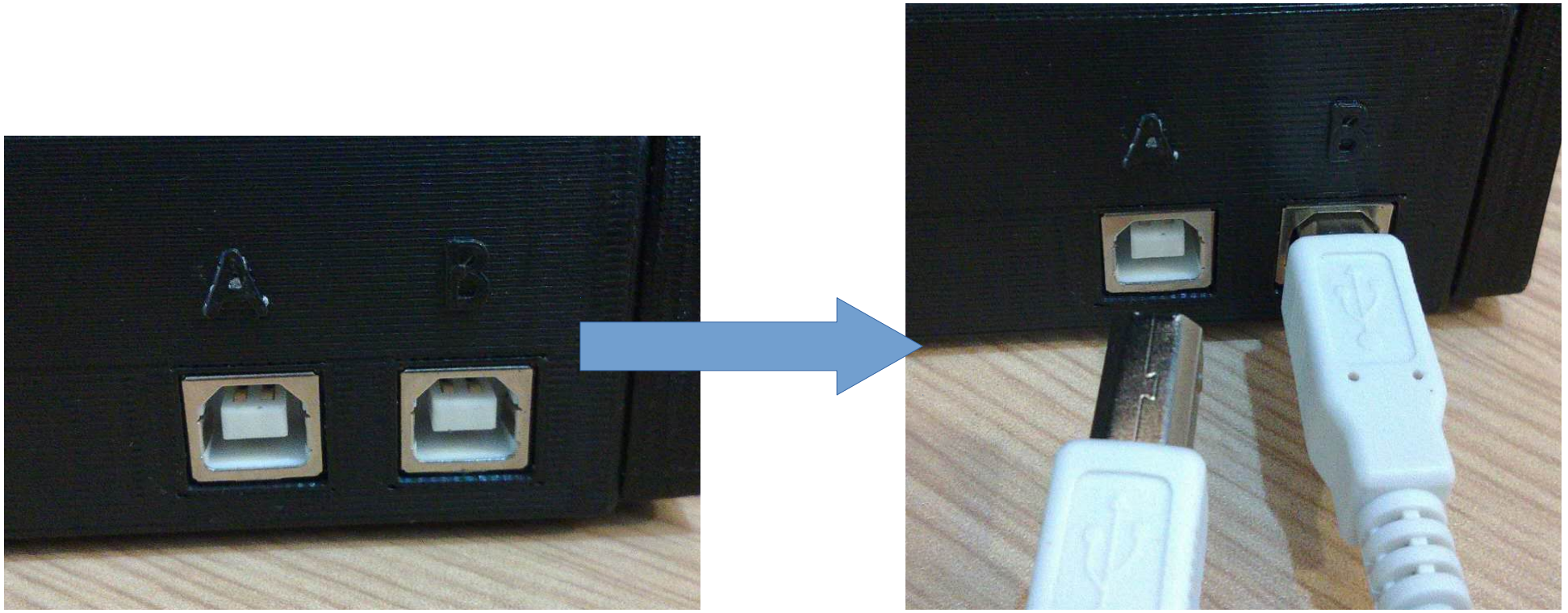


二、 如何开始使用

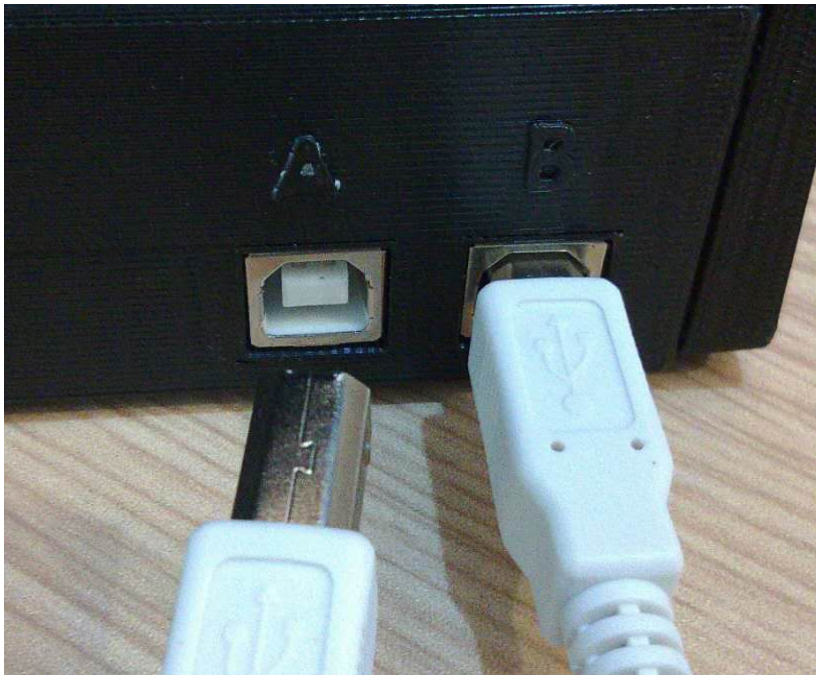
1 给灵巧手接入 12v 电源



2　将 USB 线插入灵巧手

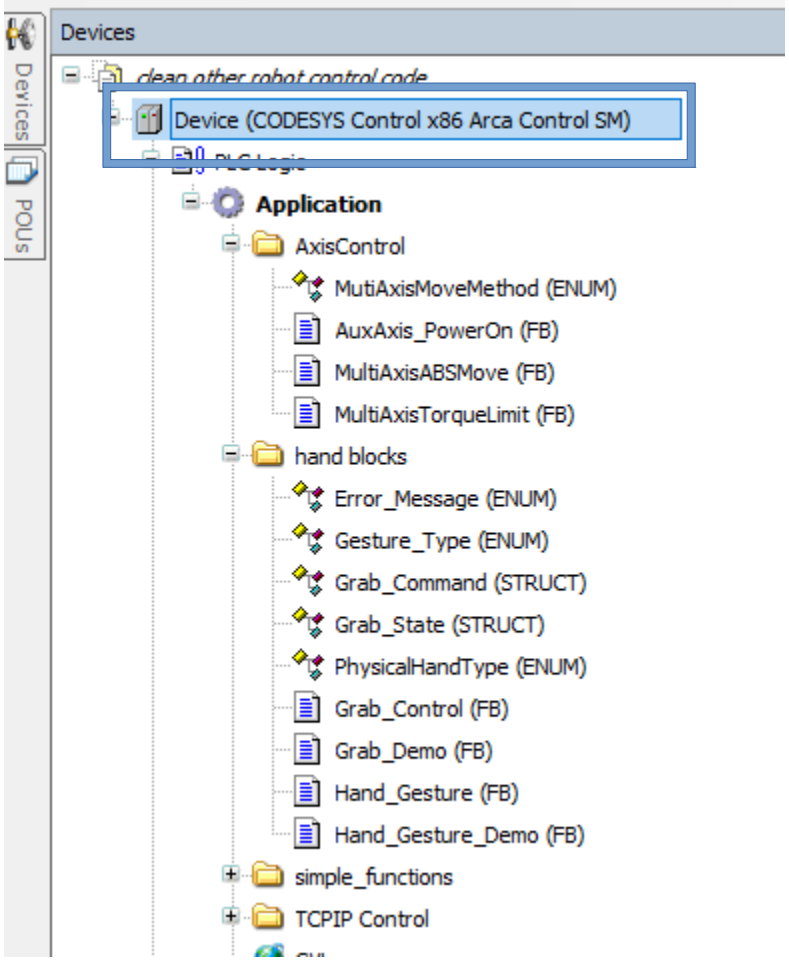


然后将 USB 的另一端插入控制器。如下图箭头所示，USB 的 A 号口对应控制器右侧的插口，而 B 号口对应控制器左侧的插口。

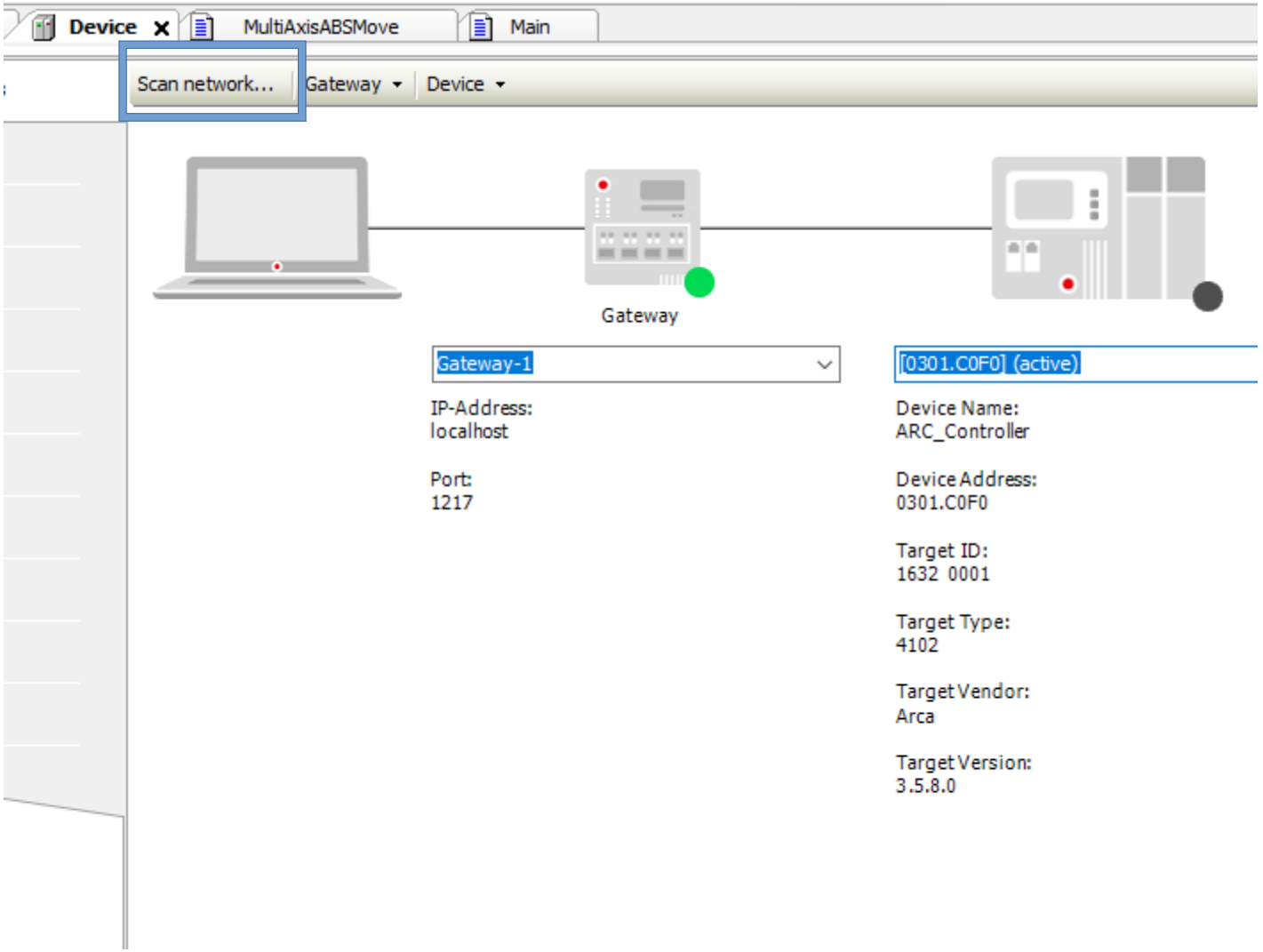


3　使用电脑工程连接控制器。

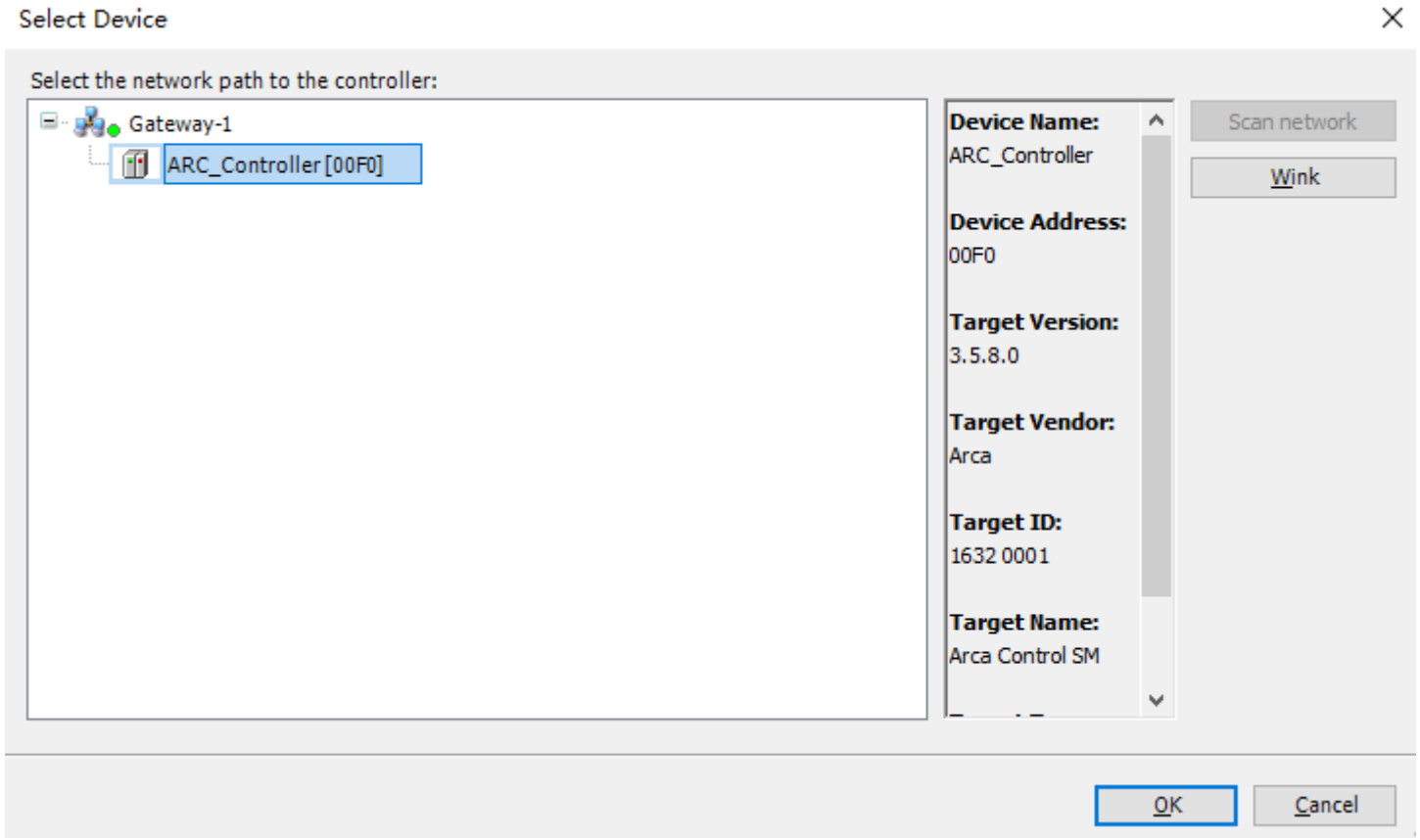
插入控制器电源启动后，通过电脑启动工程。双击列表中 Device 打开标签。



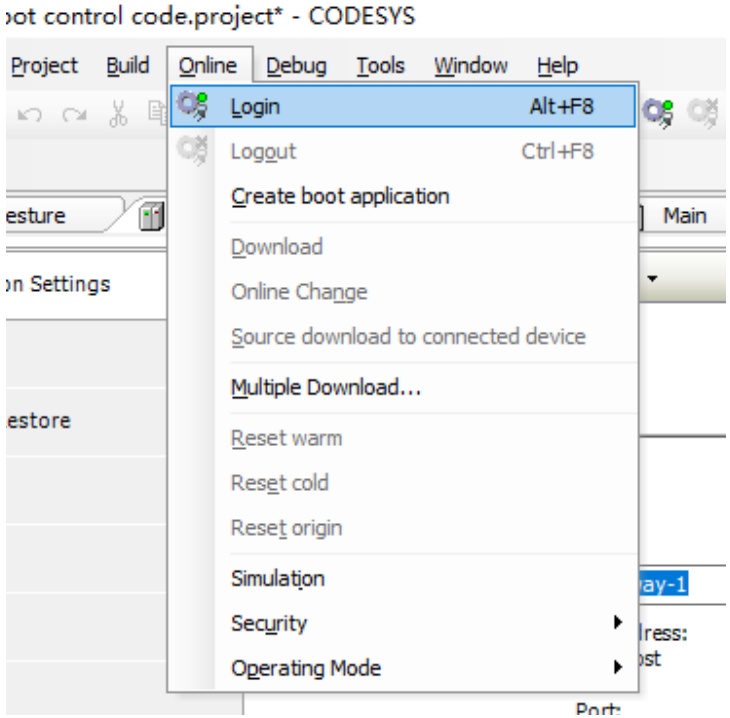
Device 标签打开后，点击 Scan network...



使用鼠标点选 Gateway 下的 ARC_Controller，然后点选 ok。

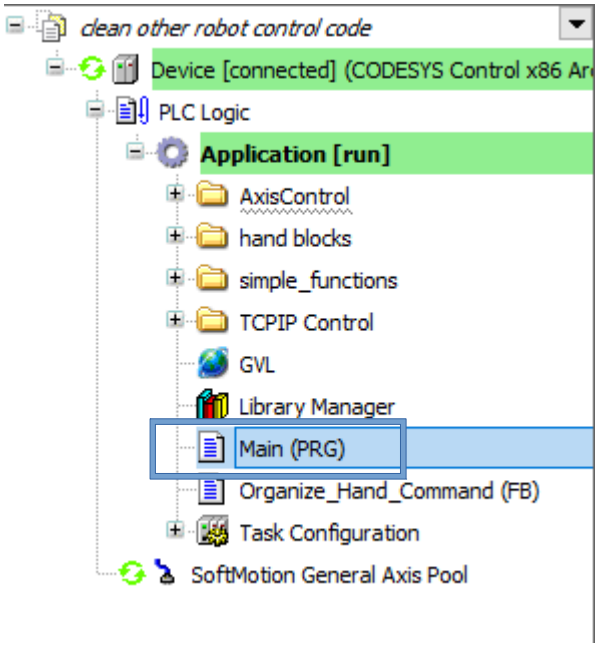


在菜单栏中选择 online，然后在下拉列表中选择 Login。此时电脑准备将工程上传至控制器上，程序将弹出新的对话框询问是否覆盖，选择是即可。

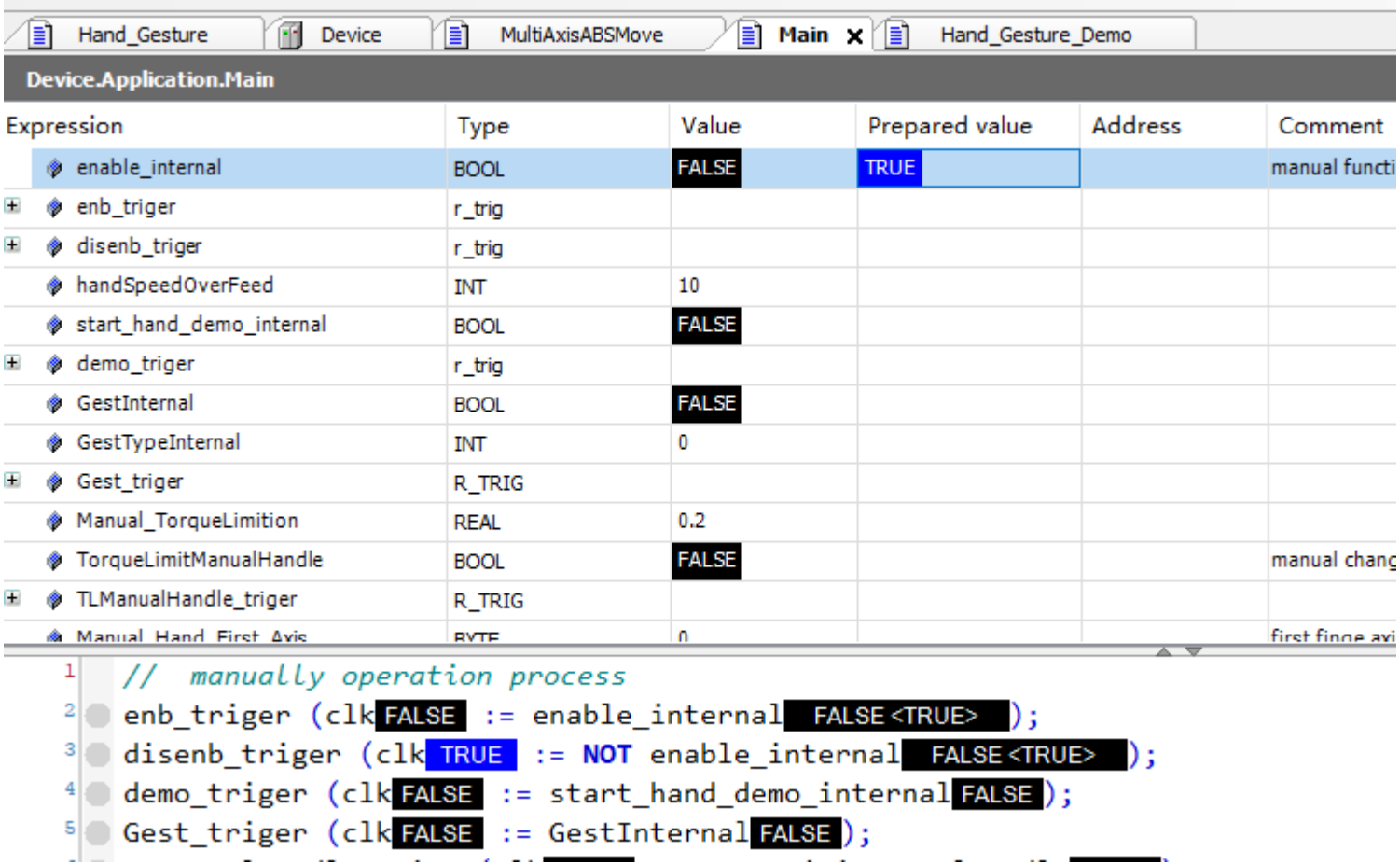


4　使用工程中的手动方式控制灵巧手

双击 Main 程序。



- 在 Main 标签中，找到想要修改的变量，使用鼠标在 Prepared Value 列处单击然后输入改变的数值。最后使用键盘快捷键 Ctrl + F7 将要改变的数值写入控制器中。
- 如下图所示，按下 Ctrl + F7 后，enable_internal 将被置为真。则控制器控制的手的电机将被使能，在这之后便可开始指示灵巧手展示手势。



- handSpeedOverFeed 控制速度。最小值为 1，最大值为 100，此值为速度百分比。
- start_hand_demo_internal 为触发信号，上升沿触发手开始循环演示。
- GestInternal 为触发信号，上升沿触发手执行一次 GestTypeInternal 中输入的值为手的编号动作。例如当 GestTypeInternal 的值为 1 时，触发 GestInternal 手将执行一次 1 号动作。
- TorqueLimitManualHandle 为触发信号，上升沿触发执行一次，按照 Manual_TorqueLimition 中的值限制电机力矩。

三、 灵巧手示例控制代码

控制灵巧手的示例代码，所有程序以及功能块全部采用 Structured Text(ST)。

主程序 main

本灵巧手示例功能全部由 Main 程序完成对其他功能块的管理。Main 程序亦是唯一的主任务程序（在 MainTask 中可以看到）。

参数

输入　N/A

输出　N/A

描述

声明栏

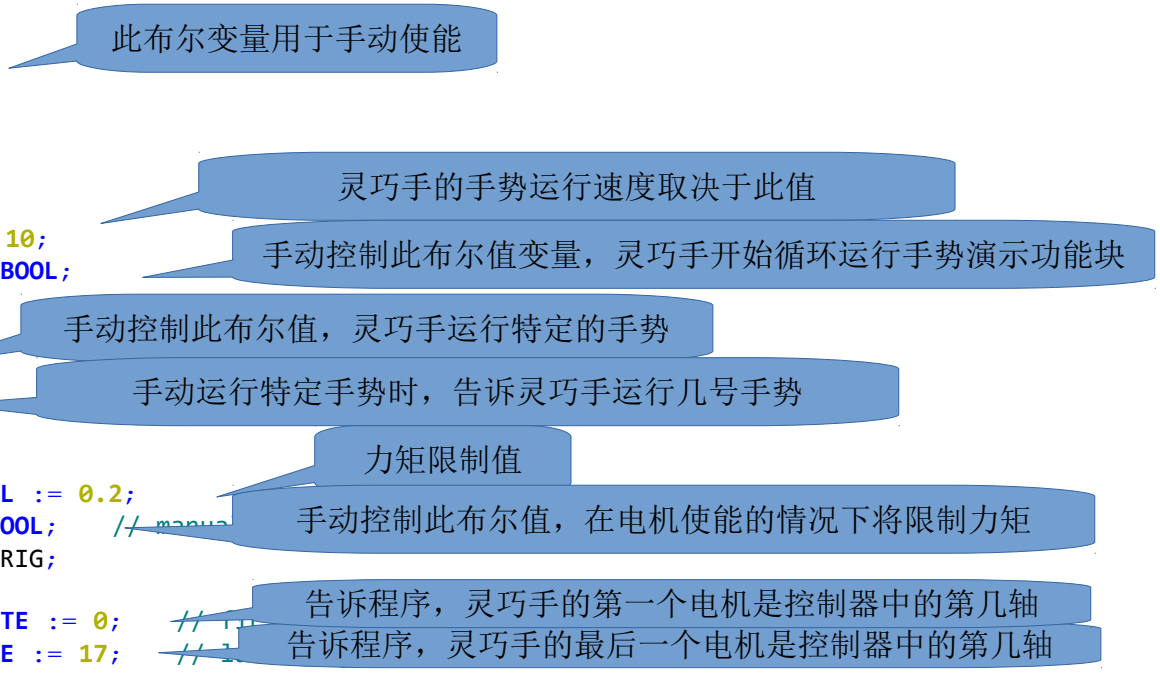
```
PROGRAM Main
VAR
    // manual function
    enable_internal : BOOL;
    MoveEn:BOOL:=FALSE;
    enb_triger : r_trig;
    disenb_triger : r_trig;

    handSpeedOverFeed : INT := 10;
    start_hand_demo_internal : BOOL;
    demo_triger : r_trig;

    GestInternal: BOOL;
    GestTypeInternal:INT;
    Gest_triger : R_TRIG;

    Manual_TorqueLimition : REAL := 0.2;
    TorqueLimitManualHandle : BOOL;    // manua
    TLManualHandle_triger : R_TRIG;

    Manual_Hand_First_Axis : BYTE := 0;    // 1
    Manual_Hand_Last_Axis : BYTE := 17;    // 1
```



```
    Manual_Hand_Axis_Allow_Move_Chart : ARRAY [0..17] OF BOOL := [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1];

    // auto function
    TCP_Server : TCPserverconnection;
    commandOut : TCP_Command_Extract_And_Analysis;
    handMov : Organize_Hand_Command;
    hand_cmd_valid : BOOL;
    hand_cmd : robot_command_via_tcp;
    old_pos_ready, new_pos_ready : BOOL;
    pick_triger, release_triger : R_TRIG;

    // Torque limition
    ToqueState : INT := 0;
    SetTorque : MultiAxisTorqueLimit;
    Torque_Limit_For_Counter : INT;
    Torque_Value_Limit_Chart : ARRAY [0..255] OF REAL;
    Torque_Axis_Limit_Chart : ARRAY [0..255] OF BOOL;
    Torque_Limit_Counter : INT;

    //debug
    AxisM : ARRAY [1..18] OF MCAT_ReadAxisMeasures;
    Aenb : BOOL := TRUE;
    AtqM : ARRAY [1..18] OF REAL;
    APosM : ARRAY [1..18] OF REAL;

    tryRPow : ARRAY [1..13] OF ArcaLibrary.MCAT_ReadPowerStatus;
    PowS : ARRAY [1..13] OF byte;

END_VAR
```

告诉程序，哪些电机是需要控制而哪些是不需要的

指令栏

```
//  manually operation process
enb_triger (clk := enable_internal);
disenb_triger (clk := NOT enable_internal);
demo_triger (clk := start_hand_demo_internal);
Gest_triger (clk := GestInternal);
TLManualHandle_triger(clk:= TorqueLimitManualHandle);

//*****************
TCP_Server(enable := TRUE, outDataReady := commandout.Response_Valid,
     OutData := commandout.Response ,outDataSize := commandout.Response_Data_Size);
```

呼叫 TCPserverconnection 功能块，
启动 TCP 服务器，准备接受和发送指令

```
commandout(indata := tcp_server.InData
     , inDataSize := TCP_Server.InDataSize
     , inData_Valid := TCP_Server.InDataReady
     , OutRobot_State := handmov.Current_Hand_State
     );

hand_cmd_valid := commandout.Command_Valid OR pick_triger.Q OR release_triger.Q OR enb_triger.Q
     OR disenb_triger.Q OR demo_triger.Q
     OR Gest_triger.Q;
hand_cmd := commandout.Command;
IF pick_triger.Q THEN
     hand_cmd.Gesture_Valid := TRUE;
     hand_cmd.Gesture_Num := 10;
END_IF
IF release_triger.Q THEN
     hand_cmd.Gesture_Valid := TRUE;
     hand_cmd.Gesture_Num := 11;
END_IF
IF enb_triger.Q THEN
     hand_cmd.Enabl_Hand_Valid := TRUE;
     hand_cmd.Enabl_Hand_state := TRUE;
END_IF
IF disenb_triger.Q THEN
     hand_cmd.Enabl_Hand_Valid := TRUE;
     hand_cmd.Enabl_Hand_state := FALSE;
END_IF
IF demo_triger.Q THEN
     hand_cmd.Gesture_Demo_Valid := TRUE;
     hand_cmd.Gesture_Demo_state := TRUE;
END_IF
IF Gest_triger.Q THEN
     hand_cmd.Gesture_Valid := TRUE;
     hand_cmd.Gesture_Num := GestTypeInternal;
END_IF

handMov(enable:= TRUE, new_cmd_valid := hand_cmd_valid
     , new_cmd := hand_cmd
     , overfeed := handSpeedOverFeed
     , HandFirstAxis := Manual_Hand_First_Axis
     , HandLastAxis := Manual_Hand_Last_Axis
     , HandAxisAllowToMov := Manual_Hand_Axis_Allow_Move_Chart
```

呼叫 TCP_Command_Extract_And_Analysis 功能块，将 TCPserverconnection 功能块接收到的指令分析，根据规则重新打包成结构体发送出去

检查手动功能和 TCP 指令然后重新打包结构体，这样既可以用手动功能控制灵巧手，亦可用 TCP 指令控制

呼叫功能块 Organize_Hand_Command，用于管理灵巧手演示程序块和灵巧手手势程序块。

```
    );

(************************
    Toque test above
*************************)

CASE ToqueState OF
    0:
        SetTorque(Enable := FALSE);
        IF handMov.Current_Hand_State.Hand_Enabled OR TLManualHandle_triger.Q THEN
            SetTorque(FirstAuxAxisNum := Manual_Hand_First_Axis, LastAuxAxisNum := Manual_Hand_Last_Axis
                , Enable := TRUE);
            FOR Torque_Limit_Counter := Manual_Hand_First_Axis TO Manual_Hand_Last_Axis DO
                Torque_Value_Limit_Chart[Torque_Limit_Counter] := Manual_TorqueLimition;
                Torque_Axis_Limit_Chart[Torque_Limit_Counter]
                    := Manual_Hand_Axis_Allow_Move_Chart[Torque_Limit_Counter - Manual_Hand_First_Axis];
            END_FOR
            toquestate := 10;
        END_IF
    10:
        SetTorque(Axis_Torque_Limit_indicator := Torque_Axis_Limit_Chart
            , Axis_Torque_Limit_Value := Torque_Value_Limit_Chart
            , execute := TRUE);
        IF SetTorque.done THEN
            SetTorque (execute := FALSE);
            toquestate := 30;
        END_IF
    30:
        IF NOT handMov.Current_Hand_State.Hand_Enabled THEN
            toquestate := 0;
        END_IF
        IF TLManualHandle_triger.q THEN
            toquestate := 10;
        END_IF

END_CASE

AllAxisGroupEnb := handMov.Current_Hand_State.Hand_Enabled;
```

力矩限制状态机，电机使能时将自动应用一次力矩限制。
此状态机同时控制手动触发力矩限制。

功能块-灵巧手功能块管理程序/Organize_Hand_Command

此功能块负责管理与协调功能块 Hand_Gesture 和 Hand_Gesture_Demo

参数

输入	`enable :BOOL;`	启动此功能块
	`New_Cmd_valid : BOOL;`	告诉功能块有新指令到达，此处应赋予单循环脉冲信号。
	`new_cmd : robot_command_via_tcp;`	输入指令
	`OverFeed : INT;`	输入速率，有效数值为 1~100
	`HandFirstAxis : BYTE;`	灵巧手的第一个电机是第几轴
	`HandLastAxis : BYTE;`	灵巧手的最后一个电机是第几轴
	`HandAxisAllowToMov : ARRAY [0..17] OF BOOL;`	告诉功能块，18 个电机中哪一些需要控制哪一些不需要。需要控制的电机 bit 设为 1，不需要的设为 0。 例如在 18 个电机中，如果除了第三轴外都需要控制的话： [1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
输出	`Done : BOOL;`	手势完成时会输出完成信号。
	`Current_Hand_State : robot_state;`	输出当前状态。
	`Err : BOOL;`	报警输出。
	`Err_Message : error_message;`	报警信息。

描述

声明

```
handMov : Organize_Hand_Command;
```

指令

```
hand_cmd_valid := commandout.Command_Valid OR pick_triger.Q OR release_triger.Q OR enb_triger.Q
    OR disenb_triger.Q OR demo_triger.Q
    OR Gest_triger.Q;
```

```
hand_cmd := commandout.Command;
IF pick_triger.Q THEN
    hand_cmd.Gesture_Valid := TRUE;
    hand_cmd.Gesture_Num := 10;
END_IF
IF release_triger.Q THEN
    hand_cmd.Gesture_Valid := TRUE;
    hand_cmd.Gesture_Num := 11;
END_IF
IF enb_triger.Q THEN
    hand_cmd.Enabl_Hand_Valid := TRUE;
    hand_cmd.Enabl_Hand_state := TRUE;
END_IF
IF disenb_triger.Q THEN
    hand_cmd.Enabl_Hand_Valid := TRUE;
    hand_cmd.Enabl_Hand_state := FALSE;
END_IF
IF demo_triger.Q THEN
    hand_cmd.Gesture_Demo_Valid := TRUE;
    hand_cmd.Gesture_Demo_state := TRUE;
END_IF
IF Gest_triger.Q THEN
    hand_cmd.Gesture_Valid := TRUE;
    hand_cmd.Gesture_Num := GestTypeInternal;
END_IF

handMov(enable:= TRUE, new_cmd_valid := hand_cmd_valid
    , new_cmd := hand_cmd
    , overfeed := handSpeedOverFeed
    , HandFirstAxis := Manual_Hand_First_Axis
    , HandLastAxis := Manual_Hand_Last_Axis
    , HandAxisAllowToMov := Manual_Hand_Axis_Allow_Move_Chart
    );
```

功能块-灵巧手演示/Hand_Gesture_Demo

此功能块通过打包 Hand_Gesture 实现手的循环演示不同动作。

参数

输入	`enable :BOOL;`	启动此功能块
	`startMov :BOOL;`	开始执行动作
	`ovr : INT;`	手势速率
	`HandFirstAxis : BYTE;`	灵巧手的第一个电机是第几轴
	`HandLastAxis : BYTE;`	灵巧手的最后一个电机是第几轴
	`HandAxisAllowToMov : ARRAY [0..17] OF BOOL;`	告诉功能块，18 个电机中哪一些需要控制哪一些不需要。需要控制的电机 bit 设为 1，不需要的设为 0。 例如在 18 个电机中，如果除了第三轴外都需要控制的话： [1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
输出	`IsEnabled : BOOL;`	此功能块已启动
	`IsMoving : BOOL;`	灵巧手正在执行动作
	`Current_Gesture : Gesture_type;`	灵巧手当前手势
	`err : BOOL;`	报警输出
	`err_message : error_message;`	报警信息

描述

声明

```
Hand_Demo : hand_gesture_demo;
```

指令

```
_old_demo_state := TRUE;
_old_ovr := 30;
HandFirstAxis := 0;
HandLastAxis := 17;
```

```
HandAxisAllowToMov := [1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1];
Hand_Demo(enable:= _old_demo_state, startMov := TRUE, ovr := _old_ovr
        , HandFirstAxis:= HandFirstAxis, HandLastAxis:= HandLastAxis, HandAxisAllowToMov:= HandAxisAllowToMov);
```

功能块-灵巧手手势/Hand_Gesture

参数

输入	Enable : BOOL;	使能功能块
	OverFeed : INT;	动作速率
	Gesture : Gesture_Type;	输入执行手势类型
	HandType : PhysicalHandType;	灵巧手种类（暂时无此功能）
	HandFirstAxis : BYTE;	灵巧手的第一个电机是第几轴
	HandLastAxis : BYTE;	灵巧手的最后一个电机是第几轴
	HandAxisAllowToMov : ARRAY [0..17] OF BOOL;	告诉功能块，18 个电机中哪一些需要控制哪一些不需要。需要控制的电机 bit 设为 1，不需要的设为 0。 例如在 18 个电机中，如果除了第三轴外都需要控制的话： [1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
输出	Done : BOOL;	完成当前动作
	isMoving : BOOL;	正在动作
	Error : BOOL;	报警输出
	Err_Message : Error_Message;	报警信息

描述

声明

```
GestureMov : Hand_Gesture;
```

指令

```
// set gesture type
GestureType[1] := Gesture_type.Gesture_Type_Num_1;
GestureType[2] := Gesture_type.Gesture_Type_Num_2;
GestureType[3] := Gesture_type.Gesture_Type_Num_3;
ovr := 40;
HandFirstAxis := 0;
HandLastAxis := 17;
HandAxisAllowToMov := [1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1]

GestureMov(Enable := TRUE, overfeed := ovr, Gesture := Current_Gesture := GestureType[cycleCounter]
        , isMoving => isMoving
        , HandFirstAxis := HandFirstAxis, HandLastAxis:= HandLastAxis, HandAxisAllowToMov:= HandAxisAllowToMov);

IF GestureMov.Done THEN
    cycleTimer(in := TRUE, pt:= cycleDelayTime);   // start timer for delay
    IF cycleTimer.Q THEN  // time reach
        cycleCounter := cycleCounter + 1; // done and self add for next gesture
        cycleTimer(in := FALSE);  // reset timer
        GestureMov(Enable := FALSE);  // reset movement block
        IF cycleCounter <= total_cycle_time THEN
            cycleState := 10;     // if not reach total number then go back keep do next gesture
        ELSE
            cycleState := 0; // if reach maximum then go back beginning
        END_IF
    END_IF
END_IF
```

功能块-TCP 服务器/TCPServerConnection

参数

输入	`Enable : BOOL;`	启动此功能块
	`OutDataReady : BOOL;`	告诉功能块有新指令到达，用于向客户端发送信息。此处应赋予单循环脉冲信号。
	`OutDataSize : CAA.SIZE;`	发送信息的长度
	`OutData : ARRAY [0..255] OF BYTE;`	发送的信息
输出	`Err : BOOL;`	报警信号
	`InDataReady : BOOL;`	接受到新信息，此布尔值为单循环脉冲信号。
	`InDataSize : CAA.SIZE;`	接受到的信息长度
	`InData : STRING;`	接收到的信息

描述

声明

```
TCP_Server : TCPserverconnection;
```

指令

```
TCP_Server(enable := TRUE, outDataReady := commandout.Response_Valid,
    OutData := commandout.Response ,outDataSize := commandout.Response_Data_Size
    , InDataReady => inDataValid, InDataSize => inDataSize, inData => indata);
```

功能块-智能手 TCP 通讯代码解析/TCP_Command_Exract_And_Analysis

参数

输入	`InData : STRING;`	从 TCP 服务器收到的指令
	`InDataSize : CAA.SIZE;`	从 TCP 服务器收到的指令的长度
	`InData_Valid : BOOL;`	告诉功能块从 TCP 服务器收到新指令，此处应赋予单循环脉冲。
	`OutRobot_State : Robot_State;`	告诉功能块当前状态，功能块会拆分并打包成通讯代码输出
输出	`Command_Valid : BOOL;`	从 TCP 服务器收到的指令解析完成，信号为单循环脉冲。
	`Command : robot_command_via_tcp;`	解析并重新打包好的从 TCP 服务器收到的指令
	`Response : ARRAY [0..255] OF BYTE;`	重新打包好的当前状态
	`Response_Data_Size : CAA.SIZE;`	当前状态的数据长度
	`Response_Valid : BOOL;`	当前状态打包完成，信号为单循环脉冲。
	`Error : BOOL;`	报警信号
	`ErrorMassage : STRING;`	报警信息

描述

声明

```
commandOut : TCP_Command_Extract_And_Analysis;
```

指令

```
commandout(indata := tcp_server.InData
    , inDataSize := TCP_Server.InDataSize
    , inData_Valid := TCP_Server.InDataReady
    , OutRobot_State := handmov.Current_Hand_State
    , Command_Valid=> TCP_Command_Valid, Command=> TCP_Command
    , Response => Out_Robot_State, Response_Data_Size => Out_Data_Size, Response_Valid => Out_Data_Ready);
```

功能块-多轴与多机器人组使能/AuxAxis_PowerOn

参数

输入	`Enable : BOOL;`	启动此功能块
输出	`Done : BOOL;`	完成所有轴与机器人组的使能
	`Error : BOOL;`	报警信号
	`Err_Message : Error_Message;`	报警信息

描述

声明

```
powerOnHand : AuxAxis_PowerOn;
```

指令

```
powerOnHand(enable := enable
    , Done => Done, Error => Err, Err_Message => ErrMsg);
```

功能块-多轴绝对位置运动/MultiAxisABSMove

参数

输入	`FirstAuxAxisNum : BYTE;`	想要控制的第一个电机号码
	`LastAuxAxisNum : BYTE;`	想要控制的最后一个电机号码
	`Enable : BOOL;`	启动此功能块。一旦启动，功能块将不接受 `FirstAuxAxisNum` 和 `LastAuxAxisNum` 的改变。

	`Axis_Mov_Indicator : ARRAY [0..255] OF BOOL;`	告诉功能块，255 个电机中哪一些需要控制哪一些不需要。需要控制的电机 bit 设为 1，不需要的设为 0。例如需要控制的第一个电机为 3，最后一个电机为 6，其中的 5 号电机无需控制时： [0, 0, 0, 1, 1, 0, 1, 0]
	`Axis_Mov_Pos : ARRAY [0..255] OF REAL;`	告诉功能块每一个需要控制的电机的位置
	`Execute : BOOL;`	执行动作。功能块开始执行动作后将不再接受 `Axis_Mov_Indicator` 输入的改变。
	`Vel, Acc, Dec, Jerk : REAL;`	所有电机使用统一的速度，加速度，减速度，加加速度。
	`MoveMethod : MutiAxisMoveMethod;`	电机动作的方法。有两种模式选择，分别为无控制模式和速度控制模式。速度控制时所有电机将同时到达位置，无控制时每个电机各走各的。
输出	`done : BOOL;`	完成动作
	`error : BOOL;`	报警信号
	`error_str : STRING;`	报警信息

描述

声明

```
MultisingleMov : MultiAxisABSMove;
```

指令

```
MultisingleMov(firstAuxAxisNum := faxis, LastAuxAxisNum := laxis, enable := Enable
    , execute := TRUE, Axis_Mov_Indicator := AxisAllowToMov, axis_mov_pos := GivHandPos
    , vel := givenVel, acc :=givenAcc, dec := givenAcc, jerk := givenJerk, MoveMethod := 10);
```

功能块-多轴力矩限制/MultiAxisTorqueLimit

参数

输入	FirstAuxAxisNum : BYTE;	想要限制力矩的第一个电机号码
	LastAuxAxisNum : BYTE;	想要限制力矩的最后一个电机号码
	Enable : BOOL;	启动此功能块。一旦启动，功能块将不接受 FirstAuxAxisNum 和 LastAuxAxisNum 的改变。
	Axis_Torque_Limit_Indicator : ARRAY [0..255] OF BOOL;	告诉功能块，255 个电机中哪一些需要控制哪一些不需要。需要控制的电机 bit 设为 1，不需要的设为 0。例如需要控制的第一个电机为 3，最后一个电机为 6，其中的 5 号电机无需控制时： [0, 0, 0, 1, 1, 0, 1, 0]
	Axis_Torque_Limit_Value : ARRAY [0..255] OF REAL;	告诉功能块，每一个电机新的限制值。
	Execute : BOOL;	执行，限制将被写入。
输出	done : BOOL;	执行完成
	error : BOOL;	报警信号
	error_str : STRING;	报警信息

描述

声明

```
SetTorque : MultiAxisTorqueLimit;
```

指令

```
SetTorque(Axis_Torque_Limit_indicator := Torque_Axis_Limit_Chart, Axis_Torque_Limit_Value := Torque_Value_Limit_Chart
          , execute := TRUE);
```

参数

输入

输出 N/A

描述

声明

指令

四、 使用 TCP 发送指令控制智能手

使用示例工程，将在控制器中建立 TCP 服务器。TCP 客户端连接上服务器后可以通过发送指令控制灵巧手，使灵巧手执行设定好的手势与动作。在默认情况下，TCP 服务器 IP 地址为 192.168.80.240，端口为 23。如果控制器的 IP 地址被更改，则工程中的 TCPServerConnection 功能块中的变量 IP_Addr 需要修改成控制器一致的地址。

使用客户端发送指令控制灵巧手，需要遵循以下格式：

一条完整的指令格式
(标签)内容;[CR][LF]

(标签)：标签由 10 位 ASCII 中的英文字母组成，有大小写之分。标签告诉程序执行什么样的动作，这些标签最终在功能块 TCP_Command_Extract_And_Analysis 中被定义。

内容：内容由 10 位 ASCII 中的数字组成，结合标签将告诉程序如何执行动作。

[CR]：CR 为 ASCII 中的回车符。[CR]总共由 3 个字符组成，以 10 进位表示为 91，13，93。

[LF]：LF 为 ASCII 中的换行符。[LF]总共由 3 个字符组成，以 10 进位表示为 91，10，93。

标签-内容对照表

标签	内容	含义
X	正负数值	用于控制机器人行走笛卡尔坐标系 X 轴的相对位置。
Y	正负数值	用于控制机器人行走笛卡尔坐标系 Y 轴的相对位置。
Z	正负数值	用于控制机器人行走笛卡尔坐标系 Z 轴的相对位置。
GRAB	0	打开爪子
	1	合拢爪子
GEST	0	灵巧手不执行任何手势。
	1	手势-数字 1
	2	手势-数字 2
	3	手势-数字 3
	4	手势-数字 4
	5	手势-数字 5
	6	手势-数字 6
	10	手势-拳头
	11	手势-手掌
	12	手势-比拇指
	13	手势-摇动食指 NONONO
	14	手势-勾引
	15	手势-手指开合
	16	手势-左右摇摆手掌
	17	手势-握拳上下摇动
	18	

HENB	0	关闭使能
	1	使能使能控制器中所有电机与机器人（包括灵巧手）
HDem	0	灵巧手停止循环演示
	1	灵巧手停止开始演示不同的手势
CNC	正数值	执行控制器中相应数值名称的 CNC 程序

样例

一条完整的指令样例	完整的 ASCII	含义
(HENB)01;[CR][LF]	40 72 69 78 66 41 48 49 59 91 13 93 91 10 93	使能控制器中所有电机与机器人（包括灵巧手）