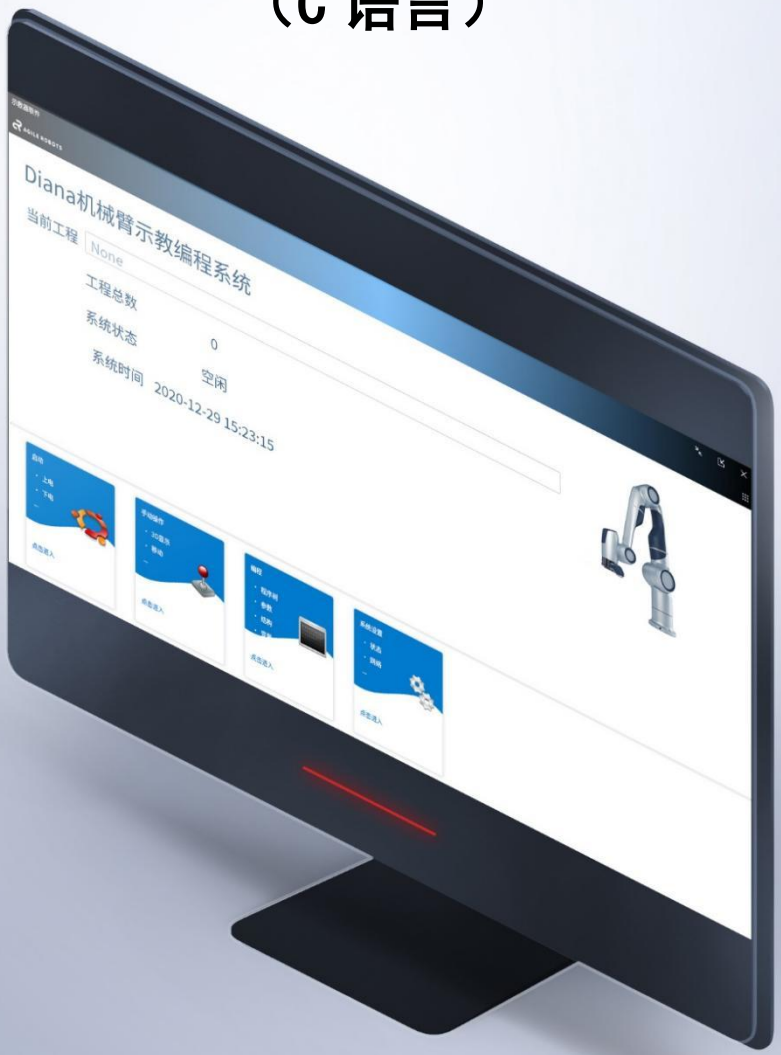
AGILE ROBOTS

北京思灵机器人科技有限责任公司

医疗版远程 API 说明文档

（C 语言）



软 件 版 本　2.14　|　文 档 版 本　3.33

目录

1 initSrvNetInfo 1
2 initSrv 1
3 destroySrv 4
4 setPushPeriod 4
5 moveTCP 5
6 rotationTCP 6
7 moveJoint 7
8 moveJToTarget 7
9 moveJToPose 8
10 moveJ 10
11 moveL 10
12 moveLToTarget 11
13 moveLToPose 12
14 speedJ 13
15 speedL 13
16 freeDriving 14
17 stop 15
18 forward 15
19 inverse 16
20 getJointPos 17
21 getJointAngularVel 18
22 getJointCurrent 18
23 getJointTorque 19
24 getTcpPos 19
25 getTcpExternalForce 20
26 releaseBrake 20
27 holdBrake 21
28 changeControlMode 21
29 getLibraryVersion 22
30 formatError 22
31 getLastError 23
32 setLastError 23
33 setDefaultActiveTcp 24
34 getLinkState 25
35 getTcpForce 25
36 getJointForce 26
37 isCollision 27
38 initDHCali 27
39 getDHCaliResult 28
40 setDH 29
41 setWrd2BasRT 29
42 setFla2TcpRT 30

43 getRobotState 31
44 resume 32
45 setJointCollision 32
46 setCartCollision 33
47 enterForceMode 33
48 leaveForceMode 34
49 setDefaultActiveTcpPose 35
50 setResultantCollision 36
51 zeroSpaceFreeDriving 36
52 createPath 37
53 addMoveL 37
54 addMoveJ 38
55 runPath 40
56 destroyPath 40
57 rpy2Axis 40
58 axis2RPY 41
59 homogeneous2Pose 41
60 pose2Homogeneous 42
61 enableTorqueReceiver 43
62 sendTorque_rt 44
63 enableCollisionDetection 44
64 setActiveTcpPayload 45
65 servoJ 45
66 servoL 46
67 servoJ_ex 47
68 servoL_ex 48
69 speedJ_ex 49
70 speedL_ex 50
71 dumpToUDisk 51
72 inverse_ext 52
73 getJointLinkPos 53
74 createComplexPath 53
75 addMoveLByTarget 54
76 addMoveLByPose 55
77 addMoveJByTarget 56
78 addMoveJByPose 57
79 addMoveCByTarget 58
80 addMoveCByPose 59
81 runComplexPath 60
82 destroyComplexPath 60
83 saveEnvironment 61
84 dumpToUDiskEx 61
85 enterForceMode_ex 62
86 readDI 63

87 readAI 64
88 setAIMode 64
89 writeDO 65
90 writeAO 66
91 readBusCurrent 66
92 readBusVoltage 67
93 getDH 67
94 getOriginalJointTorque 68
95 getJacobiMatrix 69
96 resetDH 69
97 runProgram 70
98 stopProgram 70
99 getVariableValue 71
100 setVariableValue 71
101 isTaskRunning 72
102 pauseProgram 72
103 resumeProgram 73
104 stopAllProgram 73
105 isAnyTaskRunning 74
106 cleanErrorInfo 74
107 setCollisionLevel 75
108 mappingInt8Variant 76
109 mappingDoubleVariant 76
110 mappingInt8IO 76
111 mappingDoubleIO 77
112 setMappingAddress 77
113 lockMappingAddress 79
114 unlockMappingAddress 80
115 getJointCount 80
116 setWayPoint 80
117 getWayPoint 84
118 addWayPoint 84
119 deleteWayPoint 85
120 zeroSpaceManualMove 85
121 setExternalAppendTorCutoffFreq 86
122 getHomingState 87
123 startHoming 88
124 setEndKeyEnableState 88
125 updateForce 89
126 getCartImpedanceCoordinateType 90
127 setCartImpedanceCoordinateType 90
128 getInertiaMatrix 91
129 getSixAxiaForce 91
130 inverseClosedFull 92

131 getInverseClosedResultSize 93
132 getInverseClosedJoints 94
133 destoryInverseClosedItems 94
134 nullSpaceFreeDriving 95
135 nullSpaceManualMove 95
136 calculateJacobi 95
137 calculateJacobiTF 96
138 getTcpForceInToolCoordinate 97
139 speedLOnTcp 97
140 inverseClosedIdeal 98
141 getInverseClosedIdealResultSize 100
142 getInverseClosedIdealJoints 100
143 destoryInverseClosedIdealItems 101
144 getSixAxisZeroOffset 101
145 setSixAxisZeroOffset 102
146 setDefaultActiveWorkpiece 103
147 setDefaultActiveWorkpiecePose 103
148 getDefaultActiveWorkpiece 104
149 getDefaultActiveWorkpiecePose 105
150 setControllerFeatureCode 105
151 testControllerFeature 106
152 enableSixAxis 106
153 isSixAxisEnabled 107
154 getMechanicalJointsPositionRange 109
155 getMechanicalMaxJointsVel 109
156 getMechanicalMaxJointsAcc 110
157 getMechanicalMaxCartVelAcc 111
158 getJointsPositionRange 111
159 getMaxJointsVel 112
160 getMaxJointsAcc 113
161 getMaxCartTranslationVel 114
162 getMaxCartRotationVel 114
163 getMaxCartTranslationAcc 115
164 getMaxCartRotationAcc 115
165 setJointsPositionRange 116
166 setMaxJointsVel 116
167 setMaxJointsAcc 117
168 setMaxCartTranslationVel 117
169 setMaxCartRotationVel 118
170 setMaxCartTranslationAcc 118
171 setMaxCartRotationAcc 119
172 freeDriving_ex 120
173 freeDrivingWithLockedJoint 121
174 freeDrivingWithLockedJoint_ex 121

175 freeDrivingForCurrentLoop 123
176 getCurrentLoopFreeDrivingState 123
177 setJointLockedInCartImpedanceMode 124
178 getJointLockedInCartImpedanceMode 125
179 calcRobotArmAngle 125
180 enterSafetyIdle 126
181 leaveSafetyIdle 126
182 identifyTcpPayload 127
183 setJointImpeda 130
184 getJointImpeda 130
185 setCartImpeda 131
186 getCartImpeda 131
187 getRobotNetworkInfo 132
188 setRobotNetworkInfo 133
189 getRgbLedState 134
190 setRgbLedState 135
191 getFunctionOptI4 136
192 setFunctionOptI4 138
193 setGravInfo 139
194 getGravInfo 140
195 getSixAxisInstallation 140
196 initSixAxisInstallation 141
197 getCalcSixAxisWaypoints 146
198 identifyTcpPayloadWithSixAxis 147
199 getFixedSixAxisForce 149
200 getDefaultActiveTcp 149
201 setSixAxisSensorAbsAccuracy 150
202 getSixAxisSensorAbsAccuracy 151
203 getJointsSoftLimitRange 151
204 setJointsSoftLimitRange 152
205 getTcpPayloadWithSixAxis 153
206 setTcpPayloadWithSixAxis 153
207 setThresholdTorque 154
208 getThresholdTorque 155
209 setStaticFriction 155
210 getStaticFriction 156
211 setPredictMoveMode 157
212 getPredictMoveMode 158
213 setSafetySpeedLimit 159
214 getSafetySpeedLimit 161
215 setSafetyCartZone 161
216 getSafetyCartZone 162
217 getEncoderBatteryLevel 162
218 getWarningList 163

219 formatWarning 165
220 cleanWarning 165
221 calibrateSixAxisSensor 165
222 calcInstallAngle 166
223 setFunctionOptR8 168
224 getFunctionOptR8 168
225 updateRealtimeVirtualWall 169
附录 A： 172
附录 B： 178
附录 C： 179
附录 D： 181

修订历史

文档版本	修改内容	软件版本	修订时间
V1.0	创建	-	
V2.0	1. 更改原 setCollision 接口函数为 setCartCollision 和 setJointCollision 两个接口函数。 2. 新增接口 getTcpForce，getJointForce，isCollision，initDHCali，getDHCaliResult，setDH，setWrd2BasRT，setFla2TcpRT，getRobotState，resume	-	2020-7-14
V2.1	1. 修改 moveJToTarget，moveJToPose，moveLToTarget，moveLToPose 接口函数的参数，去掉交融半径；修改 speedJ 接口函数的参数，去掉 active_tcp。 2. 修改 getRobotState，增加 free-driving 和 zero-space-free-driving 两种状态。 3. 新增接口：enterForceMode，leaveForceMode，setDefaultActiveTcpPose，setResultantCollision，setJointImpedance，getJointImpedance，setCartImpedance，getCartImpedance，zeroSpaceFreeDriving。 4. 新增多路点功能相关接口：createPath，addMoveL，addMoveJ，runPath，destroyPath。 5. 新增硬件错误码，及修改 formatError 的说明，并对 initSrv 中回调函数 fnError 的实现提出一些建议。 6. 修改 speedJ 和 speedL 的参数 t 的含义。	-	2020-9-9
V2.2	1. 新增四个接口：ToAxis，ToRPY，Homogeneous2Pose，Pose2Homogeneous。	-	2020-9-12
V2.3	1. 新增目录 2. 修改 enterForceMode 描述。 3. 新增接口：servoJ，servoL 4. 为接口函数 moveJToTarget, moveJToPose, moveLToTarget, moveLToPose, moveJ, moveL,添加 wait_move()函数示例。	-	2020-9-25

V2.4	1. 新增接口：speedJ_ex，speedL_ex，servoJ_ex，servoL_ex	-	2020-10-12
V2.5	1. 新增接口 dumpToUDisk 2. 修改 getDHCaliResult 参数个数，原有 3 个参数，现为 4 个，增加绝对定位精度参考值数组。	-	2020-10-25
V2.6	1. 修改 save 接口名为 saveEnvironment	-	2020-11-25
V2.7	2. 新增 IO 相关接口：71-81	-	2020-11-27
V2.8	1. 新增接口 getDH、getOriginalJointTorque、getJacobiM atrix 2. 修改函数 getJointTorque 含义	-	2020-11-27
V2.9	1. 新增 resetDH 2. 更新错误码	-	2020-12-10
V3.0	1. 修改 IO 读写接口 readDI、readAI、writeDO、writeAO、setAIMode 2. 新增接口 runProgram、stopProgram、getVariableValue、setVariableValue、isTaskRunning、pauseProgram、resumeProgram、stopAllProgram、isAnyTaskRunning、cleanErrorInfo、setCollisionLevel、mappingInt8Variant、mappingDoubleVariant、mappingInt8IO、mappingDoubleIO、setMappingAddress、lockMappingAddress、unlockMappingAddress	-	2021-03-01
V3.1	新增路点变量接口： setWayPoint、getWayPoint、addWayPoint、deleteWayPoint	-	2021-04-25
V3.2	新增创建复杂路径接口： createComplexPath、addMoveLByTarget、addMoveLByPose、addMoveJByTarget、addMoveJByPose、addMoveCByTarget、addMoveCByPose、runComplexPath、destroyComplexPath	-	2021-04-26
V3.3	1. API 支持同时控制多台机械臂	-	2021-06-25
V3.4	1. 修正了一些书写错误	-	2021-07-15
V3.5	1. 完善 C 版本 API	-	2021-09-02

V3.6	1. 补充函数 inverseClosedFull、getInverseClosedResultSize、getInverseClosedJoints、destoryInverseClosedItems、nullSpaceFreeDriving、nullSpaceManualMove、caculateJacobi、caculateJacobiTF、getTcpForceInToolCoordinate、initSrvNetInfo 函数	-	2021-09-15
V3.7	1. 修改 sendTorque_rt 函数的 torque 参数数组个数为 7，zeroSpaceManualMove 函数的速度和加速度参数的数组个数为 7	-	2021-12-03
V3.8	1. 添加 15 个接口说明和 附件 B，其中接口分别为：setEndKeyEnableState、inverseClosedIdeal、getInverseClosedIdealResultSize、getInverseClosedIdealJoints、destoryInverseClosedIdealItems、getSixAxisZeroOffset、setSixAxisZeroOffset、setDefaultActiveWorkpiece、setDefaultActiveWorkpiecePose、getDefaultActiveWorkpiece、getDefaultActiveWorkpiecePose、setControllerFeatureCode、testControllerFeature、enableSixAxis、isSixAxisEnabled。	-	2021-12-24
V3.9	1. 添加 18 个接口说明：getMechanicalJointsPositionRange、getMechanicalMaxJointsVel、getMechanicalMaxJointsAcc、getMechanicalMaxCartVelAcc、getJointsPositionRange、getMaxJointsVel、getMaxJointsAcc、getMaxCartTranslationVel、getMaxCartRotationVel、getMaxCartTranslationAcc、getMaxCartRotationAcc、setJointsPositionRange、setMaxJointsVel、setMaxJointsAcc、setMaxCartTranslationVel、setMaxCartRotationVel、setMaxCartTranslationAcc、setMaxCartRotationAcc	-	2022-3-16
V3.10	1. 添加 3 个接口说明：freeDriving_ex、freeDrivingWithLockedJoint、freeDrivingWithLockedJoint_ex	V2.7	2022-3-23
V3.11	1. 新增接口 freeDrivingForCurrentLoop 和 getCurrentLoopFreeDrivingState	V2.7	2022-03-26
V3.12	1. 更新错误码	V2.7	2022-03-29
V3.13	新增接口 setJointLockedInCartImpedanceMode 和 getJointLockedInCartImpedanceMode	V2.7	2022-04-18
V3.14	1. 新增接口 calcRobotArmAngle、enterSafetyIdle 和 leaveSafetyIdle 2. 更新错误码	V2.7	2022-05-18
V3.15	1. 新增接口 identifyTcpPayload	V2.8	2022-05-19

V3.16	1. 删除接口 getCartImpedance、setCartImpedance、getJointImpedance、setJointImpedance 2. 新增接口 getCartImpeda、setCartImpeda、getJointImpeda、setJointImpeda 3. 删除 updateForce_ex 接口，修订 updateForce 接口的入参	V2.8	2022-07-08
V3.17	1. 修正 getInverseClosedJoints 返回值 2. 新增接口 getRobotNetworkInfo、setRobotNetworkInfo、getRgbLedState 和 setRgbLedState 3. 新增接口 getFunctionOptI4、setFunctionOptI4	V2.9	2022-07-29
V3.18	1. 新增接口 getGravInfo、setGravInfo、getSixAxisInstallation、initSixAxisInstallation、getCalcSixAxisWaypoints、identifyTcpPayloadWithSixAxis 和 getFixedSixAxisForce	V2.9	2022-08-12
V3.19	1. 新增接口描述：getDefaultActiveTcp	V2.9	2022-08-29
V3.20	1. 修改 MoveJToPose 和 MoveLToPose 函数声明。	V2.10	2022-09-27
V3.21	1. 修改 initSrv 描述。 2. 新增接口 getLastWarning。	V2.10	2022-10-18
V3.22	1. Thor3 六轴机械臂适配	V2.11	2022-11-21
V3.23	1. 更新接口 setFunctionOptI4 描述 2. 补充接口 getJointsSoftLimitRange 和 setJointsSoftLimitRange 的描述	V2.11	2022-12-29
V3.24	1. 新增接口 getTcpPayloadWithSixAxis、setTcpPayloadWithSixAxis、getThresholdTorque、setThresholdTorque	V2.12	2023-01-13
V3.25	1. 新增接口 setStaticFricion、getStaticFriction	V2.12	2023-01-20
V3.26	1. 新增接口 setPredictMoveMode、getPredictMoveMode	V2.12	2023-02-03
V3.27	2. 新增接口 setSafetySpeedLimit、getSafetySpeedLimit	V2.13	2023-03-07
V3.28	1. 新增接口 getEncoderBatteryLevel、getWarningList、formatWarning、cleanWarning，移除 getLastWarning	V2.13	2023-03-10
V3.2	1. 新增接口 setSafetyCartZone、getSafetyCartZone	V2.1	2023

9		3	-03-17
V3.30	1. 新增接口 calibrateSixAxisSensor	V2.13	2023-03-27
V3.31	1. 新增错误码 -2232：ERROR_CODE_VEL_OR_ACC_PARAMETER_OUT_OF_RANGE 2. 修改 setStaticFriction 的 torque 参数范围 3. 删除附录 E（线程安全函数列表）	V2.13	2023-04-06
V3.32	1. 新增接口 calcInstallAngle 2. 新增错误码 -2233 : ERROR_CODE_POINTS_CANT_FORM_PLANE	V2.14	2023-05-10
V3.33	1. 新增接口 setFunctionOptR8、getFunctionOptR8、updateRealtimeVirtualWall	V2.14	2023-05-24

该操作库函数的所有输入输出参数，均采用国际单位，即力（N），扭矩（N.m），电流（A），长度（m），线速度（m/s），线加速度（m/s^2），角度（rad），角速度（rad/s），角加速度（rad/s^2），时间（s）。如无特殊说明，所有输入输出参数均为轴角或轴角转换的齐次矩阵。另外，文档中涉及关节个数的位置均用 JOINT_NUM 表示，针对 Diana，JOINT_NUM=7，针对 Thor，JOINT_NUM=6。

该操作库所有函数均为线程安全函数。

1 initSrvNetInfo

void initSrvNetInfo(srv_net_st* pInfo)
初始化网络结构体，这会使得全部端口随机分配。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** pInfo： 网络结构体，包含 IP 地址以及所需要的全部端口号。 **返回值：** 无。

调用示例：
srv_net_st* pInfo = new srv_net_st();
initSrvNetInfo(pInfo);
strcpy(pInfo->SrvIp,"192.168.10.75");
...
delete pInfo;
pInfo = nullptr;

2 initSrv

int initSrv(FNCERRORCALLBACK fnError, FNCSTATECALLBACK fnState, srv_net_st * pinfo)
初始化 API，完成其他功能函数使用前的初始化准备工作。 **注：从 2.10 版本开始，系统加入磁盘阵列状态检查。磁盘阵列存在三种状态：可用、部分可用和不可用。initSrv 过程会受到影响，当磁盘阵列中磁盘全部不可用时会自动调用 destroySrv 释放连接，由于释放连接后，lastError 也将被释放，此时本动态库会打印一条 log（"<ip address> 's initSrv failed, RAID state = 2"）提示磁盘阵列中磁盘全部不可用导致 initSrv 失败（返回-1），同时错误码-1018（ERROR_CODE_RAID_INVALID）将传递给 fnError 回调函数。但是该错误码不可以通过 getLastError 获取；当磁盘阵列中部分磁盘可用时，系统会告警，但不影响系统继续使用，getLastError 返回 0。** 适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

fnError：错误处理回调函数。函数声明形式：void fnError(int e,const char *strIpAddress) 其中 e 为错误码（包含通信错误例如版本不匹配，链路错误例如网络断开，硬件故障例如编码器错误等），可调用 formatError 获取字符串提示信息。fnError 函数会用于多线程中实时反馈，所以尽量不要在函数实现中使用 sleep 函数之类会阻塞线程的操作。

fnState：robot state 回调函数。回调函数参数名为 StrRobotStateInfo 的结构体，包含：关节角度数组（jointPos），关节角速度数组（jointAngularVel），关节角电流当前值数组（jointCurrent），关节角扭矩数组（jointTorque），TCP 位姿向量（tcpPos），TCP 外部力（tcpExternalForce），是否发生碰撞标志（bCollision），TCP 外部力是否有效标志（bTcpForceValid），TCP 六维力数组（tcpForce）和轴空间力数组（jointForce）。

pinfo：srv_net_st 结构体指针，用于配置本地连接服务器、心跳服务和状态反馈服务的端口号信息及服务器 IP。端口号如果传 0 则由系统自动分配。

返回值：

0：成功。

-1：失败。

调用示例：

```
#include  "DianaAPI.h "
const char* strIpAddress =  "192.168.10.75";
void logRobotState(StrRobotStateInfo *pinfo,const char *strIpAddress)
{
   static int staCnt = 1;
   if((staCnt++ % 1000 == 0) && pinfo)
   {
      for(int i = 0; i < JOINT_NUM; ++i)
      {
         printf( "jointPos[%d] = %f \n ", i, pinfo->jointPos[i]);
         printf( "jointCurrent [%d] = %f \n ", i, pinfo-> jointCurrent [i]);
         printf( "jointTorque [%d] = %f \n ", i, pinfo-> jointTorque [i]);
         if(i < 6)
         {
            printf( "tcpPos [%d] = %f \n ", i, pinfo-> tcpPos [i]);
         }
      }
   }
}
void errorControl(int e,const char *strIpAddress)
```

```
{
   const char * strError = formatError(e, strIpAddress);  //该函数后面会介绍
     printf( "error code (%d):%s\n", e, strError);
}

srv_net_st * pinfo1 = new srv_net_st();
srv_net_st * pinfo2 = new srv_net_st();
memset(pinfo1 ->SrvIp, 0x00, sizeof(pinfo1 ->SrvIp));
memset(pinfo2 ->SrvIp, 0x00, sizeof(pinfo2 ->SrvIp));
memcpy(pinfo1 ->SrvIp, "172.168.10.75", strlen("172.168.10.75"));
memcpy(pinfo2 ->SrvIp, "172.168.10.76", strlen("172.168.10.76"));
pinfo1->LocHeartbeatPort = 0;
pinfo1->LocRobotStatePort = 0;
pinfo1->LocSrvPort = 0;
pinfo2->LocHeartbeatPort = 0;
pinfo2->LocRobotStatePort = 0;
pinfo2->LocSrvPort = 0;
int ret = initSrv(errorControl, logRobotState,  pinfo1);
if(ret < 0)
{
   printf( "172.168.10.75 initSrv failed! Return value = %d\n ", ret);
}
ret = initSrv(errorControl, logRobotState,  pinfo2);
if(ret < 0)
{
   printf( "172.168.10.76 initSrv failed! Return value = %d, lastError = %d\n ", ret,
getLastError(strIpAddress ));
}
……do something ……
destroySrv(); //这里不给 ip 会销毁所有已注册连接
if(pinfo1)
{
  delete pinfo1;
  pinfo1 = nullptr;
}
if(pinfo2)
{
  delete pinfo2;
```

```
  pinfo2 = nullptr;
}
```

3 destroySrv

int destroySrv(const char* strIpAddress = " ")

结束调用 API，用于结束时释放指定 IP 地址机械臂的资源。如果该函数未被调用就退出系统（例如客户端程序在运行期间崩溃），服务端将因为检测不到心跳而认为客户端异常掉线，直至客户端再次运行，重新连接。除此之外不会引起严重后果。

注意：该 API 被调用以后，除了 initSrv 和 initSrvNetInfo 之外的所有 API 将无法调用。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

strIpAddress:可选参数，需要释放服务资源的机械臂的 IP 地址字符串，如果为空，则会释放全部已经成功 initSrv 的机械臂的资源。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
int ret = destroySrv(strIpAddress);
if(ret < 0)
{
   printf( "destroySrv failed! Return value = %d\n ", ret);
}
```

4 setPushPeriod

int setPushPeriod(int Period,const char* strIpAddress = " ")

设置指定 IP 地址机械臂的数据推送周期。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

intPeriod：输入参数。推送周期，单位为 ms。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
setPushPeriod(10,strIpAddress);
if(ret < 0)
{
   printf("setPushPeriod failed! Return value = %d\n", ret);
}
```

5 moveTCP

int moveTCP(tcp_direction_e d, double v, double a, double * active_tcp=nullptr, const char * strIpAddress = " ")

手动移动指定 IP 地址的机械臂末端中心点。该函数会立即返回，停止运动需要调用 stop 函数。现支持在不同工具坐标系下移动，由传入的 active_tcp 决定。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

d：表示移动方向的枚举类型，枚举及其含义如下：

1. T_MOVE_X_UP 表示沿 x 轴正向；
2. T_MOVE_X_DOWN 表示沿 x 轴负向；
3. T_MOVE_Y_UP 表示沿 y 轴正向；
4. T_MOVE_Y_DOWN 表示沿 y 轴负向；
5. T_MOVE_Z_UP 表示沿 z 轴正向；
6. T_MOVE_Z_DOWN 表示沿 z 轴负向。

v：速度，单位：m/s。

a：加速度，单位：m/s^2。

active_tcp：工具坐标系对应的位姿向量，大小为 6 的数组，为空时，将使用基坐标系。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
tcp_direction_e dtype =T_MOVE_X_UP;
double vel = 0.1;
double acc = 0.2;
const char* strIpAddress =  "192.168.10.75";
int ret = moveTCP(dtype, vel, acc, nullptr, strIpAddress);
if(ret < 0)
{
```

```
    printf( "moveTCP failed! Return value = %d\n ", ret);
}
M_SLEEP(3000);
stop(strIpAddress);
```

6 rotationTCP

```
int rotationTCP(tcp_direction_e d, double v, double a, double * active_tcp=nullptr, const char * strIpAddress =  " ")
```

使指定的 IP 地址的机械臂绕末端中心点变换位姿。该函数会立即返回，停止运动需要调用 stop 函数。现支持在不同工具坐标系下旋转，由传入的 active_tcp 决定。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

d：表示旋转方向的枚举类型，枚举及其含义如下：

1. T_MOVE_X_UP 表示沿 x 轴正向旋转；
2. T_MOVE_X_DOWN 表示沿 x 轴负向旋转；
3. T_MOVE_Y_UP 表示沿 y 轴正向旋转；
4. T_MOVE_Y_DOWN 表示沿 y 轴负向旋转；
5. T_MOVE_Z_UP 表示沿 z 轴正向旋转；
6. T_MOVE_Z_DOWN 表示沿 z 轴负向旋转。

v：速度，单位：rad/s。

a：加速度，单位：rad/s^2。

active_tcp：d 参数的参考坐标系（基于法兰坐标系），大小为 6 的数组（位置和旋转矢量（轴角）），旋转时仅参考方向，旋转中心是系统当前工具中心点；为空时使用基坐标系，旋转中心仍是系统当前工具中心点。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
tcp_direction_e dtype = T_MOVE_X_UP;
double vel = 0.1;
double acc = 0.2;
const char* strIpAddress =  "192.168.10.75";
int ret = rotationTCP(dtype, vel, acc, nullptr, strIpAddress);
if(ret < 0)
{
    printf( "rotationTCP failed! Return value = %d\n ", ret);
```

```
}
M_SLEEP(3000);
stop(strIpAddress);
```

7 moveJoint

moveJoint(joint_direction_e d, int i, double v, double a, const char * strIpAddress = " ")
手动控制指定 IP 地址的机械臂关节移动。该函数会立即返回，停止运动需要调用 stop 函数。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** d：表示关节移动方向的枚举类型。T_MOVE_ UP 表示关节沿正向旋转；T_MOVE_DOWN 表示关节沿负向旋转。 i：关节索引号。 v：速度，单位：rad/s。 a：加速度，单位：rad/s^2。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。

调用示例：
joint_direction_e dtype =T_MOVE_UP;
int index = 1;
double vel = 0.8;
double acc = 0.8;
const char* strIpAddress = "192.168.10.75";
int ret = moveJoint(dtype, index, vel, acc, strIpAddress);
if(ret < 0)
{
printf("moveJoint failed! Return value = %d\n ", ret);
}
M_SLEEP(3000);
stop(strIpAddress);

8 moveJToTarget

int moveJToTarget(double *joints, double v, double a, const char *strIpAddress = " ")

控制指定 IP 地址的机械臂以 JOINT_NUM 个关节角度为终点的 moveJ。该函数会立即返回，停止运动需要调用 stop 函数。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

joints：终点关节角度数组首地址。

v：速度，单位：rad/s。

a：加速度，单位：rad/s^2。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
void wait_move(const char* strIpAddress)
{
   M_SLEEP(20);
   while (true)
   {
      const char state = getRobotState(strIpAddress);
      if (state != 0)
      {
         break;
      }
      else
      {
         M_SLEEP(1);
      }
   }
   stop(strIpAddress);
}
double joints[JOINT_NUM] = {0.0};
double vel = 0.2;
double acc = 0.4;
int ret = moveJToTarget(joints, vel, acc, strIpAddress);
if(ret < 0)
{
   printf( "moveJToTarget failed! Return value = %d\n ", ret);
}
wait_move(strIpAddress);
```

9 moveJToPose

```
int moveJToPose(double *pose, double v, double a, double *active_tcp=nullptr, const char
```

*strIpAddress = " ", const constrain_type_e eConstrainType = E_CONSTRAIN_TYPE_NONE, const double dblConstrainValue = 0.0)
控制指定 IP 地址的机械臂，以工具中心点位姿为终点的 moveJ，可选择为该移动添加约束。该函数会立即返回，停止运动需要调用 stop 函数。如果移动失败可以通过 getLastError 查询失败原因。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** pose：终点位姿数组首地址，数组长度为 6。保存 TCP 坐标（x, y, z）和轴角（rx, ry, rz）组合的矢量数据。pose 可以相对于不同工具坐标系，由传入的 active_tcp 决定。 v：速度，单位：rad/s。 a：加速度，单位：rad/s^2。 active_tcp：需要移动的工具中心点对应的位姿向量（基于法兰坐标系），大小为 6 的数组（位置和旋转矢量（轴角））；为空时将移动系统当前工具的中心点至 pose。（注意：此处 active_tcp 代表需要移动的工具） strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 eConstrainType：约束类型。其中 E_CONSTRAIN_TYPE_NONE 表示无约束；E_CONSTRAIN_LOCKED_J3 表示系统会锁定三轴求逆解并以 moveJToTarget 的方式移动到目标位置；E_CONSTRAIN_LOCKED_J7 表示系统会锁定七轴求逆解并以 moveJToTarget 的方式移动到目标位置。该变量的默认值为无约束，即 E_CONSTRAIN_TYPE_NONE。 dblConstrainValue：约束值。当 eConstrainType 为 E_CONSTRAIN_LOCKED_J3 或 E_CONSTRAIN_LOCKED_J7 时该值有效，表示对应类型下被锁轴的角度，单位：rad。该变量的默认值为 0.0。 **返回值：** 0：成功。 -1：失败。

调用示例：
double poses[6] = { 0.65, 0, 0.65, 0,0,0};
double vel = 0.2;
double acc = 0.4;
const char* strIpAddress = "192.168.10.75";
int ret = moveJToPose(poses , vel, acc, nullptr, strIpAddress, E_CONSTRAIN_LOCKED_J3, -PI/3);
wait_move(strIpAddress);

10 moveJ

moveJ
宏定义，默认匹配 moveJToTarget。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** 同 moveJToTarget。 **返回值：** 0：成功。 -1：失败。

调用示例：
double joints[JOINT_NUM] = {0.0};
double vel = 0.2;
double acc = 0.4;
const char* strIpAddress = "192.168.10.75";
int ret = moveJ (joints, vel, acc, strIpAddress);
if(ret < 0)
{
printf("moveJ failed! Return value = %d\n ", ret);
}
wait_move(strIpAddress);

11 moveL

moveL
宏定义，默认匹配 moveLToPose。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** 同 moveLToPose。 **返回值：** 0：成功。 -1：失败。

调用示例：
double poses[6] = {0.087,0.0,1.0827,0.0,0.0,0.0};
double vel = 0.2;
double acc = 0.4;
const char* strIpAddress = "192.168.10.75";

```
int ret = moveL (poses, vel, acc, nullptr, strIpAddress);
if(ret < 0)
{
   printf( "moveL failed! Return value = %d\n ", ret);
}
wait_move(strIpAddress);
```

12 moveLToTarget

int moveLToTarget(double *joints, double v, double a, const char *strIpAddress = " ")

控制指定 IP 地址的机械臂以 JOINT_NUM 个关节角度为终点的 moveL。该函数会立即返回，停止运动需要调用 stop 函数。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

joints：终点关节角度数组首地址，数组长度为 JOINT_NUM。

v：速度，单位：m/s。

a：加速度，单位：m/s^2。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
#define DEGREE_TO_RAD(x) ((x)*PI / 180.0)
//以七轴机械臂为例：
double joints[7] = {DEGREE_TO_RAD(9.100),DEGREE_TO_RAD(-
18.901),DEGREE_TO_RAD(3.847),DEGREE_TO_RAD(133.001),DEGREE_TO_RAD(3.272
),DEGREE_TO_RAD(-74.400),DEGREE_TO_RAD(11.539)};
double vel = 0.2;
double acc = 0.4;
const char* strIpAddress =  "192.168.10.75";
int ret = moveLToTarget(joints, vel, acc);
if(ret < 0)
{
   printf( "moveLToTarget failed! Return value = %d\n ", ret);
}
wait_move(strIpAddress);
```

13 moveLToPose

int moveLToPose(double *pose, double v, double a, double *active_tcp=nullptr, const char *strIpAddress = " ", const constrain_type_e eConstrainType = E_CONSTRAIN_TYPE_NONE, const double dblConstrainValue = 0.0)
控制指定 IP 地址的机械臂，以工具中心点位姿为终点的 moveL。可选择为该移动添加约束。该函数会立即返回，停止运动需要调用 stop 函数。如果移动失败可以通过 getLastError 查询失败原因。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** pose：终点位姿数组首地址，数组长度为 6，保存 TCP 坐标（x, y, z）和轴角（rx, ry, rz）组合数据。 v：速度，单位：m/s。 a：加速度，单位：m/s^2。 active_tcp：需要移动的工具中心点对应的位姿向量（基于法兰坐标系），大小为 6 的数组（位置和旋转矢量（轴角））；为空时将移动系统当前工具的中心点至 pose。（注意：此处 active_tcp 代表需要移动的工具）。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 eConstrainType：约束类型。其中 E_CONSTRAIN_TYPE_NONE 表示无约束；E_CONSTRAIN_LOCKED_J3 表示系统会锁定三轴求逆解并以 moveLToTarget 的方式移动到目标位置；E_CONSTRAIN_LOCKED_J7 表示系统会锁定七轴求逆解并以 moveLToTarget 的方式移动到目标位置。该变量的默认值为无约束，即 E_CONSTRAIN_TYPE_NONE。 dblConstrainValue：约束值。当 eConstrainType 为 E_CONSTRAIN_LOCKED_J3 或 E_CONSTRAIN_LOCKED_J7 时该值有效，表示对应类型下被锁轴的角度，单位：rad。该变量的默认值为 0.0。 **返回值：** 0：成功。 -1：失败。

调用示例：
double poses[6] = { 0.55, 0, 0.55, 0,0,0 };
double vel = 0.2;
double acc = 0.4;
const char* strIpAddress = "192.168.10.75";

```
int ret = moveLToPose(poses, vel, acc, nullptr, strIpAddress, E_CONSTRAIN_LOCKED_J3, -PI/3);
wait_move();
```

14 speedJ

int speedJ(double *speed, double a, double t, const char *strIpAddress = " ")

控制指定 IP 地址的机械臂进入速度模式，关节空间运动。时间 t 是可选项，如果提供了 t 值，控制指定 IP 地址的机械臂将在 t 时间后减速。如果没有提供时间 t 值，机械臂将在达到目标速度时减速。该函数调用后立即返回。停止运动需要调用 stop 函数。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

speed：关节角速度数组首地址，数组长度为 JOINT_NUM。单位：rad/s。

a：加速度，单位：rad/s^2。

t：时间，单位：s。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
double speeds [JOINT_NUM] = {0.0};
speeds[0]=0.2;
double acc = 0.40;
const char* strIpAddress =  "192.168.10.75";
int ret = speedJ(speeds, acc, 0, strIpAddress);
if(ret < 0)
{
   printf( "speedJ failed! Return value = %d\n ", ret);
}
sleep(2);
stop(strIpAddress);
```

15 speedL

int speedL(double *speed, double *a, double t, double *active_tcp=nullptr, const char *strIpAddress = " ")

控制指定 IP 地址的机械臂进入速度模式，进行笛卡尔空间下直线运动，支持同步旋转，

但笛卡尔方向必须有速度或者加速度才能旋转。时间 t 是可选项，如果提供了 t 值，机械臂将在 t 时间后减速。如果没有提供时间 t 值，机械臂将在达到目标速度时减速。该函数调用后立即返回。停止运动需要调用 stop 函数。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

speed：基坐标系下的工具空间速度（位置和旋转），数组长度为 6,其中前 3 个单位为 m/s，后 3 个单位为 rad/s。位置和旋转均参考基坐标系，旋转时的旋转中心可由 active_tcp 指定。

a：加速度，单位：m/s^2，rad/s^2。

t：时间，单位：s。

active_tcp：基于法兰坐标系的位姿，指定旋转时的旋转中心，大小为 6 的数组（位置+旋转矢量（轴角））；为空时旋转中心是系统当前工具中心点。（注意：机械臂做直线运动时中心点会随动，所以无旋转运动的情况下，此参数看不出影响）

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
double speeds[6] = {0.1,0.0,0.0,0.0,0.0,0.0};
double acc[2] = {0.30, 0.50};
const char* strIpAddress =  "192.168.10.75";
int ret = speedL(speeds, acc, 0, nullptr, strIpAddress);
if(ret < 0)
{
   printf( "speedL failed! Return value = %d\n ", ret);
}
sleep(2);
stop(strIpAddress);
```

16 freeDriving

```
int freeDriving(bool enable, const char *strIpAddress =  " ")
```

实现控制指定 IP 地址的机械臂正常模式与零力驱动模式之间的切换。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

enable：bool 变量，是否进入零力驱动模式，true 表明进入零力驱动，false 为退出零力驱动进入正常模式。只有在正常模式下，才可以控制机械臂运动。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
int ret = freeDriving(true, strIpAddress);
if(ret < 0)
{
   printf( "freeDriving failed! Return value = %d\n ", ret);
}
```

17 stop

int stop(const char *strIpAddress = " ")

控制指定 IP 地址的机械臂停止当前执行的任务。将会以最大加速度停止。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
int ret = stop(strIpAddress);
if(ret < 0)
{
   printf( "stop failed! Return value = %d\n ", ret);
}
```

18 forward

int forward(double *joints, double *pose, double *active_tcp=nullptr, const char *strIpAddress = " ")

正解函数，针对指定 IP 地址机械臂，由传入的关节角都计算出的正解 TCP 位姿。现支持在不同工具坐标系下求正解，由传入的 active_tcp 决定。从 2.14 版本开始，正解 forward 函数已支持本地化计算，但因工具坐标系推送存在 1 个周期的延迟(参考 setPushPeriod)，在设定完工具坐标系后(setDefaultActiveTcp 或 setDefaultActiveTcpPose)立即求正解需要至少等待 1 个推送周期。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

joints：传入关节角度数组首地址，数组长度为 JOINT_NUM。单位：rad。

pose：输出位姿数组首地址，数组长度为 6。用于传递转化后的结果，数据为包含 active_tcp 坐标（x, y, z）和旋转矢量（轴角坐标）组合。

active_tcp：工具坐标系对应的位姿向量，大小为 6 的数组，为空时，将使用默认的工具坐标系 default_tcp。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
double joints[JOINT_NUM] = {0.0};
double pose[6] = {0.0};
const char* strIpAddress =  "192.168.10.75";
int ret = forward(joints, pose, nullptr, strIpAddress);
if(ret < 0)
{
   printf( "forward failed! Return value = %d\n ", ret);
}
```

19 inverse

```
int inverse(double *pose, double *joints, double *active_tcp=nullptr, const char *strIpAddress = " ")
```

逆解函数，针对指定 IP 地址机械臂，根据当前关节角，通过 TCP 位姿计算出最佳逆解关节角度。现支持在不同工具坐标系下求逆解，由传入的 active_tcp 决定。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

pose：输入位姿数组首地址，数据为包含 active_tcp 坐标（x, y, z）和旋转矢量（轴角坐

标）组合。 joints：输出关节角度数组首地址，用于传递转换的结果。 active_tcp：工具坐标系对应的位姿向量，大小为 6 的数组，为空时，将使用默认的工具坐标系 default_tcp。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。

调用示例：
double pose[6] = {0.599429, 0.004977, 0.704431, -0.000084, 0.000120,- 0.000048};
double joints[JOINT_NUM] = {0.0};
const char* strIpAddress = "192.168.10.75";
int ret = inverse(pose, joints, nullptr,strIpAddress);
if(ret < 0)
{
printf("inverse failed! Return value = %d\n ", ret);
}

20 getJointPos

int getJointPos(double *joints, const char *strIpAddress = " ")
获取指定 IP 地址机械臂各个关节角度的位置，库初始化后，后台会自动同步机械臂状态信息，因此所有的监测函数都是从本地缓存取数。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** joints：输出关节角数组首地址，数组长度为 JOINT_NUM。用于传递获取到的结果。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。
调用示例： double joints[JOINT_NUM] = {0.0}; const char* strIpAddress = "192.168.10.75"; int ret = getJointPos(joints, strIpAddress);

```
if(ret < 0)
{
   printf( "getJointPos failed! Return value = %d\n ", ret);
}
```

21 getJointAngularVel

int getJointAngularVel(double *vels, const char *strIpAddress = " ")
获取指定 IP 地址机械臂当前各关节的角速度。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** vels：输出关节角速度数组首地址，数组长度为 JOINT_NUM。用于传递获取到的结果。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。

调用示例：
double speeds[JOINT_NUM] = {0.0};
const char* strIpAddress = "192.168.10.75";
int ret = getJointAngularVel(speeds,strIpAddress);
if(ret < 0)
{
printf("getJointAngularVel failed! Return value = %d\n ", ret);
}

22 getJointCurrent

int getJointCurrent(double *joints, const char *strIpAddress = " ")
获取当前关节电流。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** joints：输出关节角度数组首地址，数组长度为 JOINT_NUM。用于传递获取到的结果。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。

调用示例：

```
double joints[JOINT_NUM] = {0.0};
const char* strIpAddress= "192.168.10.75";
int ret = getJointCurrent(joints, strIpAddress);
if(ret < 0)
{
   printf( "getJointCurrent failed! Return value = %d\n ", ret);
}
```

23 getJointTorque

int getJointTorque(double *torques, const char *strIpAddress = " ")

获取指定 IP 地址机械臂各关节真实扭矩数据，即减去零偏的数据。

适用臂型：通用医疗臂、定制医疗臂

参数：

torques：输出关节阻抗数组首地址，数组长度为 JOINT_NUM。用于传递获取到的结果。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
double torques[JOINT_NUM] = {0.0};
const char* strIpAddress =  "192.168.10.75";
int ret = getJointTorque(torques, strIpAddress);
if(ret < 0)
{
   printf( "getJointTorque failed! Return value = %d\n ", ret);
}
```

24 getTcpPos

int getTcpPos(double *pose, const char *strIpAddress = " ")

获取指定 IP 地址机械臂当前 TCP 位姿数据，TCP 位姿可被 setDefaultActiveTcp 函数改变。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

pose：输出关节 TCP 位姿数组首地址，数组长度为 6。用于传递获取到的结果，其中，后三个角度为轴角。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
double poses[6] = {0.0};
const char* strIpAddress =  "192.168.10.75";
int ret = getTcpPos(poses, strIpAddress);
if(ret < 0)
{
   printf( "getTcpPos failed! Return value = %d\n ", ret);
}
```

25 getTcpExternalForce

```
double getTcpExternalForce(const char *strIpAddress =  " ")
```

获取指定 IP 地址机械臂 TCP 实际感受到的合力大小，TCP 位姿可被 setDefaultActiveTcp 函数改变。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

返回力的大小。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
double force = getTcpExternalForce(strIpAddress);
```

26 releaseBrake

```
int releaseBrake(const char *strIpAddress =  " ")
```

打开指定 IP 地址机械臂的抱闸，启动机械臂。调用该接口后，需要调用者延时 2s 后再做其他操作。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
int ret = releaseBrake (strIpAddress);
if(ret < 0)
{
   printf( "releaseBrake failed! Return value = %d\n ", ret);
}
```

27 holdBrake

int holdBrake(const char *strIpAddress = " ")

关闭指定 IP 地址机械臂的抱闸，停止机械臂。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
int ret = holdBrake (strIpAddress);
if(ret < 0)
{
   printf( "holdBrake failed! Return value = %d\n ", ret);
}
```

28 changeControlMode

int changeControlMode(mode_e m, const char *strIpAddress = " ")

控制指定 IP 地址机械臂的模式切换。

适用臂型：通用医疗臂、定制医疗臂

参数：

m：枚举类型。

1. T_MODE_INVALID 无意义；
2. T_MODE_POSITION 位置模式；
3. T_MODE_JOINT_IMPEDANCE 关节空间阻抗模式；
4. T_MODE_CART_IMPEDANCE 笛卡尔空间阻抗模式。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
int ret = changeControlMode(T_MODE_POSITION, strIpAddress);
if(ret < 0)
{
   printf( "changeControlMode failed! Return value = %d\n ", ret);
}
```

29 getLibraryVersion

unsigned short getLibraryVersion()

获取当前库的版本号。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

无。

返回值：

当前版本号,高 8 位为主版本号，低 8 位为次版本号。

调用示例：

```
unsigned short uVersion = getLibraryVersion();
```

30 formatError

const char *formatError(int e, const char* strIpAddress = " ")

获取指定 IP 地址机械臂的错误码 e 的字符串描述，该错误码在初始化指定的回调函数中会作为形参传入，也可以在函数调用失败后查询得到。对于错误码为-2001 的硬件错误，会延时回馈，一般建议对此类错误延时 100 毫秒后调用 formatError 函数获取具体硬件错误提示信息，否则将提示 "refresh later ... "而看不到具体内容。

适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** e：错误码。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 错误描述信息。

调用示例：
int e = -1003;
const char* msg = formatError(e, "192.168.10.75");
printf("%s\n ",msg);

31 getLastError

int getLastError(const char* strIpAddress = " ")
返回指定 IP 地址机械臂最近发生的错误码。该错误码会一直保存，确保可以查询得到，直至库卸载，因此，当库函数调用失败后，如果想知道具体的错误原因，应该调用该函数获取错误码。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：没有错误。 其它值：具体错误码。

调用示例：
const char* strIpAddress = "192.168.10.75";
int e = getLastError(strIpAddress);
printf("getLastError code [%d] \n",e);

32 setLastError

int setLastError(int e, const char* strIpAddress = " ")
重置指定 IP 地址机械臂错误码。将系统中记录的错误码重置为 e。该操作完成后会直接影响 getLastError 的调用结果。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：**

e：错误码。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

错误码。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
int e = setLastError(0, strIpAddress);
printf("setLastError code [%d] \n",e);
```

33 setDefaultActiveTcp

int setDefaultActiveTcp(double *default_tcp , const char *strIpAddress = " ")

设置指定 IP 地址字符串的默认工具坐标系。在没有调用该函数时，默认工具中心点为法兰盘中心，调用该函数后，默认的工具坐标系将被改变。该函数将会改变 moveTCP，rotationTCP，moveJToPos，moveLToPose，speedJ，speedL，forward，inverse，getTcpPos，getTcpExternalForce 的默认行为。从 2.14 版本开始，正解 forward 函数已支持本地化计算，但因工具坐标系推送存在 1 个周期的延迟(参考 setPushPeriod)，在设定完工具坐标系后(setDefaultActiveTcp 或 setDefaultActiveTcpPose)立即求正解需要至少等待 1 个推送周期。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

default_tcp：输入参数，TCP 相对于末端法兰盘的 4*4 齐次变换矩阵的首地址，数组长度为 16。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
double default_tcp2[16] = {1,0,0,0,  0,1,0,0,  0,0,1,0,  0,0,-0.220,1};
int ret = setDefaultActiveTcp(default_tcp2, strIpAddress);
if(ret < 0)
{
   printf( "setDefaultActiveTcp failed! Return value = %d\n ", ret);
```

```
}
```

34 getLinkState

int getLinkState(const char *strIpAddress = " ")

获取与指定 IP 地址机械臂间的链路状态。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：链路正常。

-1：链路断开。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
int ret = getLinkState(strIpAddress);
if(ret == 0)
{
    printf( "network connected \n ");
}
else
{
    printf( "network disconnected \n ");
}
```

35 getTcpForce

int getTcpForce(double *forces, const char *strIpAddress = " ")

获取指定 IP 地址机械臂的 TCP 所受外部六维力，TCP 位姿可被 setDefaultActiveTcp 函数改变。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

forces：工具中心点处六维力数组的首地址，数组长度为 6。用于传递获取到的结果。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0： 获取到六维力且六维力有效。

-1： 获取不到六维力矩，或六维力数值无效（发生奇异）。

调用示例：

```
double tcpForce[6];
const char* strIpAddress =  "192.168.10.75";

int ret = getTcpForce(tcpForce,strIpAddress);
if(ret == 0)
{
printf( "get tcp force: %f, %f, %f, %f, %f, %f \n ", tcpForce[0] , tcpForce[1] , tcpForce[2] , tcpForce[3] , tcpForce[4] , tcpForce[5] );
}
else
{
printf( "get tcp force failed \n ");
}
```

36 getJointForce

int getJointForce(double *forces, const char *strIpAddress = " ")
获取指定 IP 地址机械臂的轴空间 JOINT_NUM 个关节所受力。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** forces：关节轴力矩数组的首地址，数组长度为 JOINT_NUM。用于传递获取到的结果。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：获取成功。 -1：获取失败。

调用示例：
double jointForce[JOINT_NUM];
const char* strIpAddress = "192.168.10.75";
int ret = getJointForce (jointForce, strIpAddress);
if(ret == 0)
{
printf("get joint force: %f, %f, %f, %f, %f, %f, %f \n ", jointForce [0] , jointForce [1] , jointForce [2] , jointForce [3] , jointForce [4] , jointForce [5] , jointForce [6]);
//以 7 轴机械臂为例
}
else

```
{
printf( "get joint force failed \n ");
}
```

37 isCollision

bool isCollision(const char *strIpAddress = " ")

从轴空间判断指定 IP 地址机械臂是否发生碰撞。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

true：机械臂发生碰撞。

false：机械臂未发生碰撞。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
if(isCollision(strIpAddress))
{
    printf( "Warning: Robot got collision\n ");
}
else
{
    printf( "Robot is running\n ");
}
```

38 initDHCali

int initDHCali(double *tcpMeas, double *jntPosMeas, unsigned int nrSets, const char *strIpAddress = " ")

根据输入的关节角以及 TCP 位置数组计算指定 IP 地址机械臂的 DH 参数。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

tcpMeas：输入参数。TCP 位置数据数组的首地址，数组长度为 3 * nrSets。每组数据为[x,y,z]，共 nrSets 组。单位：米。

jntPosMeas：输入参数。关节角位置数组的首地址，数组长度为 JOINT_NUM * nrSets，每组数据为各关节角位置信息，共 nrSets 组。单位：弧度。

nrSets：输入参数。测量样本数量，最少 32 组，至少保证大于或等于需要辨识的参数。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：正常。

-1：失败。

调用示例：

```
int rowNo_refData = 32;
double jntMeas[JOINT_NUM * 32] = {…};
double tcpMeas[96] = {…};
const char* strIpAddress =  "192.168.10.75";
if (initDHCali(tcpMeas, jntMeas, rowNo_refData, strIpAddress) != 0)
{
   printf("DHCaliFcns.InitDHCali failed!\n");
}
```

39 getDHCaliResult

int getDHCaliResult(double *rDH, double *wRT, double *tRT, double *confid, const char *strIpAddress = " ")

获取指定 IP 地址机械臂 DH 参数的计算结果。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

rDH：输出参数。机械臂各关节 DH 参数数组的首地址，数组长度为 4*JOINT_NUM。每 JOINT_NUM 个数为一组，共四组数据[a，alpha，d，theta]。单位：rad、m。

wRT：输出参数。机械臂基坐标系相对于世界坐标系下的位姿数组的首地址，数组长度为 6。位姿数据[x, y, z, Rx，Ry，Rz]。单位：rad、m。

tRT： 输出参数。靶球在法兰坐标系下的位置描述数组的首地址，数组长度为 3。数组为靶球位置坐标[x,y,z]。单位：m。

confid：输出参数。绝对定位精度参考值数组的首地址，数组长度为 2。其中，第一个值为标定前绝对定位精度，第二个值为标定后绝对定位精度。单位：m。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：获取成功。

1：获取结果精度可能较低。

-1：获取失败，发生异常。

调用示例：

```
double rDH[4*JOINT_NUM], wRT[6], tRT[3 ], confid[2];
const char* strIpAddress =  "192.168.10.75";
if (getDHCaliResult(rDH, wRT, tRT, confid, strIpAddress) < 0)
{
   printf("DHCaliFcns.getDHCaliResult failed.\n");
}
```

40 setDH

int setDH(double *a, double *alpha, double *d, double *theta, const char *strIpAddress = " ")

设置指定 IP 地址机械臂当前 DH 参数。特别注意，**错误的参数设置**可能引起机械臂损坏，需谨慎设置！

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

a：输入参数。各关节的 a 参数数组的首地址，数组长度为 JOINT_NUM。

alpha：输入参数。各关节的 alpha 参数数组的首地址，数组长度为 JOINT_NUM。

d：输入参数。各关节的 d 参数数组的首地址，数组长度为 JOINT_NUM。

theta：输入参数。各关节的 theta 参数数组的首地址，数组长度为 JOINT_NUM。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
double a[JOINT_NUM], alpha[JOINT_NUM], d[JOINT_NUM], theta[JOINT_NUM];
const char* strIpAddress =  "192.168.10.75";
if (setDH(a, alpha, d, theta, strIpAddress) < 0)
{
      printf("DHCaliFcns.setDH failed.\n");
}
```

41 setWrd2BasRT

int setWrd2BasRT(double *RTw2b, const char *strIpAddress = " ")

初始化世界坐标系到指定 IP 地址机械臂坐标系的平移和旋转位姿。用于 DH 参数标定前设置，若用户不能提供此参数，DH 参数标定功能依旧可以使用。如果调用此函数则使用用户自定义的位姿。特别注意，此功能每次移动机械臂与激光跟踪仪都需要重新计

算，使用错误的参数可能引起 DH 参数计算不准确或标定异常。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** RTw2b：输入参数。世界坐标系到机械臂坐标系的平移和旋转位姿数组的首地址，数组长度为 6。单位：米和弧度。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。

调用示例：
double wRT[6] = {…};
const char* strIpAddress = "192.168.10.75";
if (setWrd2BasRT(wRT, strIpAddress) < 0)
{
printf("DHCaliFcns. setWrd2BasRT failed.\n");
}

42 setFla2TcpRT

int setFla2TcpRT(double *RTf2t, const char *strIpAddress = " ")
初始化指定 IP 地址机械臂法兰坐标系到工具坐标系的平移位置。用于 DH 参数标定前设置，若用户不能提供此参数，DH 参数标定功能依旧可以使用。如果调用此函数则使用用户自定义的位姿。特别注意，此功能每次移动机械臂与激光跟踪仪都需要重新计算，使用错误的参数可能引起 DH 参数计算不准确或标定异常。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** RTf2t：输入参数。初始化法兰坐标系到工具坐标系的平移位置数组的首地址，数组长度为 3，位置信息数据[x,y,z]。单位：米。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。
调用示例： double Fla[3] = {…};

```
const char* strIpAddress =  "192.168.10.75";
if (setFla2TcpRT (Fla, strIpAddress) < 0)
{
    printf("DHCaliFcns. setFla2TcpRT failed.\n");
}
```

43 getRobotState

const char getRobotState(const char *strIpAddress = " ")

获取指定 IP 地址机械臂当前工作状态。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：running（机械臂正在运动或者示教器程序正在运行）。

1：paused（暂停正在运动的机械臂或者暂停正在运行的示教器程序）。

2：idle（空闲状态）。

3：free-driving（进入正常零力驱动模式或者安全零力驱动模式）。

4：null-space-free-driving（进入零空间零力驱动模式）。

5：hold_brake（机械臂关闭抱闸）。

6：error（机械臂发生错误）。

7：error_handling（机械臂的关节超出极限位置）。

8：torque_receiver（扭矩发送模式）。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
const char state = getRobotState(strIpAddress);
if (0 == state)
{
   printf("\t[robot state]:running\n");
}
else if (1 == state)
{
    printf("\t[robot state]:paused\n");
}
else if (2 == state)
{
```

```
        printf("\t[robot state]:idle\n");
}
else
{
        printf("\t[robot state]: unknown state \n");
}
```

44 resume

int resume(const char *strIpAddress = " ")

当指定 IP 地址机械臂发生碰撞或其他原因暂停后，恢复运行时使用。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
if (resume(strIpAddress)  <  0)
{
   printf("Diana API resume failed!\n");
}
```

45 setJointCollision

int setJointCollision(double *collision,const char *strIpAddress = " ")

设置指定 IP 地址机械臂关节空间碰撞检测的力矩阈值。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

collision：输入参数。JOINT_NUM 关节轴力矩阈值数组的首地址，数组长度为 JOINT_NUM。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：设置成功。

-1：设置失败。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
double collision[7] = {200, 200, 200, 200, 200, 200, 200}; //以 7 轴机械臂为例
if (setJointCollision(collision, strIpAddress) < 0)
{
   printf("Diana API setJointCollision failed!\n");
}
```

46 setCartCollision

int setCartCollision(double *collision,const char *strIpAddress = " ")

设置指定 IP 地址机械臂笛卡尔空间碰撞检测的六维力阈值。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

collision：输入参数。在基坐标系下的六维力数组的首地址，数组长度为 6。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：设置成功。

-1：设置失败。

调用示例：

```
double collision[6] = { 200, 200, 200, 200, 200, 200};
const char* strIpAddress =  "192.168.10.75";
if (setCartCollision (collision, strIpAddress) < 0)
{
   printf("Diana API setCartCollision failed!\n");
}
```

47 enterForceMode

int enterForceMode(int intFrameType, double *dblFrameMatrix, double *dblForceDirection, double dblForceValue, double dblMaxApproachVelocity, double dblMaxAllowTcpOffset,const char *strIpAddress = " ")

使指定 IP 地址机械臂进入力控模式。

适用臂型：通用医疗臂、定制医疗臂

参数：

intFrameType：参考坐标系类型。0：基坐标系；1：工具坐标系；2：自定义坐标系（暂不支持）。

dblFrameMatrix：自定义坐标系矩阵（暂不支持），使用时传单位矩阵即可。

dblForceDirection：表达力的方向的数组首地址，数组长度为 3。

dblForceValue：力大小。单位：N。推荐取值范围（1N~100N）。

dblMaxApproachVelocity：最大接近速度。单位：m/s。推荐取值范围（0.01m/s~0.5m/s）。

dblMaxAllowTcpOffset：允许的最大偏移。单位：m。推荐取值范围（0.01m~1m）。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
int iFrameType = 1;
double dblFrameMatrix[16] = {1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1};
double dblForceDir[3]={0,0,-1};
double dblForceValue = 2.0;
double dblMaxVel = 0.1;
double dblMaxOffset = 0.2;
const char* strIpAddress =  "192.168.10.75";
if (enterForceMode(iFrameType, dblFrameMatrix, dblForceDir, dblForceValue, dblMaxVel,
dblMaxOffset,strIpAddress) < 0)
{
   printf("Diana API enterForceMode failed!\n");
}
```

48 leaveForceMode

int leaveForceMode (int intExitMode,const char *strIpAddress = " ")

设置指定 IP 地址机械臂退出力控模式,并设置退出后机械臂的工作模式。

适用臂型：通用医疗臂、定制医疗臂

参数：

intExitMode：控制模式。

0：代表位置模式；

1：代表关节空间阻抗模式；

2：代表笛卡尔空间阻抗模式。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂

时生效。 返回值： 0：成功。 -1：失败。

调用示例：
int intExitMode = 0;
const char* strIpAddress = "192.168.10.75";
if (leaveForceMode (intExitMode,strIpAddress) < 0)
{
printf("Diana API leaveForceMode failed!\n");
}

49 setDefaultActiveTcpPose

int setDefaultActiveTcpPose (double *arrPose, const char *strIpAddress = " ")
设置指定 IP 地址机械臂默认的工具坐标系位姿。在没有调用该函数时，默认工具中心点为法兰盘中心，调用该函数后，默认的工具坐标系将被改变。该函数将会改变 moveTCP，rotationTCP，moveJToPos，moveLToPose，speedJ，speedL，forward，inverse，getTcpPos，getTcpExternalForce 等函数的默认行为。从 2.14 版本开始，正解(参考 forward)函数已支持本地化计算，但因工具坐标系推送存在 1 个周期的延迟(参考 setPushPeriod)，在设定完工具坐标系后(setDefaultActiveTcp 或 setDefaultActiveTcpPose)立即求正解需要至少等待 1 个推送周期。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** arrPose：输入参数。TCP 相对于末端法兰盘的位姿向量的首地址，数组长度为 6，其中，后三个角度为轴角。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。

调用示例：
double pose[6] = {0.1,0.1,0.1,0, 0, 0};
const char* strIpAddress = "192.168.10.75";
if (setDefaultActiveTcpPose (pose,strIpAddress) < 0)
{

```
    printf("Diana API setDefaultActiveTcpPose failed!\n");
}
```

50 setResultantCollision

int setResultantCollision (double force,const char *strIpAddress = " ")

设置指定 IP 地址机械臂笛卡尔空间碰撞检测 TCP 的合力矩阈值。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

force：合力值。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
double force = 8.9;
const char* strIpAddress =  "192.168.10.75";
if (setResultantCollision (force, strIpAddress) < 0)
{
    printf("Diana API setResultantCollision failed!\n");
}
```

51 zeroSpaceFreeDriving

int zeroSpaceFreeDriving (bool enable, const char *strIpAddress = " ")

控制指定 IP 地址机械臂进入或退出零空间自由驱动模式。

适用臂型：通用医疗臂、定制医疗臂

参数：

enable：输入参数。

true——进入零空间自由驱动模式；

false——退出零空间自由驱动模式。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
if (zeroSpaceFreeDriving (true, strIpAddress) < 0)
{
   printf("Diana API zeroSpaceFreeDriving failed!\n");
}
else
{
   printf("Diana API zeroSpaceFreeDriving succeed!\n");
   M_SLEEP(5000);
   zeroSpaceFreeDriving(false, strIpAddress);
}
```

52 createPath

int createPath (int type, unsigned int &id_path, const char *strIpAddress = " ")

为指定 IP 地址机械臂创建一个路段。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

type：输入参数。1 :表示 moveJ, 2：表示 moveL。

id_path：输出参数。用于保存新创建 Path 的 ID。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

详见 addMoveJ 或 addMoveL 调用示例。

53 addMoveL

int addMoveL (unsigned int id_path, double *joints, double vel, double acc, double blendradius, const char *strIpAddress = " ")

向指定 IP 地址机械臂已创建的路段添加 MoveL 路点。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

id_path：输入参数。要添加路点的路径 ID。

joints：输入参数。要添加的路点，即该路点的各关节角度数组的首地址，数组长度为 JOINT_NUM。单位：rad。

vel：moveL 移动到目标路点的速度。单位：m/s。

acc：moveL 移动到目标路点的加速度。单位：m/s^2。

blendradius：交融半径。单位：m。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
//以七轴机械臂为例
#define PI 3.14159265358979323846
#define DEGREE_TO_RAD(x) ((x)*PI / 180.0)
#define RAD_TO_DEGREE(x) ((x)*180.0 / PI)

double dblFirstMoveLPosition [7] = { DEGREE_TO_RAD(0.0), DEGREE_TO_RAD(0.0),
DEGREE_TO_RAD(0.0), DEGREE_TO_RAD(90.0), DEGREE_TO_RAD(0.0),
DEGREE_TO_RAD(-90.0), DEGREE_TO_RAD(0.0) };
double dblSecondMoveLPosition [7] = { DEGREE_TO_RAD(0.0), DEGREE_TO_RAD(0.0),
DEGREE_TO_RAD(0.0), DEGREE_TO_RAD(120.0), DEGREE_TO_RAD(0.0),
DEGREE_TO_RAD(-60.0), DEGREE_TO_RAD(0.0) };
double dblThirdMoveLPosition [7] = { DEGREE_TO_RAD(0.0), DEGREE_TO_RAD(0.0),
DEGREE_TO_RAD(0.0), DEGREE_TO_RAD(150.0), DEGREE_TO_RAD(0.0),
DEGREE_TO_RAD(-30.0), DEGREE_TO_RAD(0.0) };

unsigned int path_id = 0;
printf("start test moveL path.\n");
const char* strIpAddress =  "192.168.10.75";
createPath(2, path_id,strIpAddress);
addMoveL(path_id, dblFirstMoveLPosition, 0.2, 0.2, 0.3,strIpAddress);
addMoveL(path_id, dblSecondMoveLPosition, 0.2, 0.2, 0.3,strIpAddress);
addMoveL(path_id, dblThirdMoveLPosition, 0.2, 0.2, 0.3,strIpAddress);
runPath(path_id,strIpAddress);
destroyPath(path_id,strIpAddress);
wait_move(strIpAddress);
```

54 addMoveJ

```
int addMoveJ (unsigned int id_path, double *joints, double vel_percent, double acc_percent,
```

double blendradius_percent, const char *strIpAddress = " ")

向指定 IP 地址机械臂已创建的路段添加 MoveJ 路点。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

id_path：输入参数。要添加路点的路径 ID。

joints：输入参数。要添加的路点，即该路点的各关节角度数组的首地址，数组长度为 JOINT_NUM。单位：rad。

vel_percent：moveJ 移动到目标路点的速度百分比。（拟改为 rad/s）

acc_percent：moveJ 移动到目标路点的加速度百分比。（拟改为 rad/s^2）

blendradius _percent：交融半径百分比。每个关节位移的一半为交融半径最大值。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
//以七轴机械臂为例
#define PI 3.14159265358979323846
#define DEGREE_TO_RAD(x) ((x)*PI / 180.0)
#define RAD_TO_DEGREE(x) ((x)*180.0 / PI)

double dblFirstPosition[7] = { DEGREE_TO_RAD(0.0), DEGREE_TO_RAD(0.0),
DEGREE_TO_RAD(0.0), DEGREE_TO_RAD(90.0), DEGREE_TO_RAD(0.0),
DEGREE_TO_RAD(-90.0), DEGREE_TO_RAD(0.0) };
double dblSecondPosition[7] = { DEGREE_TO_RAD(0.0), DEGREE_TO_RAD(0.0),
DEGREE_TO_RAD(0.0), DEGREE_TO_RAD(120.0), DEGREE_TO_RAD(0.0),
DEGREE_TO_RAD(-60.0), DEGREE_TO_RAD(0.0) };
double dblThirdPosition[7] = { DEGREE_TO_RAD(0.0), DEGREE_TO_RAD(0.0),
DEGREE_TO_RAD(0.0), DEGREE_TO_RAD(150.0), DEGREE_TO_RAD(0.0),
DEGREE_TO_RAD(-30.0), DEGREE_TO_RAD(0.0) };

unsigned int path_id = 0;
printf("start test moveJ path.\n");
const char* strIpAddress =  "192.168.10.75";
createPath(1, path_id,strIpAddress);
addMoveJ(path_id, dblFirstPosition, 0.2, 0.2, 0.3,strIpAddress);
```

```
addMoveJ(path_id, dblSecondPosition, 0.2, 0.2, 0.3,strIpAddress);
addMoveJ(path_id, dblThirdPosition, 0.2, 0.2, 0.3,strIpAddress);
runPath(path_id,strIpAddress);
destroyPath(path_id,strIpAddress);
wait_move(strIpAddress);
```

55 runPath

int runPath (unsigned int id_path, const char *strIpAddress = " ")

为指定 IP 地址机械臂启动运行设置好的路段。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

id_path：输入参数。要运行的路径 ID。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

详见 addMoveJ 或 addMoveL 调用示例。

56 destroyPath

int destroyPath (unsigned int id_path, const char *strIpAddress = " ")

销毁指定 IP 地址机械臂某个路段。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

id_path：输入参数。要销毁的路径 ID。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

详见 addMoveJ 或 addMoveL 调用示例。

57 rpy2Axis

int rpy2Axis (double *arr)

欧拉角转轴角。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

arr：输入参数。欧拉角数组的首地址，数组长度为 3。

返回值：

0：成功。

-1：失败。

调用示例：

```
const double PI= 3.1415926;
double rpy[3]={PI, PI/3, PI/6};
int ret = rpy2Axis(rpy);
printf("Diana API rpy2Axis got: %f, %f, %f\n", rpy[0] , rpy[1] , rpy[2]);
```

输出结果：2.431323, 0.651471, -1.403725

58 axis2RPY

int axis2RPY (double *arr)

轴角转欧拉角。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

arr：输入参数。轴角数组的首地址，数组长度为 3。

返回值：

0：成功。

-1：失败。

调用示例：

```
const double PI= 3.1415926;
double rpy[3]={2.431323, 0.651471, -1.403725};
ret = axis2RPY(rpy);
printf("Diana API axis2Rpy got: %f, %f, %f\n", rpy[0] , rpy[1] , rpy[2]);
```

输出结果： -3.141593, 1.047198, 0.523599

59 homogeneous2Pose

int homogeneous2Pose (double *matrix, double *pose)

齐次变换矩阵转位姿。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

matrix：齐次变换矩阵数组的首地址，数组长度为 16。

pose：位姿数组的首地址，数组长度为 6。

返回值：

0：成功。

-1：失败。

调用示例：

```
double pose[6]={0.0};
double matrix[16]={ 0.433013, 0.250000, -0.866025, 0.000000, 0.500000, -0.866025, -
0.000000, 0.000000, -0.750000, -0.433013, -0.500000, 0.000000, -0.231000, 0.155000,
0.934000, 1.000000};
int ret = homogeneous2Pose(matrix, pose);
printf("Diana API homogeneous2Pose got: %f, %f, %f, %f, %f, %f\n",
        pose[0] , pose[1] , pose[2] , pose[3], pose[4] , pose[5]);
```

输出结果：

```
Diana API homogeneous2Pose got: -0.231000, 0.155000, 0.934000, 2.431311, 0.651465, -
1.403717
```

60 pose2Homogeneous

```
int pose2Homogeneous (double *pose, double *matrix)
```

位姿转齐次变换矩阵。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

pose：位姿数组的首地址，数组长度为 6。

matrix：齐次变换矩阵数组的首地址，数组长度为 16。

返回值：

0：成功。

-1：失败。

调用示例：

```
const double PI= 3.1415926;
double pose[6]={-0.231, 0.155, 0.934, PI, PI/3, PI/6};
double matrix[16]={0};
ret = pose2Homogeneous(pose, matrix);
printf("Diana API pose2Homogeneous
got:\n%f, %f, %f, %f\n%f, %f, %f, %f\n%f, %f, %f, %f\n%f, %f, %f, %f\n",
        matrix[0] , matrix[1] , matrix[2] , matrix[3],
        matrix[4] , matrix[5] , matrix[6] , matrix[7],
```

```
        matrix[8] , matrix[9] , matrix[10] , matrix[11],
        matrix[12] , matrix[13] , matrix[14] , matrix[15]);
```

输出结果：

```
Diana API pose2Homogeneous got:
0.758804, 0.546150, -0.354875, 0.000000
0.611590, -0.784849, -0.099843, 0.000000
0.223995, 0.292800, -0.929567, 0.000000
-0.231000, 0.155000, 0.934000, 1.000000
```

61 enableTorqueReceiver

```
int enableTorqueReceiver(bool bEnable, const char *strIpAddress =  " ")
```

在指定 IP 地址机械臂上，打开或关闭实时扭矩的接收。

适用臂型：通用医疗臂、定制医疗臂

参数：

bEnable：输入参数，是否开启实时扭矩的接收。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
bool bEnable = true;
int ret = enableTorqueReceiver (bEnable,strIpAddress);
if(ret < 0)
{
   printf( "enableTorqueReceiver failed! Return value = %d\n ", ret);
}
double torque7[7] = { 0.0,0.0,0.0,0.0,0.0,0.0,0.6 };
printf("控制七关节运动\n");
for (int i = 0; i < 3; i++)
{
   torque7[6] += 0.2;
   ret=sendTorque_rt(torque7, 1.0);
   if (ret < 0)
   {
      printf("sendTorque_rt failed!Return value = %d\n", ret);
```

```
    }
    M_SLEEP(1000);
}
ret=enableTorqueReceiver(false);
if (ret < 0)
{
    printf("exit TorqueReceiver failed!Return value = %d\n", ret);
}
```

62 sendTorque_rt

int sendTorque_rt(double *torque,double t,const char *strIpAddress = " ")

对指定 IP 地址机械臂，用户发送实时扭矩。

适用臂型：通用医疗臂、定制医疗臂

参数：

torque：输入参数，用户传入的扭矩值，大小为 JOINT_NUM 的数组。

t：持续时间，单位 s。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

详见 enableTorqueReceiver 调用示例。

63 enableCollisionDetection

int enableCollisionDetection(bool bEnable,const char *strIpAddress = " ")

开启指定 IP 地址机械臂碰撞检测。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

bEnable：输入参数，是否开启碰撞检测模式。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
int ret = enableCollisionDetection(true, strIpAddress);
if(ret < 0)
{
   printf( "enableCollisionDetection failed! Return value = %d\n ", ret);
}
```

64 setActiveTcpPayload

int setActiveTcpPayload(double *payload,const char *strIpAddress = " ")

设置指定 IP 地址机械臂的负载信息。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

payload：负载信息，大小为 10 的数组，第 1 位为质量，2~4 位为质心，5~10 位为张量。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
double dblPayload[10] = {0.0};
int ret = setActiveTcpPayload (dblPayload,strIpAddress);
if(ret < 0)
{
   printf( "setActiveTcpPayload failed! Return value = %d\n ", ret);
}
```

65 servoJ

int servoJ (double *joints, double time, double look_ahead_time, double gain, const char *strIpAddress = " ")

关节空间内，伺服指定 IP 地址机械臂到指定关节角位置。 servoJ 函数用于在线控制机械臂，lookahead 时间和 gain 能够调整轨迹是否平滑或尖锐。 注意：太高的 gain 或太短的 lookahead 时间可能会导致不稳定。由于该函数主要用于以较短位移为目标点的多次频繁调用，建议在实时系统环境下使用。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

joints：目标关节角位置数组的首地址，数组长度为 JOINT_NUM。单位：rad。

time：运动时间。单位：s。

look_ahead_time：时间（单位：s），范围（0.03-0.2）用这个参数使轨迹更平滑。

gain：目标位置的比例放大器，范围（100,2000）。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const double PI=3.1415926;
//以七轴机械臂为例
double target1[7]={0, PI/6, 0, PI/2, 0, -PI/2, 0};
const char* strIpAddress =  "192.168.10.75";
for(int i = 0; i < 100; ++i)
{
 target1[6] = target1[6]+PI/180;
 ret = servoJ( target1, 0.01, 0.1, 300, strIpAddress);
 if(ret < 0)
   break;
 sleep(0.001);
}
stop(strIpAddress);
```

66 servoL

int servoL (double *pose, double time, double look_ahead_time, double gain, double scale, double *active_tcp=nullptr, const char *strIpAddress = " ")

笛卡尔空间内，控制指定 IP 地址机械臂工具中心点到指定位姿。由于该函数主要用于以较短位移为目标点的多次频繁调用，建议在实时系统环境下使用。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

pose：基坐标系下的目标位姿数组的首地址，数组长度为 6。前三个元素单位：m；后三个元素单位：rad。注意，后三个角度需要是轴角。

time：运动时间。单位：s。

look_ahead_time：时间（单位：s），范围（0.03-0.2）用这个参数使轨迹更平滑。

gain：目标位置的比例放大器，范围（100,2000）。

scale：平滑比例系数。范围（0.0~1.0）。

active_tcp：需要移动的工具中心点对应的位姿向量（基于法兰坐标系），大小为 6 的数组（位置和旋转矢量（轴角）），为空时将移动系统当前工具的中心点至 pose。（注意：此处 active_tcp 代表需要移动的工具）

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const double PI=3.141592653;
const char* strIpAddress = "192.168.10.75";
//以七轴机械臂为例
double joints[7]={0, 0, 0, PI/2, 0, -PI/2, 0};
moveJ(joints,0.1,0.1,strIpAddress);
wait_move(strIpAddress);
double pose[6]={0};
getTcpPos(pose,strIpAddress);
for(int i = 0; i < 1000; ++i){
  pose[2] = pose[2] + 0.001;
  ret = servoL(pose, 0.01, 0.1, 300, 0.5, nullptr,strIpAddress);
  if(ret < 0)
    break;
  sleep(0.001);
}
stop(strIpAddress);
```

67 servoJ_ex

```
int servoJ_ex (double *joints, double time, double look_ahead_time, double gain, bool reliable, const char *strIpAddress =  " ")
```

关节空间内，控制指定 IP 地址机械臂到指定关节角位置优化版。 servoJ_ex 函数用于在线控制机械臂，lookahead 时间和 gain 能够调整轨迹是否平滑或尖锐。 注意：太高的 gain 或太短的 lookahead 时间可能会导致不稳定。由于该函数主要用于以较短位移为目标点的多次频繁调用，建议在实时系统环境下使用。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

joints：目标关节角位置数组的首地址，数组长度为 JOINT_NUM。单位：rad。

time：运动时间。单位：s。

look_ahead_time：时间（单位：s），范围（0.03-0.2）用这个参数使轨迹更平滑。

gain：目标位置的比例放大器，范围（100,2000）。

reliable：bool 型变量，值为 true 需要 socket 反馈通信状态，行为等同 servoJ；值为 false 则无需反馈直接返回。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
timespec next_time;
//获取系统时间存入 next_time
const char* strIpAddress =  "192.168.10.75";
const double PI=3.141592653;
double target_point[7]={0, 0, 0, PI/2, 0, -PI/2, 0};//以七轴机械臂为例
for (int i = 0; i < 50000; i++)
{
   target_point[0] = (cos(2 * PI / 5 * i * 0.001) - 1) * PI / 3;
   servoJ_ex(target_point, 0.001, 0.1, 500, false, strIpAddress);
   next_time.tv_nsec += 1000000;//计算下次唤醒时间
   //sleep 到 next_time
}
stop(strIpAddress);
```

68 servoL_ex

int servoL_ex (double *pose, double time, double look_ahead_time, double gain, double scale, bool reliable, double *active_tcp=nullptr ,const char *strIpAddress = " ")

笛卡尔空间内，控制指定 IP 地址机械臂工具中心点到指定位姿优化版。由于该函数主要用于以较短位移为目标点的多次频繁调用，建议在实时系统环境下使用。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

pose：目标位姿数组的首地址，数组长度为 6。前三个元素单位：m；后三个元素单位：rad。注意，后三个角度需要是轴角。

time：运动时间。单位：s。

look_ahead_time：时间（单位：s），范围（0.03-0.2）用这个参数使轨迹更平滑。

gain：目标位置的比例放大器，范围（100,2000）。

scale：平滑比例系数。范围（0.0~1.0）。

reliable：bool 型变量，值为 true 需要 socket 反馈通信状态，行为等同 servoL；值为 false 则无需反馈直接返回。

active_tcp：需要移动的工具中心点对应的位姿向量（基于法兰坐标系），大小为 6 的数组，为空时将移动系统当前工具的中心点至 pose。（注意：此处 active_tcp 作为工具）。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const double PI=3.141592653;
const char* strIpAddress = "192.168.10.75";
double joints[7]={0, 0, 0, PI/2, 0, -PI/2, 0};//以七轴机械臂为例
moveJ(joints,0.1,0.1,strIpAddress);
wait_move(strIpAddress);
double pose[6]={0};
getTcpPos(pose,strIpAddress);
for(int i = 0; i < 1000; ++i){
  pose[2] = pose[2] + 0.001;
  ret = servoL_ex(pose, 0.01, 0.03, 300, 1, true, nullptr, strIpAddress);
  if(ret < 0)
    break;
  sleep(0.001);
}
stop(strIpAddress);
```

69 speedJ_ex

```
int speedJ_ex (double *speed, double a, double t, bool reliable, const char *strIpAddress =  " ")
```

速度模式优化版，使指定 IP 地址机械臂进行关节空间运动。时间 t 为可选项，时间 t 是可选项，如果提供了 t 值，机械臂将在 t 时间后减速。如果没有提供时间 t 值(即 t=0)，机械臂将在达到目标速度时减速。该函数调用后立即返回。停止运动需要调用 stop 函数。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

speed：关节角速度数组首地址，数组长度为 JOINT_NUM。单位：rad/s。

a：加速度，单位：rad/s^2。

t：时间，单位：s。

reliable：bool 型变量，值为 true 需要 socket 反馈通信状态，行为等同 speedJ；值为 false 则无需反馈直接返回。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
double speeds[JOINT_NUM] = {0.0};
speeds[1]=0.2;
double acc = 0.40;
const char* strIpAddress =  "192.168.10.75";
int ret = speedJ_ex(speeds, acc, 0, true, strIpAddress);
if(ret < 0)
{
   printf( "speedJ_ex failed! Return value = %d\n ", ret);
}
sleep(2);
stop(strIpAddress);
```

70 speedL_ex

```
int speedL_ex(double *speed, double *a, double t,bool reliable, double *active_tcp=nullptr, const char *strIpAddress =  " ")
```

速度模式优化版，使指定 IP 地址机械臂笛卡尔空间下直线运动，支持同步旋转，但笛卡尔方向必须有速度或者加速度才能旋转。时间 t 为可选项，如果提供了 t 值，机械臂将在 t 时间后减速。如果没有提供时间 t 值(即 t=0)，机械臂将在达到目标速度时减速。该函数调用后立即返回。停止运动需要调用 stop 函数。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

speed：基坐标系下的工具空间速度，数组长度为 6（位置和旋转），其中前 3 个单位为 m/s，后 3 个单位为 rad/s。位置和旋转均参考基坐标系，旋转时的旋转中心可由 active_tcp

指定。

a：加速度，数组长度为 2，第 1 个单位为 m/s^2，第 2 个单位为 rad/s^2。

t：时间，单位：s。

reliable：bool 型变量，值为 true 需要 socket 反馈通信状态，行为等同 speedL；值为 false 则无需反馈直接返回。

active_tcp：基于法兰坐标系的位姿，指定旋转时的旋转中心，大小为 6 的数组（位置+旋转矢量（轴角））；为空时旋转中心是系统当前工具中心点。（注意：机械臂做直线运动时中心点会随动，所以无旋转运动的情况下，此参数看不出影响）。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
double speeds[6] = {0.1,0.0,0.0,0.0,0.0,0.0};
double acc[2] = {0.30, 0.50};
const char* strIpAddress =  "192.168.10.75";
int ret = speedL_ex(speeds, acc, 0,  true, nullptr, strIpAddress);
if(ret < 0)
{
   printf( "speedL_ex failed! Return value = %d\n ", ret);
}
sleep(2);
stop(strIpAddress);
```

71 dumpToUDisk

```
int dumpToUDisk(const char *strIpAddress =  " ")
```

导出指定 IP 地址机械臂的日志文件到 u 盘。控制箱中的系统日志文件（主要包含 ControllerLog.txt 和 DianaServerLog.txt）会自动复制到 u 盘。需要注意的是目前控制箱仅支持 FAT32 格式 u 盘，调用 dumpToUDiskEx 函数前需先插好 u 盘，如果系统日志拷贝失败将不会提示。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

1. 系统开机
2. 插入 u 盘到控制箱
3. 调用 Api 函数 dumpToUDisk("192.168.10.75")
4. 拔下 u 盘查看

72 inverse_ext

int inverse_ext(double *ref_joints, double *pose,double *joints,double *active_tcp=nullptr, const char *strIpAddress = " ")

针对指定 IP 地址机械臂，逆解函数，给定一个参考关节角，算出欧式距离最近的逆解。

现支持在不同工具坐标系下求逆解，由传入的 active_tcp 决定。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

ref_joints：参考的关节角，大小为 JOINT_NUM 的数组。

pose：输入位姿数组首地址，数据为包含 active_tcp 坐标（x, y, z）和旋转矢量（轴角坐标）组合。

joints：输出关节角度数组首地址，用于传递转换的结果，大小为 JOINT_NUM 的数组。

active_tcp：工具坐标系对应的位姿向量，大小为 6 的数组，为空时，将使用默认的工具坐标系 default_tcp。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
double ref_joints[JOINT_NUM] = {0.0};
double pose[6] = {0.584547,0.050451,0.439096,-0.000067,-0.000036,0.000027};
double joints[JOINT_NUM] = {0.0};
const char* strIpAddress =  "192.168.10.75";
int ret = inverse_ext(ref_joints,pose, joints, nullptr,strIpAddress);
if(ret < 0)
{
   printf( "inverse_ext failed! Return value = %d\n ", ret);
}
```

73 getJointLinkPos

int getJointLinkPos(double *joints,const char *strIpAddress = " ")
获取指定 IP 地址机械臂当前低速侧关节角。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** joints：输出参数。低速侧关节角,大小为 JOINT_NUM 的数组。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。

调用示例：
const char* strIpAddress = "192.168.10.75";
double joints[JOINT_NUM] = {0.0};
int ret = getJointLinkPos(joints,strIpAddress);
if(ret < 0)
{
printf("getJointLinkPos failed! Return value = %d\n ", ret);
}

74 createComplexPath

int createComplexPath (int complex_path_type, unsigned int &complex_path_id, const char *strIpAddress = " ")
在指定 IP 地址机械臂上创建一个复杂路段。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** complex_path_type：输入参数。 0 :表示 NORMAL_JOINT，该模式下路径中可以添加 moveL、moveJ、moveC 对应的路径点，路径点用 JOINT_NUM 个关节角度确定。 1：表示 MOVEP_JOINT，该模式下机械臂关节做点对点的匀速运动，路径中可以添加 moveL、moveC 对应的路径点，不可以添加 moveJ 对应的路径点，路径点用 JOINT_NUM 个关节角度确定。 2:表示 NORMAL_POSE ，该模式下路径中可以添加 moveL、moveJ、moveC 对应的路径点，路径点用末端 TCP 位姿数据确定。 3：表示 MOVEP_POSE，该模式下机械臂末端做点对点的匀速运动，路径中可以添加

moveL、moveC 对应的路径点，不可以添加 moveJ 对应的路径点，路径点用末端 TCP 位姿数据确定。

complex_path_id：输出参数。用于保存新创建 complexPath 的 ID。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

详见 addMoveJByTarget 或 addMoveLByTarget 调用示例。

75 addMoveLByTarget

int addMoveLByTarget (unsigned int complex_path_id, double *target_joints, double vel, double acc, double blendradius, const char *strIpAddress = " ")

在指定 IP 地址机械臂上，向已创建的路段添加 MoveL 路点。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

complex_path_id：输入参数。要添加路点的路径 ID。

target_joints：输入参数。要添加的路点，即该路点的各关节角度数组的首地址，数组长度为 JOINT_NUM。单位：rad。

vel：moveL 移动到目标路点的速度。单位：m/s。

acc：moveL 移动到目标路点的加速度。单位：m/s^2。

blendradius：交融半径。单位：m。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
//以七轴机械臂为例
#define PI 3.14159265358979323846
#define DEGREE_TO_RAD(x) ((x)*PI / 180.0)
#define RAD_TO_DEGREE(x) ((x)*180.0 / PI)

double dblFirstPosition [JOINT_NUM] = {DEGREE_TO_RAD(0.0),
```

```
DEGREE_TO_RAD(0.0), DEGREE_TO_RAD(0.0), DEGREE_TO_RAD(90.0),
DEGREE_TO_RAD(0.0), DEGREE_TO_RAD(-90.0), DEGREE_TO_RAD(0.0)};
double dblSecondPosition [JOINT_NUM] = {DEGREE_TO_RAD(0.0),
DEGREE_TO_RAD(0.0), DEGREE_TO_RAD(0.0), DEGREE_TO_RAD(150.0),
DEGREE_TO_RAD(0.0), DEGREE_TO_RAD(-30.0), DEGREE_TO_RAD(0.0) };
unsigned int uintComplexPathId = 0;
const char* strIpAddress =  "192.168.10.75";
createComplexPath(NORMAL_JOINT_PATH, uintComplexPathId, strIpAddress);
addMoveLByTarget(uintComplexPathId, dblFirstPosition, 0.2, 0.4, 0, strIpAddress);
addMoveLByTarget(uintComplexPathId, dblSecondPosition, 0.2, 0.2, 0, strIpAddress);
runComplexPath(uintComplexPathId, strIpAddress);
destroyComplexPath(uintComplexPathId, strIpAddress);
wait_move(strIpAddress);
```

76 addMoveLByPose

int addMoveLByPose(unsigned int complex_path_id, double *target_pose, double vel, double acc, double blendradius, const char *strIpAddress = " ")

在指定 IP 地址机械臂上，向已创建的路段添加 MoveL 路点。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

complex_path_id,：输入参数。要添加路点的路径 ID。

target_pose：输入参数。路径点位姿数组首地址，数组长度为 6，保存 TCP 坐标（x, y, z）和轴角（rx, ry, rz）组合数据。

vel：moveL 移动到目标路点的速度。单位：m/s。

acc：moveL 移动到目标路点的加速度。单位：m/s^2。

blendradius：交融半径。单位：m。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
double dblFirstPose [6] = { 0.4000, 0.0, 0.3000, 0.0, 0.0, 0.0 };
double dblSecondPose [6] = { 0.1000, 0.0, 0.8000, 0.0, 0.0, 0.0 };
 unsigned int uintComplexPathId = 0;
const char* strIpAddress =  "192.168.10.75";
```

createComplexPath(NORMAL_POSE_PATH, uintComplexPathId, strIpAddress);
addMoveLByPose(uintComplexPathId, dblFirstPose, 0.2, 0.4, 0, strIpAddress);
addMoveLByPose(uintComplexPathId, dblSecondPose, 0.2, 0.2, 0, strIpAddress);
runComplexPath(uintComplexPathId, strIpAddress);
destroyComplexPath(uintComplexPathId, strIpAddress);
wait_move(strIpAddress);

77 addMoveJByTarget

int addMoveJByTarget (unsigned int complex_path_id, double *target_joints, double vel_percent, double acc_percent, double blendradius_percent, const char * strIpAddress = " ")

在指定 IP 地址机械臂上，向已创建的路段添加 MoveJ 路点。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

complex_path_id：输入参数。要添加路点的路径 ID。

target_joints：输入参数。要添加的路点，即该路点的各关节角度数组的首地址，数组长度为 JOINT_NUM。单位：rad。

vel_percent：moveJ 移动到目标路点的速度百分比，相对于系统软限位中设定最大速度的百分比。

acc_percent：moveJ 移动到目标路点的加速度百分比，相对于系统软限位中设定最大加速度的百分比。

blendradius _percent：交融半径百分比，相对于系统软限位中设定最大交融半径的百分比。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
#define PI 3.14159265358979323846
#define DEGREE_TO_RAD(x) ((x)*PI / 180.0)
#define RAD_TO_DEGREE(x) ((x)*180.0 / PI)
//以七轴机械臂为例
double dblFirstPosition [JOINT_NUM] = {DEGREE_TO_RAD(0.0),
DEGREE_TO_RAD(0.0), DEGREE_TO_RAD(0.0), DEGREE_TO_RAD(90.0),
DEGREE_TO_RAD(0.0), DEGREE_TO_RAD(-90.0), DEGREE_TO_RAD(0.0)};
double dblSecondPosition [JOINT_NUM] = {DEGREE_TO_RAD(0.0),
```

```
DEGREE_TO_RAD(0.0), DEGREE_TO_RAD(0.0), DEGREE_TO_RAD(150.0),
DEGREE_TO_RAD(0.0), DEGREE_TO_RAD(-30.0), DEGREE_TO_RAD(0.0) };
 unsigned int uintComplexPathId = 0;
const char* strIpAddress =  "192.168.10.75";
createComplexPath(NORMAL_JOINT_PATH, uintComplexPathId, strIpAddress);
addMoveJByTarget (uintComplexPathId, dblFirstPosition, 0.2, 0.4, 0, strIpAddress);
addMoveJByTarget(uintComplexPathId, dblSecondPosition, 0.2, 0.2, 0, strIpAddress);
runComplexPath(uintComplexPathId, strIpAddress);
destroyComplexPath(uintComplexPathId, strIpAddress);
wait_move(strIpAddress);
```

78 addMoveJByPose

int addMoveJByPose (unsigned int complex_path_id, double *target_pose, double vel_percent, double acc_percent, double blendradius_percent, const char *strIpAddress = " ")

在指定 IP 地址机械臂上，向已创建的路段添加 MoveJ 路点。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

complex_path_id：输入参数。要添加路点的路径 ID。

target_pose：输入参数。要添加的路点，路径点位姿数组首地址，数组长度为 6，保存 TCP 坐标（x, y, z）和轴角（rx, ry, rz）组合数据。

vel_percent：moveJ 移动到目标路点的速度百分比。相对于系统软限位中设定最大速度的百分比。

acc_percent：moveJ 移动到目标路点的加速度百分比。相对于系统软限位中设定最大加速度的百分比。

blendradius _percent：交融半径百分比。相对于该轨迹最大交融半径的百分比。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
double dblFirstPose [6] = { 0.484540,0.150450,0.439095,-0.000052,-0.000022,0.000031 };
double dblSecondPose [6] = { 0.584547,0.050451,0.439096,-0.000067,-0.000036,0.000027 };
unsigned int uintComplexPathId = 0;
const char* strIpAddress =  "192.168.10.75";
createComplexPath(NORMAL_POSE_PATH, uintComplexPathId, strIpAddress);
```

```
addMoveJByPose(uintComplexPathId, dblFirstPose, 0.2, 0.4, 0, strIpAddress);
addMoveJByPose(uintComplexPathId, dblSecondPose, 0.2, 0.2, 0, strIpAddress);
runComplexPath(uintComplexPathId, strIpAddress);
destroyComplexPath(uintComplexPathId, strIpAddress);
wait_move(strIpAddress);
```

79 addMoveCByTarget

```
int addMoveCByTarget (unsigned int complex_path_id, double *pass_joints, double *target_joints, double vel, double acc, double blendradius, bool ignore_rotation, const char *strIpAddress = " ")
```

在指定 IP 地址机械臂上，向已创建的路段添加 MoveC 路点。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

complex_path_id：输入参数。要添加路点的路径 ID。

pass_joints：输入参数。要添加的 moveC 中间路点，即该路点的各关节角度数组的首地址，数组长度为 JOINT_NUM。单位：rad。

target_joints：输入参数。要添加的 moveC 目标路点，即该路点的各关节角度数组的首地址，数组长度为 JOINT_NUM。单位：rad。

vel：moveC 移动到目标路点的速度。单位：m/s。

acc：moveC 移动到目标路点的加速度。单位：m/s^2。

blendradius：交融半径。单位：m。

ignore_rotation：是否忽略姿态变化。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
#define PI 3.14159265358979323846
#define DEGREE_TO_RAD(x) ((x)*PI / 180.0)
#define RAD_TO_DEGREE(x) ((x)*180.0 / PI)
const char* strIpAddress =  "192.168.10.75";
//以七轴机械臂为例
double start_joint[JOINT_NUM]={ DEGREE_TO_RAD(1.138), DEGREE_TO_RAD(-5.357), DEGREE_TO_RAD(-17.288), DEGREE_TO_RAD(129.746), DEGREE_TO_RAD(-1.928), DEGREE_TO_RAD(-55.385), DEGREE_TO_RAD(-14.982) };
```

```
double vel=0.2;
double acc=0.2;
moveJToTarget(start_joint,vel,acc,strIpAddress);
wait_move(strIpAddress);
double dblFirstPosition [JOINT_NUM] = { DEGREE_TO_RAD(7.848), DEGREE_TO_RAD(4.284), DEGREE_TO_RAD(-12.640), DEGREE_TO_RAD(119.067), DEGREE_TO_RAD(1.125), DEGREE_TO_RAD(-56.756), DEGREE_TO_RAD(-5.372) };
double dblSecondPosition [JOINT_NUM] = { DEGREE_TO_RAD(13.228), DEGREE_TO_RAD(-7.620), DEGREE_TO_RAD(-6.515), DEGREE_TO_RAD(132.295), DEGREE_TO_RAD(-1.045), DEGREE_TO_RAD(-55.280), DEGREE_TO_RAD(7.366) };

unsigned int uintComplexPathId = 0;
createComplexPath(NORMAL_JOINT_PATH, uintComplexPathId, strIpAddress);
addMoveCByTarget (uintComplexPathId, dblFirstPosition, dblSecondPosition, 0.2, 0.4, 0,true, strIpAddress);
runComplexPath(uintComplexPathId, strIpAddress);
destroyComplexPath(uintComplexPathId, strIpAddress);
wait_move(strIpAddress);
```

80 addMoveCByPose

```
int addMoveCByPose (unsigned int complex_path_id, double *pass_pose, double *target_pose, double vel, double acc, double blendradius, bool ignore_rotation, const char *strIpAddress = "")
```

在指定 IP 地址机械臂上，向已创建的路段添加 MoveC 路点。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

complex_path_id：输入参数。要添加路点的路径 ID。

pass _pose：输入参数。要添加的 moveC 中间路点，即路径点位姿数组首地址，数组长度为 6，保存 TCP 坐标（x, y, z）和轴角（rx, ry, rz）组合数据。

target_pose：输入参数。要添加的 moveC 目标路点，即该路径点位姿数组首地址，数组长度为 6，保存 TCP 坐标（x, y, z）和轴角（rx, ry, rz）组合数据。

vel：moveC 移动到目标路点的速度。单位：m/s。

acc：moveC 移动到目标路点的加速度。单位：m/s^2。

blendradius：交融半径。单位：m。

ignore_rotation：是否忽略姿态变化。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂

时生效。 **返回值：** 0：成功。 -1：失败。

调用示例：
const char* strIpAddress = "192.168.10.75";
double start_pose[6]={ 0.484540,0.150450,0.439095,-0.000052,-0.000022,0.000031};
double vel=0.2;
double acc=0.2;
moveJToPose(start_pose,vel,acc,nullptr,strIpAddress);
wait_move(strIpAddress);
 	
double dblFirstPose [6] = { 0.584547,0.050451,0.439096,-0.000067,-0.000036,0.000027 };
double dblSecondPose [6] = { 0.484544,-0.049563,0.439087,-0.000066,-0.000018,0.000013 };
unsigned int uintComplexPathId = 0;
createComplexPath(NORMAL_POSE_PATH, uintComplexPathId, strIpAddress);
addMoveCByPose(uintComplexPathId, dblFirstPose, dblSecondPose, 0.2, 0.4, 0,true, strIpAddress);
runComplexPath(uintComplexPathId, strIpAddress);
destroyComplexPath(uintComplexPathId, strIpAddress);
wait_move(strIpAddress);

81 runComplexPath

int runComplexPath (unsigned int complex_path_id, const char *strIpAddress = " ")
在指定 IP 地址机械臂上，启动运行设置好的路段。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** complex_path_id：输入参数。要运行的路径 ID。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。
调用示例： 详见 addMoveJByTarget 或 addMoveLByTarget 调用示例。

82 destroyComplexPath

int destroyComplexPath (unsigned int complex_id_path, const char *strIpAddress = " ")
在指定 IP 地址机械臂上，销毁某个路段。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** complex_id_path：输入参数。要销毁的路径 ID。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。
调用示例： 详见 addMoveJByTarget 或 addMoveLByTarget 调用示例。

83 saveEnvironment

int saveEnvironment (const char *strIpAddress = " ")
将指定 IP 地址机械臂的控制器当前参数数据写入配置文件，用于重启机械臂时初始化设置各参数，包括碰撞检测阈值、阻抗参数、DH 参数等所有可通过 API 设置的参数数据。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。

调用示例：
const char* strIpAddress = "192.168.10.75";
int ret = saveEnvironment (strIpAddress);
if(ret < 0)
{
printf("Save failed!\n ");
}

84 dumpToUDiskEx

int dumpToUDiskEx (double time_out, const char *strIpAddress = " ")
导出指定 IP 地址机械臂的日志文件到 u 盘。控制箱中的系统日志文件（主要包含

ControllerLog.txt 和 DianaServerLog.txt）会自动复制到 u 盘。需要注意的是目前控制箱仅支持 FAT32 格式 u 盘，调用 dumpToUDiskEx 函数前需先插好 u 盘，如果系统日志拷贝失败将不会提示。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

time_out：单位 秒，设置超时时间，一般需要大于 3 秒,-1 表示设置超时时间无穷大。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

5. 系统开机
6. 插入 u 盘到控制箱
7. 调用 Api 函数 dumpToUDiskEx(-1,"192.168.10.75")
8. 拔下 u 盘查看

85 enterForceMode_ex

int enterForceMode_ex(double *dblForceDirection, double dblForceValue, double dblMaxApproachVelocity, double dblMaxAllowTcpOffset, double *dblActiveTcp = nullptr, const char *strIpAddress = " ")

使指定 IP 地址机械臂进入力控模式，相较于 enterForceMode 增加不同坐标系的支持。适用臂型：通用医疗臂、定制医疗臂

参数：

dblForceDirection：表达力的方向的数组首地址，数组长度为 3。

dblForceValue：力大小。单位：N。推荐取值范围（1N~100N）。

dblMaxApproachVelocity：最大接近速度。单位：m/s。推荐取值范围（0.01m/s~0.5m/s）。

dblMaxAllowTcpOffset：允许的最大偏移。单位：m。推荐取值范围（0.01m~1m）。

dblActiveTcp：支持的坐标系，大小为 6 的数组。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
double dblForceDir[3]={0,0,-1};
double dblForceValue = 2.0;
double dblMaxVel = 0.1;
double dblMaxOffset = 0.2;
double dblActiveTcp[6] = {0.0};
double rpy[3] = { to_rad(90),to_rad(90), to_rad(0) };
ret = rpy2Axis(rpy);
dblActiveTcp[2] = -0.1;
dblActiveTcp[3] = rpy[0];
dblActiveTcp[4] = rpy[1];
dblActiveTcp[5] = rpy[2];
const char* strIpAddress =  "192.168.10.75";
if (enterForceMode_ex(dblForceDir, dblForceValue, dblMaxVel, dblMaxOffset,
dblActiveTcp,strIpAddress) < 0)
{
   printf("Diana API enterForceMode_ex failed!\n");
}
```

86 readDI

```
int readDI(const char *group_name, const char *name, int &value, const char *strIpAddress =  "
")
```

读取指定 IP 地址机械臂一个数字输入的值。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

group_name：数字输入的分组，例如，’board’,’plc’；

name：数字输入的信号名，例如，’di0’；

value：读取返回的值。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
int value;
const char* strIpAddress =  "192.168.10.75";
int ret = readDI( "board",  "di0", value, strIpAddress);
if(ret < 0)
```

```
{
    printf( "readDI failed!\n ");
}
```

87 readAI

int readAI (const char *group_name, const char *name, int &mode, double &value, const char *strIpAddress = " ")

读取指定 IP 地址机械臂一个模拟输入的值和模式。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

group_name：模拟输入的分组，例如，’board’,’plc’；

name：模拟输入的信号名，例如，’ai0’；

mode：当前模拟输入模式，1 代表电流，2 代表电压；

value：读取返回的值。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
int mode;
double value;
const char* strIpAddress =  "192.168.10.75";
int ret = readAI( "board",  "ai0",mode,value, strIpAddress);
if(ret < 0)
{
   printf( "ReadAI failed!\n ");
}
```

88 setAIMode

int setAIMode (const char *group_name, const char *name, int mode, const char *strIpAddress = " ")

设置指定 IP 地址机械臂模拟输入的模式。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

group_name：模拟输入的分组，例如，’board’,’plc’；

name：模拟输入的信号名，例如，’ai0’；

mode: 模拟输入模式，1 代表电流，2 代表电压。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
int mode = 1;
const char* strIpAddress =  "192.168.10.75";
int ret = setAIMode( "board",  "ai0",mode, strIpAddress);
if(ret < 0)
{
   printf( "SetAIMode failed!\n ");
}
```

89 writeDO

int writeDO (const char *group_name, const char *name,int value, const char *strIpAddress = "")

设置指定 IP 地址机械臂一个数字输出的值。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

group_name：数字输出的分组，例如，’board’,’plc’。

name：数字输出的信号名，例如，’do0’。

value：设置的值。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
int value = 0;
const char* strIpAddress =  "192.168.10.75";
int ret = writeDO ( "board ",  "do0 ",value, strIpAddress);
if(ret < 0)
{
```

```
   printf( "WriteDO failed!\n ");
}
```

90 writeAO

int writeAO (const char *group_name, const char *name, int mode, double value, const char *strIpAddress = " ")

设置指定 IP 地址机械臂一个模拟输出的值和模式。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

group_name：模拟输出的分组，例如，’board’,’plc’；

name：模拟输出的信号名，例如，’ao0’；

mode：当前模拟输出模式；

value：设置输出的值。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
int mode = 1;
double value = 8.8;
const char* strIpAddress =  "192.168.10.75";
int ret = writeAO ( "board ",  "ao0 ", mode, value, strIpAddress);
if(ret < 0)
{
   printf( "WriteAO failed!\n ");
}
```

91 readBusCurrent

int readBusCurrent(double &current, const char *strIpAddress = " ")

读取指定 IP 地址机械臂总线电流。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

current：总线电流。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
double current;
const char* strIpAddress =  "192.168.10.75";
int ret = readBusCurrent(current, strIpAddress);
if(ret < 0)
{
   printf( "ReadBusCurrent failed!\n ");
}
```

92 readBusVoltage

int readBusVoltage (double &voltage, const char *strIpAddress = " ")

读取指定 IP 地址机械臂总线电压。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

voltage：总线电压。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
double voltage;
const char* strIpAddress =  "192.168.10.75";
int ret = readBusVoltage(voltage, strIpAddress);
if(ret < 0)
{
   printf( "ReadBusVoltage failed!\n ");
}
```

93 getDH

int getDH(double *aDH, double *alphaDH, double *dDH,double *thetaDH, const char *strIpAddress = " ")

获取指定 IP 地址机械臂的 DH 参数。

适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** aDH：连杆长度的数组，长度为 JOINT_NUM。 alphaDH:连杆转角的数组，长度为 JOINT_NUM。 dDH:连杆偏距的数组，长度为 JOINT_NUM 。 thetaDH:连杆的关节角的数组，长度为 JOINT_NUM。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。

调用示例：
double a[JOINT_NUM],alpha[JOINT_NUM],d[JOINT_NUM],theta[JOINT_NUM];
const char* strIpAddress = "192.168.10.75";
int ret = getDH(a,alpha,d,theta, strIpAddress);
if(ret < 0)
{
printf("getDH failed!\n ");
}

94 getOriginalJointTorque

int getOriginalJointTorque(double *torques, const char *strIpAddress = " ")
获取指定 IP 地址机械臂传感器反馈的扭矩值，未减去零偏。 适用臂型：通用医疗臂、定制医疗臂 **参数：** torques：反馈扭矩的数组，长度为 JOINT_NUM。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。
调用示例： double torques[JOINT_NUM]; const char* strIpAddress = "192.168.10.75"; int ret = getOriginalJointTorque(torques, strIpAddress);

```
if(ret < 0)
{
   printf( "getOriginalJointTorque failed!\n ");
}
```

95 getJacobiMatrix

int getJacobiMatrix(double *matrix_jacobi, const char *strIpAddress = " ")

获取指定 IP 地址机械臂的雅各比矩阵。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

matrix_jacobi：雅各比矩阵的数组，长度为 6 * JOINT_NUM。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
double matrix_jacobi [6 * JOINT_NUM];
const char* strIpAddress =  "192.168.10.75";
int ret = getJacobiMatrix (matrix_jacobi, strIpAddress);
if(ret < 0)
{
   printf( "getJacobiMatrix failed!\n ");
}
```

96 resetDH

int resetDH(const char *strIpAddress = " ")

重置指定 IP 地址机械臂用户自定义 DH 参数。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
int ret = resetDH(strIpAddress);
if(ret < 0)
{
   printf( "resetDH failed!\n ");
}
```

97 runProgram

int runProgram(const char *programName, const char *strIpAddress = " ")
运行指定 IP 地址机械臂某个程序。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** programName：程序的名字。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。

调用示例：
const char* strIpAddress = "192.168.10.75";
int ret=runProgram("AgileRobots", strIpAddress);
if(ret < 0)
{
   printf(" runProgram failed!\n ");
}

98 stopProgram

int stopProhram(const char *programName, const char *strIpAddress = " ")
停止指定 IP 地址机械臂某个程序。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** programName：程序的名字。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。

-1：失败。

调用示例：
const char* strIpAddress = "192.168.10.75";
int ret=stopProgram("AgileRobots", strIpAddress);
if(ret < 0)
{
printf(" stopProgram failed!\n ");
}

99 getVariableValue

int getVariableValue(const char *variableName,double &value, const char *strIpAddress = " ")
获取指定 IP 地址机械臂某个全局变量的值。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** variableName：全局变量的名字 value: 获取的全局变量的值 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。

调用示例：
double value;
const char* strIpAddress = "192.168.10.75";
int ret = getVariableValue ("GLOBAL", value, strIpAddress);
if(ret < 0)
{
printf("getVariableValue failed!\n ");
}

100 setVariableValue

int setVariableValue(const char *variableName,const double &value, const char *strIpAddress = " ")
设置指定 IP 地址机械臂某个全局变量的值。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：**

variableName：全局变量的名字。

value: 设置的全局变量的值。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
double value=2.0;
const char* strIpAddress =  "192.168.10.75";
int ret = setVariableValue ( "GLOBAL",value, strIpAddress);
if(ret < 0)
{
   printf( "setVariableValue failed!\n ");
}
```

101 isTaskRunning

```
int isTaskRunning(const char *programName, const char *strIpAddress =  " ")
```

判断指定 IP 地址机械臂某个程序是否在运行。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

programName：程序名称。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：运行中。

-1：没有运行中。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
int ret = isTaskRunning ( "AgileRobots", strIpAddress);
if(ret < 0)
{
   printf( "AgileRobots is not running!\n ");
}
```

102 pauseProgram

int pauseProgram(const char *strIpAddress = " ")
暂停指定 IP 地址机械臂所有程序。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。

调用示例：
const char* strIpAddress = "192.168.10.75";
int ret = pauseProgram(strIpAddress);
if(ret == 0)
{
printf("All program is paused!\n ");
}

103 resumeProgram

int resumeProgram(const char *strIpAddress = " ")
恢复运行指定 IP 地址机械臂已经暂停的程序。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。

调用示例：
const char* strIpAddress = "192.168.10.75";
int ret = resumeProgram(strIpAddress);
if(ret == 0)
{
printf("All program is resume!\n ");
}

104 stopAllProgram

int stopAllProgram(const char *strIpAddress = " ")
停止指定 IP 地址机械臂所有程序。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。

调用示例：
const char* strIpAddress = "192.168.10.75";
int ret = stopAllProgram(strIpAddress);
if(ret == 0)
{
printf("All program is stop!\n ");
}

105 isAnyTaskRunning

int isAnyTaskRunning(const char *strIpAddress = " ")
判断指定 IP 地址机械臂是否有程序在运行。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：有程序在运行。 -1：无程序在运行。

调用示例：
const char* strIpAddress = "192.168.10.75";
int ret = isAnyTaskRunning(strIpAddress);
if(ret < 0)
{
printf("Any program is not running!\n ");
}

106 cleanErrorInfo

int cleanErrorInfo(const char *strIpAddress = " ")
清除指定 IP 地址机械臂的错误信息。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。

调用示例：
const char* strIpAddress = "192.168.10.75";
int ret = cleanErrorInfo(strIpAddress);
if(ret < 0)
{
printf("cleanErrorInfo failed!\n ");
}

107 setCollisionLevel

int setCollisionLevel(int level, const char *strIpAddress = " ")
设置指定 IP 地址机械臂的碰撞检测类型。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** level：碰撞等级，int 类型，值为 0 或者下面几种值的组合（0 表示只考虑后台最大保护阈值）： 0x01：关节空间检测 0x02：笛卡尔空间检测 0x04： TCP 合力检测 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。
调用示例： const char* strIpAddress = "192.168.10.75"; int ret = setCollisionLevel(0x01|0x02, strIpAddress);

```
if(ret < 0)
{
   printf( "setCollision failed!\n ");
}
```

108 mappingInt8Variant

int mappingInt8Variant(const char *variableName,int index, const char *strIpAddress = " ")

在指定 IP 地址机械臂上映射 int8 型变量。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

variableName：变量名称。

index: 映射变量序号。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

详见 setMappingAddress 示例。

109 mappingDoubleVariant

int mappingDoubleVariant(const char *variableName,int index, const char *strIpAddress = " ")

在指定 IP 地址机械臂上映射 double 型变量。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

variableName：变量名称。

index: 映射变量序号。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

详见 setMappingAddress 示例。

110 mappingInt8IO

int mappingInt8IO(const char *groupName,const char *name,int index, const char *strIpAddress = " ")
在指定 IP 地址机械臂上映射数字信号 IO。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** groupName：IO 组名 name：IO 名字 index: 映射序号 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。
调用示例： 详见 setMappingAddress 示例。

111 mappingDoubleIO

int mappingDoubleIO(const char *groupName,const char *name,int index, const char *strIpAddress = " ")
在指定 IP 地址机械臂上映射模拟信号 IO。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** groupName：IO 组名。 name: IO 名称。 index: 映射序号。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。
调用示例： 详见 setMappingAddress 示例。

112 setMappingAddress

int setMappingAddress(double *dblAddress,int doubleItemCount,int8_t *int8Address,int

int8ItemCount, const char *strIpAddress = " ")

在指定 IP 地址机械臂上设置地址映射。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

dblAddress：double 型数据的映射地址，double 数组。

doubleItemCount：double 型数据的映射数量，int 型，最大支持 40 个。

int8Address：int8 型数据的映射地址，int8 数组。

int8ItemCount: int8 型数据的映射数量，int 型，最大支持 160 个。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

mapping 的使用说明：

mapping 的作用是将机械臂上 double 型变量，int8 型变量（程序变量表中），数字信号 IO，模拟信号 IO 通过索引与自定义的内存地址（数组）建立映射连接，通过自定义的内存地址访问上述变量

1、步骤 1：首先定义一些 INT8 型和 DOUBLE 型的索引值，通过枚举方式：

```
enum INT8_VALUE_INDEX
{
   DI0 = 0,
   DI1,
   DI2,
   DI3,
};

enum DOUBLE_VALUE_INDEX
{
   AI0 = 0,
   AI1,
   TEST_VALUE_0,
   TEST_VALUE_1,
};
```

2、步骤 2：将机械臂上 double 型变量，int8 型变量（程序变量表中），数字信号 IO，模拟信号 IO 与索引值建立映射：

```
//mapping double value
   mappingDoubleIO("board", "ai0", AI0);
   mappingDoubleIO("board", "ai1", AI1);
   mappingDoubleVariant("test_value_0", TEST_VALUE_0);
   mappingDoubleVariant("test_value_1", TEST_VALUE_1);

//mapping int8 value
   mappingInt8IO("board", "di0", DI0);
   mappingInt8IO("board", "di1", DI1);
   mappingInt8IO("board", "di2", DI2);
   mappingInt8IO("board", "di3", DI3);
```

3、步骤 3：设置映射地址（自定义的内存地址（数组））

```
   int8_t int8_array[4] = {0};
   double double_array[4] = {0.0};
   setMappingAddress(double_array, 4, int8_array, 4);
```

4、步骤 4：使用自定义的 double_array 和 int8_array 地址名通过索引值访问机械臂上 double 型变量，int8 型变量（程序变量表中），数字信号 IO，模拟信号 IO 的值

```
   lockMappingAddress();//给数据加锁
   printf("数字信号 DI[0-3] = {%d, %d, %d, %d}\n", int8_array[DI0], int8_array[DI1],
int8_array[DI2], int8_array[DI3]);
   printf("模拟信号 AI[2] = {%f, %f}\n", double_array[AI0], double_array[AI1]);
   printf("double 型变量 test_value_0=%f, test_value_1=%f\n",
double_array[TEST_VALUE_0], double_array[TEST_VALUE_1]);
   unlockMappingAddress();//给数据解锁
```

113 lockMappingAddress

int lockMappingAddress(const char *strIpAddress = " ")
在指定 IP 地址机械臂上访问数据时加锁。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：**

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

详见 setMappingAddress 示例。

114 unlockMappingAddress

int unlockMappingAddress(const char *strIpAddress = " ")

在指定 IP 地址机械臂上解锁数据锁。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

详见 setMappingAddress 示例。

115 getJointCount

int getJointCount(const char *strIpAddress = " ")

在指定 IP 地址机械臂上，获取其机械臂的关节数目。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

int 型数值关节数量。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
int count = getJointCount(strIpAddress);
printf( "Joint Count = %d\n ",count);
```

116 setWayPoint

int setWayPoint(const char *strWayPointName,double *dblTcppos,double *dblJoints, const char *strIpAddress = " ")

修改指定 IP 地址机械臂上路点变量的值。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

strWayPointName：路点变量名称 。

dblTcppos：位姿信息，大小为 6 的数组，其中，后三个角度为轴角。

dblJoints：关节角信息。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
//以七轴机械臂为例：
    #define DEGREE_TO_RAD(x) ((x)*PI / 180.0)
    double Target[7] = { DEGREE_TO_RAD(0.0), DEGREE_TO_RAD(0.0), DEGREE_TO_RAD(0.0), DEGREE_TO_RAD(90.0), DEGREE_TO_RAD(0.0), DEGREE_TO_RAD(-90.0), DEGREE_TO_RAD(0.0) };
    moveJ(Target, 0.8, 0.4);
    wait_move();

    const char *strWayPointName = "api_01";
    double add_dblTcppos[6] = { 0.0 };
    getTcpPos(add_dblTcppos);
    double add_dblJoints[7] = { 0.0 };
    getJointPos(add_dblJoints);

    int ret = addWayPoint(strWayPointName, add_dblTcppos, add_dblJoints);
    if (ret < 0)
    {
        printf( "addWayPoint failed!\n");
    }
    else
    {
        printf( "addWayPoint succeed!\n");
```

```
    }

    double get_dblTcppos[6] = { 0.0 };
    double get_dblJoints[7] = { 0.0 };
    ret = getWayPoint(strWayPointName, get_dblTcppos, get_dblJoints);
    if (ret < 0)
    {
        printf( "getWayPoint failed!\n");
    }
    else
    {
        printf( "getWayPoint succeed!\n");
        printf( "addWayPoint-Tcppos: %f, %f, %f, %f, %f, %f\n",
            add_dblTcppos[0], add_dblTcppos[1], add_dblTcppos[2], add_dblTcppos[3],
add_dblTcppos[4], add_dblTcppos[5]);
        printf( "getWayPoint-Tcppos: %f, %f, %f, %f, %f, %f\n\n\n",
            get_dblTcppos[0], get_dblTcppos[1], get_dblTcppos[2], get_dblTcppos[3],
get_dblTcppos[4], get_dblTcppos[5]);

        printf( "addWayPoint-Joints: %f, %f, %f, %f, %f, %f, %f\n",
            add_dblJoints[0], add_dblJoints[1], add_dblJoints[2], add_dblJoints[3],
add_dblJoints[4], add_dblJoints[5], add_dblJoints[6]);
        printf( "getWayPoint-Joints: %f, %f, %f, %f, %f, %f, %f\n\n\n",
            get_dblJoints[0], get_dblJoints[1], get_dblJoints[2], get_dblJoints[3],
get_dblJoints[4], get_dblJoints[5], get_dblJoints[6]);
    }

    double Target1[7] = { DEGREE_TO_RAD(0.0), DEGREE_TO_RAD(-60.0),
DEGREE_TO_RAD(0.0), DEGREE_TO_RAD(150.0), DEGREE_TO_RAD(0.0),
DEGREE_TO_RAD(-90.0), DEGREE_TO_RAD(0.0) };
    moveJ(Target1, 0.8, 0.4);
    wait_move();

    double set_dblTcppos[6] = { 0.0 };
    getTcpPos(set_dblTcppos);
    double set_dblJoints[7] = { 0.0 };
    getJointPos(set_dblJoints);
    ret = setWayPoint(strWayPointName, set_dblTcppos, set_dblJoints);
```

```
    if (ret < 0)
    {
        printf( "setWayPoint failed!\n");
    }
    else
    {
        printf( "setWayPoint succeed!\n");
    }

    ret = getWayPoint(strWayPointName, get_dblTcppos, get_dblJoints);
    if (ret < 0)
    {
        printf( "getWayPoint failed!\n");
    }
    else
    {
        printf( "getWayPoint succeed!\n");
        printf( "setWayPoint-Tcppos: %f, %f, %f, %f, %f, %f\n",
            set_dblTcppos[0], set_dblTcppos[1], set_dblTcppos[2], set_dblTcppos[3],
set_dblTcppos[4], set_dblTcppos[5]);
        printf( "getWayPoint-Tcppos: %f, %f, %f, %f, %f, %f\n\n\n",
            get_dblTcppos[0], get_dblTcppos[1], get_dblTcppos[2], get_dblTcppos[3],
get_dblTcppos[4], get_dblTcppos[5]);

        printf( "setWayPoint-Joints: %f, %f, %f, %f, %f, %f, %f\n",
            set_dblJoints[0], set_dblJoints[1], set_dblJoints[2], set_dblJoints[3],
set_dblJoints[4], set_dblJoints[5], set_dblJoints[6]);
        printf( "getWayPoint-Joints: %f, %f, %f, %f, %f, %f, %f\n\n\n",
            get_dblJoints[0], get_dblJoints[1], get_dblJoints[2], get_dblJoints[3],
get_dblJoints[4], get_dblJoints[5], get_dblJoints[6]);
    }

    ret = deleteWayPoint(strWayPointName);
    if (ret < 0)
    {
        printf( "deleteWayPoint failed!\n");
    }
    else
```

```
{
    ret = getWayPoint(strWayPointName, get_dblTcppos, get_dblJoints);
    if (ret < 0)
    {
        printf( "deleteWayPoint succeed!\n");
    }
    else
    {
        printf( "deleteWayPoint failed!\n");
    }
}
```

117 getWayPoint

int getWayPoint(const char *strWayPointName,double *dblTcppos,double *dblJoints, const char *strIpAddress = " ")

获取指定 IP 地址机械臂上路点变量信息。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

strWayPointName：路点变量名称 。

dblTcppos：位姿信息，大小为 6 的数组，其中，后三个角度为轴角。

dblJoints：关节角信息。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

详见 setWayPoint 示例。

118 addWayPoint

int addWayPoint(const char *strWayPointName,double *dblTcppos,double *dblJoints, const char *strIpAddress = " ")

在指定 IP 地址机械臂上新增路点变量。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

strWayPointName：路点变量名称 。

dblTcppos：位姿信息，大小为 6 的数组，其中，后三个角度为轴角。 dblJoints：关节角信息。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。
调用示例： 详见 setWayPoint 示例。

119 deleteWayPoint

int deleteWayPoint(const char *strWayPointName, const char *strIpAddress = " ")
在指定 IP 地址机械臂上删除路点变量。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** strWayPointName：路点变量名称。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。
调用示例： 详见 setWayPoint 示例。

120 zeroSpaceManualMove

int zeroSpaceManualMove (int direction,double *dblJointVel,double *dblJointAcc,const char *strIpAddress = " ")
控制指定 IP 地址机械臂进入或退出零空间手动移动模式。 适用臂型：通用医疗臂、定制医疗臂 **参数：** direction：输入参数，控制零空间手动运动的方向，1 表示向前，-1 表示向后运动。 dblJointVel：输入参数，各关节运动的速度，单位 rad/s。 dblJointAcc：输入参数，各关节运动的加速度，单位 rad/s^2 。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
//以七轴机械臂为例：
const char* strIpAddress = "192.168.10.75";
double dblJointVel[7]={0.1,0.1,0.1,0.1,0.1,0.1,0.1};
double dblJointAcc[7]={0.1,0.1,0.1,0.1,0.1,0.1,0.1};
int ret = zeroSpaceManualMove(1,dblJointVel, dblJointAcc, strIpAddress);
if (ret < 0){
   printf("Diana API zeroSpaceManualMove failed!\n");
}
else{
   M_SLEEP(5000);
}
stop(strIpAddress);
```

121 setExternalAppendTorCutoffFreq

```
int setExternalAppendTorCutoffFreq(double dblFreq, const char *strIpAddress =  " ")
```

在指定 IP 地址机械臂上，设置各关节附加力矩的滤波截止频率。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

dblFreq：设置各关节附加力矩的滤波截止频率。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
double dblFreq = 1.0;
const char* strIpAddress =  "192.168.10.75";
int ret = setExternalAppendTorCutoffFreq(dblFreq, strIpAddress);
if(ret < 0)
{
   printf( "setExternalAppendTorCutoffFreq failed! Return value = %d\n ", ret);
}
```

122 getHomingState

int getHomingState(const char *strIpAddress = " ")

针对指定 IP 地址的医疗臂，在其控制箱每次上电后，第七关节需要进行零位状态查询。并根据返回状态决定是否要进行寻零操作。

适用臂型：定制医疗臂 1.0。

参数：

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

3：当前不在零位（需要寻零）。

2：寻零过程中出错。

1：寻零进行中。

0：当前已在零位（寻零成功）。

-1：调用失败。

调用示例：

```
ret = releaseBrake();
M_SLEEP(2000);
if(ret < 0){
  int error_code = getLastError();
  string msg = formatError(error_code);
  printf("release brake failed(%s)\n", msg.c_str());
  int state = getHomingState();
  printf("getHomingState = %d\n", state);
  if(state != 0){
    ret = startHoming();
    printf("startHoming = %d\n", ret);
    int cnt = 0;
    while(state != 0){
      M_SLEEP(100);
      state = getHomingState();
      printf("getHomingState(%d) = %d\n", cnt, state);
      if(cnt++ == 60000){ //6s
        break;
      }
    }
    ret = stop();
    printf("stop = %d\n", ret);
    state = getHomingState();
    printf("getHomingState(%d) = %d\n", cnt, state);
  }
}
```

```
ret = releaseBrake();
printf("releaseBrake again = %d\n", ret);
M_SLEEP(2000);
```

123 startHoming

int startHoming(const char *strIpAddress = " ")

在指定 IP 的地址的机械臂上，在其控制箱每次上电后，第七关节需要进行寻零操作时调用该函数启动寻零程序。需要与 stop 函数配对使用。当寻零结束或失败后，调用 stop 函数。当七关节不在零位时，调用 *releaseBrake* 开报闸会失败，*getLastError* 会返回 *ERROR_CODE_HOME_POSITION_ERROR*（*-2305*）。

适用臂型：定制医疗臂 1.0。

参数：

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

参照 getHomingState 中示例。

124 setEndKeyEnableState

int setEndKeyEnableState(bool bEnable,const char *strIpAddress = " ")

在指定 IP 地址的机械臂上，使能或禁用末端按键进入自由驱动的绑定。当用户希望赋予末端按键其他功能时，先调用该功能禁用末端按键进入自由驱动的绑定，再通过 readDI 读取末端按键状态以绑定自定义的功能。

适用臂型：通用医疗臂、定制医疗臂

参数：

bEnable：输入参数，bool 量，使能或禁用末端按键进入自由驱动的绑定。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
```

```
int ret = setEndKeyEnableState (false, strIpAddress);
if(ret < 0)
{
   printf( "setEndKeyEnableState failed! Return value = %d\n ", ret);
}
```

125 updateForce

int updateForce(double dblForceValue,const char *strIpAddress = " ")
在指定 IP 地址机械臂上，在力控模式下，实时改变力指令的大小。 适用臂型：通用医疗臂、定制医疗臂 **参数：** dblForceValue：输入参数，力的大小。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。

调用示例：
int iFrameType = 1;
double dblFrameMatrix[16] = {1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1};
double dblForceDirection[3]={0,0,-1};
double dblForceValue = 1.0;
double dblMaxVel = 0.1;
double dblMaxOffset = 0.2;
const char* strIpAddress = "192.168.10.75";
double joints[7] = {0,0,0,3.141592653/2,0,0,0};//以七轴机械臂为例
moveJ(joints,0.2,0.2,strIpAddress);
wait_move(strIpAddress);
if (enterForceMode(iFrameType, dblFrameMatrix, dblForceDirection, dblForceValue, dblMaxVel, dblMaxOffset,strIpAddress) < 0)
{
printf("Diana API enterForceMode failed!\n");
}else{
int count = 0;
while(count++ < 2000){
dblForceValue -=0.001;
updateForce(dblForceValue,strIpAddress);

```
        M_SLEEP(1);
    }
    int intExitMode = 0;
    if (leaveForceMode(intExitMode,strIpAddress) < 0)
    {
        printf("Diana API leaveForceMode failed!\n");
    }
}
```

126 getCartImpedanceCoordinateType

int getCartImpedanceCoordinateType(const char *strIpAddress = " ")

在指定 IP 地址机械臂上，获取设置笛卡尔阻抗模式时参考坐标系种类。

适用臂型：通用医疗臂、定制医疗臂

参数：

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

坐标系的类型，0：基坐标系， 1：工具坐标系。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
int ret = getCartImpedanceCoordinateType(strIpAddress);
if(ret == 0){
    printf( "it’s base coordinate\n ");
}
else{
    printf( "it’s tool coordinate\n ");
}
```

127 setCartImpedanceCoordinateType

int setCartImpedanceCoordinateType(const int intCoordinateType, const char *strIpAddress = " ")

在指定 IP 地址机械臂上，设置笛卡尔阻抗模式时参考坐标系种类。

适用臂型：通用医疗臂、定制医疗臂

参数：

intCoordinateType：参考坐标系的类型，0：基坐标系， 1：工具坐标系。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
int ret = setCartImpedanceCoordinateType(0,strIpAddress);
if(ret < 0){
   printf( "setCartImpedanceCoordinateType failed! Return value = %d\n ", ret);
}
```

128 getInertiaMatrix

```
int getInertiaMatrix(double *dblMotorPosition, double *dblInerMatrix,const char *strIpAddress = " ")
```

在指定 IP 地址机械臂上，获取惯性参数矩阵。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

dblMotorPosition：输入参数，机械臂的关节角，大小为 JOINT_NUM 的数组。

dblInerMatrix：输出参数，惯性矩阵,大小为 JOINT_NUM * JOINT_NUM 的数组。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
double dblMotorPosition[JOINT_NUM] = {0};
double dblInerMatrix[JOINT_NUM*JOINT_NUM] = {0};
int ret = getInertiaMatrix (dblMotorPosition, dblInerMatrix,strIpAddress);
if(ret < 0)
{
   printf( "getInertiaMatrix failed! Return value = %d\n ", ret);
}
```

129 getSixAxiaForce

```
int getSixAxiaForce(double *dblForce, const char *strIpAddress =  " ")
```

在指定 IP 地址机械臂上，获取六维力传感器的读数。

适用臂型：通用医疗臂、Thor3

参数：

dblForce：输出参数，六维力传感器的读数，大小为 6 的数组。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
double dblForce[6] = {0};
int ret = getSixAxiaForce (dblForce,strIpAddress);
if(ret < 0)
{
   printf( "getSixAxiaForce failed! Return value = %d\n ", ret);
}
```

130 inverseClosedFull

```
int inverseClosedFull(const double *pose,const int lock_joint_index, const double lock_joint_position, double *ref_joints, double *active_tcp=nullptr,const char *strIpAddress = "")
```

在指定 IP 地址机械臂上，基于工具坐标系，给定一个参考关节角，约束单轴求逆解，注意，医疗臂不能锁定四轴。现支持在不同工具坐标系下求逆解，由传入的 active_tcp 决定。

适用臂型：通用医疗臂、定制医疗臂

参数：

pose：输入位姿数组首地址，数据为包含 active_tcp 坐标（x, y, z）和旋转矢量（轴角坐标）组合。

lock_joint_index：输入参数，被约束的关节号。

lock_joint_position：输入参数，被约束关节的角度，单位为弧度。

ref_joints：参考的关节角，大小为 JOINT_NUM 的数组。

active_tcp：工具坐标系对应的位姿向量，大小为 6 的数组，为空时，将使用默认的工具坐标系 default_tcp。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

非负数：生成逆解对应的 ID。

-1：失败。

调用示例：

```
const char *strIpAddress = "192.168.10.75";
double pose[6] = {0.087, 0.0, 1.0827, 0.0, 0.0, 0.0};
int lock_joint_index = 6;
double lock_joint_position = 0;
double ref_joints[JOINT_NUM] = {0.0};
double joints[JOINT_NUM]={0.0};
int id = inverseClosedFull(pose, lock_joint_index, lock_joint_position, ref_joints, nullptr,strIpAddress);
if (id == -1){
   printf("inverseClosedFull failed! Return value = %d\n", id);
}
else{
   int size = getInverseClosedResultSize(id,strIpAddress);
   int ret = -1;
   if (size > 0){
      for (int i = 0; i < size; ++i){
         if (getInverseClosedJoints(id, i, joints, strIpAddress) == 0){
            printf("(%lf,%lf,%lf,%lf,%lf,%lf,%lf)", joints[0], joints[1], joints[2], joints[3], joints[4], joints[5], joints[6]);//以七轴机械臂为例
         }
      }
      ret = destoryInverseClosedItems(id);
   }
}
```

131 getInverseClosedResultSize

```
int getInverseClosedResultSize(const int id,const char *strIpAddress =  " ")
```

在指定 IP 地址的机械臂上，根据 ID 获取约束单轴求逆解结果的组数。

适用臂型：通用医疗臂、定制医疗臂

参数：

id：输入参数，所求逆解对应的 ID。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值： 非负数：ID 对应逆解的组数。 -1：失败。
调用示例： 见 inverseClosedFull 示例。

132 getInverseClosedJoints

int getInverseClosedJoints(const int id,const int index,double *joints,const char *strIpAddress = " ")
在指定 IP 地址机械臂上，根据 ID 按索引获取对应关节角。 适用臂型：通用医疗臂、定制医疗臂 **参数：** id：输入参数，所求逆解对应的 ID。 index：输入参数，对应的多组逆解中的编号。 joints：输出参数，需要求的多组逆解中编号对应的逆解值。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功 -1：失败。
调用示例： 见 inverseClosedFull 示例。

133 destoryInverseClosedItems

int destoryInverseClosedItems(const int id,const char *strIpAddress = " ")
在指定 IP 地址机械臂上，根据 ID 删除约束单轴求逆解的结果数据集。 适用臂型：通用医疗臂、定制医疗臂 **参数：** id：输入参数，逆解对应的 ID。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。
调用示例：

见 inverseClosedFull 示例。

134 nullSpaceFreeDriving

int nullSpaceFreeDriving(bool enable,const char *strIpAddress = " ")
zeroSpaceFreeDriving 的宏，用法详见 zeroSpaceFreeDriving。 适用臂型：通用医疗臂、定制医疗臂
调用示例： 见 zeroSpaceFreeDriving 示例。

135 nullSpaceManualMove

int nullSpaceManualMove(bool enable,const char *strIpAddress = " ")
zeroSpaceManualMove 的宏，用法详见 zeroSpaceManualMove。 适用臂型：通用医疗臂、定制医疗臂
调用示例： 见 zeroSpaceManualMove 示例。

136 calculateJacobi

int calculateJacobi(double *dblJacobiMatrix, const double *dblJointPosition, int intJointCount, const char *strIpAddress = " ")
在指定 IP 地址机械臂上，求解末端法兰中心点坐标系相对于基坐标系的雅各比矩阵。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** dblJacobiMatrix：输出参数，雅各比矩阵，大小为 6*JOINT_NUM 的矩阵。 dblJointPosition：输入参数，用于计算雅各比矩阵的关节角（该关节角与机械臂当前反馈的位姿无关），大小为 JOINT_NUM 的数组。 intJointCount：输入参数，关节数量。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。

调用示例：
const char *strIpAddress = "192.168.10.75";
const int intJointCount = 7; //以七轴机械臂为例
const int intTcpCount = 6;
double dblJacobiMatrix[intTcpCount*intJointCount] = {0};

```
double dblJointPosition[intJointCount] = {0};
int ret = calculateJacobi(dblJacobiMatrix,dblJointPosition,intJointCount,strIpAddress);
if(ret == - 1){
   printf("cannot get jacobi matrix");
}else{
   for(int i = 0;i< intJointCount;++i){
      for(int j=0;j< intTcpCount;++j){
         printf("%lf,",dblJacobiMatrix[i* intTcpCount + j]);
      }
      printf("\n");
   }
}
```

137 calculateJacobiTF

```
int calculateJacobiTF(double *dblJacobiMatrix, const double *dblJointPosition, int intJointCount,
double *active_tcp=nullptr, const char *strIpAddress =  " ")
```

在指定 IP 地址机械臂上，求解工具中心点坐标系相对于基坐标系的雅各比矩阵 。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

dblJacobiMatrix：输出参数，雅各比矩阵，大小为 6*JOINT_NUM 的矩阵。

dblJointPosition：输入参数，用于计算雅各比矩阵的关节角（该关节角与机械臂当前反馈的位姿无关），大小为 JOINT_NUM 的数组。

intJointCount：输入参数，关节数量。

active_tcp：当前工具坐标系对应的位姿向量，大小为 6 的数组，为空时，将使用默认的工具坐标系 default_tcp。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char *strIpAddress = "192.168.10.75";
const int intTcpCount = 6;
double dblJacobiMatrix[6*JOINT_NUM] = {0};
double dblJointPosition[JOINT_NUM] = {0};
int ret = calculateJacobiTF(dblJacobiMatrix,dblJointPosition, JOINT_NUM, nullptr, strIpAddress);
```

```
if(ret == - 1){
   printf("cannot get jacobi matrix");
}else{
   for(int i = 0;i<intTcpCount;++i){
      for(int j=0;j<JOINT_NUM;++j){
         printf("%lf,",dblJacobiMatrix[i* intTcpCount + j]);
      }
      printf("\n");
   }
}
```

138 getTcpForceInToolCoordinate

int getTcpForceInToolCoordinate(double *forces,const char *strIpAddress = " ")

在指定 IP 地址机械臂上，获取工具坐标系的 Tcp 外力值。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

forces：输出参数，Tcp 外力，大小为 6。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char *strIpAddress = "192.168.10.75";
double forces[6] = {0.0};
int ret = getTcpForceInToolCoordinate(forces,strIpAddress);
for(int j=0;j<6;++j){
   printf("%lf,",forces[j]);
}
```

139 speedLOnTcp

int speedLOnTcp(double *speed, double *a, double t, double *active_tcp=nullptr, const char *strIpAddress = " ")

速度模式优化版，使指定 IP 地址机械臂笛卡尔空间下直线运动。时间 t 是可选项，如果提供了 t 值，机械臂将在 t 时间后减速。如果没有提供时间 t 值，机械臂将在达到目标速度时减速。该函数调用后立即返回。停止运动需要调用 stop 函数。现支持在不同工具坐标系下移动，由传入的 active_tcp 决定。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

speed：指定系统当前工具中心点的空间速度，数组长度为 6（位置和旋转），其中前 3 个单位为 m/s，后 3 个单位为 rad/s。平移速度和旋转速度的参考坐标系由 active_tcp 指定。

a：加速度，单位：m/s^2，rad/s^2。

t：时间，单位：s。

active_tcp：基于法兰中心的位姿，大小为 6 的数组（位置和旋转矢量（轴角）），平移和旋转方向参考此位姿，旋转中心是系统当前工具中心点；为空时平移和旋转方向参考系统当前工具坐标系，旋转中心是系统当前工具中心点。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
double speeds[6] = {0.1,0.0,0.0,0.0,0.0,0.0};
double acc[2] = {0.30, 0.50};
const char* strIpAddress =  "192.168.10.75";
int ret = speedLOnTcp(speeds, acc, 0,  nullptr, strIpAddress);
if(ret < 0)
{
   printf( "speedLOnTcp failed! Return value = %d\n ", ret);
}
```

140 inverseClosedIdeal

int inverseClosedIdeal (const double *pose, const double lock_robot_arm_angle, double *ref_joints, double *active_tcp=nullptr, const char *strIpAddress = " ")

根据末端位姿，在固定臂角的情况下，求更接近某组关节位置的逆解集合。现支持在不同工具坐标系下计算，由传入的 active_tcp 决定。

适用臂型：定制医疗臂

参数：

pose：输入终点位姿数组首地址，数组长度为 6。包含 TCP 坐标（x, y, z）和轴角（rx, ry, rz）组合的矢量数据。pose 可以相对于不同工具坐标系，由传入的 active_tcp 决定。

lock_robot_arm_angle：输入参数，臂角，单位：rad。

ref_joints：输入参数，逆解集合参照的关节角数组首地址，数组长度为 JOINT_NUM，

单位：rad。

active_tcp：基于法兰中心的位姿，大小为 6 的数组（位置和旋转矢量（轴角）），平移和旋转方向参考此位姿，旋转中心是系统当前工具中心点；为空时平移和旋转方向参考系统当前工具坐标系，旋转中心是系统当前工具中心点。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

非负数：生成逆解集合对应的 ID。

-1：失败。

调用示例：

```
    double joint[JOINT_NUM] = {0};//vel = 0.2,  acc = 0.4;
   double pose[6] = {};
   double active_tcp[6] = {0.1,0.1,0.1,0,0,0};
   double start_point[JOINT_NUM]={0.188914,0.100130,-0.196280,2.046503,0.023230,-
0.996984,-0.019024};
   forward(start_point, pose, active_tcp);
   printf("start_point = {%f,%f,%f,%f,%f,%f,%f}\n"
          , start_point[0]
          , start_point[1]
          , start_point[2]
          , start_point[3]
          , start_point[4]
          , start_point[5]
          , start_point[6]);
   printf("pose = {%f,%f,%f,%f,%f,%f}\n"
          , pose[0]
          , pose[1]
          , pose[2]
          , pose[3]
          , pose[4]
          , pose[5]);
   int id = inverseClosedIdeal(pose, 30*PI/180,start_point,active_tcp);
   printf("inverseClosed return id = %d\n",id);
   int size = getInverseClosedIdealResultSize(id);
   printf("getInverseClosedResultSize return size = %d\n",size);
   for(int i = 0; i < size; ++i){
      double retJoints[JOINT_NUM] = {0};
      int ret = getInverseClosedIdealJoints(id, i, retJoints);
      printf("retJoints=%d, {%f,%f,%f,%f,%f,%f,%f}\n", ret
          , retJoints[0]
          , retJoints[1]
          , retJoints[2]
          , retJoints[3]
          , retJoints[4]
          , retJoints[5]
          , retJoints[6]); //以七轴机械臂为例
          double retPose[6] = {0};
      forward(retJoints, retPose,active_tcp);
      printf("retPose[%d] = {%f,%f,%f,%f,%f,%f}\n", i
            , retPose[0]
```

```
            , retPose[1]
            , retPose[2]
            , retPose[3]
            , retPose[4]
            , retPose[5]);
    }
    ret = destoryInverseClosedIdealItems(id);
```

141 getInverseClosedIdealResultSize

int getInverseClosedIdealResultSize (const int id, const char *strIpAddress = " ")

在指定 IP 地址的机械臂上，根据 ID 获取约束臂角求逆解结果的组数。

适用臂型：定制医疗臂

参数：

id：输入参数，所求逆解集合对应的 ID。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

非负数：ID 对应逆解的组数。

-1：失败。

调用示例：

见 inverseClosedIdeal 示例。

142 getInverseClosedIdealJoints

int getInverseClosedIdealJoints (const int id, int index, double *joints, const char *strIpAddress = " ")

在指定 IP 地址机械臂上，根据 ID 按索引获取约束臂角求逆解集合中对应关节角。

适用臂型：定制医疗臂

参数：

id：输入参数，所求逆解对应的 ID。

index：输入参数，对应的多组逆解中的编号。

joints：输出参数，需要求的多组逆解中编号对应的逆解值，单位：rad。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

见 inverseClosedIdeal 示例。

143 destoryInverseClosedIdealItems

int destoryInverseClosedIdealItems (const int id, const char *strIpAddress = " ")
在指定 IP 地址机械臂上，根据 ID 删除约束臂角求逆解的结果数据集。 适用臂型：定制医疗臂 **参数：** id：输入参数，逆解集合对应的 ID。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。
调用示例： 见 inverseClosedIdeal 示例。

144 getSixAxisZeroOffset

int getSixAxisZeroOffset (double *dblZeroOffset, const char *strIpAddress = " ")
获取指定 IP 地址机械臂上六维力传感器的零偏数据。 适用臂型：通用医疗臂、Thor3 **参数：** dblZeroOffset：输出参数，用于接收六维力传感器零偏数据的数组首地址，数组长度为 10。零偏数据的含义依次为：m、mx、my、mz、x、y、z、rx、ry、rz。其中： m 为六维力传感器承载的且能感知到的负载质量，单位：Kg； mx、my、mz 为六维力负载的质心坐标（在六维力自身坐标系下的表达），单位：m； x、y、z、rx、ry、rz 为六维力在各个维度的零偏值，其中 x、y、z 的单位为 N，rx、ry、rz 的单位为 N.m。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。
调用示例： double dblZeroOffset [10] = {0}; const char* strIpAddress = "192.168.10.75"; int ret = getSixAxisZeroOffset (dblZeroOffset, strIpAddress);

```
if(ret < 0){
   printf( "getSixAxisZeroOffset failed! Return value = %d\n ", ret);
} else{
   printf( "getSixAxisZeroOffset success! Return value
={%f, %f, %f, %f, %f, %f, %f, %f, %f, %f}\n "
   , dblZeroOffset[0]
   , dblZeroOffset[1]
   , dblZeroOffset[2]
   , dblZeroOffset[3]
   , dblZeroOffset[4]
   , dblZeroOffset[5]
   , dblZeroOffset[6]
   , dblZeroOffset[7]
   , dblZeroOffset[8]
   , dblZeroOffset[9]);
}
```

145 setSixAxisZeroOffset

int setSixAxisZeroOffset (double *dblZeroOffset, const char *strIpAddress = " ")

设置指定 IP 地址机械臂上六维力传感器的零偏数据。设置后需调用 saveEnvironment 函数才能使设置永久生效。

适用臂型：通用医疗臂、Thor3

参数：

dblZeroOffset：输入参数，用于存放零偏数据的数组首地址，数组长度为 10。零偏数据的含义依次为：m、mx、my、mz、x、y、z、rx、ry、rz。其中：

m 为六维力传感器承载的且能感知到的负载质量，单位：Kg；

mx、my、mz 为六维力负载的质心坐标（在六维力自身坐标系下的表达），单位：m；

x、y、z、rx、ry、rz 为六维力在各个维度的零偏值，其中 x、y、z 的单位为 N，rx、ry、rz 的单位为 N.m。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
```

```
double dblSixAxis[10] = {0.4732,0.0,0.0064,0.0361,-6.6443,-11.7713,-1.6271,-
0.5637,0.6157,0.007};
ret = setSixAxisZeroOffset(dblSixAxis, strIpAddress);
if(ret >= 0){
   printf("setSixAxisZeroOffset = %d\n",ret);
   ret = saveEnvironment(strIpAddress);
}
```

146 setDefaultActiveWorkpiece

int setDefaultActiveWorkpiece (double *dblWop, const char *strIpAddress = " ")

在指定 IP 地址机械臂上设置默认工件坐标系。设置后需调用 saveEnvironment 函数才能使设置永久生效。

适用臂型：通用医疗臂、Thor3

参数：

dblWop ：输入参数，工件坐标系相对于基坐标系的 4*4 齐次变换矩阵的首地址，数组长度为 16。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
double dblWop [16] = {1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1};
ret = setDefaultActiveWorkpiece (dblWop, strIpAddress);
if(ret >= 0){
   printf("setDefaultActiveWorkpiece = %d\n",ret);
   ret = saveEnvironment(strIpAddress);
}
```

147 setDefaultActiveWorkpiecePose

int setDefaultActiveWorkpiecePose (double *dblWopPose, const char *strIpAddress = " ")

在指定 IP 地址机械臂上设置默认工件坐标系。设置后需调用 saveEnvironment 函数才能使设置永久生效。

适用臂型：通用医疗臂、Thor3

参数：

dblWopPose：输入参数，工件坐标系相对于基坐标系位姿向量的首地址，数组长度为6。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
double dblWopPose [6] = {0.1,0,0,0,0,0};
ret = setDefaultActiveWorkpiecePose (dblWopPose, strIpAddress);
if(ret >= 0){
   printf("setDefaultActiveWorkpiecePose = %d\n",ret);
   ret = saveEnvironment(strIpAddress);
}
```

148 getDefaultActiveWorkpiece

```
int getDefaultActiveWorkpiece (double *dblWop, const char *strIpAddress =  " ")
```

获取指定 IP 地址机械臂上默认工件坐标系相对于基坐标系的 4*4 齐次变换矩阵。

适用臂型：通用医疗臂、Thor3

参数：

dblWop：输出参数，工件坐标系相对于基坐标系的 4*4 齐次变换矩阵的首地址，数组长度为 16。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
double dblWop [16] = {0};
ret = getDefaultActiveWorkpiece (dblWop, strIpAddress);
if(ret < 0){
   printf("getDefaultActiveWorkpiece failed ret = %d\n",ret);
}
```

149 getDefaultActiveWorkpiecePose

int getDefaultActiveWorkpiecePose (double *dblWopPose, const char *strIpAddress = " ")
获取指定 IP 地址机械臂上默认工件坐标系相对于基坐标系的转换位姿向量。 适用臂型：通用医疗臂、Thor3 **参数：** dblWopPose：输出参数，工件坐标系相对于基坐标系位姿向量的首地址，数组长度为 6。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。

调用示例：
const char* strIpAddress = "192.168.10.75";
double dblWopPose [6] = {0};
ret = getDefaultActiveWorkpiecePose (dblWopPose, strIpAddress);
if(ret < 0){
printf("getDefaultActiveWorkpiecePose failed ret = %d\n",ret);
}

150 setControllerFeatureCode

int setControllerFeatureCode (controller_feature intFeatureCode, bool bEnable, const char *strIpAddress = " ")
在指定 IP 地址机械臂上禁用或使能某项控制器的特性。设置后需调用 saveEnvironment 函数才能使设置永久生效。 目前不支持该功能 **参数：** intFeatureCode：输入参数，controller_feature 枚举类型变量。目前仅支持 NONE_FEATURE（int 值为 0）代表无特性；AUTO_SWITCH_TO_IMPEDANCE_MODE_WHEN_COLLISION_DETECTED （int 值为 1）碰撞后退出到阻抗模式特性。 bEnable：输入参数，bool 型变量，使能或禁用 intFeatureCode 代表的特性。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：**

0：成功。

-1：失败。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
int ret = setControllerFeatureCode
(AUTO_SWITCH_TO_IMPEDANCE_MODE_WHEN_COLLISION_DETECTED, true, strIpAddress);
if(ret >= 0){
   printf("setControllerFeatureCode = %d\n",ret);
   ret = saveEnvironment(strIpAddress);
}
```

151 testControllerFeature

int testControllerFeature (controller_feature intFeatureCode, bool &bEnable, const char *strIpAddress = " ")

在指定 IP 地址机械臂上验证是否具备某项特性。

目前不支持该功能

参数：

intFeatureCode：输入参数，controller_feature 枚举类型变量。目前仅支持 NONE_FEATURE（int 值为 0）代表无特性；AUTO_SWITCH_TO_IMPEDANCE_MODE_WHEN_COLLISION_DETECTED （int 值为 1）碰撞后退出到阻抗模式特性。

bEnable：输出参数，bool 型变量，用于接收要查询特性的使能状态。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
bool bEnabled = false;
int ret = testControllerFeature (AUTO_SWITCH_TO_IMPEDANCE_MODE_WHEN_COLLISION_DETECTED,
bEnabled, strIpAddress);
if(ret >=  0)
{
      printf( "testControllerFeature return bEnabled = %d\n ", (int) bEnabled);
}
```

152 enableSixAxis

int enableSixAxis (six_axis_type_e type, bool &bEnable, const char *strIpAddress = " ")

在指定 IP 地址机械臂上六维力零偏参数在各种模式下是否使用。

适用臂型：通用医疗臂、Thor3

参数：

type：输入参数，six_axis_type_e 枚举变量。目前仅支持 SIX_AXIS_IN_FREE_DRIVING（零力驱动模式下使能或禁用六维力参数）；未来可能会支持 SIX_AXIS_IN_FORCE_MODE（力控模式下使能或禁用六维力参数）、SIX_AXIS_IN_JOINT_IMPEDANCE（关节空间阻抗中使能或禁用六维力参数）和 SIX_AXIS_IN_CART_IMPEDANCE（笛卡尔空间阻抗中使能或禁用六维力参数）。

bEnable：既是输入又是输出参数，bool 型变量，使能或禁用六维力参数在 type 模式中的使用，同时在函数调用后返回当前系统状态，可用于判断是否设置成功。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
   bool bFreeDrivingHasSixAxis = false;
   bool bEnabled = bFreeDrivingHasSixAxis;
   int ret = enableSixAxis(six_axis_type_e::SIX_AXIS_IN_FREE_DRIVING, bEnabled);
   if(ret >= 0){
     if(bEnabled != bFreeDrivingHasSixAxis){
        printf("enableSixAxis->six_axis_type_e::SIX_AXIS_IN_FREE_DRIVING = %d
failed!!!\n", (int)bEnabled);
     } else {
        printf("enableSixAxis->six_axis_type_e::SIX_AXIS_IN_FREE_DRIVING = %d
success!!!\n", (int)bEnabled);
     }
   } else {
        printf( "enableSixAxis failed! ret = %d\n ", ret);
   }
```

153 isSixAxisEnabled

bool isSixAxisEnabled (six_axis_type_e type, const char *strIpAddress = " ")

在指定 IP 地址机械臂上查询六维力零偏参数在某种模式下的使能状态。

适用臂型：通用医疗臂、Thor3

参数：

type：输入参数，six_axis_type_e 型枚举变量。目前仅支持 SIX_AXIS_IN_FREE_DRIVING（零力驱动模式下使能或禁用六维力参数）；未来可能会

支持 SIX_AXIS_IN_FORCE_MODE（力控模式下使能或禁用六维力参数）、SIX_AXIS_IN_JOINT_IMPEDANCE（关节空间阻抗中使能或禁用六维力参数）和SIX_AXIS_IN_CART_IMPEDANCE（笛卡尔空间阻抗中使能或禁用六维力参数）。

bEnable：输出参数，bool 型变量，用于接收要查询的 type 模式下对应的使能状态。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

true：使能。

false：禁用。

调用示例：

```
bool bFreeDrivingHasSixAxis =
isSixAxisEnabled(six_axis_type_e::SIX_AXIS_IN_FREE_DRIVING);
printf("isSixAxisEnabled->six_axis_type_e::SIX_AXIS_IN_FREE_DRIVING = %d\n",
(int)bFreeDrivingHasSixAxis);
```

154 getMechanicalJointsPositionRange

int getMechanicalJointsPositionRange (double *dblMinPos, double *dblMaxPos, const char *strIpAddress = " ")

获取指定 IP 地址机械臂各关节角的机械限位。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

dblMinPos：输出参数，用于存放各关节角的最小机械限位，大小为 JOINT_NUM 的数组，单位：rad。

dblMaxPos：输出参数，用于存放各关节角的最大机械限位，大小为 JOINT_NUM 的数组，单位：rad。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

-1：失败。

0：成功。

调用示例：

```
const char* robot0 =  "192.168.10.75";
double dblMinPos[JOINT_NUM] = {0}, dblMaxPos[JOINT_NUM]={0};
int ret = getMechanicalJointsPositionRange(dblMinPos, dblMaxPos, robot0);
printf("getMechanicalJointsPositionRange ret = %d dblMinPos = {%f,%f,%f,%f,%f,%f,%f}
dblMaxPos = {%f,%f,%f,%f,%f,%f,%f}\n"
            , ret
            , dblMinPos[0]
            , dblMinPos[1]
            , dblMinPos[2]
            , dblMinPos[3]
            , dblMinPos[4]
            , dblMinPos[5]
            , dblMinPos[6]
            , dblMaxPos[0]
            , dblMaxPos[1]
            , dblMaxPos[2]
            , dblMaxPos[3]
            , dblMaxPos[4]
            , dblMaxPos[5]
            , dblMaxPos[6]);//以七轴机械臂为例
```

155 getMechanicalMaxJointsVel

int getMechanicalMaxJointsVel (double *dblVel, const char *strIpAddress = " ")

获取指定 IP 地址机械臂各关节最大机械速度。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

dblVel：输出参数，用于存放各关节的最大机械速度，大小为 JOINT_NUM 的数组，单位：rad/s。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

-1：失败。

0：成功。

调用示例：

```
const char* robot0 =  "192.168.10.75";
double dblMaxVel[JOINT_NUM]={0};
int ret = getMechanicalMaxJointsVel(dblMaxVel, robot0);
printf("getMechanicalJointsMaxVel ret = %d dblMaxVel = {%f,%f,%f,%f,%f,%f,%f}\n"
, ret
, dblMaxVel[0]
, dblMaxVel[1]
, dblMaxVel[2]
, dblMaxVel[3]
, dblMaxVel[4]
, dblMaxVel[5]
, dblMaxVel[6]); //以七轴机械臂为例
```

156 getMechanicalMaxJointsAcc

int getMechanicalMaxJointsAcc (double *dblAcc, const char *strIpAddress = " ")

获取指定 IP 地址机械臂各关节最大机械加速度。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

dblAcc：输出参数，用于存放各关节的最大机械加速度，大小为 JOINT_NUM 的数组，单位：rad/s^2。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

-1：失败。

0：成功。

调用示例：

```
const char* robot0 =  "192.168.10.75";
double dblMaxAcc[JOINT_NUM]={0};
int ret = getMechanicalMaxJointsAcc(dblMaxAcc, robot0);
printf("getMechanicalJointsMaxAcc ret = %d dblMaxAcc = {%f,%f,%f,%f,%f,%f,%f}\n"
, ret
, dblMaxAcc[0]
, dblMaxAcc[1]
, dblMaxAcc[2]
, dblMaxAcc[3]
```

```
, dblMaxAcc[4]
, dblMaxAcc[5]
, dblMaxAcc[6]); //以七轴机械臂为例
```

157 getMechanicalMaxCartVelAcc

int getMechanicalMaxCartVelAcc (double &dblMaxCartTranslationVel, double &dblMaxCartRotationVel, double &dblMaxCartTranslationAcc, double &dblMaxCartRotationAcc, const char *strIpAddress = " ")

获取指定 IP 地址笛卡尔空间最大机械平移速度、最大机械旋转速度、最大机械平移加速度和最大机械旋转加速度。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

dblMaxCartTranslationVel：输出参数，用于存放笛卡尔空间最大机械平移速度，double 型变量，单位：m/s。

dblMaxCartRotationVel：输出参数，用于存放笛卡尔空间最大机械旋转速度，double 型变量，单位：rad/s。

dblMaxCartTranslationAcc：输出参数，用于存放笛卡尔空间最大机械平移加速度，double 型变量，单位：m/s^2。

dblMaxCartRotationAcc：输出参数，用于存放笛卡尔空间最大机械旋转加速度，double 型变量，单位：rad/s^2。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

-1：失败。

0：成功。

调用示例：

```
const char* robot0 =  "192.168.10.75";
double dblLinearVel = 0, dblRotationVel = 0, dblLinearAcc = 0, dblRotationAcc = 0;
int ret = getMechanicalMaxCartVelAcc(dblLinearVel, dblRotationVel, dblLinearAcc,
dblRotationAcc, robot0);
printf("getMechanicalCartMaxVelAcc ret = %d
{dblLinearVel(%f),dblRotationVel(%f),dblLinearAcc(%f),dblRotationAcc(%f)}\n"
, ret
, dblLinearVel
, dblRotationVel
, dblLinearAcc
, dblRotationAcc);
```

158 getJointsPositionRange

int getJointsPositionRange (double *dblMinPos, double *dblMaxPos, const char *strIpAddress

= " ")
获取指定 IP 地址机械臂各关节的极限位置。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** dblMinPos：输出参数，用于存放关节角的最小软限位，大小为 JOINT_NUM 的数组，单位：rad。 dblMaxPos：输出参数，用于存放关节角的最大软限位，大小为 JOINT_NUM 的数组，单位：rad。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** -1：失败。 0：成功。

调用示例：
const char* robot0 = "192.168.10.75";
double dblMinPos[JOINT_NUM] = {0}, dblMaxPos[JOINT_NUM]={0};
int ret = getJointsPositionRange(dblMinPos, dblMaxPos, robot0);
printf("getJointsPositionRange ret = %d dblMinPos = {%f,%f,%f,%f,%f,%f,%f} dblMaxPos = {%f,%f,%f,%f,%f,%f,%f}\n"
, ret
, dblMinPos[0]
, dblMinPos[1]
, dblMinPos[2]
, dblMinPos[3]
, dblMinPos[4]
, dblMinPos[5]
, dblMinPos[6]
, dblMaxPos[0]
, dblMaxPos[1]
, dblMaxPos[2]
, dblMaxPos[3]
, dblMaxPos[4]
, dblMaxPos[5]
, dblMaxPos[6]); //以七轴机械臂为例

159 getMaxJointsVel

int getMaxJointsVel (double *dblVel, const char *strIpAddress = " ")
获取指定 IP 地址机械臂各关节最大速度。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** dblVel：输出参数，用于存放各关节的最大速度，大小为 JOINT_NUM 的数组，单位：rad/s。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

-1：失败。

0：成功。

调用示例：

```
const char* robot0 =  "192.168.10.75";
double dblMaxVel[JOINT_NUM]={0};
int ret = getMaxJointsVel(dblMaxVel, robot0);
printf("getMaxJointsVel ret = %d dblMaxVel = {%f,%f,%f,%f,%f,%f,%f}\n"
, ret
, dblMaxVel[0]
, dblMaxVel[1]
, dblMaxVel[2]
, dblMaxVel[3]
, dblMaxVel[4]
, dblMaxVel[5]
, dblMaxVel[6]); //以七轴机械臂为例
```

160 getMaxJointsAcc

```
int getMaxJointsAcc (double *dblAcc, const char *strIpAddress =  " ")
```

获取指定 IP 地址机械臂各关节最大加速度。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

dblAcc：输出参数，用于存放各关节的最大加速度，大小为 JOINT_NUM 的数组，单位：rad/s^2。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

-1：失败。

0：成功。

调用示例：

```
const char* robot0 =  "192.168.10.75";
double dblMaxAcc[JOINT_NUM]={0};
int ret = getMaxJointsAcc(dblMaxAcc, robot0);
printf("getMaxJointsAcc ret = %d dblMaxAcc = {%f,%f,%f,%f,%f,%f,%f}\n" //以七轴机械臂为例


, ret
, dblMaxAcc[0]
, dblMaxAcc[1]
, dblMaxAcc[2]
, dblMaxAcc[3]
```

```
, dblMaxAcc[4]
, dblMaxAcc[5]
, dblMaxAcc[6]); //以七轴机械臂为例
```

161 getMaxCartTranslationVel

int getMaxCartTranslationVel (double &dblMaxCartTranslationVel, const char *strIpAddress = " ")

获取指定 IP 地址机械臂笛卡尔空间最大平移速度。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

dblMaxCartTranslationVel：输出参数，用于存放笛卡尔空间最大平移速度，double 型变量，单位：m/s。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

-1：失败。

0：成功。

调用示例：

```
const char* robot0 =  "192.168.10.75";
double dblLinearVel = 0;
int ret = getMaxCartTranslationVel(dblLinearVel, robot0);
printf("getMaxCartTranslationVel ret = %d dblLinearVel=%f\n"
, ret
, dblLinearVel
);
```

162 getMaxCartRotationVel

int getMaxCartRotationVel (double &dblMaxCartRotationVel, const char *strIpAddress = " ")

获取指定 IP 地址机械臂笛卡尔空间最大机软旋转速度。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

dblMaxCartRotationVel：输出参数，用于存放笛卡尔空间最大机软旋转速度，double 型变量，单位：rad/s。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

-1：失败。

0：成功。

调用示例：

```
const char* robot0 =  "192.168.10.75";
double dblRotationVel = 0;
int ret = getMaxCartRotationVel(dblRotationVel, robot0);
printf("getMaxCartRotationVel ret = %d dblRotationVel=%f\n"
, ret
, dblRotationVel);
```

163 getMaxCartTranslationAcc

int getMaxCartTranslationAcc (double &dblMaxCartTranslationAcc, const char *strIpAddress = " ")

获取指定 IP 地址机械臂笛卡尔空间最大软平移加速度。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

dblMaxCartTranslationAcc：输出参数，用于存放笛卡尔空间最大软平移加速度，double 型变量，单位：m/s^2。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

-1：失败。

0：成功。

调用示例：

```
const char* robot0 =  "192.168.10.75";
double dblLinearAcc = 0;
int ret = getMaxCartTranslationAcc(dblLinearAcc, robot0);
printf("getMaxCartTranslationAcc ret = %d dblLinearAcc=%f\n"
, ret
, dblLinearAcc);
```

164 getMaxCartRotationAcc

int getMaxCartRotationAcc (double &dblMaxCartRotationAcc, const char *strIpAddress = " ")

在指定 IP 地址机械臂笛卡尔空间最大软旋转加速度。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

dblMaxCartRotationAcc：输出参数，用于存放笛卡尔空间最大软旋转加速度，double 型变量，单位：rad/s^2。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

-1：失败。

0：成功。

调用示例：

```
const char* robot0 =  "192.168.10.75";
double dblRotationAcc = 0;
int ret = getMaxCartRotationAcc(dblRotationAcc, robot0);
printf("getMaxCartRotationAcc ret = %d dblRotationAcc=%f\n"
, ret
, dblRotationAcc);
```

165 setJointsPositionRange

int setJointsPositionRange (double *dblMinPos, double *dblMaxPos, const char *strIpAddress = " ")

设置指定 IP 地址机械臂各关节极限位置。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

dblMinPos：输入参数，关节角的最小软限位，大小为 JOINT_NUM 的数组，单位：rad。

dblMaxPos：输入参数，关节角的最大软限位，大小为 JOINT_NUM 的数组，单位：rad。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

-1：失败。

0：成功。

调用示例：

```
const char* robot0 =  "192.168.10.75";
double dblMinPos[JOINT_NUM] = {0}, dblMaxPos[JOINT_NUM]={0};
int ret = getJointsPositionRange(dblMinPos, dblMaxPos, robot0);
for (int i = 0; i < JOINT_NUM; ++i) {
	dblMinPos[i]++;
	dblMaxPos[i]--;
}
ret = setJointsPositionRange(dblMinPos,dblMaxPos,robot0);
printf("setJointsPositionRange return %d\n",ret);
ret = saveEnvironment(robot0);
printf("saveEnvironment return %d\n",ret);
```

166 setMaxJointsVel

int setMaxJointsVel (double *dblVel, const char *strIpAddress = " ")

设置指定 IP 地址机械臂各关节的最大速度。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

dblVel：输入参数，各关节的最大软速度，大小为 JOINT_NUM 的数组，单位：rad/s。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

-1：失败。

0：成功。

调用示例：

```
const char* robot0 =  "192.168.10.75";
double dblMaxVel[JOINT_NUM]={0};
int ret = getMaxJointsVel(dblMaxVel, robot0);
for (int i = 0; i < JOINT_NUM ; ++i) {
      dblMaxVel[i]--;
}
ret = setMaxJointsVel(dblMaxVel, robot0);
printf("setJointsVel return %d\n",ret);
ret = saveEnvironment(robot0);
printf("saveEnvironment return %d\n",ret);
```

167 setMaxJointsAcc

int setMaxJointsAcc (double *dblAcc, const char *strIpAddress = " ")

设置指定 IP 地址机械臂各关节的最大加速度。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

dblAcc：输入参数，用于存放各关节的最大软加速度，大小为 JOINT_NUM 的数组，单位：rad/s^2。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

-1：失败。

0：成功。

调用示例：

```
const char* robot0 =  "192.168.10.75";
double dblMaxAcc[JOINT_NUM]={0};
int ret = getMaxJointsAcc(dblMaxAcc, robot0);
for (int i = 0; i < JOINT_NUM; ++i) {
      dblMaxAcc[i]--;
}
ret = setMaxJointsAcc(dblMaxAcc, robot0);
ret = saveEnvironment(robot0);
printf("saveEnvironment return %d\n",ret);
```

168 setMaxCartTranslationVel

int setMaxCartTranslationVel (double dblMaxCartTranslationVel, const char *strIpAddress = "

")
设置指定 IP 地址机械臂笛卡尔空间最大平移速度。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** dblMaxCartTranslationVel：输入参数，用于存放笛卡尔空间最大软平移速度，double 型变量，单位：m/s。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** -1：失败。 0：成功。

调用示例：
const char* robot0 = "192.168.10.75";
double dblLinearVel = 0.5;
int ret = setMaxCartTranslationVel(dblLinearVel, robot0);
printf("setMaxCartTranslationVel return = %d\n",ret);
ret = saveEnvironment(robot0);
printf("saveEnvironment return %d\n",ret);

169 setMaxCartRotationVel

int setMaxCartRotationVel (double dblMaxCartRotationVel,const char *strIpAddress = " ")
设置指定 IP 地址机械臂笛卡尔空间最大旋转速度。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** dblMaxCartRotationVel：输入参数，卡尔空间最大旋转速度，double 型变量，单位：rad/s。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** -1：失败。 0：成功。

调用示例：
const char* robot0 = "192.168.10.75";
double dblRotationVel = 1;
int ret = setMaxCartRotationVel(dblRotationVel, robot0);
printf("setMaxCartRotationVel return = %d\n",ret);
ret = saveEnvironment(robot0);
printf("saveEnvironment return %d\n",ret);

170 setMaxCartTranslationAcc

int setMaxCartTranslationAcc (double dblMaxCartTranslationAcc, const char *strIpAddress = "")
设置指定 IP 地址机械臂笛卡尔空间最大平移加速度。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** dblMaxCartTranslationAcc：输入参数，笛卡尔空间最大平移加速度，double 型变量，单位：m/s^2。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** -1：失败。 0：成功。

调用示例：
const char* robot0 = "192.168.10.75";
double dblLinearAcc = 1;
int ret = setMaxCartTranslationAcc(dblLinearAcc, robot0);
printf("setMaxCartTranslationAcc return = %d\n",ret);
ret = saveEnvironment(robot0);
printf("saveEnvironment return %d\n",ret);

171 setMaxCartRotationAcc

int setMaxCartRotationAcc (double dblMaxCartRotationAcc, const char *strIpAddress = " ")
设置指定 IP 地址机械臂笛卡尔空间最大旋转加速度。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** dblMaxCartRotationAcc：输入参数，笛卡尔空间最大旋转加速度，double 型变量，单位：rad/s^2。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** -1：失败。 0：成功。

调用示例：
const char* robot0 = "192.168.10.75";
double dblRotationAcc = 1;
int ret = setMaxCartRotationAcc(dblRotationAcc, robot0);
printf("setMaxCartRotationAcc return = %d\n",ret);
ret = saveEnvironment(robot0);
printf("saveEnvironment return %d\n",ret);

172 freeDriving_ex

int freeDriving_ex(bool enable, int intFrameType, double *dblForceDirection, const char *strIpAddress = " ")

实现控制指定 IP 地址的机械臂正常模式与定向零力拖动模式之间的切换。定向零力拖动是指可选择笛卡尔空间可拖动方向的轴空间零力拖动。

适用臂型：通用医疗臂、定制医疗臂

参数：

enable：bool 变量，是否进入定向零力拖动模式，true 表明进入定向零力拖动，false 为退出定向零力拖动进入正常模式。只有在正常模式下，才可以控制机械臂运动。

intFrameType： int 型变量，用于指定在哪个坐标系下选择可拖动方向。0：不限制拖动方向，等同于关节空间虚拟墙；1：在基坐标系下描述可拖动方向；2：在工具坐标系下描述可拖动方向；3：在工件坐标系下描述可拖动方向。

dblForceDirection：double 型数组变量。表达力的方向（三个平移维度和三个旋转维度）的数组首地址，数组长度为 6。非“0”代表对应维度可拖动，“0”代表对应维度不可拖动。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
double direct[6]={0,0,1,0,0,0};
printf("任意键开始测试[自由拖动 ex]……\n");
char ch = getchar();
ret = freeDriving_ex(true, 1, direct, strIpAddress);
if (ret < 0)
{
   printf("进入自由拖动 ex 失败!Return value = %d\n", ret);
}
else
{
   printf("进入自由拖动 ex 成功！ \n");
}

printf("任意键退出测试[自由拖动 ex]……\n");
ch = getchar();
ret = freeDriving_ex(false, 1, direct, strIpAddress);
if (ret < 0)
{
   printf("退出自由拖动 ex 失败!Return value = %d\n", ret);
}
else
```

```
{
   printf("退出自由拖动 ex 成功！\n");
}
```

173 freeDrivingWithLockedJoint

int freeDrivingWithLockedJoint(bool enable, int intLockedJntIndex = -1, const char *strIpAddress = " ")

实现控制指定 IP 地址的机械臂正常模式与锁轴零力拖动模式之间的切换。

适用臂型：通用医疗臂、定制医疗臂

参数：

enable：bool 变量，是否进入锁轴零力拖动模式，true 表明进入锁轴零力拖动，false 为退出锁轴零力拖动进入正常模式。只有在正常模式下，才可以控制机械臂运动。

intLockedJntIndex：int 型变量，锁定轴的索引值，-1 代表不锁轴，0 代表 1 轴。目前仅支持锁定 3 轴，即该值设为 2，其他值视为不锁轴。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
int ret = freeDrivingWithLockedJoint(true, 2, strIpAddress);
if (ret < 0)
{
   printf("（2）带锁轴自由拖动失败!Return value = %d\n", ret);
}
else
{
   printf("（2）带锁轴自由拖动成功！\n");
}
printf("敲任意键退出测试[（2）带锁轴自由拖动]……\n");
char ch = getchar();
ret = freeDrivingWithLockedJoint(false, 2, strIpAddress);
if (ret < 0)
{
   printf("退出（2）带锁轴自由拖动失败!Return value = %d\n", ret);
}
else
{
   printf("退出（2）带锁轴自由拖动成功！\n");
}
```

174 freeDrivingWithLockedJoint_ex

int freeDrivingWithLockedJoint_ex(bool enable, int intFrameType, double *dblForceDirection,

int intLockedJntIndex = -1, const char *strIpAddress = " ")

实现控制指定 IP 地址的机械臂正常模式与锁轴定向零力拖动模式之间的切换。定向零力拖动是指可选择笛卡尔空间可拖动方向的轴空间零力拖动。

适用臂型：通用医疗臂、定制医疗臂

参数：

enable：bool 变量，是否进入锁轴定向零力拖动模式，true 表明进入锁轴定向零力拖动，false 为退出锁轴定向零力拖动进入正常模式。只有在正常模式下，才可以控制机械臂运动。

intFrameType： int 型变量，用于指定在哪个坐标系下选择可拖动方向。0：不限制拖动方向，等同于关节空间虚拟墙；1：在基坐标系下描述可拖动方向；2：在工具坐标系下描述可拖动方向；3：在工件坐标系下描述可拖动方向。

dblForceDirection：double 型数组变量。表达力的方向（三个平移维度和三个旋转维度）的数组首地址，数组长度为 6。非“0”代表对应维度可拖动，“0”代表对应维度不可拖动。

intLockedJntIndex：int 型变量，锁定轴的索引值，-1 代表不锁轴，0 代表 1 轴。目前仅支持锁定 3 轴，即该值设为 2，其他值视为不锁轴。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
double direct[6]={0,0,1,0,0,0};
int ret = freeDrivingWithLockedJoint_ex(true, 1, direct, 2, strIpAddress);
if (ret < 0)
{
   printf("（2）带锁轴定向自由拖动失败!Return value = %d\n", ret);
}
else
{
   printf("（2）带锁轴定向自由拖动成功！\n");
}
printf("敲任意键退出测试[（2）带锁轴定向自由拖动]……\n");
char ch = getchar();
ret = freeDrivingWithLockedJoint_ex(false, 1, direct, 2, strIpAddress);
if (ret < 0)
{
   printf("退出（2）带锁轴定向自由拖动失败!Return value = %d\n", ret);
}
else
{
   printf("退出（2）带锁轴定向自由拖动成功！\n");
```

```
}
```

175 freeDrivingForCurrentLoop

int freeDrivingForCurrentLoop (bool enable, const char *strIpAddress = " ")

使指定 IP 地址机械臂开启或关闭电流环自由驱动。开启电流环自由驱动的前提是系统已进入安全处理模式。安全处理模式是非正常工作模式，通常用于处理机械臂的异常情况，如关节超极限位置、机械臂发生严重碰撞或发生 JCU 过流等硬件故障。开启电流环自由驱动后，抱闸自动打开，用户可以手动将机械臂关节拖回预期的工作区间。关闭电流环自由驱动后，抱闸自动关闭。

适用臂型：通用医疗臂、定制医疗臂、Thor3

注：该 API 需要结合 enterSafetyIdle 和 leaveSafetyIdle 进行使用。只有进入安全处理模式以后，才能成功开启电流环自由驱动。当关闭电流环自由驱动后，需要退出安全处理模式。典型的调用过程如下：

1、调用 enterSafetyIdle 进入安全处理模式；

2、调用 freeDrivingForCurrentLoop(true)开启电流环自由驱动，将机械臂关节拖回正常工作区间；

3、调用 freeDrivingForCurrentLoop(false)关闭电流环自由驱动；

4、调用 leaveSafetyIdle 退出安全处理模式。

参数：

enable：输入参数，开启或关闭电流环自由驱动，bool 型变量。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

-1：失败。

0：成功

调用示例：

```
const char* strIpAddress=  "192.168.10.75";
int ret = enterSafetyIdle(strIpAddress);
printf("enterSafetyIdle return value = %d\n", ret);
ret = freeDrivingForCurrentLoop (true, strIpAddress);
printf("enter current loop free driving return = %d\n",ret);
M_SLEEP(15); // 该时间段（此处举例 15 秒）用于将机械臂超限关节驱回正常工作区间
ret = freeDrivingForCurrentLoop (false, strIpAddress);
printf("leave current loop free driving return = %d\n",ret);
ret = leaveSafetyIdle(strIpAddress);
printf("leaveSafetyIdle return value = %d\n", ret);
```

176 getCurrentLoopFreeDrivingState

int getCurrentLoopFreeDrivingState (const char *strIpAddress = " ")
查询指定 IP 地址机械臂电流环自由驱动的状态。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** -1：网络调用失败 0：退出电流环自由驱动 1：处在电流环自由驱动

调用示例：
const char* strIpAddress = "192.168.10.75";
freeDrivingForCurrentLoop (true, strIpAddress);
int ret = getCurrentLoopFreeDrivingState(strIpAddress);
Sleep(5);
printf("get current loop free driving state return = %d\n",ret);
freeDrivingForCurrentLoop (false, strIpAddress);
ret = getCurrentLoopFreeDrivingState(strIpAddress);
printf("get current loop free driving state return = %d\n",ret);

177 setJointLockedInCartImpedanceMode

int setJointLockedInCartImpedanceMode(const bool bLock, const int intLockedJointIndex, const char *strIpAddress = "")
设置指定 IP 地址机械臂在笛卡尔阻抗和/或力控模式下锁定/解锁某轴（当前版本仅支持 3 轴）。该设置将在机械臂进入笛卡尔阻抗或者力控模式时正式生效。 适用臂型：通用医疗臂、定制医疗臂 **参数：** bLock：输入参数，如果该值为 true，表示机械臂在笛卡尔阻抗和/或力控模式下将锁定某轴（当前版本仅支持锁定 3 轴，即 intLockedJointIdex 必须为 2，否则锁定无效）；如果该值为 false，则不论 intLockedJointIdex 取值多少，均表示机械臂在笛卡尔阻抗和/或力控模式下解锁某轴（当前版本仅支持解锁 3 轴）。 intLockedJointIdex：输入可选参数，表示轴的索引值（索引从 0 开始），缺省值为 2（即 3 轴）。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** -1：失败。

0：成功。

调用示例：
// 设置锁定 3 轴
const char* robot0 = "192.168.10.75";
const bool bLock = true;
int intLockedJointIndex = 2;
int ret = setJointLockedInCartImpedanceMode(bLock, intLockedJointIndex, robot0);
printf("setJointLockedInCartImpedanceMode return = %d\n",ret);
 	
// 设置解锁 3 轴
const char* robot0 = "192.168.10.75";
const bool bLock = false;
int intLockedJointIndex = -1;
int ret = setJointLockedInCartImpedanceMode(bLock, intLockedJointIndex, robot0);
printf("setJointLockedInCartImpedanceMode return = %d\n",ret);

178 getJointLockedInCartImpedanceMode

int getJointLockedInCartImpedanceMode(bool &isLocked, const char *strIpAddress = "")
查询指定 IP 地址机械臂在笛卡尔阻抗和/或力控模式下某轴（当前版本仅支持 3 轴）是否属于锁定/解锁状态。 适用臂型：通用医疗臂、定制医疗臂 **参数：** isLocked：输出参数，如果该值为 true，表示机械臂在笛卡尔阻抗和/或力控模式下某轴（当前版本仅支持 3 轴）处于锁定状态；如果该值为 false，则表示机械臂在笛卡尔阻抗和/或力控模式下某轴（当前版本仅支持 3 轴）处于解锁状态。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** -1：失败。 0：成功。

调用示例：
const char* robot0 = "192.168.10.75";
bool bIsLocked = false;
int ret = getJointLockedInCartImpedanceMode(bIsLocked, robot0);
printf("getJointLockedInCartImpedanceMode return = %d\n",ret);

179 calcRobotArmAngle

int calcRobotArmAngle(double* dblJoints,double& dblAngle, const char *strIpAddress = "")
对指定 IP 地址机械臂，计算传入关节角对应的臂角。 适用臂型：定制医疗臂

参数：

dblJoints：输入参数，机械臂各关节的关节角数组，弧度。

dblAngle：输出参数，机械臂计算出的臂角，角度

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

-1：失败。

0：成功。

调用示例：

```
const char* robot =  "192.168.10.75";
double dblJoints[7] = {0,90 * PI /180,90 * PI / 180,0,0,0,0}; //以七轴机械臂为例

double dblAngle = 0;
int ret = calcRobotArmAngle(dblJoints, dblAngle ,robot);
printf("calcRobotArmAngle get angle = %lf\n", dblAngle);
```

180 enterSafetyIdle

```
int enterSafetyIdle(const char *strIpAddress =  "")
```

对指定 IP 地址机械臂，进入安全处理模式。安全处理模式是非正常工作模式，通常用于处理机械臂的异常情况，如关节超极限位置、机械臂发生严重碰撞或发生 JCU 过流等硬件故障。进入安全处理模式以后，抱闸会自动关闭，此时可以通过开启电流环自由驱动来将机械臂的关节拖回预期的工作区间。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

-1：失败。

0：成功。

调用示例：

详见 freeDrivingForCurrentLoop 的调用示例。

181 leaveSafetyIdle

```
int leaveSafetyIdle(const char *strIpAddress =  "")
```

对指定 IP 地址机械臂，退出安全处理模式。安全处理模式是非正常工作模式，通常用于处理机械臂的异常情况，如关节超极限位置、机械臂发生严重碰撞或发生 JCU 过流等硬件故障。退出安全处理模式以后，抱闸仍处于关闭状态。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

-1：失败。

0：成功。

调用示例：

详见 freeDrivingForCurrentLoop 的调用示例。

182 identifyTcpPayload

int identifyTcpPayload(double *dblResult, FNCSTOP fnNeedStop = nullptr, LoadIdentificationParam *ptrParam = nullptr, const char *strIpAddress = "")

对指定 IP 地址的机械臂执行一系列移动指令进行负载辨识。该功能会执行一段时间，移动至每个路点后大概会停留 3 秒左右。请不要使用多线程同时调用其他移动指令或修改系统参数配置。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

dblResult：输出参数，用于接收负载信息，大小为 10 的数组，第 1 位为质量（单位：kg），2~4 位为质心（单位：m），5~10 位为张量（单位：$kg•m^2$）。该结果可以作为输入参数传给 setActiveTcpPayload 使用。目前该函数仅支持辨识质量和质心，即当返回结果有效时 dblResult 数组只有前四个有值。

fnNeedStop：可选参数，停止负载辨识功能的回调函数。函数声明形式：bool fnNeedStop (const char *strIpAddress)。当 fnNeedStop 函数返回 true 时，将中止正在执行的负载辨识程序。该参数缺省值为空，如果不传入该参数，用户开启负载辨识程序后将无法干预程序执行。fnNeedStop 函数将在负载辨识程序开始执行后，每 200 毫秒检测一次，尽量不要在函数实现中使用 sleep 函数之类会阻塞线程的操作。

ptrParam：可选参数，LoadIdentificationParam 结构体指针（LoadIdentificationParam 的定义参见 DianaAPI.h 头文件），用于个性化配置质量、负载辨识路点等信息(暂时仅支持 32 个路点作为负载辨识的路点)。缺省值为空。当该值为空时，使用系统默认的 32 个路点作为负载辨识路点。其中 dblMass 用于已知负载质量时使用。当 dblMass 有值时需要同时将 bIsMassValid 设置为 true，表示 dblMass 值有效；否则 dblMass 的值将被忽略。

arrWaypoints 为 LoadIdentificationWaypoint 结构体指针（LoadIdentificationWaypoint 的定义参见 DianaAPI.h 头文件），用于自定义负载辨识的路点信息，每个路点主要包含各关节角度（单位：rad）和速度百分比（如 30 表示 30%，建议速度百分比不超过 30%）。arrWaypoints 数组的长度通过 arrWaypointsSize 传入。在负载辨识过程中将以传入路点的

第一组值作为初始点，机械臂先移动至初始点，然后从第二个路点开始依次移动至最后一个路点，做为第一轮正向辨识。第一轮结束后，机械臂回到初始点，再进行第二轮反向识别，即回到最后一个点，逆向依次移动至第二个路点后结束整个辨识过程。因此如果路点数过少可能会导致辨识失败、结果不准确等问题。由于整个辨识过程相对独立，部分失败原因会以 identifyTcpPayload 函数返回值的方式通知用户。请配合 getLastError 函数一起判断辨识结果的有效性。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：其他错误。

-2：无法读取指定 IP 地址的机械臂的关节扭矩。

-3：初始化失败。

-4：采样失败。

-5：传送辨识数据失败。

-6：结果不可信。

-7：负载辨识失败。

-8：回调函数中断负载辨识程序执行。当用户传入的 fnNeedStop 函数返回 true 时触发此错误。可以不作处理。

-9：指定 IP 地址的机械臂已经在执行负载辨识任务。

-10：开始执行负载辨识任务失败。

-11：结束负载辨识任务失败。

-12：指定 IP 地址的机械臂正在运行示教程序，无法执行负载辨识任务。

-17：未打开抱闸。

调用示例：

```
//以七轴机械臂为例
bool isStopLoadIdentification(const char *strIpAddress){
   static int cnt = 30;
   printf("isStopLoadIdentification[%d](%s)\n",cnt, strIpAddress);
   if(cnt-- > 0){
      return false;
   } else {
      return true;
   }
}
……
const char* robot =  "192.168.10.75";
double waypoints[32][7]={{0.0, 0.0, 0.0, 1.57, -1.047, -1.57, -1.57}
                                    ,{0.0, 0.0, 0.0, 1.57, -0.979, -1.503, -1.469}
                                    ,{0.0, 0.0, 0.0, 1.57, -0.912, -1.435, -1.368}
```

```
                                ,{0.0, 0.0, 0.0, 1.57, -0.844, -1.368, -1.266}
                                ,{0.0, 0.0, 0.0, 1.57, -0.776, -1.300, -1.165}
                                ,{0.0, 0.0, 0.0, 1.57, -0.709, -1.232, -1.064}
                                ,{0.0, 0.0, 0.0, 1.57, -0.641, -1.165, -0.962}
                                ,{0.0, 0.0, 0.0, 1.57, -0.574, -1.097, -0.861}
                                ,{0.0, 0.0, 0.0, 1.57, -0.506, -1.030, -0.760}
                                ,{0.0, 0.0, 0.0, 1.57, -0.439, -0.962, -0.658}
                                ,{0.0, 0.0, 0.0, 1.57, -0.371, -0.895, -0.557}
                                ,{0.0, 0.0, 0.0, 1.57, -0.304, -0.827, -0.456}
                                ,{0.0, 0.0, 0.0, 1.57, -0.236, -0.760, -0.354}
                                ,{0.0, 0.0, 0.0, 1.57, -0.168, -0.692, -0.253}
                                ,{0.0, 0.0, 0.0, 1.57, -0.101, -0.624, -0.152}
                                ,{0.0, 0.0, 0.0, 1.57, -0.033, -0.557, -0.050}
                                ,{0.0, 0.0, 0.0, 1.57, 0.033, -0.489, 0.050}
                                ,{0.0, 0.0, 0.0, 1.57, 0.101, -0.422, 0.152}
                                ,{0.0, 0.0, 0.0, 1.57, 0.168, -0.354, 0.253}
                                ,{0.0, 0.0, 0.0, 1.57, 0.236, -0.287, 0.354}
                                ,{0.0, 0.0, 0.0, 1.57, 0.304, -0.219, 0.456}
                                ,{0.0, 0.0, 0.0, 1.57, 0.371, -0.152, 0.557}
                                ,{0.0, 0.0, 0.0, 1.57, 0.439, -0.084, 0.658}
                                ,{0.0, 0.0, 0.0, 1.57, 0.506, -0.016, 0.760}
                                ,{0.0, 0.0, 0.0, 1.57, 0.574, 0.050, 0.861}
                                ,{0.0, 0.0, 0.0, 1.57, 0.641, 0.118, 0.962}
                                ,{0.0, 0.0, 0.0, 1.57, 0.709, 0.185, 1.064}
                                ,{0.0, 0.0, 0.0, 1.57, 0.776, 0.253, 1.165}
                                ,{0.0, 0.0, 0.0, 1.57, 0.844, 0.320, 1.266}
                                ,{0.0, 0.0, 0.0, 1.57, 0.912, 0.388, 1.368}
                                ,{0.0, 0.0, 0.0, 1.57, 0.979, 0.456, 1.469}
                                ,{0.0, 0.0, 0.0, 1.57, 1.047, 0.523, 1.57}};
   LoadIdentificationWaypoint wps[32];
   for(int i = 0; i < 32; ++i) {
      memcpy(wps[i].joints, waypoints[i], sizeof(double) * 7);
      wps[i].vel = 30.0;
   }
   double result[10] = {0};
   LoadIdentificationParam param;
   param.bIsMassValid = true;
   param.dblMass = 1.5;
   param.arrWaypoints = wps;
   param.arrWaypointsSize = 32;
   ret = identifyTcpPayload(result, isStopLoadIdentification, &param, robot);
   if (ret < 0)
   {
      printf("identifyTcpPayload 失败! Return value = %d result =
{%f,%f,%f,%f,%f,%f,%f,%f,%f,%f} error=%d\n"
            , ret, result[0], result[1], result[2], result[3], result[4], result[5], result[6], result[7]
            , result[8], result[9], getLastError());
   }
   else
   {
      printf("identifyTcpPayload 成功！  {%f,%f,%f,%f,%f,%f,%f,%f,%f,%f} error=%d \n"
         , result[0], result[1], result[2], result[3], result[4], result[5], result[6], result[7]
         , result[8], result[9] , getLastError());
   }
```

183 setJointImpeda

int setJointImpeda(double *arrStiff, double dblDampRatio,const char *strIpAddress = " ")
设置指定 IP 地址机械臂各关节阻抗参数，包含刚度 Stiffiness 和阻尼比 DampRatio 的数据。 适用臂型：通用医疗臂、定制医疗臂 **参数：** arrStiff：表示各关节刚度 Stiffiness 的数组的首地址，数组长度为 JOINT_NUM。 dblDampRatio：表示各关节公用的阻尼比 DampRatio，类型为 double。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。

调用示例：
double arrStiff[JOINT_NUM] = { 3000, 3000, 3000, 1000, 500, 1000, 1000};//以七轴机械臂为例
double dblDampRatio = 0.7;
const char* strIpAddress = "192.168.10.75";
if (setJointImpeda(arrStiff, dblDampRatio, strIpAddress) < 0)
{
printf("Diana API setJointImpeda failed!\n");
}

184 getJointImpeda

int getJointImpeda (double *arrStiff, double *dblDampRatio,const char *strIpAddress = " ")
获取指定 IP 地址机械臂各关节阻抗参数，包含刚度 Stiffiness 和阻尼比 DampRatio 的数据。 适用臂型：通用医疗臂、定制医疗臂 **参数：** arrStiff：表示各关节刚度 Stiffiness 的数组的首地址，数组长度为 JOINT_NUM，用于接收获取到的值。 dblDampRatio：表示各关节共用的阻尼比 dblDampRatio 的地址，类型为 double*，用于接收获取到的值。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂

时生效。 **返回值：** 0：成功。 -1：失败。

调用示例：
double arrStiff [JOINT_NUM] = { 0};
double dblDampRatio = 0;
const char* strIpAddress = "192.168.10.75";
if (getJointImpeda(arrStiff, &dblDampRatio, strIpAddress) < 0)
{
printf("Diana API getJointImpeda failed!\n");
}

185 setCartImpeda

int setCartImpeda (double *arrStiff, double dblDampRatio, const char *strIpAddress = " ")
设置指定 IP 地址机械臂笛卡尔空间阻抗参数。 适用臂型：通用医疗臂、定制医疗臂 **参数：** arrStiff：表示笛卡尔空间，各维度刚度 Stiffiness 的数组的首地址，数组长度为 6。 dblDampRatio：表示笛卡尔空间，各维度共用的阻尼比 DampRatio。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。

调用示例：
double arrStiff[6] = { 1000, 1000, 1000, 500, 500, 500};
double dblDampRatio = 0.7;
const char* strIpAddress = "192.168.10.75";
if (setCartImpeda(arrStiff, dblDampRatio, strIpAddress) < 0)
{
printf("Diana API setCartImpeda failed!\n");
}

186 getCartImpeda

int getCartImpeda (double *arrStiff, double *dblDampRatio, const char *strIpAddress = " ")

获取指定 IP 地址机械臂笛卡尔空间各维度阻抗参数，包含刚度 Stiffiness 和阻尼比 DampRatio 的数据。

适用臂型：通用医疗臂、定制医疗臂

参数：

arrStiff：表示笛卡尔空间，各维度刚度 Stiffiness 的数组的首地址，数组长度为 6，用于接收获取到的值。

dblDampRatio：表示笛卡尔空间，各维度共用的阻尼比 DampRatio，double*类型，用于接收获取到的值。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
double arrStiff [6] = {0};
double dblDampRatio= {0};
const char* strIpAddress =  "192.168.10.75";
if (getCartImpeda(arrStiff, &dblDampRatio, strIpAddress) < 0)
{
   printf("Diana API getCartImpeda failed!\n");
}
```

187 getRobotNetworkInfo

```
int getRobotNetworkInfo(char *strNetworkInfo, const char *strIpAddress =  " ")
```

获取指定 IP 地址机械臂上的网络配置信息，包含网卡名、ip 地址和子网掩码的数据。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

strNetworkInfo：表示远端机器网络信息的字符串，不同网卡的信息以分号(;)进行分割，网卡名与网络信息以冒号(:)进行分割，IP 地址与子网掩码以正斜杠(/)作为分割。例如:”enp2s0:192.168.10.75/24;enp3s0:192.168.1.75/24”

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
char strNetworkInfo[256] ={'\0'};
const char* strIpAddress =  "192.168.10.75";
if (getRobotNetworkInfo(strNetworkInfo,  strIpAddress) < 0)
{
   printf("Diana API getRobotNetworkInfo failed!\n");
}
else
{
  printf("%s\n",strNetworkInfo);
}
```

188 setRobotNetworkInfo

```
int setRobotNetworkInfo(const char *strCurrentNetworkInfo, const char *strTargetNetworkInfo,
const char *strIpAddress =  " ")
```

设置指定 IP 地址机械臂上的网络配置信息，目前支持设置 IP 地址和子网掩码。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

strCurrentNetworkInfo：表示远端机器期望修改的网络信息，现支持修改 IP 地址和子网掩码。如果需要修改多个 IP 地址，需要用分号(;)进行分割，比如”192.168.10.75/24;192.168.1.75/24”。如果不需要修改子网掩码，则可以不写，比如上述例子也可以直接写成”192.168.10.75;192.168.1.75”，但是如果携带子网掩码，则必须确保子网掩码为当前 IP 所使用。

strTargetNetworkInfo：表示远端机器中期望变成的网络信息，现支持修改 IP 地址和子网掩码，如果需要修改多个 IP 地址，需要用分号(;)进行分割，比如”192.168.1.75/24;192.168.10.75/24”。如果不需要修改子网掩码，可以不写。注意，需要确保 strTargetNetworkInfo 中的 IP 信息的数量与 strCurrentNetworkInfo 中一致。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例 1（单 IP 修改）：

```
const char* strCurrentNetworkInfo = "192.168.10.75";
const char* strTargetNetworkInfo = "192.168.10.88";
const char* strIpAddress =  "192.168.10.75";
```

```
if (setRobotNetworkInfo (strCurrentNetworkInfo,strTargetNetworkInfo, strIpAddress) < 0)
{
   printf("Diana API setRobotNetworkInfo failed!\n");
}
```

调用示例 2（多 IP 修改）：

```
const char* strCurrentNetworkInfo = "192.168.10.75/24;192.168.1.75/24";
const char* strTargetNetworkInfo = "192.168.1.75;192.168.10.75";
const char* strIpAddress =  "192.168.10.75";
if (setRobotNetworkInfo (strCurrentNetworkInfo,strTargetNetworkInfo, strIpAddress) < 0)
{
   printf("Diana API setRobotNetworkInfo failed!\n");
}
```

189 getRgbLedState

```
int getRgbLedState(led_color *rgbLedColor, const char *strIpAddress = ‘’)
```

获取指定 IP 地址机械臂各轴灯环的颜色。

适用臂型：定制医疗臂

注意：该 API 仅适用于定制医疗臂 2.0 及以上关节协议。

参数：

rgbLedColor：表示各轴灯环颜色的数组，长度为 JOINT_NUM，类型为结构体 led_color，如下:

```
typedef struct color_rgb {
public:
   color_rgb(uint8_t r = 0, uint8_t g = 0, uint8_t b = 0) : red(r), green(g), blue(b) {}
public:
   uint8_t red;
   uint8_t green;
   uint8_t blue;
} led_color;
```

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
//以七轴机械臂为例
```

```
led_color rgb_color[7] =
{{0,0,0},{0,0,10},{0,10,0},{10,0,0},{0,10,10},{10,0,10},{10,10,10}};
const char* strIpAddress =  "192.168.10.75";
if (setRgbLedState(rgb_color, strIpAddress) < 0)
{
   printf("Diana API setRgbLedState failed!\n");
}
else
{
      if (getRgbLedState(rgb_color, strIpAddress) == 0)
      {
            for(int i = 0; i < JOINT_NUM; ++i)
            {
                  printf("joint %d got color rgb(%d,%d,%d)\n",i+1, rgb_color[i].red,
rgb_color[i].green, rgb_color[i].blue);
            }
      }
      else
      {
            printf("Diana API setRgbLedState failed!\n");
      }
}
```

190 setRgbLedState

int setRgbLedState(led_color *rgbLedColor, const char *strIpAddress = ‘’)

设置指定 IP 地址机械臂各轴灯环的颜色。

适用臂型：定制医疗臂

注意：该 API 仅适用于定制医疗臂 2.0 及以上关节协议。

参数：

rgbLedColor：表示各轴灯环颜色的数组，长度为 JOINT_NUM，类型为结构体 led_color。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

见 getRgbLedState

191 getFunctionOptI4

int getFunctionOptI4 (int intFunctionIndex, int intOptName, int *intOptVal, strIpAddress = " ")

获取指定功能的可选整型参数。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

intFunctionIndex：指定功能索引，取值为 function_index_e 枚举类型，枚举值及含义如下表所示：

intFunctionIndex	对应枚举	含义
0	function_index_e::T_FREEDRIVING	自由驱动相关
1	function_index_e::T_CART_IMPEDANCE	力控和笛卡尔阻抗相关

intOptName：功能的可选参数，

当 intFunctionIndex 为 0 时，表示自由驱动相关参数，对应含义如下：

intOptName	对应枚举值	含义
0x10005	freedriving_opt_name_e::T_ENABLE_REALTIME_VIRTUAL_WALL	获取实时虚拟墙功能是否开启的状态
0x10008	freedriving_opt_name_e::T_CONTROL_RULE	获取自由驱动的控制方式

intOptVal：用于存放参数的值，随 intOptName 的值变化：

intOptName	对应 OptName 枚举值	intOptValue 值及其含义
0x10005	freedriving_opt_name_e::T_ENABLE_REALTIME_VIRTUAL_WALL	0 表示失败； 1 表示成功。
0x10008	freedriving_opt_name_e::T_CONTROL_RULE	0 表示基于阻抗的自由驱动； 1 表示基于导纳的自由驱动。

当 intFunctionIndex 为 1 时，表示力控和笛卡尔阻抗相关参数，对应含义如下：

intOptName	对应枚举	含义
0x10100	cart_impedance_opt_name_e:: T_COORDINATE_TYPE	获取力控和笛卡尔阻抗的坐标系类型。
0x10101	cart_impedance_opt_name_e::T_IS_SINGLE_AXIS_LOCKED	获取力控和笛卡尔阻抗是否锁轴
0x10103	cart_impedance_opt_name_e::T_CONSTRAIN_TYPE	获取力控和笛卡尔阻抗的零空间约束条件

intOptVal：用于存放参数的值，随 intOptName 的值变化：

intOptName	对应枚举	intOptValue 值及其含义
0x10100	cart_impedance_opt_name_e:: T_COORDINATE_TYPE	0 表示基坐标系 1 表示工具坐标系 2 表示工件坐标系
0x10101	cart_impedance_opt_name_e::T_IS_SINGLE_AXIS_LOCKED	1 表示锁 3 轴
0x10103	cart_impedance_opt_name_e::T_CONSTRAIN_TYPE	0 表示零空间无约束；1 表示以 3 轴趋向正负 10 度作为零空间的约束条件。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

-1：失败。

0：成功。

调用示例：

```
// 如下以力控和笛卡尔阻抗时施加零空间以 3 轴趋向正负 10 度的约束为例
int ret = 0;
ret = setFunctionOptI4(1, 0x10103, 1);
if (ret == 0) printf("力控和笛卡尔阻抗时施加零空间以 3 轴趋向正负 10 度的约束成功！\n");

int intOptValue = 0;
ret = getFunctionOptI4(1, 0x10103, &intOptValue);
if (ret == 0) {
   if (intOptValue == 1)
      printf("力控和笛卡尔阻抗时已施加零空间以 3 轴趋向正负 10 度的约束！\n");
   else
      printf("力控和笛卡尔阻抗时未施加零空间以 3 轴趋向正负 10 度的约束！\n");
}
```

192 setFunctionOptI4

int setFunctionOptI4 (int intFunctionIndex, int intOptName, int intOptVal, const char * strIpAddress = " ")

设置指定功能的可选整型参数。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

intFunctionIndex：指定功能索引,如下表所示。

intFunctionIndex	对应枚举	含义
0	function_index_e::T_FREEDRIVING	自由驱动相关
1	function_index_e::T_CART_IMPEDANCE	力控和笛卡尔阻抗相关

intOptName：功能的可选参数，

当 intFunctionIndex 为 0 时，代表自由驱动相关参数，对应含义如下：

intOptName	对应枚举	含义
0x10005	freedriving_opt_name_e::T_ENABLE_REALTIME_VIRTUAL_WALL	使能实时虚拟墙功能
0x10008	freedriving_opt_name_e::T_CONTROL_RULE	设置自由驱动的控制方式

intOptVal：用于存放参数的值，随 intOptName 的值变化：

intOptName	对应 OptName 枚举值	intOptValue 值及其含义

0x10005	freedriving_opt_name_e::T_ENABLE_REALTIME_VIRTUAL_WALL	0 表示 disable； 1 表示 enable。
0x10008	freedriving_opt_name_e::T_CONTROL_RULE	0 表示基于阻抗的自由驱动； 1 表示基于导纳的自由驱动。

当 intFunctionIndex 为 1 时，代表力控和笛卡尔阻抗相关参数，对应含义如下：

intOptName	对应枚举	含义
0x10100	cart_impedance_opt_name_e:: T_COORDINATE_TYPE	设置力控和笛卡尔阻抗的坐标系类型。
0x10102	cart_impedance_opt_name_e::T_LOCKED_SINGLE_AXIS	设置力控和笛卡尔阻抗是否锁轴
0x10103	cart_impedance_opt_name_e::T_CONSTRAIN_TYPE	设置力控和笛卡尔阻抗的零空间约束条件

intOptVal：用于存放参数的值，随 intOptName 的值变化：

intOptName	对应 OptName 枚举值	intOptValue 值及其含义
0x10100	cart_impedance_opt_name_e:: T_COORDINATE_TYPE	0 表示基坐标系 1 表示工具坐标系 2 表示工件坐标系
0x10102	cart_impedance_opt_name_e::T_LOCKED_SINGLE_AXIS	1 表示锁 3 轴；
0x10103	cart_impedance_opt_name_e::T_CONSTRAIN_TYPE	0 表示零空间无约束； 1 表示以 3 轴趋向正负 10 度作为零空间的约束条件。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

-1：失败。

0：成功。

调用示例：

详见 getFunctionOptI4 调用示例。

193 setGravInfo

int setGravInfo(double *dblGravityAccInBase, const char *strIpAddress = " ")

针对指定 IP 地址的机械臂，设置其重力矢量。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

dblGravityAccInBase：输入参数，重力矢量，大小为 3 的数组。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。 -1：失败。

调用示例：
const char* ipAddress = "192.168.10.75";
double grav[3] = {0.0};
int ret = setGravInfo(grav,ipAddress);
if(ret < 0){
printf("setGravInfo failed! Return value = %d\n", ret);
}

194 getGravInfo

int getGravInfo(double *dblGravityAccInBase, const char *strIpAddress = " ")
针对指定 IP 地址的机械臂，获取其安装信息的重力矢量。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** dblGravityAccInBase：输出参数，重力矢量，大小为 3 的数组。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。

调用示例：
const char* ipAddress = "192.168.10.75";
double grav_axis[3] = {0.0};
int ret = getGravInfo (grav_axis,ipAddress);
if(ret < 0){
printf("getGravInfo failed! Return value = %d\n", ret);
}
else{
printf("getGravInfo :%f, %f, %f\n", grav_axis[0], grav_axis[1], grav_axis[2]);
}

195 getSixAxisInstallation

int getSixAxisInstallation(dblMass, dblCenterOfMass, dblPoseOfSixAxisInFalan, dblGravityAccInBase, const char *strIpAddress = " ")
针对指定 IP 地址的机械臂，获取系统中存储的六维力传感器和转接盘安装信息。安装信

息包括：质量、质心、六维力坐标系相对法兰的位姿和表达机械臂安装姿态的重力矢量。常用于配合 initSixAxisInstallation 函数一起使用。

适用臂型：通用医疗臂、Thor3

参数：

dblMass：输出参数，系统中存储的六维力传感器和转接盘的质量。单位：kg。

dblCenterOfMass：输出参数，系统中存储的六维力传感器和转接盘的质心向量的首地址，数组长度为 3。单位：m。

dblPoseOfSixAxisInFalan：输出参数，系统中存储的六维力坐标系相对法兰的位姿向量的首地址，数组长度为 16。

dblGravityAccInBase：输出参数，重力加速度的矢量在机械臂基座标系中的表达，数组个数为 3。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

详见 initSixAxisInstallation 的调用示例。

196 initSixAxisInstallation

int initSixAxisInstallation(double dblMass, double *dblCenterOfMass, double *dblPoseOfSixAxisInFalan, double *dblGravityAccInBase, const char *strIpAddress = " ")

针对指定 IP 地址的机械臂，初始化六维力传感器和转接盘的安装信息。安装信息包括：质量、质心、六维力坐标系相对法兰的位姿和表达机械臂安装姿态的重力矢量。

在使用跟六维力传感器相关功能时应首先调用此函数。为了方便用户使用可以通过 getSixAxisInstallation 获取系统中存储的安装信息，确认与 robot 上安装的传感器信息匹配后作为初始化参数传入。该参数传入系统后除了设置给六维力相关功能，还会保存用于下次 getSixAxisInstallation 调用。

适用臂型：通用医疗臂、Thor3

参数：

dblMass：输入参数，六维力传感器和转接盘的质量。单位：kg。

dblCenterOfMass：输入参数，六维力传感器和转接盘的质心向量的首地址，数组长度为 3。单位：m。

dblPoseOfSixAxisInFalan：输入参数，六维力坐标系相对法兰的位姿向量的首地址，数组

长度为 16。

dblGravityAccInBase：输入参数，重力加速度的矢量在机械臂基座标系中的表达，数组个数为 3。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
bool isStopIdentification(const char *strIpAddress){
   static int cnt = 50;
   printf("isStopLoadIdentification[%d](%s)\n",cnt, strIpAddress);
   if(cnt-- > 0){
      return false;
   } else {
      return true;
   }
}
************************************************************
   double start[7] = {0,-PI/6, 0, PI/2, 0, -PI*2/3, 0};//以七轴机械臂为例
   ret = moveJToTarget(start, 0.3,0.3);
   if(ret == 0) {
      wait_move();
      stop();
   }

   double dblMass = 0.0, dblCenter[3] = {0}, dblFalan[16] = {0}, dblGravity[3] = {0};
   ret=getSixAxisInstallation(&dblMass, dblCenter, dblFalan, dblGravity);
   if (ret < 0)
   {
      printf("getSixAxisInstallation failed! Return value = %d\n",  ret);
   } else {
      printf("getSixAxisInstallation success! Return value = %d, \n\t\tmass = %f, \n\t\tcenter=
{%f,%f,%f},
\n\t\tfalan={%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f}\n\t\tgravity={%f,%f,%f}\
n",
            ret,
```

```
            dblMass,
            dblCenter[0],
            dblCenter[1],
            dblCenter[2],
            dblFalan[0],
            dblFalan[1],
            dblFalan[2],
            dblFalan[3],
            dblFalan[4],
            dblFalan[5],
            dblFalan[6],
            dblFalan[7],
            dblFalan[8],
            dblFalan[9],
            dblFalan[10],
            dblFalan[11],
            dblFalan[12],
            dblFalan[13],
            dblFalan[14],
            dblFalan[15],
            dblGravity[0],
            dblGravity[1],
            dblGravity[2]);
    }
    dblMass = 0.185;
    dblCenter[0] = 0;
    dblCenter[1] = 0;
    dblCenter[2] = -0.018;
    //(0,0,-9.8)
    dblGravity[0] = 0;
    dblGravity[1] = 0;
    dblGravity[2] = -9.8;
    memset(dblFalan, 0x0, sizeof(double) * 16);
    dblFalan[0] = cos(PI/6);
    dblFalan[1] = sin(PI/6);
    dblFalan[4] = sin(PI/6);
    dblFalan[5] = -cos(PI/6);
    dblFalan[10] = -1.0;
```

```
    dblFalan[14] = -0.0315;
    dblFalan[15] = 1.0;
    ret=initSixAxisInstallation(dblMass, dblCenter, dblFalan, dblGravity);
    if (ret < 0) {
        printf("initSixAxisInstallation failed! Return value = %d\n", ret);
    } else {
        printf("initSixAxisInstallation success! Return value = %d, \n\t\tmass = %f, \n\t\tcenter=
{%f,%f,%f}, "

"\n\t\tfalan={%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f}\n\t\tgravity={%f,%f,%f}
\n",
                ret,
                dblMass,
                dblCenter[0],
                dblCenter[1],
                dblCenter[2],
                dblFalan[0],
                dblFalan[1],
                dblFalan[2],
                dblFalan[3],
                dblFalan[4],
                dblFalan[5],
                dblFalan[6],
                dblFalan[7],
                dblFalan[8],
                dblFalan[9],
                dblFalan[10],
                dblFalan[11],
                dblFalan[12],
                dblFalan[13],
                dblFalan[14],
                dblFalan[15],
                dblGravity[0],
                dblGravity[1],
                dblGravity[2]);
    }
    ret = saveEnvironment();
    if (ret < 0) {
```

```
        printf("saveEnvironment failed! Return value = %d \n", ret);
    } else {
        printf("saveEnvironment success! Return value = %d \n", ret);
    }
    double dblWPs[36] = {0}, dblVel[6] = {0}, dblAcc[6] = {0}, dblHome[JOINT_NUM] = {0},
dblHomeVel = 0.0, dblHomeAcc = 0.0;
        ret = getCalcSixAxisWaypoints(dblHome, dblHomeVel, dblHomeAcc, dblWPs, dblVel,
dblAcc);
    if(ret < 0) {
        printf("getCalcSixAxisWaypoints failed! Return value = %d  \n", ret);
    } else {
        printf("getCalcSixAxisWaypoints success! Return value = %d  \n", ret);
        for(int i = 0; i < JOINT_NUM; ++i) {
            printf("home[%d] = %f,", i,dblHome[i]);
        }
        printf("velHome = %f, accHome = %f\n", dblHomeVel, dblHomeAcc);
    for(int i = 0; i < 6; ++i) {
        for(int j = 0; j < 6; ++j) {
            printf("wp[%d][%d] = %f,", i,j, dblWPs[i*6 + j]);
        }
        printf("vel = %f, acc = %f\n", dblVel[i], dblAcc[i]);
    }
}
dblHome[0] = 0.0;
dblHome[1] = 0.0;
dblHome[2] = 0.0;
dblHome[3] = PI/2;
dblHome[4] = 0.0;
dblHome[5] = -PI/2;
dblHome[6] = 0.0;
dblHomeVel = 0.2;
dblHomeAcc = 0.4;
double payload[10] = {0};
ret = identifyTcpPayloadWithSixAxis(payload, dblHome, dblHomeVel, dblHomeAcc, dblWPs,
dblVel, dblAcc, 6, isStopIdentification);
if(ret < 0) {
        printf("identifyTcpPayloadWithSixAxis failed! Return value = %d  \n", ret);
} else {
```

```
        printf("identifyTcpPayloadWithSixAxis success! Return value = %d payload{%f,%f,%f,%f,%f,%f,%f,%f,%f,%f} \n",
                ret,
                payload[0],
                payload[1],
                payload[2],
                payload[3],
                payload[4],
                payload[5],
                payload[6],
                payload[7],
                payload[8],
                payload[9]);
    }
    printf("save the payload result\n");
    char ch = getchar();
    double force[6] = {0};
    for ( int i = 0; i < 100; ++i) {
        ret = getFixedSixAxisForce(force);
        if(ret < 0) {
            printf("getFixedSixAxisForce failed! Return value = %d  \n", ret);
        } else {
            printf("getFixedSixAxisForce success! Return value = %d [%d] force{%f,%f,%f,%f,%f,%f} \n",
                    ret,
                    i,
                    force[0],
                    force[1],
                    force[2],
                    force[3],
                    force[4],
                    force[5]);
        }
        M_SLEEP(10);
    }
```

197 getCalcSixAxisWaypoints

```
int getCalcSixAxisWaypoints(double *dblHome, double &dblHomeVel, double &dblHomeAcc,
```

double *dblWaypoints, double *dblVel, double *dblAcc, const char *strIpAddress = " ")
针对指定 IP 地址的机械臂，获取六维力传感器标定过程推荐的点位和 home 点。每次调用 initSixAxisInstallation 后，该点位信息会根据传入的参数重新计算，并通过本函数获取。如果没有调用 initSixAxisInstallation，直接调用本函数，将得到上次成功标定六维力时使用的信息。此处获取到的点位可能受限于 robot 实际安装位姿，需谨慎使用。 适用臂型：通用医疗臂、Thor3 **参数：** dblHome ：输出参数，六维力传感器标定过程推荐的 Home 点位向量的首地址，数组大小为 JOINT_NUM 的关节角信息。单位：rad。 dblHomeVel ：输出参数，用于接收系统推荐的 Home 点移动速度。 dblHomeAcc ：输出参数，用于接收系统推荐的 Home 点移动加速度。 dblWaypoints：输出参数，六维力传感器标定过程推荐的点位向量的首地址，数组为 6*6 的 pose 信息，每组点位后三个量为轴角。 dblVel：输出参数，六维力传感器标定过程推荐的点位对应的速度向量的首地址，数组长度为 6。单位：m/s dblAcc：输出参数，六维力传感器标定过程推荐的点位对应的加速度向量的首地址，数组长度为 6。单位：m/s^2 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。
调用示例： 详见 initSixAxisInstallation 的调用示例。

198 identifyTcpPayloadWithSixAxis

int identifyTcpPayloadWithSixAxis (double *dblPayload, double *dblHome, double dblHomeVel, double dblHomeAcc, double *dblWaypoints, double *dblVel, double *dblAcc, int size, FNCSTOP fnNeedStop, const char *strIpAddress = " ")
针对指定 IP 地址的机械臂，通过执行一系列移动指令来标定六维力传感器，并返回包含六维力传感器和转接盘的负载信息。六维力传感器零偏参数和计算所得的质量质心结果会直接存入后台。该功能会执行一段时间，移动至每个路点后大概会停留 10 秒左右。请不要使用多线程同时调用其他移动指令或修改系统参数配置。 适用臂型：通用医疗臂、Thor3

参数：

dblPayload：输出参数，用于接收负载信息，大小为 10 的数组，第 1 位为质量（单位：kg），2~4 位为质心（单位：m），5~10 位为张量（单位：kg•m^2）。该结果可以作为输入参数传给 setActiveTcpPayload 使用。目前该函数仅支持辨识质量和质心，即当返回结果有效时 dblResult 数组只有前四个有值。

dblHome ：输入参数，六维力传感器标定过程的 Home 点位向量的首地址，数组大小为 JOINT_NUM 的关节角信息。单位：rad。

dblHomeVel ：输入参数，移动至 Home 点的速度。

dblHomeAcc ：输入参数，移动至 Home 点的加速度。

dblWaypoints：输入参数，六维力传感器标定过程的点位向量的首地址，数组为 6*6 的 pose 信息，每组点位后三个量为轴角。该路点信息可以通过 getCalcSixAxisWaypoints 获取。

dblVel：输入参数，六维力传感器标定过程推荐的点位对应的速度向量的首地址，数组长度为 6。单位：m/s

dblAcc：输入参数，六维力传感器标定过程推荐的点位对应的加速度向量的首地址，数组长度为 6。单位：m/s^2

size：路点个数，一般为 6。

fnNeedStop：可选参数，停止标定功能的回调函数。函数声明形式：bool fnNeedStop (const char *strIpAddress)。当 fnNeedStop 函数返回 true 时，将中止正在执行的负载辨识程序。该参数缺省值为空，如果不传入该参数，用户开启负载辨识程序后将无法干预程序执行。fnNeedStop 函数将在负载辨识程序开始执行后，每 200 毫秒检测一次，尽量不要在函数实现中使用 sleep 函数之类会阻塞线程的操作。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

2：正在运行六维力标定。

-1：其他错误

-2：六维力标定相应 callback 函数停止。

-3：六维力标定路点数量不匹配

-4：六维力标定设置路点或 home 点失败。

-5：开始六维力标定失败。

-6：停止六维力标定失败。

-7：获取六维力标定结果失败。

-8：六维力标定路径规划失败。

-9：六维力标定 move 失败。

-11：非位置模式无法进行六维力标定

-12：系统正在执行其他程序无法进行六维力标定。

-13：负载辨识中无法进行六维力标定。

-14：力控模式无法进行六维力标定。

-15：系统处于非空闲态，无法进行六维力标定。

-16：路点存储失败。

-17：未打开抱闸。

调用示例：

详见 initSixAxisInstallation 的调用示例。

199 getFixedSixAxisForce

int getFixedSixAxisForce(double *dblFixedForce, const char *strIpAddress = " ")

针对指定 IP 地址的机械臂，（重新标定六维力传感器后）获取修正后的六维力值。

适用臂型：通用医疗臂、Thor3

参数：

fixedForce：输出参数，修正后的六维力传感器感应到的力向量的首地址，数组长度为 6。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

详见 initSixAxisInstallation 的调用示例。

200 getDefaultActiveTcp

int getDefaultActiveTcp(double *default_tcp, const char *strIpAddress = "")

获取指定 IP 地址机械臂默认的工具坐标系。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

default_tcp：输出参数，Tcp 相对于末端法兰盘的 4*4 齐次变换矩阵的首地址，数组长度

为 16。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* strIpAddress = "192.168.10.75";
double default_tcp[16] = {0.0};
int ret = getDefaultActiveTcp(default_tcp, strIpAddress);
if(ret < 0)
{
   printf("getDefaultActiveTcp failed! Return value = %d\n", ret);
}
```

201 setSixAxisSensorAbsAccuracy

int setSixAxisSensorAbsAccuracy(double *dblAccuracy, const char *strIpAddress = " ")

针对指定 IP 地址的机械臂，设置六维力传感器绝对测量精度。其中平移力分量的有效值范围为[0.001, 2.0]，扭矩分量的有效值范围为[0.0001, 0.1]。其中，平移分量的默认值都为 1.0，扭矩分量的默认值为 0.02。

适用臂型：通用医疗臂、Thor3

参数：

dblAccuracy：输入参数，绝对测量精度，大小为 6 的数组，依次对应平移力 x 分量、平移力 y 分量、平移力 z 分量、扭矩 x 分量、扭矩 y 分量和扭矩 z 分量。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* ipAddress = "192.168.10.75";
double accuracy[6] = {0.0};
int ret = setSixAxisSensorAbsAccuracy (accuracy,ipAddress);
if(ret < 0){
   printf("setSixAxisSensorAbsAccuracy failed! Return value = %d\n", ret);
}
```

202 getSixAxisSensorAbsAccuracy

int getSixAxisSensorAbsAccuracy (double *dblAccuracy, const char *strIpAddress = " ")
针对指定 IP 地址的机械臂，获取六维力传感器绝对测量精度。 适用臂型：通用医疗臂、Thor3 **参数：** dblAccuracy：输出参数，绝对测量精度，大小为 6 的数组，依次对应平移力 x 分量、平移力 y 分量、平移力 z 分量、扭矩 x 分量、扭矩 y 分量和扭矩 z 分量。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。

调用示例：
const char* ipAddress = "192.168.10.75";
double accuracy[6] = {0.0};
int ret = getSixAxisSensorAbsAccuracy (accuracy, ipAddress);
if(ret < 0){
printf("getSixAxisSensorAbsAccuracy failed! Return value = %d\n", ret);
}
else{
printf("getSixAxisSensorAbsAccuracy success\n");
}

203 getJointsSoftLimitRange

int getJointsSoftLimitRange(double *dblMinPos, double *dblMaxPos, const char *strIpAddress = "")
获取指定 IP 地址机械臂各关节的软限位。 适配臂型：通用医疗臂、定制医疗臂、Thor3。 **参数：** dblMinPos：输出参数，JOINT_NUM 个轴关节角度的极限位置下限组成的元组。单位：rad。 dblMaxPos：输出参数，JOINT_NUM 个轴关节角度的极限位置上限组成的元组。单位：rad。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

注：JOINT_NUM 表示机械臂的关节个数，Diana 机械臂的 JOINT_NUM 为 7，Thor3 为 6。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* robot0 =  "192.168.10.75";
double dblMinPos[JOINT_NUM] = {0}, dblMaxPos[JOINT_NUM]={0};
int ret = getJointsSoftLimitRange (dblMinPos, dblMaxPos, robot0);
printf("getJointsSoftLimitRange ret = %d dblMinPos = {%f,%f,%f,%f,%f,%f,%f} dblMaxPos = {%f,%f,%f,%f,%f,%f,%f}\n"
, ret
, dblMinPos[0]
, dblMinPos[1]
, dblMinPos[2]
, dblMinPos[3]
, dblMinPos[4]
, dblMinPos[5]
, dblMinPos[6]
, dblMaxPos[0]
, dblMaxPos[1]
, dblMaxPos[2]
, dblMaxPos[3]
, dblMaxPos[4]
, dblMaxPos[5]
, dblMaxPos[6]); //以七轴机械臂为例
```

204 setJointsSoftLimitRange

```
int setJointsSoftLimitRange(double *dblMinPos, double *dblMaxPos, const char *strIpAddress = "")
```

设置指定 IP 地址机械臂各关节软限位。

适配臂型：通用医疗臂、定制医疗臂、Thor3。

参数：

dblMinPos：输入参数，关节角的最小软限位，大小为 JOINT_NUM 的数组，单位：rad。

dblMaxPos：输入参数，关节角的最大软限位，大小为 JOINT_NUM 的数组，单位：rad。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

注：JOINT_NUM 表示机械臂的关节个数，Diana 机械臂的 JOINT_NUM 为 7，Thor3 为 6。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* robot0 =  "192.168.10.75";
double dblMinPos[JOINT_NUM] = {0}, dblMaxPos[JOINT_NUM]={0};
int ret = getJointsSoftLimitRange (dblMinPos, dblMaxPos, robot0);
for (int i = 0; i < JOINT_NUM; ++i) {
      dblMinPos[i]++;
      dblMaxPos[i]--;
}
ret = setJointsSoftLimitRange (dblMinPos,dblMaxPos,robot0);
printf("setJointsSoftLimitRange return %d\n",ret);
ret = saveEnvironment(robot0);
printf("saveEnvironment return %d\n",ret);
```

205 getTcpPayloadWithSixAxis

```
int getTcpPayloadWithSixAxis(double &dblMass, double *dblMassCenter, const char
*strIpAddress =  " ")
```

获取指定 IP 地址机械臂上六维力传感器感知到的负载质量和质心。

适用臂型：通用医疗臂、Thor3

参数：

dblMass：输出参数，负载质量，单位：Kg。

dblMassCenter：输出参数，用于存放负载质心的数组首地址，数组长度为 3。质心数据的含义依次为：mx、my、mz，单位：m。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
double dblMassCenterArr[3] = {0.0, 0.0, 0.0}, dblMass = 0.0;
const char* strIpAddress =  "192.168.10.75";
int ret = getTcpPayloadWithSixAxis (dblMass, dblMassCenterArr, strIpAddress);
if(ret < 0){
   printf( " getTcpPayloadWithSixAxis failed! Return value = %d\n ", ret);
} else{
printf( " getTcpPayloadWithSixAxis success! Return mass = %f, center = {%f, %f, %f }\n "
, dblMass
, dblMassCenterArr [0]
, dblMassCenterArr [1]
, dblMassCenterArr [2]
);
}
```

206 setTcpPayloadWithSixAxis

int setTcpPayloadWithSixAxis(const double dblMass, double *dblMassCenter, const char *strIpAddress = " ")

设置指定 IP 地址机械臂上六维力传感器感知到的负载质量和质心。设置后需调用 saveEnvironment 函数才能使设置永久生效。

适用臂型：通用医疗臂、Thor3

参数：

dblMass：输入参数，负载质量，单位：Kg。

dblMassCenter：输入参数，用于存放负载质心的数组首地址，数组长度为 3。负载质心数据的含义依次为：mx、my、mz，单位：m。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* strIpAddress =  "192.168.10.75";
double dblMassCenterArr[3] = {0.4732,0.0,0.0361}, dblMass = 0.24;
ret = setTcpPayloadWithSixAxis (dblMass, dblMassCenterArr, strIpAddress);
if(ret >= 0){
    printf("setTcpPayloadWithSixAxis return %d\n",ret);
ret = saveEnvironment(strIpAddress);
}
```

207 setThresholdTorque

int setThresholdTorque (double *arrThreshold, const char *strIpAddress = "")

设置指定 IP 地址机械臂各关节传感器检测阈值，此阈值在切换至阻抗模式时用于检测是否允许切换至阻抗模式。

适用臂型：通用医疗臂、定制医疗臂

参数：

arrThreshold：输入参数，表示各关节阈值 arrThreshold 的数组的首地址，数组长度为 JOINT_NUM,单位（Nm)。各关节允许最大值[18,18,15,15,6,6,6]，最小值皆为 0。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
double arrThreshold[7] = { 6, 6, 5, 5, 2, 2, 2};
const char* strIpAddress = "192.168.10.75";
if (setThresholdTorque (arrThreshold, strIpAddress) < 0)
{
   printf("Diana API setThresholdTorque failed!\n");
}
```

208 getThresholdTorque

int getThresholdTorque(double *arrThreshold, const char *strIpAddress = "")

获取指定 IP 地址机械臂各关节传感器检测阈值，此阈值在切换至阻抗模式时用于检测是否允许切换至阻抗模式。

适用臂型：通用医疗臂、定制医疗臂

参数：

arrThreshold：输出参数，表示各关节阈值 arrThreshold 的数组的首地址，数组长度为 JOINT_NUM,单位（Nm)

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
double arrThreshold [JOINT_NUM] = {0};
const char* strIpAddress = "192.168.10.75";
if ( getThresholdTorque (arrThreshold, strIpAddress) < 0)
{
   printf("Diana API getThresholdTorque failed!\n");
}
else
{
   printf(" getThresholdTorque : arrThreshold=%f, %f, %f, %f, %f, %f, %f\n",
   arrThreshold[0], arrThreshold[1], arrThreshold[2], arrThreshold[3], arrThreshold[4],
arrThreshold[5], arrThreshold[6]);
}
```

209 setStaticFriction

int setStaticFriction (bool bEnable, double dblVelocity, double dblForce, double dblTorque , const char *strIpAddress = "")

设置指定 IP 地址机械臂的静摩擦力，静摩擦力可以在自由驱动时用于弥补负载不匹配导致的机械臂漂移。

适用臂型：通用医疗臂、定制医疗臂

参数：

bEnable：输入参数，表示静摩擦力的开启与关闭，bool 类型。

dblVelocity：输入参数，当任意关节的速度超过该值时，静摩擦力停止生效，为 double 类型，单位 rad/s，输入范围[0.01,0.5]。

dblForce：输入参数，表示当静摩擦力生效时，在平移方向上，末端所允许补偿的最大力值，double 类型，单位 N，输入范围[0,10]。

dblTorque：输入参数，表示当静摩擦力生效时，在旋转方向上，末端所允许补偿的最大力矩值，double 类型，单位为 Nm，输入范围[0,2]。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
double dblVelocity = 0.1,dblForce = 3,dblTorque = 0.5;
const char* strIpAddress = "192.168.10.75";
if (setStaticFriction(true, dblVelocity, dblForce, dblTorque ,strIpAddress) < 0)
{
   printf("Diana API setStaticFriction failed!\n");
}
```

210 getStaticFriction

```
int getStaticFriction(bool &bEnable, double &dblVelocity, double &dblForce, double &dblTorque , const char *strIpAddress = "")
```

获取指定 IP 地址机械臂的静摩擦力状态，静摩擦力可以在自由驱动时用于弥补负载不匹配导致的机械臂漂移。

适用臂型：通用医疗臂、定制医疗臂

参数：

bEnable：输出参数，表示静摩擦力的开启与关闭，bool 类型。

dblVelocity：输出参数，当任意关节的速度超过该值时，静摩擦力停止生效，为 double 类型，单位 rad/s。

dblForce：输出参数，表示当静摩擦力生效时，在平移方向上，末端所允许补偿的最大

力值，double 类型，单位 N。

dblTorque：输出参数，表示当静摩擦力生效时，在旋转方向上，末端所允许补偿的最大力矩值，double 类型，单位为 N.m。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
bool enable = false;
double dblVelocity = 0,dblForce = 0,dblTorque = 0;
const char* strIpAddress = "192.168.10.75";
if (getStaticFriction(enable, dblVelocity, dblForce, dblTorque, strIpAddress) < 0)
{
   printf("Diana API getStaticFriction failed!\n");
}
else
{
   printf("getStaticFriction: enable=%d,velocity = %f,force = %f,torque= %f\n",
   enable,dblVelocity,dblForce,dblTorque);
}
```

211 setPredictMoveMode

int setPredictMoveMode(const bool bEnable, const int intSamples = 1000, const char *strIpAddress = “”)

设置当前系统是否处于运动预测模式。

a. 当系统处于运动预测模式时，所有可以使机器人移动的功能都将被禁用。

b. 设置系统模式为运动预测模式或普通模式（即非运动预测模式）。仅当当前系统为位置模式时才可以进入运动预测模式。如果当前为非位置模式，进入运动预测模式会失败，但退出运动预测模式会返回成功，但对系统无任何操作。

c. 当系统被设置为运动预测模式后，如需修改每个路段插补个数，无需退出运动预测模式，再次调用进入该模式即可更新。

注 1：仅当系统处于位置模式时可以切换到运动预测模式。在运动预测模式下，仅支持 moveL、moveJ、moveC、runPath、moveJToPose、moveJToTarget、moveLToPose、moveLToTarget、runComplexPath 几个指令的预测。并且在该模式下除 initSrv、destroySrv、releaseBrake、holdBrake、enterSafetyIdle、leaveSafetyIdle、

freeDrivingForCurrentLoop、getCurrentLoopFreeDrivingState、cleanErrorInfo、定制臂 1.0 的寻零和所有 getXXX 系列 API 可以正常使用外，其他 API 均返回失败（返回值-1），但不报错误码。

注 2：预测模式下，当前仅支持一条 move 指令（包括：moveL、moveJ、moveC、moveJToPose、moveJToTarget、moveLToPose、moveLToTarget）的预测，如需要多路段预测请使用 runPath 或 runComplexPath。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数说明：

bEnable ：输入参数，bool 型变量。当 bEnable=true，代表进入预测模式，此时 intSample 的值有效；当 bEnable=false，代表退出预测模式，此时 intSample 的值会被忽略。

intSamples：输入参数，int 型变量。代表每个路段插补个数，仅当 bEnable 为 true 时有效。该参数的取值范围为[10, 1000]。默认值为 1000。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功

-1：失败

调用示例：

```
const char* strIpAddress = "192.168.10.75";
ret = setPredictMoveMode(true, 933, strIpAddress);
printf("setPredictMoveMode ret %d error=%d\n", ret, getLastError(strIpAddress));
ret = getPredictMoveMode(strIpAddress);
printf("getPredictMoveMode ret %d\n", ret);
double joints1[7] = {to_rad(0),to_rad(0),to_rad(0),to_rad(90),to_rad(0),to_rad(-90),to_rad(0)};
ret = moveJToTarget(joints1, 0.2,0.2, strIpAddress);
printf("predict mode: moveJToTarget 1 return %d error=%d\n", ret, getLastError(strIpAddress));
wait_move(strIpAddress);
ret = setPredictMoveMode(false, -1, strIpAddress);
printf("setPredictMoveMode ret %d error=%d\n", ret, getLastError(strIpAddress));
```

212 getPredictMoveMode

bool getPredictMoveMode(const char *strIpAddress = “”)

获取当前系统是否处于运动预测模式。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数说明：

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

true：当前系统为运动预测模式。

false：当前系统为普通模式（即非运动预测模式）。

调用示例：

详见 setPredictMoveMode

213 setSafetySpeedLimit

int setSafetySpeedLimit(double* dblTcpSpeedLimit, double* dblJointsSpeedLimit, const char *strIpAddress = “”)

设置指定 IP 地址机械臂在全模式下所允许的最大速度，当满足某个限制条件时，机械臂会自动进入立即停止，关闭抱闸，并切换至位置模式。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数说明：

dblTcpSpeedLimit：输入参数，大小为 6 的数组，指定笛卡尔空间允许的最大速度。其中前三个参数表示平移方向的速度限值（单位 m/s），设定的值应不小于 0.05 且不大于机械限定的笛卡尔空间平移速度的最大值(参考 getMechanicalMaxCartVelAcc)，当超过该值时，会自动修改设定的值为该最大值。后三个参数表示旋转方向的速度限值（单位 rad/s），设定的值应不小于 0.2 且不大于机械限定的笛卡尔空间旋转速度的最大值(参考 getMechanicalMaxCartVelAcc)，当超过该值时，会自动修改设定的值为该最大值。如果将某方向设为 0，则默认为该方向上的速度不做限制。

dblJointsSpeedLimit：输入参数，大小为关节数量的数组，指定关节空间允许的最大速度（单位 rad/s），设定的值应不小于 0.2 且不大于机械限定的关节空间转动速度的最大值(参考 getMechanicalMaxJointsVel)，当超过该值时，会自动修改设定的值为该最大值。如果将某关节设为 0，则默认为该关节上的速度不做限制。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
const char* strIpAddress = "192.168.10.75";
double dblTcpSpeedLimit[6] = {1.2,1.2,1.2,1.57,1.57,1.57};
```

```
double dblJointsSpeedLimit[7] = {1.57,1.57,1.57 ,0.5,1.57,1.57,1.57};
int ret = setSafetySpeedLimit (dblTcpSpeedLimit, dblJointsSpeedLimit , strIpAddress);
printf("setSafetySpeedLimit ret %d error=%d\n", ret, getLastError(strIpAddress));
ret = getSafetySpeedLimit (dblTcpSpeedLimit, dblJointsSpeedLimit ,strIpAddress);
printf(“Tcp Space:%lf, %lf, %lf, %lf, %lf, %lf\n”, dblTcpSpeedLimit[0], dblTcpSpeedLimit[1],
dblTcpSpeedLimit[2], dblTcpSpeedLimit[3], dblTcpSpeedLimit[4], dblTcpSpeedLimit[5]);
printf(“Joint Space:%lf, %lf, %lf, %lf, %lf, %lf,%lf\n”, dblJointsSpeedLimit [0],
dblJointsSpeedLimit [1], dblJointsSpeedLimit [2], dblJointsSpeedLimit [3],
dblJointsSpeedLimit [4], dblJointsSpeedLimit [5], dblJointsSpeedLimit[6]);
```

214 getSafetySpeedLimit

int getSafetySpeedLimit(double* dblTcpSpeedLimit, double* dblJointsSpeedLimit, const char *strIpAddress = “”)
获取指定 IP 地址机械臂在全模式下所允许的最大速度。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数说明：** dblTcpSpeedLimit：输出参数，大小为 6 的数组，指定笛卡尔空间允许的最大速度，其中前三个参数单位为 m/s，后三个参数单位为 rad/s。 dblJointsSpeedLimit：输出参数，大小为关节数量的数组，指定关节空间允许的最大速度，单位 rad/s。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。
调用示例： 详见 setSafetySpeedLimit

215 setSafetyCartZone

int setSafetyCartZone(/*IN[3]*/const double* dblTcpPos,/*IN[3]*/const double * dblCartFreeZone, /*IN*/const char *strIpAddress = “”)
设置指定 IP 地址机械臂的安全运动空间，该空间是一个以指定 TCP 位置为中心点，指定长、宽、高而形成的立方体。该安全空间在非安全处理模式下均生效。当机械臂在运动过程中 TCP 超出该安全空间后，机械臂会立即停止、关闭抱闸，并切换至位置模式。此时用户可以通过安全处理方式将机械臂驱回安全运动空间。 适用臂型：通用医疗臂、定制医疗臂、Thor3 **参数：** dblTcpPos：输入参数，大小为 3 的数组，表示安全立方体的中心点，为基坐标系下指定 TCP 的位置，单位为 m。 dblCartFreeZone：输入参数，大小为 3 的数组，表示安全立方体的长、宽、高，单位为 m。当设置为 0 时，则表示该方向不做限制。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：**

0：成功。

-1：失败。

调用示例：

```
const char* strIpAddress = "192.168.10.75";
double dblTcpPose[3] = {0.5994,0.005,0.7045};
double dblCartZoneParameter[3] = {0.6,0.6,0.6};
int ret = setSafetyCartZone(dblTcpPose, dblCartZoneParameter, strIpAddress);
printf("setSafetyCartZone ret %d error=%d\n", ret, getLastError(strIpAddress));
double dblTcpPose1[3] = {0.0};
double dblCartZoneParameter1[3] = {0.0};
ret = getSafetyCartZone(dblTcpPose1, dblCartZoneParameter1,strIpAddress);
printf("Cart Zone Center:%lf, %lf, %lf\n", dblTcpPose1[0], dblTcpPose1[1], dblTcpPose1[2]);
printf("Cart Zone Parameter:%lf, %lf, %lf\n", dblCartZoneParameter1[0], dblCartZoneParameter1[1], dblCartZoneParameter1[2]);
```

216 getSafetyCartZone

```
bool getSafetyCartZone(/*OUT[3]*/const double* dblTcpPos,/*OUT[3]*/const double * dblCartFreeZone, /*OUT*/const char *strIpAddress = "")
```

获取指定 IP 地址机械臂 TCP 的安全运动空间，该空间是一个以指定 TCP 位置为中心点，指定长、宽、高而形成的立方体。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数说明：

dblTcpPos：输出参数，大小为 3 的数组，表示安全立方体的中心点，为基坐标系下指定 TCP 的位置，单位为 m。

dblCartFreeZone：输出参数，大小为 3 的数组，表示安全立方体的长、宽、高，单位为 m。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功

-1：失败

调用示例：

详见 setSafetyCartZone

217 getEncoderBatteryLevel

```
int getEncoderBatteryLevel(/*OUT[JOINT_COUNT]*/int *arrBatteryLevels, /*IN*/const char
```

*strIpAddress = “”)
获取机械臂关节编码器电源电量等级。 适用臂型： Thor3 **参数：** arrBatteryLevels：输出参数，关节编码器电源电量等级数组首地址，数组长度为 JOINT_NUM。1 代表电源电量正常；0 代表电源电量低；-1 代表编码器不可用，电量耗尽。 strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。 **返回值：** 0：成功。 -1：失败。
调用示例： int arrBatteryLevels[7] = {0}; int ret = getEncoderBatteryLevel(arrBatteryLevels); printf("getEncoderBatteryLevel return %d， battery={%d,%d,%d,%d,%d,%d,%d}\n", ret, arrBatteryLevels[0], arrBatteryLevels[1], arrBatteryLevels[2], arrBatteryLevels[3], arrBatteryLevels[4], arrBatteryLevels[5], arrBatteryLevels[6]);

218 getWarningList

int getWarningList(/*OUT[MAX_WARNING_NUM]*/int *arrWarningCodes, /*IN*/const int len, /*IN*/const char *strIpAddress = “”)
获取指定 IP 地址机械臂的系统告警码列表。 适用臂型：通用医疗臂、定制医疗臂、 Thor3 **参数：** arrWarningCodes：输出参数，机械臂系统告警码列表数组首地址，数组最大长度为 MAX_WARNING_LIST_SIZE（暂定为 10）。数组用于接收系统当前所有告警码（WARNING_CODE）。 len: arrWarningCodes 数组的长度，范围在(0, MAX_WARNING_LIST_SIZE]内。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

>=0：当前系统告警数量。

-1：失败。

调用示例：

```
int arrWarningList[10] = {0};
int cnt = getWarningList(arrWarningList, 10);
printf("getWarningList[%d]={%d,%d,%d,%d,%d,%d,%d,%d,%d,%d}\n",
        cnt,
        arrWarningList[0],
        arrWarningList[1],
        arrWarningList[2],
        arrWarningList[3],
        arrWarningList[4],
        arrWarningList[5],
        arrWarningList[6],
        arrWarningList[7],
        arrWarningList[8],
        arrWarningList[9]);
for (int i = 0; i < cnt; ++i) {
    printf("warningcode = %d, warningMsg = %s \n", arrWarningList[i],
formatWarning(arrWarningList[i]));
}
ret = cleanWarning();
printf("cleanWarning ret %d\n", ret);
cnt = getWarningList(arrWarningList, 10);
printf("getWarningList[%d]={%d,%d,%d,%d,%d,%d,%d,%d,%d,%d}\n",
        cnt,
        arrWarningList[0],
        arrWarningList[1],
        arrWarningList[2],
        arrWarningList[3],
        arrWarningList[4],
        arrWarningList[5],
        arrWarningList[6],
        arrWarningList[7],
```

```
            arrWarningList[8],
            arrWarningList[9]);
for (int i = 0; i < cnt; ++i) {
      printf("warningcode = %d, warningMsg = %s \n", arrWarningList[i],
formatWarning(arrWarningList[i]));
}
```

219 formatWarning

const char *formatWarning (/*IN*/int w, /*IN*/const char *strIpAddress = “”)

获取指定 IP 地址机械臂的告警码 w 的字符串描述，该告警码可以通过调用 getWarningList 得到。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

w：告警码。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

告警码对应的描述信息。

调用示例：

详见 getWarningList 。

220 cleanWarning

int cleanWarning (/*IN*/const char *strIpAddress = “”)

清除指定 IP 地址机械臂的所有普通告警。

适用臂型：通用医疗臂、定制医疗臂、Thor3

参数：

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

详见 getWarningList 。

221 calibrateSixAxisSensor

int calibrateSixAxisSensor (/*IN*/const char *strIpAddress = “”)

标定指定 IP 地址机械臂的零偏，使用本功能需要六维力传感器已进行过一次标定。标定结果不会存储到配置文件，如希望下次开机后结果依然生效，则应当调用 saveEnvironment。本功能仅支持位置模式下使用。

适用臂型：通用医疗臂、Thor3

参数：

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
enableSixAxis(six_axis_type_e::SIX_AXIS_IN_FREE_DRIVING,true);
int ret = calibrateSixAxisSensor();
if(ret == 0){
   printf("calibrateSixAxisSensor succeed.\n");
   saveEnvironment();
}else{
   printf("calibrateSixAxisSensor failed.\n");
}
```

222 calcInstallAngle

```
int calcInstallAngle(/*IN[6]*/ const double *dblPose1, /*IN[6]*/ const double *dblPose2, /*IN[6]*/ const double *dblPose3, /*OUT[3]*/ double *angle, /*IN*/ const char *strIpAddress = " ")
```

用户通过自由驱动 (freeDriving) 在水平面上示教三个点，获取 (getTcpPos) 其位姿，通过这三个点的位姿计算基坐标系安装角度。

示教的三个输入点位的要求：

1. 三点需要可以构成有效平面，即三点不重合且不共线。

2. 三点在水平面按照逆时针分布 (俯视图)。

适用臂型：通用医疗臂、定制医疗臂、 Thor3

参数：

dblPose1：输入参数，大小为 6 的数组，用户示教的第一个点的位姿，其中前三个参数单位为 m，后三个参数单位为 rad，后三个角度是轴角。

dblPose2：输入参数，大小为 6 的数组，用户示教的第二个点的位姿，其中前三个参数单位为 m，后三个参数单位为 rad，后三个角度是轴角。

dblPose3：输入参数，大小为 6 的数组，用户示教的第三个点的位姿，其中前三个参数单位为 m，后三个参数单位为 rad，后三个角度是轴角。

angle：输出参数，大小为 3 的数组，机械臂基坐标的安装角度，单位为 rad，轴角表示。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
#define TO_DEGREE(x)  (x * 180 / PI)

double tcpPose1[6] = { 0 };
double tcpPose2[6] = { 0 };
double tcpPose3[6] = { 0 };

printf("请在平面上逆时针示教三个点:\n");
freeDriving(true);
printf("示教第一个点，示教到点位后按任意键：\n");
getchar();
getTcpPos(tcpPose1);

printf("示教第二个点，示教到点位后按任意键：\n");
getchar();
getTcpPos(tcpPose2);

printf("示教第三个点，示教到点位后按任意键：\n");
getchar();
getTcpPos(tcpPose3);
freeDriving(false);

double angle[3] = { 0.0 };
int ret = calcInstallAngle(tcpPose1, tcpPose2, tcpPose3, angle);
if (ret == -1) {
   printf("calcInstallAngle Failed\n");
} else {
   printf("install angle:[%f, %f, %f]\n", TO_DEGREE(angle[0]), TO_DEGREE(angle[1]),
```

```
TO_DEGREE(angle[2]));
}
```

223 setFunctionOptR8

int setFunctionOptR8 (int intFunctionIndex, int intOptName, double dblOptVal, const char * strIpAddress = " ")

设置指定功能的高精度浮点型参数。

适用臂型：通用医疗臂

参数：

intFunctionIndex：指定功能索引, 取值为 function_index_e 枚举类型，枚举值及含义如下表所示。

intFunctionIndex	对应枚举	含义
0	function_index_e::T_FREEDRIVING	自由驱动相关
1	function_index_e::T_CART_IMPEDANCE	力控和笛卡尔阻抗相关

intOptName：功能的可选参数，

当 intFunctionIndex 为 0 时，支持参数如下：

intOptName	对应枚举	含义
0x10006	freedriving_opt_name_e::T_MAX_TRANSLATION_DECELERATION	设置实时虚拟墙内减速区域的最大平移速度
0x10009	freedriving_opt_name_e::T_MAX_ALLOWED_MOVE_ZONE	设置导纳自由驱动的最大活动距离。如果当前为阻抗自由驱动，则此参数设置无效。
0x1000A	freedriving_opt_name_e::T_MAX_ALLOWED_MOVE_SPEED	设置导纳自由驱动的极限速度。如果当前为阻抗自由驱动，则此参数设置无效。

dblOptVal：设置 intFunctionIndex 与 intOptName 组合所指定的属性的 double 值。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

-1：失败。

0：成功。

调用示例：

参考 updateRealtimeVirtualWall。

224 getFunctionOptR8

int getFunctionOptR8 (int intFunctionIndex, int intOptName, double *dblOptVal, const char * strIpAddress = " ")

获取指定功能的高精度浮点型参数。

适用臂型：通用医疗臂

参数：

intFunctionIndex：指定功能索引,如下表所示。

intFunctionIndex	对应枚举	含义
0	function_index_e::T_FREEDRIVING	自由驱动相关
1	function_index_e::T_CART_IMPEDANCE	力控和笛卡尔阻抗相关

intOptName：功能的可选参数，

当 intFunctionIndex 为 0 时，参数如下：

intOptName	对应枚举	含义
0x10006	freedriving_opt_name_e:: T_MAX_TRANSLATION_DECELERATION	获取实时虚拟墙内减速区域的最大平移速度
0x10009	freedriving_opt_name_e::T_MAX_ALLOWED_MOVE_ZONE	获取导纳自由驱动的最大活动距离。
0x1000A	freedriving_opt_name_e::T_MAX_ALLOWED_MOVE_SPEED	获取导纳自由驱动的极限速度。

dblOptVal：获取 intFunctionIndex 与 intOptName 组合所指定的属性的 double 值。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

-1：失败。

0：成功。

调用示例：

参考 updateRealtimeVirtualWall。

225 updateRealtimeVirtualWall

int updateRealtimeVirtualWall(bool enteredDecelerationBoundary,double distance2HazardousBoundary,double *safetyDirection,/*IN*/const char *strIpAddress = “”)

实时指令，机械臂会根据上位机发送的实时指令参数实现任意形状的虚拟墙效果。目前仅支持在导纳模式（需保证六维力传感器在线），进行定向自由驱动（需要保证自由度只存在 1~5 个分量）时设置。目前虚拟墙的效果如下：机械臂会按照收到的危险边界距离参数和当前自身速度在危险方向上的速度分量判断是否会导致机械臂触碰危险边界，如判断导致碰撞，则开始在危险方向上进行减速，确保机械臂以确定的加速度停止在危险边界上。

适用臂型：通用医疗臂

关联函数：

开启实时虚拟墙请参考函数 setFunctionOptI4。

查询实时虚拟墙的状态请参考 getFunctionOptI4。

设置虚拟墙的减速区的最大平移速度请参考 setFunctionOptR8。

查询虚拟墙的减速区的最大平移速度请参考 getFunctionOptR8。

参数：

enteredDecelerationBoundary：输入参数，bool 类型，表示是否进入减速边界。

distance2HazardousBoundary：输入参数，double 类型，表示距离危险边界的距离，单位 m，由上位机实时发送。

safetyDirection：输入参数，大小为 3 的 double 类型数组，安全方向矢量[X, Y, Z]，由上位机实时发送。

strIpAddress：可选参数，需要控制机械臂的 IP 地址字符串，不填仅当只连接一台机械臂时生效。

返回值：

0：成功。

-1：失败。

调用示例：

```
int ret = setFunctionOptI4(function_index_e::T_FREEDRIVING,
freedriving_opt_name_e::T_CONTROL_RULE, 1);
if (ret == 0) {
   printf("succeed to enable admittance control!\n");
   ret = setFunctionOptI4(function_index_e::T_FREEDRIVING,
freedriving_opt_name_e::T_ENABLE_REALTIME_VIRTUAL_WALL, 1);
   if (ret == 0) {
      printf("succeed to enable realtime virtual wall!\n");
      int intOptValue = 0;
      ret = getFunctionOptI4(function_index_e::T_FREEDRIVING,
freedriving_opt_name_e::T_ENABLE_REALTIME_VIRTUAL_WALL,
                        &intOptValue);
      if (ret == 0) {
         if (intOptValue == 1) {
            printf("realtime virtual wall is enabled!\n");
            double dblOptValue = 0.1;
            ret = setFunctionOptR8(function_index_e::T_FREEDRIVING,
freedriving_opt_name_e::T_MAX_TRANSLATION_DECELERATION, dblOptValue);
            if (ret == 0) {
               printf("succeed to set max translation deceleration in realtime virtual wall!\n");
```

```
            ret = getFunctionOptR8(function_index_e::T_FREEDRIVING,

freedriving_opt_name_e::T_MAX_TRANSLATION_DECELERATION, &dblOptValue);
            if (ret == 0) {
               printf("current max translation deceleration is %lf in realtime virtual wall!\n",
dblOptValue);
               double dblForceDirection[6] ={1,0,0,0,0,0};
               freeDriving_ex(true,0,dblForceDirection);
               bool enteredDecelerationBoundary = false;
               double distance2HazardousBoundary = 1;
               double safetyDirection[3] = {1, 0, 0};
               for (int i = 0; i < 1000; ++i) {
                  enteredDecelerationBoundary = (i < 500 ? false : true);
                  distance2HazardousBoundary = 1 - i * 0.0005;
                  updateRealtimeVirtualWall(enteredDecelerationBoundary,
distance2HazardousBoundary,
                                            safetyDirection);
                  M_SLEEP(1)
               }
               freeDriving_ex(false,0,dblForceDirection);
            }
         }
      }
   }
}
int ret = setFunctionOptI4(function_index_e::T_FREEDRIVING,
freedriving_opt_name_e::T_CONTROL_RULE, 0);
if (ret == 0){
   printf("succeed to disable admittance control!\n");
}
ret = setFunctionOptI4(function_index_e::T_FREEDRIVING,
freedriving_opt_name_e::T_ENABLE_REALTIME_VIRTUAL_WALL, 0);
if (ret == 0) {
   printf("succeed to disable realtime virtual wall!\n");
}
```

附录 A：

表 1：Diana API 接口错误码表

系统错误宏定义	错误码	说明
ERROR_CODE_WSASTART_FAIL	-1001	加载 windows 系统 socket 库失败
ERROR_CODE_CREATE_SOCKET_FAIL	-1002	创建 socket 对象失败
ERROR_CODE_BIND_PORT_FAIL	-1003	socket 绑定端口失败
ERROR_CODE_SOCKET_READ_FAIL	-1004	socket 的 select 调用失败
ERROR_CODE_SOCKET_TIMEOUT	-1005	socket 的 select 调用超时
ERROR_CODE_RECVFROM_FAIL	-1006	socket 接收数据失败
ERROR_CODE_SENDTO_FAIL	-1007	socket 发送数据失败
ERROR_CODE_LOST_HEARTBEAT	-1008	服务端的心跳连接丢失
ERROR_CODE_LOST_ROBOTSTATE	-1009	服务端信息反馈丢失
ERROR_CODE_GET_DH_FAILED	-1010	获取 DH 信息失败
ERROR_CODE_RELEASE_BRAKE_FAILED	-1011	打开抱闸失败
ERROR_CODE_HOLD_BRAKE_FAILED	-1012	关闭抱闸失败
ERROR_CODE_IP_ADDRESS_NOT_REGISTER	-1013	该 IP 机械臂尚未 initSrv
ERROR_CODE_ROBOTARM_OVERNUMBER	-1014	超过最大支持机械臂数
ERROR_CODE_SOCKET_OTHER_ERROR	-1015	其他 socket 连接错误
ERROR_CODE_SOCKET_CONNECTED_ALREADY	-1016	socket 已建立连接
ERROR_CODE_UNKNOWN_ROBOTSTATE	-1017	未知机械臂状态
ERROR_CODE_RAID_INVALID	-1018	全部磁盘阵列不可用
ERROR_CODE_JOINT_REGIST_ERROR	-2001	硬件错误

ERROR_CODE_CURRENT_OFFSET_ERROR	-2002	电流偏置错误
ERROR_CODE_EQEP_ENCODER	-2003	高速侧编码器正交编码信号错误
ERROR_CODE_SPIENCODER	-2004	与低速侧编码器通信断开
ERROR_CODE_HALL_SENSOR	-2005	开关 Hall 相序错误
ERROR_CODE_CURRENT_BUS_OVERTIME	-2006	母线电流长时间过载
ERROR_CODE_CURRENT_IQ_OVERTIME	-2007	电机三相电流长时间过载
ERROR_CODE_POSITION_CMD_STEP	-2008	位置指令阶跃错误
ERROR_CODE_TORQUE_SENSOR	-2009	扭矩传感器信号故障 1
ERROR_CODE_EEPROM_READ	-2010	关节读 EEPROM 参数错误
ERROR_CODE_EEPROM_WRITE	-2011	关节写 EEPROM 参数错误
ERROR_CODE_LS_ENCODER_OVERSPEED	-2012	低速侧编码器反馈位置超速
ERROR_CODE_LS_ENCODER_FB_ERROR	-2013	低速侧编码器反馈数据错误
ERROR_CODE_MS_SINGAL_Z_ERROR	-2014	高速侧编码器 Z 信号异常
ERROR_CODE_THREE_PHASE_CURRENT	-2015	电机三相电流瞬时过流
ERROR_CODE_TORQUE_SENSOR_READ_ERROR	-2016	扭矩传感器读取故障
ERROR_CODE_COMMUNICATE_ERROR	-2101	底层通信失败（ln）
ERROR_CODE_LOST_HEART_WITH_DIANAROBOT_ERROR	-2102	与后台服务心跳断开
ERROR_CODE_CALLING_CONFLICT_ERROR	-2201	暂停状态执行操作失败
ERROR_CODE_COLLISION_ERROR	-2202	发生碰撞
ERROR_CODE_NOT_FOLLOW_POSITION_CMD	-2203	力控模式关节位置与指令发生滞后

ERROR_CODE_NOT_FOLLOW_TCP_CMD	-2204	力控模式 TCP 位置与指令发生滞后
ERROR_CODE_NOT_ALL_AT_OP_STATE	-2205	有关节未进入正常状态
ERROR_CODE_OUT_RANGE_FEEDBACK	-2206	关节角反馈超软限位
ERROR_CODE_EMERGENCY_STOP	-2207	急停已拍下
ERROR_CODE_NO_INIT_PARAMETER	-2208	找不到关节初始参数
ERROR_CODE_NOT_MATCH_LOAD	-2209	负载与理论值不匹配
ERROR_CODE_CANNOT_MOVE_WHILE_FREE_DRIVING	-2210	自由驱动模式不能执行其他运动
ERROR_CODE_CANNOT_MOVE_WHILE_ZERO_SPACE_FREE_DRIVING	-2211	零空间自由驱动模式下不能执行其他运动
ERROR_CODE_PASSTHROUGH_FIFO_OVERFLOW_UP_ERROR	-2212	透传上溢出
ERROR_CODE_PASSTHROUGH_FIFO_OVERFLOW_DOWN_ERROR	-2213	透传下溢出
ERROR_CODE_ROBOT_IN_VIRTUAL_WALL	-2214	有关节在虚拟墙内
ERROR_CODE_CONFLICT_TASK_RUNNING	-2215	运动任务冲突
ERROR_CODE_OUT_OF_PHYSICAL_RANGE_FEEDBACK	-2216	超出物理限位
ERROR_CODE_PARAMETER_POINTER_EQUALS_NULLPTR	-2217	非法参数传递
ERROR_CODE_ PARAMETER_EQUALS_NAN_OR_INF	-2218	输入参数存在 nan 或 inf
ERROR_CODE_UNKNOWN_FIRMWARE_VERSION	-2219	未知的固件版本
ERROR_CODE_INPUT_OUT_OF_EXTREME_POSITION_RANGE	-2220	输入超出关节极限位置
ERROR_CODE_INPUT_OUT_OF_PHYSICAL_POSITION_RANGE	-2221	输入超出物理限位
ERROR_CODE_NOT_IN_SAFETY_HANDLE_MODE	-2222	当前未处于安全处理模式
ERROR_CODE_INCOMPATIBLE_FIRMWARE_VERSION	-2223	电流环自由驱动不兼容的固件版本
ERROR_CODE_VEL_OR_ACC_PARAMETER_OUT_OF_RANGE	-2232	速度或加速度超限
ERROR_CODE_POINTS_CANT_FORM_PLANE	-2233	三点无法构成有

		效平面
ERROR_CODE_PLAN_ERROR	-2301	路径规划失败
ERROR_CODE_INTERPOLATE_POSITION_ERROR	-2302	位置模式插补失败
ERROR_CODE_INTERPOLATE_TORQUE_ERROR	-2303	阻抗模式插补失败
ERROR_CODE_SINGULAR_VALUE_ERROR	-2304	奇异位置
ERROR_CODE_ PLANNER_ERROR	-2305	规划失败
ERROR_CODE_ HOME_POSITION_ERROR	-2306	需要寻零
ERROR_CODE_ FATAL	-2307	严重错误(关节位置超出物理极限)
ERROR_CODE_ POS_LIMIT	-2308	位置超出限制
ERROR_CODE_ FORCE_LIMIT	-2309	关节力矩超出限制
ERROR_CODE_ SPEED_LIMIT	-2310	速度超出限制
ERROR_CODE_ ACC_LIMIT	-2311	加速度超出限制
ERROR_CODE_ JERK_LIMIT	-2312	加加速度超出限制
ERROR_CODE_ MOTION_LIMIT	-2313	位置超出限制
ERROR_CODE_ IK_TRACK	-2314	轨迹跟踪过程逆解求解失败
ERROR_CODE_ IK_GENERAL	-2315	通用位置逆解求解失败
ERROR_CODE_ PLAN_INPUT	-2316	轨迹规划输入错误
ERROR_CODE_ PLAN_MOVJ	-2317	关节空间轨迹规划失败
ERROR_CODE_ PLAN_MOVL	-2318	直线轨迹规划失败
ERROR_CODE_ PLAN_MOVC	-2319	圆弧轨迹规划失败
ERROR_CODE_ PLAN_BLEND	-2320	过渡轨迹规划失败
ERROR_CODE_ PLAN_SPDJ	-2321	SpeedJ 轨迹规划失败
ERROR_CODE_ PLAN_SPDL	-2322	SpeedL 轨迹规划失败
ERROR_CODE_ PLAN_SRVJ	-2323	ServoJ 轨迹规划

		失败
ERROR_CODE_ PLAN_SRVL	-2324	ServoL 轨迹规划失败
ERROR_CODE_ MOVE_UNKNOWN	-2325	未知运动类型或运动类型不匹配
ERROR_CODE_ MOVE_UNPLAN	-2326	轨迹未规划
ERROR_CODE_ MOVE_INPUT	-2327	轨迹插补输入错误
ERROR_CODE_ MOVE_INTERP	-2328	轨迹插补失败
ERROR_CODE_ PLAN_TRANSLATION	-2329	移动规划失败
ERROR_CODE_ PLAN_ROTATION	-2330	旋转规划失败
ERROR_CODE_ PLAN_JOINTS	-2331	关节规划失败
ERROR_CODE_ UNMATCHED_JOINTS_NUMBER	-2332	零空间自由驱动关节数不匹配
ERROR_CODE_ TCPCALI_FUTILE_WPS	-2333	示教点不合理
ERROR_CODE_ TCPCALI_FIT_FAIL	-2334	拟合 TCP 失败
ERROR_CODE_ DHCALI_FIT_WF_FAIL	-2335	DH 参数初始化世界坐标系失败
ERROR_CODE_ DHCALI_FIT_TF_FAIL	-2336	DH 参数初始化工具坐标系失败
ERROR_CODE_DHCALI_FIT_DH_FAIL	-2337	DH 参数拟合失败
ERROR_CODE_ DHCALI_INIT_FAIL	-2338	DH 参数初始化失败
ERROR_CODE_ SLFMOV_SINGULAR	-2339	零空间运动至奇异位置
ERROR_CODE_ SLFMOV_FUTILE	-2340	零空间运动在笛卡尔空间内无效
ERROR_CODE_SLFMOV_JNTLIM	-2341	零空间运动至关节限位
ERROR_CODE_ SLFMOV_SPDLIM	-2342	零空间运动至关节限位
ERROR_CODE_ SLFMOV_FAIL	-2343	零空间运动插补失败
ERROR_CODE_ SLFMOV_FFC_FAIL	-2344	零空间运动前馈补偿错误
ERROR_CODE_ LOADIDENT_INIT_FAIL	-2345	负载辨识初始化失败
ERROR_CODE_ LOADIDENT_UFB_FAIL	-2346	负载辨识更新反

		馈数据错误
ERROR_CODE_ LOADIDENT_FIT_FAIL	-2347	负载辨识失败
ERROR_CODE_ LOADIDENT_NONLOADED	-2348	未检测到有效负载
ERROR_CODE_RESOURCE_UNAVAILABLE	-3001	参数错误
ERROR_CODE_DUMP_LOG_TIMEOUT	-3002	导出 Log 文件超时
ERROR_CODE_DUMP_LOG_FAILED	-3003	导出 Log 文件失败
ERROR_CODE_RESET_DH_FAILED	-3004	重置 DH 参数失败

注：表 1 中 ERROR_CODE_JOINT_REGIST_ERROR（-2001）硬件错误和 ERROR_CODE_NOT_ALL_AT_OP_STATE(-2205)的 OP 状态错误需要通过调用 holdBrake()合抱闸函数或重启硬件来清除错误。

附录 B：

表 2：Diana API 接口告警码表

系统告警宏定义	告警码	说明
WARNING_CODE_RAID_PARTIAL_INVALID	-1001	部分磁盘阵列不可用
WARNING_CODE_JOINT_1_ENCODER_BATTERY_LOW	-1002	关节 1 编码器电源电量低
WARNING_CODE_JOINT_2_ENCODER_BATTERY_LOW	-1003	关节 2 编码器电源电量低
WARNING_CODE_JOINT_3_ENCODER_BATTERY_LOW	-1004	关节 3 编码器电源电量低
WARNING_CODE_JOINT_4_ENCODER_BATTERY_LOW	-1005	关节 4 编码器电源电量低
WARNING_CODE_JOINT_5_ENCODER_BATTERY_LOW	-1006	关节 5 编码器电源电量低
WARNING_CODE_JOINT_6_ENCODER_BATTERY_LOW	-1007	关节 6 编码器电源电量低
WARNING_CODE_JOINT_7_ENCODER_BATTERY_LOW	-1008	关节 7 编码器电源电量低
WARNING_CODE_PREDICT_CONFLICTED_CALLING	-1009	预测模式下的冲突调用

附录 C：

确保运动学逆解唯一性的解决方案

Diana 机械臂为七自由度机械臂，由于多了一个冗余自由度，理论上存在无数多组逆解(根据末端位姿求解关节角)，在实际应用中，当执行逆解运算或者进行笛卡尔空间运动时，有可能出现逆解不唯一的情况。为了确保逆解的唯一性，可采取如下解决方案。

第一种情况: 已知其中某个路点对应的关节角。

例：已知 A 点关节角 Joints_A 和 B 点位姿 Pose_B，机械臂在两点之间进行往复运动。

解决方案：向目标点 A 运动时，调用 moveJToTarget 或 moveLToTarget 函数。

```
const char* strIpAddress = "192.168.10.75";
double Joints_A[7] = {0.000000, 0.523599, 0.000000, 1.570796, 0.000000, -0.872665, 0.000000}; // A 点关节角
double pose_B[6] = {0.5, 0.5, 0.5, 0, 0, 0};    // B 点位姿
double velL = 0.2, accL = 0.8;  // 直线运动的速度与加速度
moveJToTarget(Joints_A, velL, accL,strIpAddress);
wait_move(strIpAddress);
int count = 10;
for (int i = 0; i < count; i++)
{
    // 调用 moveJToTarget 或 moveLToTarget 函数移动至目标点 A
    moveLToTarget(Joints_A, velL, accL, strIpAddress);
    wait_move(strIpAddress);
    // 调用 moveJToPose 或 moveLToPose 函数移动至目标点 B
    moveLToPose(pose_B, velL, accL, nullptr, strIpAddress);
    wait_move(strIpAddress);
}
```

第二种情况: 所有路点对应的关节角均未知。

例：已知 A 点位姿 Pose_A 和 B 点位姿 Pose_B，机械臂在两点之间进行往复运动。

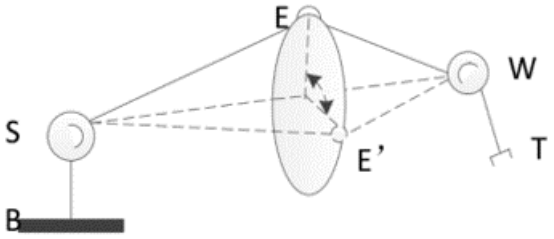
解决方案：首先在 A 点和 B 点附近分别示教一个参考点，并记录下参考点位下的机械臂关节角 q_ref_A 和 q_ref_B，然后利用 inverse_ext 函数，求解目标位姿下相对于参考点关节角距离最近的逆解，最后调用 moveJToTarget 或 moveLToTarget 函

数进行运动。

```
const char* strIpAddress = "192.168.10.75";
double Pose_A[6] = {0.5, 0.5, 0.5, 0, 0, 0};
double Pose_B[6] = {0.4, 0.6, 0.2, 0, 0, 0};
// 示教两个参考点位并记录下关节角 q_ref_A, q_ref_B
double q_ref_A [7] = {-0.645772, 0.261799, -0.157080, 1.675516, 0.052360, -1.186824, -0.802851};
double q_ref_B[7] = {-0.645772, 0.261799, -0.157080, 1.675516, 0.052360, -1.186824, -0.802851};
double velJ = 0.25, accJ = 1.0;      // 关节空间运动的速度与加速度
double velL = 0.1, accL = 0.4;       // 直线运动的速度与加速度
int count = 10;
for ( int i = 0; i < count; i++)
{
    // 调用 inverse_ext 函数, 根据 q_ref_A 计算 Pose_A 所对应的关节角 Joints_A
    double Joints_A [7] = {0.0};
    inverse_ext(q_ref_A, Pose_A, Joints_A);
    // 调用 moveJToTarget 或 moveLToTarget 函数移动到目标点 A
    moveJToTarget(Joints_A, velJ, accJ);
    wait_move(strIpAddress);
    // 调用 inverse_ext 函数, 根据 q_ref_B 计算 Pose_B 所对应的关节角 Joints_B
    double Joints_B [7] = {0.0};
    inverse_ext(q_ref_B, Pose_B, Joints_B);
    // 调用 moveJToTarget 或 moveLToTarget 函数移动到目标点 B
    moveLToTarget(Joints_B, velL, accL);
    wait_move(strIpAddress);
}
```

附录 D：

关于臂角



S，E，W 三点构成平面。当此平面与机械臂安装面垂直且 E 在上方时，臂角为 0 度，如图 1；

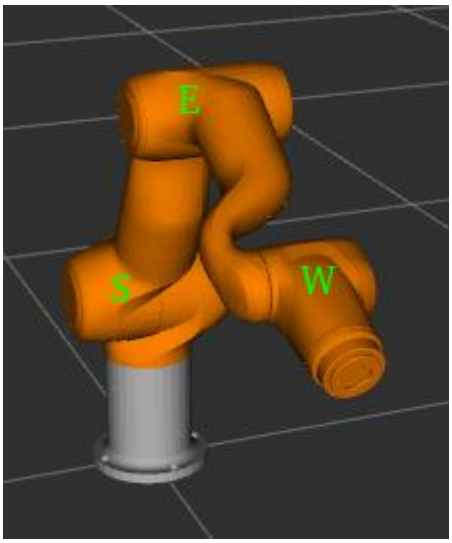


图 1

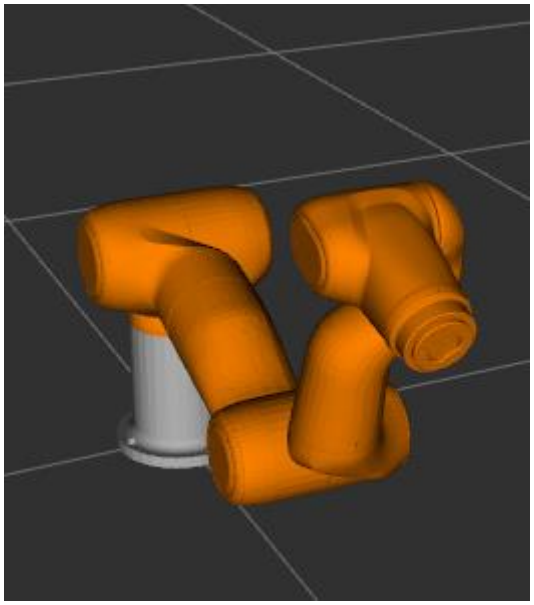


图 2

如图 2，E 在下方，此时臂角为±180 度

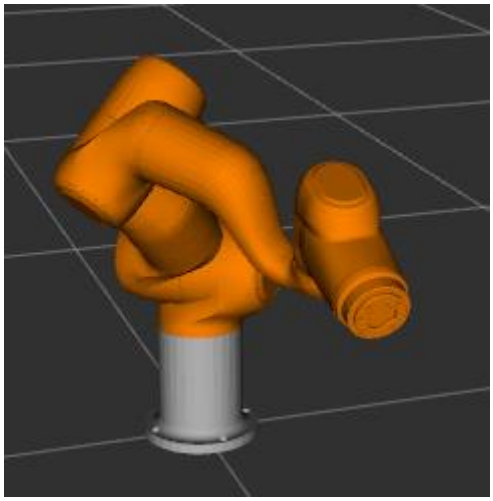


图 3

如图 3，臂角为正 30 度。臂角为±30 度时，添加一句话：此时 S, E, W 三点构成的平面与机械臂安装面夹角为 150 度。

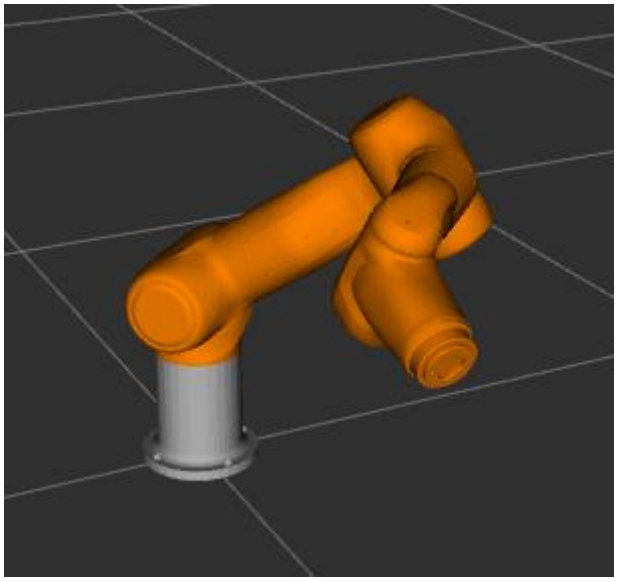


图 4

如图 4，臂角为负 30 度。臂角为±30 度时，添加一句话：此时 S, E, W 三点构成的平面与机械臂安装面夹角为 150 度。