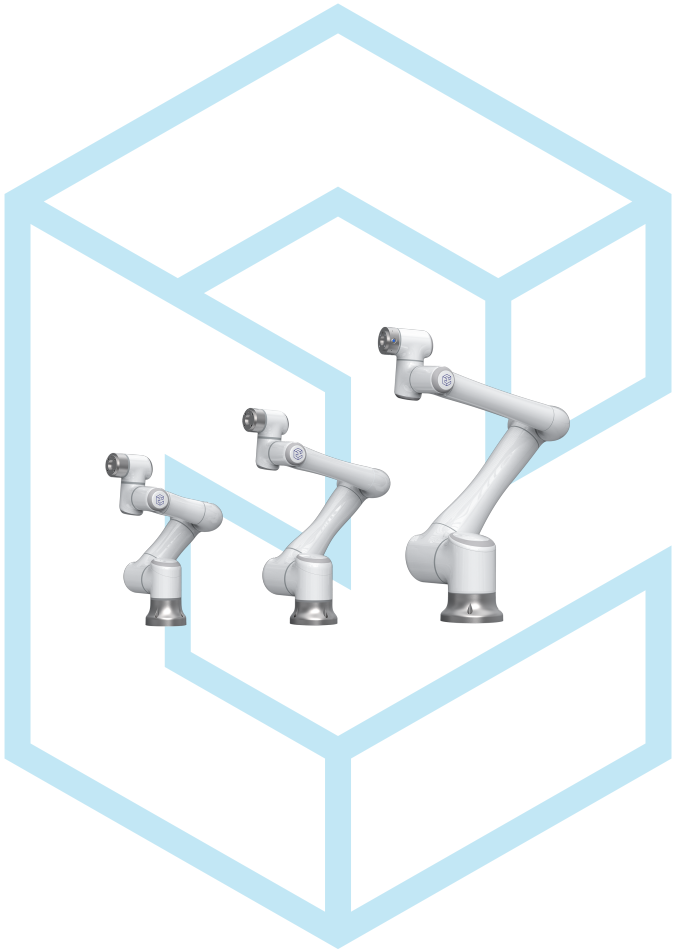
ELITE ROBOTS
艾利特机器人

# ELITE ROBOTS EC Series Programming Manual



## SDK-Socket Script Manual

Suzhou Elite Robot Co., Ltd
2023-01-09
Version: Ver3.8.2

# Contents

# Chapter 1 Introduction

**CAUTION**

The applicable software version of this manual is: Ver3.8.2.

Elite Robot has opened robot controller ports to support users in secondary development, as shown in **Table** 1-1 .

**Table 1-1 .** Controller port

| Port Number | Name | Function |
|---|---|---|
| 8055 | Control Interface | receive json string in specified format |
| 8056 | Monitor Interface | output robot information |

The user can connect to the corresponding controller port through socket communication to perform some operations to realize the corresponding function.

# Chapter 2 Control Interface

The user can send a json string in the specified format to the control interface of the controller through socket communication to achieve related functions, as shown below.

```
send

{"jsonrpc":"2.0","method":"method name","params":parameters,"id":id}

receive

  Normal
    {"jsonrpc":"2.0","result":"result","id":id}
  Error
    {"jsonrpc":"2.0","error":{"code":error code,"message":"error
       message"},"id":id}
```

**CAUTION**

This function is applicable to 2.13.0 and above versions.

The id when sending the json string is the same as the id when receiving the result, as shown below.

```
Send
    {"jsonrpc":"2.0","method":"cmd_set_payload","params":{"cog"
       :[1,2,3],"tool_num":1,"m":12},"id":1}

    {"jsonrpc":"2.0","method":"checkJbiExist","params":{"filename":"
       123123"},"id":1}

    {"jsonrpc":"2.0","method":"getRobotState","params":[],"id":1}


  Receive
    Normal
      {"jsonrpc":"2.0","result":"false","id":1}

    Error
      {"jsonrpc":"2.0","error":{"code":-32601,"message":"Method not
         found."},"id":1}
```

**NOTICE**

Both sending and receiving end with \n

Currently, there are two common exceptions returned by the json protocol:

JRPC_METHOD_NOT_FOUND -32601, JRPC_INTERNAL_ERROR -32693.

- 32601 means that the corresponding interface is not found. You need to check whether the interface name is correct or whether the current version supports the interface.
- 32693 is an exception defined inside the interface. Corresponding parameters are not found, parameters are out of range, and execution conditions are not met. Such exceptions are reported. Such errors only need to check whether the parameters and their ranges and execution conditions are satisfied according to the error information.

## 2.1 Python data processing example

The examples in this chapter are all in Python. Users can modify the code according to the examples in this section.

**CAUTION**

The Python language version must be Python3.

```
1  import socket
2  import json
3  import time
4
5  def connectETController(ip,port=8055):
6    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
7    try:
8      sock.connect((ip,port))
9      return (True,sock)
10   except Exception as e:
11     sock.close()
12     return (False,)
13
14 def disconnectETController(sock):
15   if(sock):
```

```
    sock.close()
    sock=None
  else:
    sock=None

def sendCMD(sock,cmd,params=None,id=1):
  if(not params):
    params=[]
  else:
    params=json.dumps(params)
  sendStr="{{\"method\":\"{0}\",\"params\":{1},\"jsonrpc\":\"2.0\",\"id
    \":{2}}}".format(cmd,params,id)+"\n"
  try:
    sock.sendall(bytes(sendStr,"utf-8"))
    ret=sock.recv(1024)
    jdata=json.loads(str(ret,"utf-8"))
    if("result" in jdata.keys()):
      return (True,json.loads(jdata["result"]),jdata["id"])
    elif("error" in jdata.keys()):
      return (False,jdata["error"],jdata["id"])
    else:
      return (False,None,None)
  except Exception as e:
    return (False,None,None)

if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc):
    # Get robot status
    suc, result, id = sendCMD(sock, "getRobotState")
    # Print results
    print(result)
```

# 2.2 Interface service

## 2.2.1 Servo Service(ServoService)

### 2.2.1.1 Get the servo status of the robotic arm

```
{"jsonrpc":"2.0","method":"getServoStatus","id":id}
```

Function: Get the servo status of the robotic arm

Parameter: None

Return: Enable true, not enable false

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc):
    # Get the servo status of the robotic arm
    suc, result ,id=sendCMD(sock,""getServoStatus)
```

### 2.2.1.2 Set servo status of robotic arm

```
{"jsonrpc":"2.0","method":"set_servo_status","params":{"status":status},"
   id":id}
```

Function: Set servo enable state

Parameter: status: servo switch, range: int[0,1], 1 is on, 0 is off

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc):
    # Get the servo status of the robotic arm
    suc, result ,id=sendCMD(sock, "getServoStatus")
    if( result == 0):
      # Set the servo status of the robot arm to ON
      suc, result ,id=sendCMD(sock,"set_servo_status",{"status":1})
      time.sleep(1)
```

Note: This command is only supported in remote mode.

### 2.2.1.3 Synchronous Servo Encoder Data

```
{"jsonrpc":"2.0","method":"syncMotorStatus","id":id}
```

Function: Synchronize servo encoder data

Parameter: None

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        # Get synchronization status
        suc, result, id = sendCMD(sock, "getMotorStatus")
        if(result == 0):
            # Synchronize servo encoder data
            suc, result, id = sendCMD(sock, "syncMotorStatus")
            time.sleep(0.5)
```

Note: This command is only supported in remote mode.

### 2.2.1.4 Clear alarm

```
{"jsonrpc":"2.0","method":"clearAlarm","id":id}
```

Function: Clear alarm

Parameter: None

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        # Clear alarm
        ret, result, id = sendCMD(sock, "clearAlarm")
```

Note: This command is only supported in remote mode.

#### 2.2.1.5 Get synchronization status

```
{"jsonrpc":"2.0","method":"getMotorStatus","id":id}
```

Function: Get synchronization status

Parameter: None

Return: True for synchronization, false for unsynchronization

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc):
    # Get synchronization status
    suc, result , id = sendCMD(sock, "getMotorStatus")
```

### 2.2.2 Parameter Service(ParamService)

#### 2.2.2.1 Get robot status

```
{"jsonrpc":"2.0","method":"getRobotState","id":id}
```

Function: Get robot status

Parameter: None

Return: Stop state 0, pause state 1, emergency stop state 2, running state 3, alarm state 4, collision state 5

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc):
    # Get robot status
    suc, result , id = sendCMD(sock, "getRobotState")
```

Note: The emergency stop status obtained by this instruction will only exist for a short time and will be covered by the alarm soon. If you need to get the emergency stop status, please refer to Subsection 2.2.2.58.

#### 2.2.2.2 Get Robot Mode

```
{"jsonrpc":"2.0","method":"getRobotMode","id":id}
```

Function: Get Robot Mode

Parameter: None

Return: Teaching mode 0, operating mode 1, remote mode 2

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        # Get Robot Mode
        suc, result , id = sendCMD(sock, "getRobotMode")
```

#### 2.2.2.3 Obtain the joint position information of the robot output terminal

```
{"jsonrpc":"2.0","method":"get_joint_pos","id":id}
```

Function: Obtain the joint position information of the robot output terminal

Parameter: None

Return: The current position information of the robot double pos[6]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        # Get information about the joint position of the robot output terminal
        suc, result , id = sendCMD(sock, "get_joint_pos")
```

Note: getRobotPos is deprecated, please use get_joint_pos to obtain the joint position information of the robot output terminal

#### 2.2.2.4 Get the current pose information of the robot

```
{"jsonrpc":"2.0","method":"get_tcp_pose","params":{"coordinate_num":
   coordinate_num,"tool_num":tool_num,"unit_type": unit_type},"id":id}
```

Function: Get the current pose information of the robot

Parameter: coordinate_num: coordinate number; int[-1,7], -1: base coordinate system, 0~7: corresponding user coordinate system
tool_num: tool number: int[-1,7], -1: current tool number, 0~7: corresponding tool number
unit_type: int[0,1], optional parameter, returns the unit type of pose rx, ry, rz, 0: angle, 1: radians, if not written, the default value is radians.

Return: The current pose information of the robot double pose[6]

Example:
```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc):
    # Get the current pose information of the robot
    suc, result ,id=sendCMD(sock,"get_tcp_pose",{"coordinate_num": 0,"tool_num": 0})
```

Note: If the parameter coordinate_num and the parameter tool_num are used at the same time, the corresponding tool number and the user coordinate in the corresponding user coordinate system are returned; if neither or only the parameter tool_num is used, the robot pose in the base coordinate system is returned. The parameter coordinate_num cannot be used alone.
getRobotPose and getTcpPose are deprecated, please use get_tcp_pose to get the current robot pose.

### 2.2.2.5 Get robot motor speed

```
{"jsonrpc":"2.0","method":"get_motor_speed","id":id}
```

Function: Get robot motor speed

Parameter: None

Return: Robot motor speed double speed[6]

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController( robot_ip )
  point = []
  point.append([0.0065,-103.9938,102.2076,-88.2138,
  90.0000,0.0013)
  point.append([-16.2806,-82.4996,81.9848,-89.4851,
  90.0000,-16.2858])
  point.append([3.7679, -71.7544, 68.7276, -86.9732,
  90.0000, 3.7627])
  point.append([12.8237,-87.3028,87.2361,-89.9333,
  90.0000,12.8185])
  if(conSuc):
    for i in range(0, 4, 1):
      # Joint movement
      suc, result ,id=sendCMD(sock,"moveByJoint",{"targetPos":point[i ], "speed":30})
      while(True):
        # Get robot motor speed
        suc, result ,id =sendCMD(sock,"get_motor_speed")
        print ( result )
        # Get robot status
        suc, result ,id=sendCMD(sock,"getRobotState")
        if ( result == 0):
          break
```

Note: getMotorSpeed is deprecated, please use get_motor_speed to obtain the motor speed of the robot.

### 2.2.2.6 Get the current coordinate system of the robot

```
{"jsonrpc":"2.0","method":"getCurrentCoord","id":id}
```

Function: Get the current coordinate system of the robot

Parameter: None

Return: Joint 0, Base 1, Tool 2, User 3, Cylinder 4

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        # Get the current coordinate system of the robot
        suc, result, id = sendCMD(sock, "getCurrentCoord")
```

### 2.2.2.7 Get robot cycle mode

```
{"jsonrpc":"2.0","method":"getCycleMode","id":id}
```

Function: Get robot cycle mode

Parameter: None

Return: Single step 0, single loop 1, continuous loop 2

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        # Get robot cycle mode
        suc, result, id = sendCMD(sock, "getCycleMode")
```

### 2.2.2.8 Get the current job running line number of the robot

```
{"jsonrpc":"2.0","method":"getCurrentJobLine","id":id}
```

Function: Get the current job running line number of the robot

Parameter: None

Return: JBI line number

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        # Get the servo status of the robotic arm
        suc, result ,id=sendCMD(sock, "getServoStatus")
        if( result == 0):
            # Set the servo status of the robot arm to ON
            suc, result ,id=sendCMD(sock,"set_servo_status",{"status":1})
            time.sleep(1)
        # Check if the jbi file exists
        suc, result ,id=sendCMD(sock,"checkJbiExist",{"filename": jbi_filename })
        if(suc and result ==1):
            # Run jbi file
            suc, result ,id=sendCMD(sock,"runJbi",{"filename": jbi_filename })
            if(suc and result ):
                checkRunning=3
                while(checkRunning==3):
                    # Get jbi file running status
                    suc, result ,id=sendCMD(sock,"getJbiState")
                    checkRunning=result["runState"]
                    # Get the current job running line number of the robot
                    # The line number needs to count the line number of the point information, not
                        the line number of the teach pendant program
                    suc, result ,id=sendCMD(sock,"getCurrentJobLine")
                    print( result )
                    time.sleep(0.1)
```

### 2.2.2.9 Get the current encoder value list of the robot

```
{"jsonrpc":"2.0","method":"getCurrentEncode","id":id}
```

Function: Get the current encoder value list of the robot

Parameter:

Return: The current encoder value list of the robot double encode[6]

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if (conSuc):
    # Get the current encoder value list of the robot
    suc, result , id = sendCMD(sock, "getCurrentEncode")
```

### 2.2.2.10 Get the current tool number of the robot in the teaching mode

```
{"jsonrpc":"2.0","method":"getToolNumber","id":id}
```

Function: Get the current tool number of the robot in the teaching mode

Parameter: None

Return: The current tool number of the robot, range: int[0,7]

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if (conSuc):
    # Get the current tool number of the robot in the teaching mode
    suc, result , id = sendCMD(sock, "getToolNumber")
```

### 2.2.2.11 Switch the current tool number of the robot in the teaching mode

```
{"jsonrpc":"2.0","method":"setToolNumber","params":{"tool_num":tool_num
  },"id":id}
```

Function: Switch the current tool number in the teaching mode

Parameter: tool_num: Tool number, range: int[0,7]

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Switch the current tool number of the robot in the teaching mode
        suc, result ,id=sendCMD(sock,"setToolNumber",{"tool_num":7})
        time.sleep (0.5)
        # Get the current tool number of the robot in the teaching mode
        suc, result , id = sendCMD(sock, "getToolNumber")
```

Note: This command can only switch the current tool number in teaching mode.

This command is only supported in remote mode.

### 2.2.2.12 Get the current user coordinate number of the robot

```
{"jsonrpc":"2.0","method":"getUserNumber","id":id}
```

Function: Get the current user coordinate numberr of the robot

Parameter: None

Return: Coordinate number of the current user of the robot, range: int[0,7]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Get the current user coordinate number of the robot
        suc, result , id = sendCMD(sock, "getUserNumber")
```

### 2.2.2.13 Switch the current user coordinate number of the robot

```
{"jsonrpc":"2.0","method":"setUserNumber","params":{"user_num":user_num},
   id":id}
```

Function: Switch the current user coordinate number of the robot

Parameter: user_num: user coordinate number, range: int[0,7]

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc)::
    # Switch the current user coordinate number of the robot
    suc, result ,id=sendCMD(sock,"setUserNumber",{"user_num":7})
    time.sleep(0.5)
    # Get the current user coordinate number of the robot
    suc, result , id = sendCMD(sock, "getUserNumber")
```

Note: This command is only supported in remote mode.

### 2.2.2.14 Get the current torque information of the robot

```
{"jsonrpc":"2.0","method":"get_motor_torque","id":id}
```

Function: Get the current torque information of the robot

Parameter: None

Return: Robot current torque information double torques[6], The ratio per thousand of the ratedtorque of the joint, unit ‰.

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc):
    # Get the current torque information of the robot
    suc, result , id = sendCMD(sock, "get_motor_torque")
```

Note: getRobotTorques is deprecated, please use get_motor_torque to get the current torque information of the robot

### 2.2.2.15 Get the current running point number of the trajectory

```
{"jsonrpc":"2.0","method":"getPathPointIndex","id":id}
```

Function: Get the current running point number of the trajectory

Parameter: None

Return: Store the serial number of the current running point, -1 is non-waypoint movement

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController( robot_ip )
  C000 = [0.0065,-103.9938,102.2076,-88.2138,
   90.0000,0.0013]
  C001 = [-16.2806,-82.4996,81.9848,-89.4851,
  90.0000, -16.2858]
  C002 = [3.7679, -71.7544, 68.7276, -86.9732,
  90.0000, 3.7627]
  if (conSuc):
    # Clear waypoint information 2.0
    suc, result , id = sendCMD(sock, "clearPathPoint")
    if ( result == True):
      # Add waypoint information 2.0
      suc, result , id = sendCMD(sock, "addPathPoint", {"wayPoint": C000,"moveType": 0, "
          speed": 50, "circular_radius":20})
      suc, result , id = sendCMD(sock, "addPathPoint", {"wayPoint": C001,"moveType":0, "
          speed": 50, "circular_radius":20})
      suc, result , id = sendCMD(sock, "addPathPoint", {"wayPoint": C002,"moveType": 0, "
          speed": 50, "circular_radius":0})
      # Trajectory movement 2.0
      suc, result , id = sendCMD(sock, "moveByPath")
      while(True):
        # Get the current running point number of the trajectory
        suc, result , id = sendCMD(sock, "getPathPointIndex")
        print ( result )
        # Get robot status
        suc, result , id = sendCMD(sock, "getRobotState")
        if ( result == 0):
          break
```

### 2.2.2.16 Specify coordinate system

```
{"jsonrpc":"2.0","method":"setCurrentCoord","params":{"coord_mode":
   coord_mode},"id":id}
```

Function: Specify coordinate system

Parameter: coord_mode: coordinate system,range int[0,4]. joint: 0, right angle: 1, tool: 2, user: 3, cylinder: 4

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc):
    for i in range(0, 5, 1)
      # Specify coordinate system
      suc, result ,id=sendCMD(sock,"setCurrentCoord",{"coord_mode":i})
      time.sleep(0.5)
      # Specify coordinate system
      suc, result ,id=sendCMD(sock,"getCurrentCoord")
      print(result)
```

Note: This command is only supported in remote mode.

### 2.2.2.17 Drag the teaching switch

```
{"jsonrpc":"2.0","method":"drag_teach_switch","params":{"switch":switch
   },"id":id}
```

Function: Drag the teaching switch

Parameter: switch: switch, range: int[0,1],0 is off, 1 is on

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc):
    # Drag the teaching switch
    suc, result ,id=sendCMD(sock,"drag_teach_switch",{"switch":1})
```

Note: This command is only supported in remote mode.

### 2.2.2.18 Get the drag-enabling state of the robot

```
{"jsonrpc":"2.0","method":"get_drag_state","id":id}
```

Function: Get the drag-enabling state of the robot

Parameter: None

Return: True means that the drag button is enabled, false means that the button is disenabled

Example:

```
if __name__ == "__main__":
```

```
# Robot IP address
robot_ip="192.168.1.200"
conSuc,sock=connectETController(robot_ip)
if (conSuc):
  # Get the drag-enabling state of the robot
  suc, result ,id=sendCMD(sock,"get_drag_state")
```

### 2.2.2.19 Set robot arm load and center of gravity

```
{"jsonrpc":"2.0","method":"cmd_set_payload","params":{"tool_num":tool_num
  ,"m":m,"point":point},"id":id} or {"jsonrpc":"2.0","method":"
  cmd_set_payload","params":{"tool_num":tool_num,"m":m,"cog":cog},"id":
  id}
```

Function: Set robot arm load and center of gravity

Parameter: tool_num: Tool number, range: int[0,7]
m: load weight, unit Kg, double, range: EC63 [0,3.6] EC66 [0,7.2] EC612 [0,14.4]
point or cog: center of gravity, x, y, z, unit mm, range: double[-5000,5000]

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if (conSuc):
    # Set robot arm load and center of gravity
    suc, result , id = sendCMD(sock, "cmd_set_payload",{"tool_num":0, "m":5, "cog":[10, 20,
        30]})
```

Note: Version 2.16.2 and above support the selection of tool number. The value of tool_num is set as the load and center of gravity of the tool.
The parameter point is deprecated.
This command is only supported in remote mode.

### 2.2.2.20 Set Robotic Arm Tool Center

```
{"jsonrpc":"2.0","method":"cmd_set_tcp","params":{"point":point,"tool_num
  ":tool_num,"unit_type":unit_type},"id":id}
```

Function: Set Robotic Arm Tool Center

Parameter: tool_num: tool number, range: int[0,7]
point: tool center, the first three items are in millimeters, the range: double[-500,500], the last three units: radian, range: double[-π,π] or angle, range: double[-180,180]
unit_type: optional parameter, int[0,1], sets the unit type of rx, ry, rz in the tool center, 0: angle, 1: radian, if not written, the default value is radians.

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip = "192.168.1.202"
  conSuc, sock = connectETController(robot_ip)
  if (conSuc):
    # Set up the robotic arm tool center
    suc, result , id = sendCMD(sock, "cmd_set_tcp", {"point": [10, 0, 0, 30, 0, 0], "
        tool_num": 1, "unit_type":0})
    print(suc, result , id)
  else:
    print("Connection failed")
  disconnectETController(sock)
```

Note: This command is only supported in remote mode.

### 2.2.2.21 Get collision status

```
{"jsonrpc":"2.0","method":"getCollisionState","id":id}
```

Function: Get collision status

Parameter: None

Return: Collision: 1, No collision: 0

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc):
    # Get collision status
    suc, result , id = sendCMD(sock, "getCollisionState )
```

### 2.2.2.22 Get user coordinate system data

```
{"jsonrpc":"2.0","method":"getUserFrame","params":{"user_num": user_num,"
   unit_type":unit_type},"id":id}
```

Function: Get user coordinate system data

Parameter: user_num: User coordinate number, range: int[0,7]
unit_type: int[0,1], optional parameter, returns the unit type of pose rx, ry, rz, range int [0,1]
Default: radians, 0: angle, 1: radians

Return: User coordinate system data double pose[6]

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc):
    for i in range(8):
      suc, result ,id=sendCMD(sock,"getUserFrame",{"user_num":i,"unit_type":1})
      print("User coordinate number = ",i)
      print("suc=",suc,"","id =",id)
      if (suc):
        print(" result =", result )
      else:
        print("err_msg=", result ["message"])
```

Note: The unit_type parameter is only applicable to v2.15.2 and above.

### 2.2.2.23 Specify cycle mode

```
{"jsonrpc":"2.0","method":"setCycleMode","params":{"cycle_mode":
   cycle_mode},"id":id}
```

Function: Specify cycle mode

Parameter: cycle_mode: cycle mode, range: int[0,2] single step: 0, single cycle: 1, continuous cycle: 2

Return: Success true, failure false

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController( robot_ip )
  if (conSuc):
    #Set the loop mode to single loop
    ret , result , id = sendCMD(sock, "setCycleMode",{"cycle_mode":1})
    if ret :
      print (" result =", result )
    else :
      print ("err_msg=", result ["message"])
```

Note: This command is only supported in remote mode.

### 2.2.2.24 Set user coordinate system data

```
{"jsonrpc":"2.0","method":"setUserFrame","params":{"user_num":user_num,"
   user_frame":user_frame,"unit_type":unit_type},"id":id}
```

Function: Set user coordinate system data

Parameter: user_num: user number, range int [0,7]
user_frame: user coordinate system data, double user_frame[6], range: [-1e+9,1e+9], x, y, z unit: millimeter, rx, ry, rz unit: degree/rad
unit_type: the unit type of rx, ry, rz of the user coordinate system, int[0,1], optional parameters, the unit type of rx, ry, rz, 0: angle, 1: radians, if not written, the default is radians value.

Return: True for success, false for failure

Example:
```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController( robot_ip )
  if (conSuc):
    user_frame=[499.011212,570.517817,
    247.082805, -3.141593, -0.000000, -0.773067]
    ret , result , id=sendCMD(sock,"setUserFrame",{"user_num":0,"user_frame":user_frame,"
        unit_type":1})
    if ret :
      print (" result =", result )
    else :
      print ("err_msg=", result ["message"])
```

Note: This command is only supported in remote mode.
The unit_type parameter is only applicable to v2.15.2 and above.

### 2.2.2.25 Get tool coordinate system data

```
{"jsonrpc":"2.0","method":"getTcpPos","params":{"tool_num":tool_num,"
   unit_type":unit_type},"id":id}
```

Function: Get tool coordinate system data

Parameter: tool_num: tool coordinate number, range int [0,7]
unit_type: int[0,1], return the unit type of rx, ry, rz of the coordinate system, 0:angle, 1:radian, optional parameter, if not written, the default value is radians.

Return: Tool coordinate system data double pose[6]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        ret, result, id = sendCMD(sock, "getTcpPos", {"tool_num": 0})
        if ret:
            print ("result =", result)
        else:
            print ("err_msg=", result["message"])
```

### 2.2.2.26 Get tool load quality

```
{"jsonrpc":"2.0","method":"getPayload","params":{"tool_num":tool_num},"id
    ":id}
```

Function: Get tool load quality

Parameter: tool_num: tool coordinate number, range int [0,7]

Return: Tool load quality, double m

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        ret, result, id = sendCMD(sock, "getPayload", {"tool_num": 0})
        if ret:
            print ("result =", result)
        else:
            print ("err_msg=", result["message"])
```

Note: This command is deprecated.

### 2.2.2.27 Get tool cog

```
{"jsonrpc":"2.0","method":"getCentreMass","params":{"tool_num":tool_num
    },"id":id}
```

Function: Get tool centroid

Parameter: tool_num: tool coordinate number, range int [0,7]

Return: Tool load cog double cog

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController( robot_ip )
  if (conSuc):
    ret , result , id = sendCMD(sock, "getCentreMass", {"tool_num": 0})
    if ret :
      print (" result =", result )
    else :
      print ("err_msg=", result ["message"])
```

Note: This command is deprecated.

### 2.2.2.28 Get robot type

```
{"jsonrpc":"2.0","method":"getRobotType","id":id}
```

Function: Get robot type

Parameter: None

Return: Robot type int 62 (six-axis collaborative robot), 60 (vertical multi-joint series robot), 41 (four-axis rotary joint robot), 40 (palletizing robot), 43 (SCARA robot), 30 (Delta parallel robot)

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController( robot_ip )
  if (conSuc):
    ret , result , id = sendCMD(sock, "getRobotType")
    if ret :
      print (" result =", result )
    else :
      print ("err_msg=", result ["message"])
```

### 2.2.2.29 Get robot DH parameters

```
{"jsonrpc":"2.0","method":"getDH","params":{"index":index},"id":id}
```

Function: Get robot DH parameters

Parameter: index: range int [0,11], corresponding to connecting rod parameter d1~d12

Return: DH parameters

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if (conSuc):
    # Get the value of connecting rod parameter d1
    ret, result, id = sendCMD(sock, "getDH", {"index":0})
    if ret:
      print (" result =", result)
    else:
      print ("err_msg=", result ["message"])
```

### 2.2.2.30 Set collision enable

```
{"jsonrpc":"2.0","method":"setCollisionEnable","params":{"enable":
    enable},"id":id}
```

Function: Set collision enable

Parameter: enable:int[0,1], 1: turn on the collision switch, 0: turn off the collision switch

Return: True for success, False for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if (conSuc):
    # Turn on the collision switch
    ret, result, id = sendCMD(sock, "setCollisionEnable", {"enable": 1})
    if ret:
      print (" result =", result)
    else:
      print ("err_msg=", result ["message"])
```

Note: This command is deprecated.

### 2.2.2.31 Set collision sensitivity

```
{"jsonrpc":"2.0","method":"setCollisionSensitivity","params":{"value":
   value}, "id":id}
```

Function: Set collision sensitivity

Parameter: value: sensitivity range int [10,100]

Return: True for success, False for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Set collision sensitivity to 50%
        ret, result, id = sendCMD(sock, "setCollisionSensitivity", {"value": 50})
        if ret:
            print ("result =", result)
        else:
            print ("err_msg=", result["message"])
```

Note: This command is only supported in remote mode. This command is deprecated.

### 2.2.2.32 Get the automatically generated encrypted string

```
{"jsonrpc":"2.0","method":"get_remote_sys_password","id":id}
```

Function: Get the automatically generated encrypted string

Parameter: None

Return: Automatically generated encrypted string

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.200"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        ret, str, id = sendCMD(sock, "get_remote_sys_password")
        print (str)
    else:
        print ("Connection failed")
    disconnectETController(sock)
```

### 2.2.2.33 Set security parameters

```
{"jsonrpc":"2.0","method":"setSafetyParams","params":{"password":password
    ,"enable":enable,"mode":mode,"power":power,"momentum": momentum,"
    tool_force":tool_force,"elbow_force": elbow_force,"speed":speed,"
    collision_enable":collision_enable,"collision_sensitivity":
    collision_sensitivity},"id":id}
```

Function: Set security parameters

Parameter: password: When the remote mode password is not set on the interface: the default password is "123456"; after the remote mode user password is set on the interface: firstly, you need to use "get_remote_sys_password" to obtain the encrypted string generated by the system, and the remote The user password combination of the mode, calculate its MD5 value as the password required for json security parameter setting
enable: safety restriction parameter enable, int[0,1], 1: enable, 0: disable
mode: mode, int[0,1], 0: normal mode, 1: reduced mode
power: power, range: double [80,1500], unit:W
momentum: momentum, range: double [5,90], unit:kg·m/s
tool_force: Tool force, range: double [100,400], unit:N
elbow_force: elbow force: double [100,400], unit:N
speed: speed percentage, double [0-100], unit:%
collision_enable: int[0,1], optional parameter, set the collision detection switch, 0: turn off the collision detection switch, 1: turn on the collision detection switch
collision_sensitivity: optional parameter, set collision detection sensitivity, range int[10,100], unit:%

Return: True for success, False for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        ret , str1 , id = sendCMD(sock, "get_remote_sys_password")
        word = hashlib.md5()
        str2 = "123456"
        word.update(str1.encode("utf8"))
        word.update(str2.encode("utf8"))
        password = word.hexdigest()
        print(password)
        ret , result , id = sendCMD(sock, "setSafetyParams", {"password": password, "enable": 1,
                "mode": 1, "power": 400, "momentum": 90, "tool_force": 400 , "elbow_force": 400,
            "speed": 0.5, "collision_enable": 0, "collision_sensitivity": 10})
        print( result )
    else:
        print("Connection failed")
    disconnectETController(sock)
```

Note: This command is supported in remote and play mode.

### 2.2.2.34 Get robot running speed

```
{"jsonrpc":"2.0","method":"getSpeed","id":id}
```

Function: Get the robot's automatic speed

Parameter: no

Return: Automatic speed double

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        ret , result , id = sendCMD(sock, "getSpeed")
        if ret :
            print (" result =", result )
        else :
            print ("err_msg=", result ["message"])
```

### 2.2.2.35 Clear collision status

```
{"jsonrpc":"2.0","method":"resetCollisionState","id":id}
```

Function: Clear collision status

Parameter: None

Return: True for success, False for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        ret , result , id = sendCMD(sock, " resetCollisionState ")
        if ret :
            print (" result =", result )
        else :
            print ("err_msg=", result ["message"])
```

Note: This command is only supported in remote mode.

### 2.2.2.36 Get the current tool number of the robot in remote mode

```
{"jsonrpc":"2.0","method":"getAutoRunToolNumber","id":id}
```

Function: Get the current tool number of the robot in remote mode

Parameter: None

Return: The current tool number of the robot in remote mode, range: int[0,7]
Note: The tool number in automatic mode is the same as the tool number in remote mode.

Example:

```
if __name__ == "__main__":
    ip = "192.168.1.202"
    conSuc, sock = connectETController(ip)
    # print(conSuc)
    if conSuc:
        ret, result, id = sendCMD(sock, "getAutoRunToolNumber")
        if ret:
            print("result = ", result)
        else:
            print("err_msg = ", result["message"])
```

Note: This command is applicable to v2.14.4 and above.

### 2.2.2.37 Set the current tool number of the robot in remote mode

```
{"jsonrpc":"2.0","method":"setAutoRunToolNumber","params":{"tool_num":
    tool_num},"id":id}
```

Function: Set the current tool number of the robot in remote mode

Parameter: tool_num: tool number, range: int[0,7]

Return: True for success, False for failure

Example:

```
if __name__ == "__main__":
    ip = "192.168.1.202"
    conSuc, sock = connectETController(ip)
    if conSuc:
        ret, result, id=sendCMD(sock,"setAutoRunToolNumber",{"tool_num": 0})
        if ret:
            print("result = ", result)
        else:
            print("err_msg = ", result["message"])
```

Note: This command is only supported in remote mode.
This command is applicable to v2.14.4 and above.

### 2.2.2.38 Get the center pose of flange in Base coordinate system

```
{"jsonrpc":"2.0","method":"get_base_flange_pose", "params":{"unit_type":
   unit_type},"id":id}
```

Function: Get the center pose of the flange in the Base coordinate system

Parameter: unit_type: int[0,1], return the unit type of the pose rx, ry, rz, 0: return angle, 1: return radian, optional parameter, if not written, the default value is radians.

Return: The center pose of the flange in the Base coordinate system, Double[6]

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.202"
  conSuc,sock=connectETController( robot_ip )
  if (conSuc):
    # Get the center pose of the flange in the Base coordinate system
    ret , result , id = sendCMD(sock,"get_base_flange_pose",{"unit_type": 0})
    print (" result =", result )
  else :
    print ("Connection failed ")
  disconnectETController (sock)
```

Note: This command is applicable to v2.14.4 and above.

### 2.2.2.39 Get the center pose of the flange in the user coordinate system

```
{"jsonrpc":"2.0","method":"get_user_flange_pose","params":{"unit_type":
   unit_type},"id":id}
```

Function: Get the center pose of the flange in the user coordinate system

Parameter: unit_type: int[0,1], return the unit type of the pose rx, ry, rz, 0: return angle, 1: return radian, optional parameter, if not written, the default value is radians.

Return: Center pose of flange in user coordinate system, Double[6]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.202"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Get the center pose of the flange in the user coordinate system
        ret , result , id = sendCMD(sock,"get_user_flange_pose",{"unit_type": 0})
        print (" result =", result )
    else :
        print ("Connection failed")
    disconnectETController (sock)
```

Note: This command is applicable to v2.14.4 and above.

### 2.2.2.40 Get robot subtype

```
{"jsonrpc":"2.0","method":"getRobotSubtype","id":id}
```

Function: Get robot subtype

Parameter: None

Return: Robot subtype int

Example:

```
if __name__ == "__main__":
    ip = "192.168.1.202"
    conSuc,sock=connectETController(ip)
    if conSuc:
        # Get robot subtype
        suc, result , id=sendCMD(sock,"getRobotSubtype")
        print ( result )
```

Note: This command is applicable to v2.16.2 and above.

### 2.2.2.41 Get security parameter enable status

```
{"jsonrpc":"2.0","method":"getRobotSafetyParamsEnabled","id":id}
```

Function: Get security parameter enable status

Parameter: None

Return: Close 0, open 1

Example:
```
if __name__ == "__main__":
    ip = "192.168.1.202"
    conSuc, sock = connectETController(ip)
    if conSuc:
        # Get security parameter enable status
        suc, result ,id=sendCMD(sock,"getRobotSafetyParamsEnabled")
        print ( result )
```

Note: This command is applicable to v2.16.2 and above.

### 2.2.2.42 Get safe power

```
{"jsonrpc":"2.0","method":"getRobotSafetyPower","id":id}
```

Function: Get safe power

Parameter: None

Return: The power value in normal mode and reduced mode double

Example:
```
if __name__ == "__main__":
    ip = "192.168.1.202"
    conSuc, sock = connectETController(ip)
    if conSuc:
        # Get robot safety power
        suc, result ,id=sendCMD(sock,"getRobotSafetyPower")
        print ( result )
```

Note: This command is applicable to v2.16.2 and above.

### 2.2.2.43 Get Safe Momentum

```
{"jsonrpc":"2.0","method":"getRobotSafetyMomentum","id":id}
```

Function: Gain safety momentum

Parameter: None

Return: Momentum value double in normal mode and reduced mode

Example:
```
if __name__ == "__main__":
    ip = "192.168.1.202"
    conSuc, sock = connectETController(ip)
```

```
if conSuc:
  # Get robot safety momentum
  suc, result ,id =sendCMD(sock,"getRobotSafetyMomentum")
  print ( result )
```

Note: This command is applicable to v2.16.2 and above.

### 2.2.2.44 Access to safety tools

```
{"jsonrpc":"2.0","method":"getRobotSafetyToolForce","id":id}
```

Function: Acquire the power of safety tools

Parameter: None

Return: Tool force double in normal mode and reduced mode

Example:

```
if __name__ == "__main__":
  ip = "192.168.1.202"
  conSuc, sock = connectETController(ip)
  if conSuc:
    # Acquire robot safety tool power
    suc, result ,id =sendCMD(sock,"getRobotSafetyToolForce")
    print ( result )
```

Note: This command is applicable to v2.16.2 and above.

### 2.2.2.45 Get safe elbow force

```
{"jsonrpc":"2.0","method":"get_tcp_force","params":{"ref_tcp":ref_tcp},"id":id}
```

Function: Get the "external" force and torque of the current TCP coordinate system

Parameter: ref_tcp: reference coordinate system, int[0,1], optional parameter, 0: base coordinate system, 1: tcp coordinate system, If not written, the default is tcp coordinate system.

Return: The "external" force and torque of the current tcp double force[6], the first three are external force and the last three are torque

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="172.16.11.248"
  conSuc,sock=connectETController(robot_ip)
  if (conSuc):
    # Get the force and torque of the current TCP in the base coordinate system
    suc, result , id = sendCMD(sock, "get_tcp_force",{"get_tcp_force": 0})
```

```
    print ( suc, result, id )
  else :
    print ("Connection  failed ")
  disconnectETController (sock)
```

Note: The default direction of force and torque is TCP.

### 2.2.2.46 Get the current joint torque

```
{"jsonrpc": "2.0", "method": "get_joint_torques", "id": id}
```

Function: Get the current joint torque

Parameter: None

Return: The current joint torque double torques[6]

Example:

```
if  __name__ == "__main__":
  # Robot IP  address
  robot_ip  = "172.16.11.248"
  conSuc, sock  = connectETController( robot_ip )
  if  (conSuc):
    # Get the current joint torque of the robot
    suc,  result ,  id  = sendCMD(sock, "get_joint_torques")
    print ( suc, result, id )
  else :
    print ("Connection  failed ")
  disconnectETController (sock)
```

Note: The joint torque is the motor torque minus the "torque required to drive itself", reflecting the "external" torque.

### 2.2.2.47 Get safe elbow

```
{"jsonrpc":"2.0","method":"getRobotSafetyElbowForce","id":id}
```

Function: Obtain safe elbow force

Parameter: None

Return: Elbow force in normal mode and reduced mode double

Example:

```
if  __name__ == "__main__":
  ip  = "192.168.1.202"
  conSuc, sock  = connectETController( ip )
  if  conSuc:
    # Obtain  the  robot  safety  elbow  force
    suc, result , id  =sendCMD(sock,"getRobotSafetyElbowForce")
```

```
    print ( result )
```

Note: This command is applicable to v2.16.2 and above.

### 2.2.2.48 Get speed percentage

```
{"jsonrpc":"2.0","method":"getRobotSpeedPercentage","id":id}
```

Function: Get the robot's speed percentage

Parameter: None

Return: Speed percentage in normal mode and reduced mode double

Example:

```
if __name__ == "__main__":
    ip = "192.168.1.202"
    conSuc, sock = connectETController(ip)
    if conSuc:
        # Get robot speed percentage
        suc, result , id =sendCMD(sock,"getRobotSpeedPercentage")
        print ( result )
```

Note: This command is applicable to v2.16.2 and above.

### 2.2.2.49 Get the maximum starting speed of dragging

```
{"jsonrpc":"2.0","method":"getRobotDragStartupMaxSpeed","id":id}
```

Function: Get the maximum start speed of drag

Parameter: None

Return: The maximum starting speed of the robot during dragging double

Example:

```
if __name__ == "__main__":
    ip = "192.168.1.202"
    conSuc, sock = connectETController(ip)
    if conSuc:
        # Get the maximum start speed of the drag
        suc, result , id =sendCMD(sock,"getRobotDragStartupMaxSpeed")
        print ( result )
```

Note: This command is applicable to v2.16.2 and above.

### 2.2.2.50 Get the maximum torque error percentage

```
{"jsonrpc":"2.0","method":"getRobotTorqueErrorMaxPercents","id":id}
```

Function: Get the maximum torque error percentage of the machine

Parameter: None

Return: The maximum torque error percentage in the robot force control parameters double

Example:

```
if __name__ == "__main__":
    ip = "192.168.1.202"
    conSuc, sock = connectETController(ip)
    if conSuc:
        # Get the maximum torque error percentage of the machine
        suc, result , id =sendCMD(sock,"getRobotTorqueErrorMaxPercents")
        print ( result )
```

Note: This command is applicable to v2.16.2 and above.

### 2.2.2.51 Set end button state

```
{"jsonrpc":"2.0","method":"setFlangeButton","params":{"button_num":
    button_num,"state":state},"id":id}
```

Function: Set the end button state

Parameter: button_num: button, int[0,1], 0: blue button, 1: green button
state: state, int[0,2], 0: disable, 1: drag, 2: mark

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    ip = "192.168.1.202"
    conSuc, sock = connectETController(ip)
    if conSuc:
        # Set the end button state
        suc, result , id =sendCMD(sock,"setFlangeButton",{"button_num":0,"state":1})
        print ( result )
```

Note: This command is only supported in remote mode.
This command is applicable to v2.16.2 and above.

### 2.2.2.52 Get end button status

```
{"jsonrpc":"2.0","method":"checkFlangeButton","params":{"button_num":
    button_num},"id":id}
```

Function: Get the end button state

Parameter: button_num: button, int[0,1], 0: blue button, 1: green button

Return: Disable 0, drag 1, mark 2

Example:

```
if __name__ == "__main__":
    ip = "192.168.1.202"
    conSuc, sock = connectETController(ip)
    if conSuc:
        # Get the status of the end button
        suc, result ,id=sendCMD(sock,"checkFlangeButton",{"button_num":0})
        print (suc, result , id)
```

Note: This command is applicable to v2.16.2 and above.

### 2.2.2.53 Get collision detection enable state

```
{"jsonrpc":"2.0","method":"get_collision_enable_status","id":id}
```

Function: Get the status of collision detection

Parameter: None

Return: 0: not enabled, 1: enabled

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.202"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        # Get collision detection enable state
        suc, result , id = sendCMD(sock, "get_collision_enable_status")
        print ( result )
    else :
        print ("Connection failed")
    disconnectETController(sock)
```

### 2.2.2.54 Get collision sensitivity

```
{"jsonrpc": "2.0", "method": "getCollisionSensitivity", "id": id}
```

Function: Get collision sensitivity

Parameter: None

Return: Collision sensitivity int

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
```

```
    # Get collision sensitivity
    suc, result , id = sendCMD(sock, "getCollisionSensitivity")
    print ( result )
else :
    print ("Connection failed ")
disconnectETController (sock)
```

### 2.2.2.55 Get the pose of the current tcp in the current user coordinate system

```
{"jsonrpc":"2.0","method":"getTcpPoseInUser","params":{"unit_type":
    unit_type},"id":id}
```

Function: Get the pose of the current tcp in the current user coordinate system

Parameter: unit_type: int[0,1], optional parameter, returns the unit type of pose rx, ry, rz,0: angle, 1: radians, if not written, the default value is radians.

Return: The current pose of tcp in the user coordinate system

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.202"
    conSuc,sock=connectETController( robot_ip )
    if (conSuc):
        # Get the pose information of the current tcp in the current user coordinate system
        suc, result , id = sendCMD(sock, "getTcpPoseInUser",{"unit_type": 0})
        print ( result )
    else :
        print ("Connection failed ")
    disconnectETController (sock)
```

### 2.2.2.56 Get the serial number of the robot alarm information

```
{"jsonrpc":"2.0","method":"getAlarmNum","id":id}
```

Function: Get the serial number of the robot alarm information

Parameter: None

Return: Returns the serial numbers of the last 5 robot alarm messages successfully, otherwise returns false

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        # Get the abnormality of the robot arm body
        suc, result , id = sendCMD(sock, "getAlarmNum")
        print ( result )
    else :
        print ("Connection failed")
    disconnectETController(sock)
```

### 2.2.2.57 Get joint motion speed

```
{"jsonrpc":"2.0","method":"get_joint_speed","id":id}
```

Function: Get joint speed

Parameter: None

Return: Joint movement speed double speed[6], unit: degree/s

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        while 1:
            suc, result , id = sendCMD(sock, "get_joint_speed")
            print (suc, result , id)
            time.sleep (0.001)
        else :
            print ("Connection failed")
    disconnectETController(sock)
```

### 2.2.2.58 Get tcp acceleration

```
{"jsonrpc":"2.0","method":"get_tcp_acc","id":id}
```

Function: Get tcp acceleration

Parameter: None

Return: tcp motion acceleration double tcp_acc, unit mm/$s^2$

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        while 1:
            suc, result , id = sendCMD(sock, "get_tcp_acc")
            print(suc, result , id)
            time.sleep(0.01)
    else:
        print("Connection failed")
    disconnectETController(sock)
```

### 2.2.2.59 Get joint acceleration

```
{"jsonrpc":"2.0","method":"get_joint_acc","id":id}
```

Function: Get joint acceleration

Parameter: None

Return: Joint motion acceleration double joint_acc[6], unit: degree/$s^2$

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        while 1:
            suc, result , id = sendCMD(sock, "get_joint_acc")
            print(suc, result , id)
            time.sleep(0.01)
    else:
        print("Connection failed")
    disconnectETController(sock)
```

### 2.2.2.60 Get tcp movement speed

```
{"jsonrpc": "2.0", "method": "get_tcp_speed", "id": id}
```

Function: Get tcp movement speed

Parameter: None

Return: Current tcp movement speed double tcp_speed, unit: mm/sec

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip = "192.168.1.202"
  conSuc, sock = connectETController(robot_ip)
  if (conSuc):
    # Get tcp movement speed
    suc, result , id = sendCMD(sock, "get_tcp_speed")
    print ( result )
  else :
    print ("Connection failed")
  disconnectETController(sock)
```

### 2.2.2.61 Get the emergency stop status of the robot

```
{"jsonrpc":"2.0","method":"get_estop_status","id":id}
```

Function: Get the emergency stop status of the robot

Parameter: None

Return: int[0,1], whether the robot is in an emergency stop state, 1: emergency stop, 0: non-emergency stop

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip = "192.168.1.202"
  conSuc, sock = connectETController(robot_ip)
  if (conSuc):
    suc, result , id = sendCMD(sock,"get_estop_status")
    print ( result )
```

### 2.2.2.62 Get tool load and center of mass

```
{"jsonrpc":"2.0","method":"get_tool_payload","params":{"tool_num":
   tool_num},"id":id}
```

Function: Get tool load and centroid

Parameter: tool_num: tool number, range int[0,7]

Return: m: tool load quality, double
tool_cog: tool load centroid, double cog[3]

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.202"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc):
    for i in range(0,8):
      # Get tool load and centroid
      suc, result , id = sendCMD(sock, "get_tool_payload",{"tool_num": i})
      print( result )
    else:
      print("Connection failed")
  disconnectETController(sock)
```

### 2.2.2.63 Get joint position information of robot input

```
{"jsonrpc":"2.0","method":"get_motor_pos","id":id}
```

Function: Obtain the joint position information of the robot input

Parameter: None

Return: double pos[6]: joint position information of the robot input

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="172.16.11.248"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Get the joint position information of the robot input
        suc, result , id = sendCMD(sock, "get_motor_pos")
        print ( result )
    else :
        print ("Connection failed")
    disconnectETController (sock)
```

### 2.2.2.64 Get the precise state of the robot servo encoder

```
{"jsonrpc":"2.0","method":"get_servo_precise_position_status","id":id}
```

Function: Get the precise status of the robot servo encoder

Parameter: None

Return: 1: exact, 0: inexact

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        # Get the precise status of the robot servo encoder
        suc, result , id = sendCMD(sock, " get_servo_precise_position_status ")
        print ( result )
    else :
        print ("Connection failed")
    disconnectETController (sock)
```

### 2.2.2.65 Get the state of the robot servo alarm

```
{"jsonrpc":"2.0","method":"get_servo_alarm_state","id":id}
```

Function: Get the state of the robot servo alarm

Parameter: None

Return: 1: in servo alarm state, 0: not in servo alarm state

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "172.16.11.248"
```

```
conSuc, sock = connectETController(robot_ip)
if (conSuc):
  # Get the state of the robot servo alarm
  suc, result , id = sendCMD(sock, " get_servo_alarm_state ")
  print ( result )
else :
  print ("Connection failed ")
disconnectETController(sock)
```

### 2.2.2.66 Clear the booking queue

```
{"jsonrpc":"2.0","method":"book_program_clear","id":id}
```

Function: Clear the booking queue

Parameter: None

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip = "172.16.11.248"
  conSuc, sock = connectETController(robot_ip)
  if (conSuc):
    # Clear the booking queue
      suc, result , id = sendCMD(sock, "book_program_clear")
      print (suc, result , id)
      time.sleep (0.001)
    else :
      print ("Connection failed ")
  disconnectETController(sock)
```

Note: The command is only supported in the remote mode and it can clear the queue only when the robot stops or gives an alarm (under the static condition).

### 2.2.2.67 Get the actual tcp pose

```
{"jsonrpc":"2.0","method":"get_actual_tcp","params":{"tool_num":tool_num,
   "user_num":user_num},"id":id}
```

Function: Get the actual tcp pose data in the base coordinate system or the specified user coordinate system

Parameter: tool_num: tool coordinate number, optional parameter, int[0,7], when it is not entered, the current tool is used. Otherwise, the specified tool is used
user_num: user coordinate number, optional parameter, int[0,7], when it is not entered, the user gets the pose in the base coordinate system. Otherwise, the user gets the pose in the user-specific coordinate.

Return: Robot pose information double pose [6]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="172.16.11.248"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Get the actual tcp pose of the tool 1 in the user 1 coordinate system
        suc, result , id=sendCMD(sock,"get_actual_tcp",{"tool_num":1," user_num ":1})
        print ( result )
    else :
        print ("Connection failed")
    disconnectETController (sock)
```

### 2.2.2.68 Get the target interpolation tcp pose

```
{"jsonrpc":"2.0","method":"get_target_tcp","params":{"tool_num":tool_num,
    "user_num":user_num},"id":id}
```

Function: Get the target interpolation tcp pose data in the base coordinate system or the specified user coordinate system

Parameter: tool_num: tool coordinate number, optional parameter, int[0,7], when it is not entered, the current tool is used. Otherwise, the specified tool is used
user_num: user coordinate number, optional parameter, int[0,7], when it is not entered, the user gets the pose in the base coordinate system. Otherwise, the user gets the pose in the specified user coordinate

Return: Robot pose information double pose [6]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="172.16.11.248"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Get the target interpolation tcp pose of the tool 1 in the user 1 coordinate system
        suc, result , id=sendCMD(sock,"get_target_tcp",{"tool_num":1," user_num ":1})
        print ( result )
    else :
        print ("Connection failed")
    disconnectETController (sock)
```

### 2.2.2.69 Get the actual joint

```
{"jsonrpc":"2.0","method":"get_actual_joint","id":id}
```

Function: Get the current actual joint data

Parameter: none

Return: Robot joint information double joint[6]

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="172.16.11.248"
  conSuc,sock=connectETController( robot_ip )
  if (conSuc):
    # Get the current actual joint
    suc, result , id=sendCMD(sock,"get_actual_joint")
    print ( result )
  else :
    print ("Connection failed")
  disconnectETController (sock)
```

### 2.2.2.70 Get the target interpolation joint

```
{"jsonrpc":"2.0","method":"get_target_joint","id":id}
```

Function: Get the current target interpolation joint data

Parameter: none

Return: Robot joint information double joint [6]

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="172.16.11.248"
  conSuc,sock=connectETController( robot_ip )
  if (conSuc):
    # Get the current target interpolation joint
    suc, result , id=sendCMD(sock,"get_target_joint")
    print ( result )
  else :
    print ("Connection failed")
  disconnectETController (sock)
```

### 2.2.2.71 Get the linear interpolation pose

```
{"jsonrpc":"2.0","method":"get_interp_pose","params":{"data1":data1,
   "data2":data2, "ratio":ratio},"id":id}
```

Function: Get the linear interpolation pose data between two given poses

Parameter: data1: pose data, standard parameter, double pose [6], the first three stand for position, unit x, y, z is mm, range is [-∞, +∞], the last three stand for pose, unit Rx, Ry, Rz is radian, range is [-π,π]

data2: pose data, standard parameter, double pose[6], the first three stand for position, unit x, y, z is mm, range is [-∞, +∞], the last three stand for pose, unit Rx, Ry, Rz is radian, range is [-π,π]

ratio: floating-point data, standard parameter, it stands for the proportional value. The range is [0,1]. When the value is equal to 0, the robot will return to the first pose. When the value is equal to 1, the robot will return to the second pose

Return: Robot pose information double pose [6]

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="172.16.11.248"
    conSuc,sock=connectETController(robot_ip)
    point1=[371.533, 101.636, 3.038, 0, -0.174, 2.861]
    point2=[346.312, -256.945, -91.131, -0.014, 0.521, 1.903]
    if (conSuc):
        # Get the linear interpolation pose data between two given poses
        suc, result , id=sendCMD(sock,"get_interp_pose",{"data1":point1,"data2":point2,"ratio":0.5})
        print ( result )
    else:
        print ("Connection failed")
    disconnectETController (sock)
```

#### 2.2.2.72 Get the joint temperature

```
{"jsonrpc":"2.0","method":"get_joint_temp","id":id}
```

Function: Get the joint temperature

Parameter: none

Return: double joint_temp[6]

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="172.16.11.248"
  conSuc,sock=connectETController( robot_ip )
  if (conSuc):
    suc, result , id=sendCMD(sock,"get_joint_temp")
    print ( suc, result, id )
  else:
    print ("Connection failed")
  disconnectETController (sock)
```

### 2.2.3 Movement Service(MovementService)

#### 2.2.3.1 Joint Movement

```
{"jsonrpc":"2.0","method":"moveByjoint","params":{"targetPos":targetPos,"
   speed":speed,"acc":acc,"dec":dec,"cond_type":cond_type,"cond_num":
   cond_num,"cond_value":cond_value,"cond_judgment:":cond_judgment},
   "id":id}
```

Function: Joint movement

Parameter: targetpos: the target joint point double pos[6], the range is [-360,360]
speed: operating speed, range: double[0.01,100]
cond_type: optional parameter, 0 is digital input X, 1 is digital output Y, 2 is user-defined input, range: int [0,2]
cond_num: IO address, optional parameter, range int[0,63]
cond_value: IO status, optional parameter, range int[0,1], when the actual IO status is consistent with this value, the unfinished movement is immediately abandoned and the next instruction is executed.
cond_judgment: conditional judgment, optional parameter, string type, it is a user-defined IF statement. When the conditions are met, the unfinished movement will be immediately abandoned and the next instruction will be executed.

acc: acceleration percentage, range: int [1,100], optional parameter, the default values 80 if not written.
dec: deceleration percentage, range: int [1,100], optional parameter, the default value is acc if not written.

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc, sock = connectETController(robot_ip)
  point = []
  point.append([0.0065,-103.9938,102.2076,-88.2138,
   90.0000,0.0013])
  point.append([-16.2806,-82.4996,81.9848,-89.4851,
  90.0000, -16.2858])
  point.append([3.7679, -71.7544, 68.7276, -86.9732,
  90.0000, 3.7627])
  point.append([12.8237,-87.3028,87.2361,-89.9333,
   90.0000,12.8185])
  if(conSuc):
    # Get the servo status of the robotic arm
    suc, result, id = sendCMD(sock, "getServoStatus")
    if(result == 0):
      # Set the servo status of the robotic arm to ON
      suc, result, id = sendCMD(sock,"set_servo_status",{"status":1})
      time.sleep(1)
      for i in range(4):
        # Joint movement
        suc, result ,id=sendCMD(sock,"moveByJoint",{"targetPos":point[i],"speed":30,"acc"
            :10,"dec":10,"cond_type":0,"cond_num ":7,"cond_value":1})
        while(True):
          # Get robot status
          suc, result, id = sendCMD(sock, "getRobotState")
          if (result == 0):
            break
```

Note: When cond_type is equal to 0 or 1, the con_num and con_value are valid. When cond_type is equal to 2, the con_judgment is valid. For the user-defined conditions, please refer to the syntax format of UNTIL in the JBI command.

### 2.2.3.2 Linear motion

```
{"jsonrpc":"2.0","method":"moveByLine","params":{"targetPos":targetPos,"
   speed_type":speed_type,"speed":speed,"acc":acc,"dec":dec,"cond_type"
   :cond_type,"cond_num":cond_num,"cond_value:":cond_value},
   "cond_judgment":cond_judgment},"id":id}
```

Function: Linear motion

Parameter: targetpos: the target joint point double pos[6], the range is [-360,360]
speed: operating speed. double Type is linear speed, range: 1-3000; is rotation angular speed, range: 1-300; is absolute linear speed, range: linear minimum speed parameter value-linear maximum speed parameter value; absolute rotation angular speed, range: rotation angle minimum speed parameter value-rotation angle maximum speed param-eter value
speed_type: Speedtype, 0 is V (linear speed), 1 is VR (rotational angular speed), 2 is AV (absolute linear speed), and 3 is AVR (absolute rotation angular speed). Optional.
cond_type: optional parameter, 0 is digital input X, 1 is digital output Y, 2 is user-defined input, range: int [0,2]
cond_num: IO address, optional parameter, range int[0,63]
cond_value: IO status, optional parameter, range int[0,1], when the actual IO status is consistent with this value, immediately abandon the unfinished movement and execute the next instruction
cond_judgment: conditions, optional parameter, string type, it is a user-defined IF statement. When the conditions are met, the unfinished movement will be immediately abandoned and the next instruction will be executed.
acc: acceleration percentage, range: int [1,100], optional parameter, the default value is 80 if not written.
dec: deceleration percentage, range: int [1,100], optional parameter, the default value is acc if not written.

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.205"
  conSuc, sock = connectETController(robot_ip)
  point = []
  point.append([0.0065,-103.9938,102.2076,-88.2138,
   90.0000,0.0013])
  point.append([-16.2806,-82.4996,81.9848,-89.4851,
  90.0000, -16.2858])
  point.append([3.7679, -71.7544, 68.7276, -86.9732,
  90.0000, 3.7627])
  point.append([12.8237,-87.3028,87.2361,-89.9333,
   90.0000,12.8185])
  if(conSuc):
    # Set the servo status of the robotic arm to ON
    suc, result, id = sendCMD(sock,"set_servo_status",{"status":1})
    time.sleep(1)
    for i in range(4):
      # Linear motion
      suc, result ,id=sendCMD(sock,"moveByLine",{"targetPos":point[i],"speed_type":0,"speed"
          :200,"cond_type":0,"cond_num":7,"cond_value ":1})
      while(True):
        # Get robot status
        suc, result, id = sendCMD(sock, "getRobotState")
        if (result == 0):
          break
```

Note: When there is no speed_type parameter, speed means absolute linear speed. When cond_type is equal to 0 or 1, the con_num and con_value are valid. When cond_type is equal to 2, the con_judgment is valid. For the user-defined conditions, please refer to the syntax format of UNTIL in the JBI command.

### 2.2.3.3 Circular Movement

```
{"jsonrpc":"2.0","method":"moveByArc","params":{"midPos":midPos,"
   targetPos":targetPos,"speed_type":speed_type,"speed":speed,"acc":acc,"
   dec":dec,"cond_type":cond_type ,"cond_num:":cond_num:,"cond_value":
   cond_value,"cond_judgment":cond_judgment},"id":id}
```

Function: Circular motion

Parameter: midpos: middle joint point double pos[6], the range is [-360,360]
targetpos: the target joint point double pos[8], the range is [-360,360]
speed: operating speed. double Type is linear speed, range: [1,3000]; is rotation an-gular speed, range: [1,300]; is absolute linear speed, range: linear minimum speed parameter value-linear maximum speed parameter value; absolute rotation angular speed, range: rotation angle minimum speed parameter value-rotation angle maxi-mum speed parameter value
speed_type: speed type, int[0,3], 0 is V (linear speed), 1 is VR (rotational angular speed), 2 is AV (absolute linear speed), and 3 is AVR (absolute rotation angular speed). Optional.
cond_type: optional parameter, 0 is digital input X, 1 is digital output Y, 2 is user-defined input, range int[0,2]
cond_num: IO address, optional parameter, range int[0,63]
cond_value: IO status, optional parameter, range int[0,1], when the actual IO status is consistent with this value, the unfinished movement is immediately abandoned and the next instruction is executed.
cond_judgment: conditions, optional parameter, string type, it is a user-defined IF statement. When the conditions are met, the unfinished movement will be immediately abandoned and the next instruction will be executed.
acc: acceleration percentage, range: int [1,100], optional parameter, the default value is 80 if not written.
dec: deceleration percentage, range: int [1,100], optional parameter, the default value is acc if not written.

Return: True for success, false for failure

Example:

```
if __name__ == ”__main__”:
  robot_ip=”192.168.1.200”
  conSuc,sock=connectETController( robot_ip )
  P000 = [0.0065,−103.9938,102.2076,−88.2138,
```

```
 90.0000,0.0013]
P001 = [-16.2806,-82.4996,81.9848,-89.4851,
90.0000, -16.2858]
if (conSuc):
  # Get the servo status of the robotic arm
  suc, result , id=sendCMD(sock,"getServoStatus")
  if ( result == 0):
    # Set the servo status of the robotic arm to ON
    suc, result , id=sendCMD(sock,"set_servo_status",{"status":1})
    time.sleep (1)
  #Circular movement
  suc, result , id=sendCMD(sock,"moveByArc",{"midPos":P000,"targetPos":P001,"speed_type"
      :0,"speed":20,"cond_type":0,"cond_num":7 ,"cond_value":1})
```

Note: When there is no speed_type parameter, speed means absolute linear speed. When cond_type is equal to 0 or 1, the con_num and con_value are valid. When cond_type is equal to 2, the con_judgment is valid. For the user-defined conditions, please refer to the syntax format of UNTIL in the JBI command.

### 2.2.3.4 Rotation Movement

```
{"jsonrpc":"2.0","method":"moveByRotate","params":{"targetPos":targetPos
  ,"speed_type":speed_type,"speed":speed,"acc":acc,"dec":dec,"cond_type
  ":cond_type,"cond_num:": cond_num:,"cond_value":cond_value},"id":id}
```

Function: Rotational movement

Parameter: targetpos: the target joint point double pos[6], the range is [-360,360]
speed: operating speed. double Type is linear speed, range: [1,3000]; is rotation an-gular speed, range: [1,300]; is absolute linear speed, range: linear minimum speed parameter value-linear maximum speed parameter value; absolute rotation angular speed, range: rotation angle minimum speed parameter value-rotation angle maxi-mum speed parameter value
speed_type: speed type, int[0,1], 0 is V (linear speed), 1 is VR (rotational angular speed), 2 is AV (absolute linear speed), and 3 is AVR (absolute rotation angular speed). Optional.
cond_type: optional parameter, 0 is digital input X, 1 is digital output Y, 2 is user-defined input, range int[0,2]
cond_num: IO address, optional parameter, range int[0,63]
cond_value: IO status, optional parameter, range int[0,1], when the actual IO status is consistent with this value, the unfinished movement is immediately abandoned and the next instruction is executed.

acc: acceleration percentage, range: int [1,100], optional parameter, the default value is 80 if not written.
dec: deceleration percentage, range: int [1,100], optional parameter, the default value is acc if not written.

Return: True for success, false for failure

Example:

```
if __name__ ==
  # Robot IP
  robot_ip="192.168.1.200
  conSuc,sock=connectETController( robot_ip
  P000 = [0.0065,-103.9938,102.2076,-88.2138,
   90.0000,0.0013]
  if (conSuc):
    # Get the servo status of the robotic arm
    suc, result , id=sendCMD(sock,"getServoStatus")
    if ( result == 0):
      # Set the servo status of the robotic arm to ON
      suc, result , id=sendCMD(sock,"set_servo_status",{"status":1})
      time. sleep (1)
    # Rotational movement
    suc, result , id=sendCMD(sock,"moveByRotate",{"targetPos":P000,"speed_type":0,"speed":20,
          "cond_type":0,"cond_num":7,"cond_value":1 })
```

Note: When there is no speed_type parameter, speed represents the absolute rotational angular speed.
This command is deprecated.

**CAUTION**

The above commands are only supported in remote mode.
Before executing the above commands, please make sure that the robot is in a stopped state. If the robot is running, send the stop command first and wait for the robot to stop.

### 2.2.3.5 Add waypoint info 2.0

```
{"jsonrpc":"2.0","method":"addPathPoint","params":{"wayPoint":wayPoint,"
   moveType":moveType,"speed_type":speed_type,"speed":speed,"acc":acc,"
   dec":dec,"smooth":smooth,"cond_type":cond_type,"cond_num:":cond_num:,"
   cond_value":cond_value},"id":id} or {"jsonrpc":"2.0","method":"
   addPathPoint","params" :{"wayPoint":wayPoint,"moveType":moveType,"
   speed_type":speed_type,"speed":speed,"acc":acc,"dec":dec,"
   circular_radius":circular_radius,"cond_type":cond_type,"cond_num:":
   cond_num:,"cond_value":cond_value,"cond_judgment":cond_judgment},"id":id}
```

Function: Add Waypoint Info 2.0

Parameter: waypoint: target joint point double pos[6], the range is [-360,360]
moveType: 0 joint motion, 1 linear motion, 2 rotational motion around the tool tip point, 3 arc motion
speed: running speed. double, joint speed range: [1,100] when the joint moves. For linear, rotary and circular motion, the type is linear speed range: [1,3000]; it is rotational angular speed, range: [1,300]; is the absolute linear speed, the range: the minimum linear speed parameter value - the linear maximum speed parameter value; is the absolute rotational angular velocity, the range: the rotational angle minimum speed parameter value - the rotational angle maximum speed parameter value When there is no speed_type parameter, it is expressed as: motion speed, joint mo-tion speed range [1,100], linear and arc speed range [1,3000], rotational motion speed range [1,300]
speed_type: Speed type, int[0,3], 0 is V (linear velocity), 1 is VR (rotational angular velocity), 2 is AV (absolute linear velocity), 3 is AVR (absolute rotational angular velocity). Optional.
smooth: smoothness, range: int[0,7], optional parameter, the smoothness of the last point must be 0. deprecated.
circular_radius: blending radius, range: double[0, 2608], unit millimeter, op-tional parameter, default is 0 if not written. The blend radius of the last point must be 0.
cond_type: optional parameter, 0 is digital input X, 1 is digital output Y, 2 is user-defined input, range int[0,2]
cond_num: IO address, optional parameter, range int[0,63]
cond_value: IO status, optional parameter, range int[0,1], when the actual IO status is consistent with this value, immediately give up the unfinished motion and execute the next instruction
cond_judgment: conditions, optional parameter, string type, it is a user-defined IF statement. When the conditions are met, the unfinished movement will be immediately abandoned and the next instruction will be executed.
acc: acceleration percentage, range: int [1,100], optional parameter, the default value is 80 if not written.
dec: deceleration percentage, range: int [1,100], optional parameter, the default value is acc if not written.

Return: True for success, False for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  C000 = [0.0065,-103.9938,102.2076,-88.2138,
  90.0000, 0.0013]
  C001 = [-16.2806,-82.4996,81.9848,-89.4851,
  90.0000,-16.2858]
  # clear waypoint info 2.0
  suc, result , id = sendCMD(sock, "clearPathPoint")
  if( result == True):
    # Add waypoint info 2.0
    suc, result ,id=sendCMD(sock,"addPathPoint",{"wayPoint":C000,"moveType":0,"speed":50,"
        circular_radius":50})
    suc, result ,id=sendCMD(sock,"addPathPoint",{"wayPoint":C001,"moveType":1,"speed_type":
```

```
0,"speed":50,"circular_radius" :0})
```

Note: When cond_type is equal to 0 or 1, the con_num and con_value are valid. When cond_type is equal to 2, the con_judgment is valid. For the user-defined conditions, please refer to the syntax format of UNTIL in the JBI command.

**CAUTION**

This command is only supported in remote mode.
If the motion type is joint motion, the speed_type parameter is invalid and not recommended.
The parameter circular_radius and the parameter smooth can be used either. It is recommended to use the parameter circular_radius.

### 2.2.3.6 Clear waypoint information 2.0

```
{"jsonrpc":"2.0","method":"clearPathPoint","id":id}
```

Function: Clear waypoint information 2.0

Parameter: None

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc):
    # Get the servo status of the robotic arm
    suc, result ,id=sendCMD(sock,"getServoStatus")
    if( result == 0):
      # Clear waypoint information 2.0
      suc, result , id = sendCMD(sock, "clearPathPoint")
```

Note: This command is only supported in remote mode.

### 2.2.3.7 Track Movement 2.0

```
{"jsonrpc":"2.0","method":"moveByPath","id":id}
```

Function: Track Movement 2.0

Parameter: None

Return: Failure: -1, Success: the total number of waypoints

Example:

```
if __name__ == "__main__":
```

```
# Robot IP address
robot_ip="192.168.1.200"
conSuc,sock=connectETController(robot_ip)
C000 = [0.0065,-103.9938,102.2076,-88.2138,
 90.0000,0.0013]
C001 = [-16.2806,-82.4996,81.9848,-89.4851,
90.0000, -16.2858]
C002 = [3.7679, -71.7544, 68.7276, -86.9732,
90.0000, 3.7627]
if(conSuc):
  # Clear waypoint information 2.0
  suc, result , id = sendCMD(sock, "clearPathPoint")
  if( result == True):
    # Add waypoint information 2.0
    suc, result , id = sendCMD(sock, "addPathPoint", {"wayPoint": C000,"moveType": 0, "
        speed": 50, "circular_radius":20})
    suc, result , id = sendCMD(sock, "addPathPoint", {"wayPoint": C001,"moveType":0, "
        speed": 50, "circular_radius":20})
    suc, result , id = sendCMD(sock, "addPathPoint", {"wayPoint": C002,"moveType": 0, "
        speed": 50, "circular_radius":0})
    # Trajectory movement 2.0
    suc, result , id = sendCMD(sock, "moveByPath")
    while(True):
      # Get the current running point number of the trajectory
      suc, result , id = sendCMD(sock, "getPathPointIndex")
      print( result )
      # Get robot status
      suc, result , id = sendCMD(sock, "getRobotState")
      if( result == 0):
        break
```

Note: This command is only supported in remote mode.
Before executing this command, make sure that the robot is in a stopped state. If the robot is running, send the stop command first and wait for the robot to stop.

### 2.2.3.8 Jog Movement

```
{"jsonrpc":"2.0","method":"jog","params":{"index":index,"speed":speed},"
   id":id}
```

Function: Jog movement

Parameter: index: axis direction or coordinate system direction number, range: int[0,11]
speed: manual speed percentage, range double [0.05,100] (optional parameter, not required)

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if (conSuc):
    # Get the servo status of the robotic arm
    suc, result, id = sendCMD(sock, "getServoStatus")
    if (result == 0):
      # Set the servo status of the robot arm to ON
      suc, result, id=sendCMD(sock,"set_servo_status",{"status":1})
      time.sleep(1)
    # Specify the coordinate system
    suc, result, id=sendCMD(sock, "setCurrentCoord", {"coord_mode":1})
    for i in range(0, 10, 1):
      # x-axis negative direction jog movement
      suc, result, id = sendCMD(sock, "jog", {"index":0,"speed":10})
      print(suc, result, id)
      time.sleep(0.1)
    suc, result, id = sendCMD(sock, "stop")
```

Note: After stopping sending the jog command, the robot will not stop immediately. Instead, the robot needs to be stopped immediately by the "stop robot operation" command below.
This command is only supported in remote mode.
If the next jog motion instruction is not received for more than 1 second, stop receiving the jog instruction, and the robot jog motion will stop.

### 2.2.3.9 Stop robot operation

```
{"jsonrpc":"2.0","method":"stop","id":id}
```

Function: Stop robot operation

Parameter: None

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Robot stop
        suc, result, id = sendCMD(sock, "stop")
```

Note: This command is only supported in remote mode.

### 2.2.3.10 Robot runs automatically

```
{"jsonrpc":"2.0","method":"run","id":id}
```

Function: Robot runs automatically

Parameter: None

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        # Robot stop
        suc, result, id = sendCMD(sock, "pause")
        time.sleep(1)
        # Robot start
        suc, result, id = sendCMD(sock, "run")
```

Note: This command is only supported in remote mode.

### 2.2.3.11 Robot stop

```
{"jsonrpc":"2.0","method":"pause","id":id}
```

Function: Robot stop

Parameter: None

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    if(conSuc):
        # Robot stop
        suc, result , id = sendCMD(sock, "pause")
        time.sleep(1)
```

Note: This command is only supported in remote mode.

### 2.2.3.12 Check if the jbi file exists

```
{"jsonrpc":"2.0","method":"checkJbiExist","params":{"filename":filename
    },"id":id}
```

Function: Check if the jbi file exists

Parameter: filename: the name of the file to be checked, string

Return: 0 means not present, 1 means present

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    jbi_filename ="test"
    if(conSuc):
        # Check if the jbi file exists
        suc, result ,id=sendCMD(sock,"checkJbiExist",{"filename": jbi_filename })
```

### 2.2.3.13 Run jbi file

```
{"jsonrpc":"2.0","method":"runJbi","params":{"filename":filename},"id":id
    }
```

Function: Run jbi file

Parameter: filename the name of the file to be run, string

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.200"
    conSuc,sock=connectETController(robot_ip)
    jbi_filename ="test"
```

```
if (conSuc):
  # Check if the jbi file exists
  suc, result ,id=sendCMD(sock,"checkJbiExist",{"filename": jbi_filename })
  if (suc and result ==1):
    # Run jbi file
    suc, result ,id=sendCMD(sock,"runJbi",{"filename": jbi_filename })
```

Note: This command is only supported in remote mode.
Before executing this command, make sure that the robot is in a stopped state. If the robot is running, send the stop command first and wait for the robot to stop.

### 2.2.3.14 Get jbi file running status

```
{"jsonrpc":"2.0","method":"getJbiState","id":id}
```

Function: Get jbi file running status

Parameter: None

Return: jbiName: file name
runState: 0 stop state, 1 pause state, 2 emergency stop state, 3 running state, 4 error state

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  jbi_filename ="test"
  if (conSuc):
    # Check if the jbi file exists
    suc, result ,id=sendCMD(sock,"checkJbiExist",{"filename": jbi_filename })
    if (suc and result ==1):
      # Run jbi file
      suc, result ,id=sendCMD(sock,"runJbi",{"filename": jbi_filename })
      if (suc and result ):
        checkRunning=3
        while(checkRunning==3):
          # Get jbi file running status
          suc, result ,id=sendCMD(sock,"getJbiState")
          checkRunning=result["runState"]
          time.sleep (0.1)
```

### 2.2.3.15 Set robot running speed

```
{"jsonrpc":"2.0","method":"setSpeed","params":{"value":value},"id":id}
```

Function: Set the robot running speed

Parameter: value: speed, range: double [0.05,100]

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if (conSuc):
    # Set the robot running speed to 30%
    suc, result , id = sendCMD(sock, "setSpeed", {"value": 30})
  else:
    print ("Connection failed")
  disconnectETController(sock)
```

Note: This command is applicable to v2.13.1 and above.
This command is only supported in remote mode.

### 2.2.3.16 Joint motion at uniform speed

```
{"jsonrpc":"2.0","method":"moveBySpeedj","params":{"vj":vj,"acc":acc,"t":
    t}," id":id}
```

Function: Uniform motion of joints

Parameter: vj: double type, speed value of 6 joints, six-digit unit: degree/second
acc: joint acceleration, int, range: greater than 0, unit: degree/$s^2$
t: SPEEDJ execution time, double, range: greater than 0, unit: second

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  speed_j   =[1.0,1.0,1.0,1.0,1.0,1.0]
  if (conSuc):
    # Uniform motion of joints
    suc, result ,id=sendCMD(sock,"moveBySpeedj",{"vj":speed_j,"acc":20,"t":5})
    print (suc, result , id)
```

Note: This command is applicable to v2.16.2 and above.
This command is only supported in remote mode.
During moveBySpeedj movement, when multiple moveBySpeedj instructions are sent, or stopj instruction is sent, the robot will not decelerate after executing the moveBySpeedj instruction, and continue to execute the last instruction sent by the user during the movement.

### 2.2.3.17 Stop joint motion at uniform speed

```
{"jsonrpc":"2.0","method":"stopj","params":{"acc":acc},"id":id}
```

Function: Stop joint movement at a constant speed

Parameter: acc: int, joint acceleration, stop motion at this acceleration, unit: degree/$s^2$, range: greater than 0

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if (conSuc):
    suc, result ,id=sendCMD(sock,"moveBySpeedj",{"vj" :[20,0,0,0,0,0], "acc" :50,"t" :2})
    print (suc, result ,id)
    time.sleep (1)
    suc, result ,id=sendCMD(sock,"stopj",{"acc":10})
    print ( result )
```

### 2.2.3.18 Linear uniform motion

```
{"jsonrpc":"2.0","method":"moveBySpeedl","params":{"v":v,"acc":acc,"arot
   ":arot,"t":t},"id":id}
```

Function: Linear uniform motion

Parameter: v: double type, the speed value of moving along 6 directions, unit: the first three are mm/sec, the last three are degrees/sec
acc: int displacement acceleration, range: greater than 0, unit: mm/$s^2$
arot: int optional parameter, attitude acceleration, range: greater than 0, unit: degree/$s^2$
t: SPEEDJ execution time, double, range: greater than 0, unit: second

Return: True for success, false for failure

Example:
```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  speed_l =[1.0,1.0,1.0,1.0,1.0,1.0]
  if(conSuc):
    # Linear uniform motion
    suc, result ,id=sendCMD(sock,"moveBySpeedl",{"v":speed_l,"acc":100,"arot":10,"t":3.0})
    print(suc, result ,id)
```

Note: This command is applicable to v2.16.2 and above.
This command is only supported in remote mode.
During moveBySpeedl movement, when multiple moveBySpeedl instructions are sent, or stopl instruction is sent, the robot will not decelerate after executing the moveBySpeedl instruction, and continue to execute the last instruction sent by the user during the movement.

### 2.2.3.19 Stop linear motion at constant speed

```
{"jsonrpc":"2.0","method":"stopl","params":{"acc":acc,"arot":arot},"id":
   id}
```

Function: Stop linear motion at constant speed

Parameter: acc: int, acceleration, stop motion at this acceleration, unit: mm/$sec^2$, range: greater than 0
arot: int optional parameter, attitude acceleration, range: greater than 0, unit: degree/$s^2$

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc):
    suc, result ,id=sendCMD(sock,"moveBySpeedl",{"v":[20,0,0,0,0,0],"acc":50,"arot":10,"t":2
        })
    print(suc, result ,id)
    time.sleep(1)
    suc, result ,id=sendCMD(sock,"stopl",{"acc":10})
    print( result )
```

### 2.2.3.20 Linear motion under specified coordinate system

```
{"jsonrpc":"2.0","method":"moveByLineCoord","params":{"targetUserPose":
   targetUserPose,"speed_type":speed_type,"speed":speed,"acc":acc,"dec":
   dec,"user_coord":user_coord,"cond_type":cond_type,"cond_num:" :
   cond_num:,"cond_value":cond_value,"unit_type":unit_type},"id":id}
```

Function: Linear motion in the specified coordinate system

Parameter: targetUserPose: specify the pose in the user coordinate system, where rx, ry, rz are radians, range: double [-π,π]] or angle, range: double[-180,180]
speed: operating speed. double Type is linear speed, range: [1,3000]; is rotation angular speed, range: [1,300]; is absolute linear speed, range: linear minimum speed

parameter value-linear maximum speed parameter value; absolute rotation angular speed, range: rotation angle minimum speed parameter value-rotation angle maximum speed parameter value

speed_type: speed type, optional parameter, int [0,3], 0 is V (linear speed), 1 is VR (rotational angular speed), 2 is AV (absolute linear speed), and 3 is AVR (absolute rotation angular speed).

user_coord: user coordinate system data, double[6], where rx, ry, rz are radians, range: double[-π,π] or angle, range: double[-180,180], do not write the current coordinate system.

cond_type: optional, 0 is digital input X, 1 is digital output Y, 2 is user-defined input, range int[0,2]

cond_num: IO address, optional, range int[0,63]

cond_value: IO status, optional, range int[0,1], when the actual IO status is consistent with this value, immediately abandon the unfinished movement and execute the next instruction.

cond_judgment: conditions, optional parameter, string type, it is a user-defined IF statement. When the conditions are met, the unfinished movement will be immediately.

acc: acceleration percentage, range: int [1,100], optional parameter, the default value is 80 if not written.

dec: deceleration percentage, range: int [1,100], optional parameter, the default value is acc if not written.

unit_type: the unit type of rx, ry, rz of user coordinate and user coordinate system, int [0,1], 0: angle, 1: radians, optional parameters, if not written, the default value is radians.

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  point =[211,126,343,-2.58,-0.013,-1.813]
  if(conSuc):
    # Specify the linear motion in the coordinate system
    suc, result ,id=sendCMD(sock,"moveByLineCoord",{"targetUserPose": point,"user_coord"
        :[0,0,0,0,0,0], "speed_type":1,"speed":30,"unit_type":1})
print(suc, result , id)
```

Note: This command is applicable to v2.16.2 and above. This command is only supported in remote mode. When there is no speed_type parameter, speed means absolute linear speed. When cond_type is equal to 0 or 1, the con_num and con_value are valid. When cond_type is equal to 2, the con_judgment is valid. For the user-defined conditions, please refer to the syntax format of UNTIL in the JBI command.

#### 2.2.3.21 Encoder zero calibration

```
{"jsonrpc":"2.0","method":"calibrate_encoder_zero_position","id":id}
```

Function: Encoder Zero Calibration

Parameter: None

Return: success true, failure false

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.0.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        suc, result , id = sendCMD(sock, "getServoStatus")
        print(suc, result , id)
        time.sleep(0.5)
        if result == 0:
            # Set the arm servo state
            ret , result ,id=sendCMD(sock, "set_servo_status", {"status": 1})
            print( result )
        time.sleep(1)
        # encoder zero calibration
        suc, result ,id=sendCMD(sock, "calibrate_encoder_zero_position")
        print(suc, result , id)
    else:
        print("Connection failed")
    disconnectETController(sock)
```

Note: This command is only supported in remote mode.

### 2.2.4 Kinematics Service(KinematicsService)

#### 2.2.4.1 Inverse solution function

```
{"jsonrpc":"2.0","method":"inverseKinematic","params":{"targetPose":
    targetPose,"referencePos":referencePos},"id":id}
```

Function: Inverse solution function, inverse solution with reference point position, according to the pose information to get the corresponding manipulator joint angle information

Parameter: targetPose: target pose information
referencePos: inverse solution reference point joint angle information double pos[6], the range is [-360,360], optional parameter
unit_type: the unit type of the rx, ry, rz of the input pose, int [0,1], 0: angle, 1: radian, optional parameter, if not written, the default value is radians.

Return: Joint coordinates double pos[6]

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  # Reference point
  P000 = [0, -90, 90, -90, 90, 0]
  conSuc,sock=connectETController(robot_ip)
  if(conSuc):
    # Get the current pose information of the robot
    suc, result , id = sendCMD(sock, "get_tcp_pose")
    # Inverse solution function 2.0, with reference point position
    suc, result ,id=sendCMD(sock,"inverseKinematic",{"targetPose": result ,"referencePos":
        P000})
```

### 2.2.4.2 Positive solution function

```
{"jsonrpc":"2.0","method":"positiveKinematic","params":{"targetPos":
   targetPos,"unit_type":unit_type},"id":id}
```

Function: Positive solution function, get the corresponding pose information according to the joint angle information of the manipulator

Parameter: targetpos: target joint angle information double pos[6], the range is [-360,360]
unit_type: return the unit type of the pose rx, ry, rz, int [0,1], 0: return angle, 1: return radian, optional parameter, if not written, the default value is radians.

Return: Response pose information: double pose[6]

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc):
    # Get the current position information of the robot
    suc, result , id = sendCMD(sock, "get_joint_pos")
    # Positive solution function
    suc, result ,id=sendCMD(sock,"positiveKinematic",{"targetPos": result ,"unit_type":1})
```

### 2.2.4.3 Base coordinate to user coordinate pose conversion

```
{"jsonrpc":"2.0","method":"convertPoseFromCartToUser","params":{"
   targetPose":targetPose,"userNo":userNo,"unit_type":unit_type},"id":id}
```

Function: Base coordinate to user coordinate pose conversion function, in the current user coordinate system, the pose information in the corresponding user coordinate system is obtained according to the pose information of the base coordinate

Parameter: targetPose: pose information in the base coordinate system, double pose[6], rx, ry, rz range: radian is [-π,π], angle is [-180,180]
userNo:User coordinate number, range: int[0,7]
unit_type: input pose and return pose of rx, ry, rz unit type, int [0,1], 0: angle, 1: radian, optional parameter, if not written, the default value is radians.

Return: Pose information under the user standard system: double user_pose[6]

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc):
    # Get the current pose information of the robot
    suc, result , id = sendCMD(sock, "get_tcp_pose")
    # Base coordinate to user coordinate pose conversion
    suc, result ,id=sendCMD(sock,"convertPoseFromCartToUser",{"targetPose":result,"userN":0,
        "unit_type":1})
```

### 2.2.4.4 User coordinate to base coordinate pose conversion

```
{"jsonrpc":"2.0","method":"convertPoseFromUserToCart","params":{"
   targetPose":targetPose,"userNo":userNo,"unit_type":unit_type},"id":id}
```

Function: User coordinate to base coordinate pose conversion, in the current user coordinate system, the pose information in the corresponding base coordinate system is obtained according to the pose information of the user coordinate

Parameter: targetPose: the pose information in the user coordinate system, double pose[6], the range of rx, ry, rz: radian is [-π,π], angle is [-180,180]
userNo: User coordinate number, range: int[0,7]
unit_type: input pose and return pose of rx, ry, rz unit type, int [0,1], 0: angle, 1: radian, optional parameter, if not written, the default value is radians.

Return: Pose information in the base coordinate system: double base_pose[6]

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if (conSuc):
    # Get the current pose information of the robot
    suc, result , id = sendCMD(sock, "get_tcp_pose", {"coordinate_num":1, "tool_num":7})
    # User coordinate to base coordinate pose conversion
    suc, result ,id=sendCMD(sock,"convertPoseFromUserToCart",{"targetPose":result,"userNo"
        :0})
```

### 2.2.4.5 Position multiplication

```
{"jsonrpc":"2.0","method":"poseMul","params":{"pose1":pose1,"pose2":pose2
   ,"unit_type":unit_type},"id":id}
```

Function: Position multiplication

Parameter: pose1: pose information, double pose[6], the range of rx, ry, rz: radian is [-π,π], angle is [-180,180]
pose2: pose information, double pose[6], the range of rx, ry, rz is: [-π,π] in radians, and the angle is [-180,180]
unit_type: input pose and return pose of rx, ry, rz unit type, int [0,1], 0: angle, 1: radian, optional parameter, if not written, the default value is radians.

Return: Posture multiplication result information: :double response_pose[6]

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  # pose1
  V000 = [10, −10, 10, 0, 0, 0]
  conSuc,sock=connectETController( robot_ip )
  if (conSuc):
    # Get the current pose information of the robot
    suc, result , id = sendCMD(sock, "get_tcp_pose")
    # Position multiplication
    suc, result ,id=sendCMD(sock,"poseMul",{"pose1":V000,"pose2":result,"unit_type":1})
```

### 2.2.4.6 Pose inversion

```
{"jsonrpc":"2.0","method":"poseInv","params":{"pose":pose,"unit_type":
   unit_type},"id":id}
```

Function: Pose inversionPose inversion

Parameter: pose: pose information, double pose[6], the range of rx, ry, rz: radian is [-π,π], angle is [-180,180]
unit_type: input pose and return pose of rx, ry, rz unit type, int [0,1], 0: angle, 1: radian, optional parameter, if not written, the default value is radians.

Return: Pose inversion result information: double response_pose[6]

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController( robot_ip )
  if (conSuc):
    # Get the current pose information of the robot
    suc, result , id = sendCMD(sock, "get_tcp_pose")
    # Pose inversion
    suc, result , id = sendCMD(sock, "poseInv", {"pose": result ,"unit_type":1})
```

## 2.2.5 IO Service(IOService)

### 2.2.5.1 Get input IO status

```
{"jsonrpc":"2.0","method":"getInput","params":{"addr":addr},"id":id}
```

Function: Get input IO status

Parameter: addr: input IO address, range: int[0,19][48,51]

Return: Input IO status, int[0,1], 0 is off, 1 is on

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc):
    for i in range(0, 19 ,1):
      # Get input IO status
      suc, result , id = sendCMD(sock, "getInput", {"addr":i})
      print( result )
```

### 2.2.5.2 Get output IO status

```
{"jsonrpc":"2.0","method":"getOutput","params":{"addr":addr},"id":id}
```

Function: Get output IO status

Parameter: addr: output IO address, range: int[0,19][48,51]

Return: Output IO status, int[0,1], 0 is off, 1 is on

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc):
    for n in range(0, 20 ,1):
      # Get output IO status
      suc, result , id = sendCMD(sock, "getOutput", {"addr":n})
      print( result )
```

### 2.2.5.3 Set output IO status

```
{"jsonrpc":"2.0","method":"setOutput","params":{"addr":addr,"status":
   status},"id":id}
```

Function: Set output IO status

Parameter: addr: output IO address, range: int[0,19][48,49]
status: IO status, int[0,1], 0 is off, 1 is on

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc):
    for i in range(0, 20 ,1):
      # Set output IO status
      suc, result ,id=sendCMD(sock,"setOutput",{"addr":i," status ":1})
      print ( result )
```

Note: This command is only supported in remote mode.

### 2.2.5.4 Get virtual input IO status

```
{"jsonrpc":"2.0","method":"getVirtualInput","params":{"addr":addr},"id":
   id}
```

Function: Get virtual input IO status

Parameter: addr: virtual IO address, range: int[0,399]

Return: Input IO status, int[0,1], 0 is off, 1 is on

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc):
    for i in range(0, 400 ,1):
      # Get virtual input IO status
      suc, result ,id=sendCMD(sock,"getVirtualInput",{"addr":i})
      print ( result )
```

### 2.2.5.5 Get virtual output IO status

```
{"jsonrpc":"2.0","method":"getVirtualOutput","params":{"addr":addr},"id":
   id}
```

Function: Get virtual output IO status

Parameter: addr: virtual IO address, range: int [400,1535]

Return: Output IO status, int[0,1], 0 is off, 1 is on

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc):
    for n in range(528, 800 ,1):
    # Get virtual output IO status
    suc, result ,id=sendCMD(sock,"getVirtualOutput",{"addr":n})
    print( result )
```

### 2.2.5.6 Set virtual output IO status

```
{"jsonrpc":"2.0","method":"setVirtualOutput","params":{"addr":addr,"
   status":status},"id":id}
```

Function: Set virtual output IO status

Parameter: addr: output IO address, range: int[528,799]
status: output IO status, int[0,1], 0 is off, 1 is on

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc):
    for i in range(528, 800 ,1):
      # Set virtual output IO status
      suc, result ,id=sendCMD(sock,"setVirtualOutput",{"addr":i," status ":1})
```

Note: This command is only supported in remote mode.

### 2.2.5.7 Read multiple M virtual IO

```
{" jsonrpc ":"2.0" ," method ":" getRegisters ","params":{"addr":addr,"
   len":len},"id":id}
```

Function: Read multiple M virtual IO

Parameter: addr: virtual IO address range int [0,1535]
len: The starting address starts to read backward and the length is (16*len) virtual IO range int [1,96]
The range of addr+16*len is int[0,1535]

Return: List of virtual IO values (each 16 virtual IO values are represented by a decimal integer, and the length of the list is len)

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc):
    # Get the value of M0~M16
    ret, result, id = sendCMD(sock, "getRegisters", {"addr": 0, "len": 1})
    if ret:
      print("result =", result)
    else:
      print("err_msg=", result["message"])
```

### 2.2.5.8 Get analog input

```
{"jsonrpc":"2.0","method":"getAnalogInput","params":{"addr":addr},"id":id
   }
```

Function: Get analog input

Parameter: addr: analog address, range: int[0,2]

Return: Analog value, range: double[-10,10]

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc):
    for i in range(0, 2, 1):
      # Get analog input
      suc , result , id = sendCMD(sock, "getAnalogInput", {"addr":i})
```

### 2.2.5.9 Get analog output

```
{"jsonrpc":"2.0","method":"get_analog_output","params":{"addr":addr},"id
    ":id}
```

Function: Get analog output

Parameter: addr: analog output address, range: int [0,4]

Return: Analog output value, double

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.202"
  conSuc,sock=connectETController(robot_ip)
  if (conSuc):
    for i in range(0,5):
      # Get analog output voltage value
      suc, result, id = sendCMD(sock,"get_analog_output",{"addr":i})
      print(result)
```

#### 2.2.5.10 Set analog output

```
{"jsonrpc":"2.0","method":"setAnalogOutput","params":{"addr":addr,"value
    ":value},"id":id}
```

Function: Set analog output

Parameter: addr: analog address, range: int[0,4]
value: analog value, when addr is 0-3, range: double[-10,10]; when addr is 4, range: double[0,10]

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc):
    # Set analog output
    suc, result ,id=sendCMD(sock,"setAnalogOutput",{"addr":0,"value":-10})
    suc, result ,id=sendCMD(sock,"setAnalogOutput",{"addr":1,"value":-3.5})
    suc, result ,id=sendCMD(sock,"setAnalogOutput",{"addr":2,"value":0})
    suc, result ,id=sendCMD(sock,"setAnalogOutput",{"addr":3,"value":0.5})
    suc, result ,id=sendCMD(sock,"setAnalogOutput",{"addr":4,"value":0.5})
```

Note: This command is only supported in remote mode.

### 2.2.6 Variable service(VarService)

#### 2.2.6.1 Get system B variable value

```
{"jsonrpc":"2.0","method":"getSysVarB","params":{"addr":addr},"id":id}
```

Function: Get system B variable value

Parameter: addr: variable address, range: int [0,255]

Return: Variable value, range: int[0,2147483647]

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc)
    for n in range(0, 11 ,1):
      # Get system B variable value
      suc, result , id = sendCMD(sock, "getSysVarB", {"addr":n})
      print( result )
```

### 2.2.6.2 Set system B variable value

```
{"jsonrpc":"2.0","method":"setSysVarB","params":{"addr":addr,"value":
   value},"id":id}
```

Function: Set system B variable value

Parameter: addr: variable address, range: int [0,255]
value: variable value, range: int [0,2147483647]

Return: True for success, false for failure

Example:
```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc)
    for n in range(0, 11 ,1):
      # Set system B variable value
      suc, result ,id=sendCMD(sock,"setSysVarB",{"addr":i,"value":100})
```

Note: This command is only supported in remote mode.

### 2.2.6.3 Get system I variable value

```
{"jsonrpc":"2.0","method":"getSysVarI","params":{"addr":addr},"id":id}
```

Function: Get system I variable value

Parameter: addr: variable address, range: int [0,255]

Return: Variable value, range: int[-32767,32767]

Example:
```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc)
    for n in range(0, 11 ,1):
      # Get system I variable value
      suc, result , id = sendCMD(sock, "getSysVarI", {"addr":n})
      print( result )
```

### 2.2.6.4 Set system I variable value

```
{"jsonrpc":"2.0","method":"setSysVarI","params":{"addr":addr,"value":
   value},"id":id}
```

Function: Set system I variable value

Parameter: addr: variable address, range: int [0,255]
value: variable value, range: int[-32767,32767]

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc)
    for n in range(0, 11 ,1):
      # Set system I variable value
      suc, result ,id=sendCMD(sock,"setSysVarI",{"addr":i,"value":100})
```

Note: This command is only supported in remote mode.

### 2.2.6.5 Get system D variable value

```
{"jsonrpc":"2.0","method":"getSysVarD","params":{"addr":addr},"id":id}
```

Function: Get system D variable value

Parameter: addr: variable address, range: int [0,255]

Return: Variable value, range: double[-1e+09,1e+09]

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc)
    for n in range(0, 11 ,1):
      # Get system D variable value
      suc, result , id = sendCMD(sock, "getSysVarD", {"addr":n})
      print ( result )
```

### 2.2.6.6 Set system D variable value

```
{"jsonrpc":"2.0","method":"setSysVarD","params":{"addr":addr,"value":
   value},"id":id}
```

Function: Set system D variable value

Parameter: addr: variable address, range: int [0,255]
value Variable value, range: double[-1e+09,1e+09]

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc)
    for n in range(0, 11 ,1):
      # Set system D variable value
      suc, result ,id=sendCMD(sock,"setSysVarD",{"addr":i,"value":100})
```

Note: This command is only supported in remote mode.

### 2.2.6.7 Get whether the system P variable is enabled

```
{"jsonrpc":"2.0","method":"getSysVarPState","params":{"addr":addr},"id":
   id}
```

Function: Get whether the system P variable is enabled

Parameter: addr: variable address, range: int [0,255]

Return: Store the enabled state of the P variable, 0: not enabled, 1: enabled

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc)
    for i in range(0, 101 ,1):
      # Get whether the system P variable is enabled
      suc, result , id = sendCMD(sock, "getSysVarPState", {"addr": i})
```

### 2.2.6.8 Get the value of P variable

```
{"jsonrpc":"2.0","method":"getSysVarP","params":{"addr":addr},"id":id}
```

Function: Get the value of P variable

Parameter: addr: variable address, range: int [0,255]

Return: System P variable value double pos[8]

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController( robot_ip )
  if (conSuc)
    for i in range(0, 101 ,1) :
      # Get whether the system P variable is enabled
      suc, result , id = sendCMD(sock, "getSysVarPState", {"addr": i })
        if ( result == 1):
          # Get system P variable value
          suc, result , id = sendCMD(sock, "getSysVarP", {"addr":i })
          print ( result )
```

### 2.2.6.9 Set the value of the P variable

```
{"jsonrpc":"2.0","method":"setSysVarP","params":{"addr:":addr,"pos:":pos
   },"id":id}
```

Function: Set the value of the system P variable

Parameter: addr: variable address, range int[0,255]

pos: the value of the p variable, double pos[6], range [-360,360]

Return: True for success, False for failure

Example:

```
if __name__ == "__main__":
  ip = "192.168.1.202"
  conSuc, sock = connectETController(ip)
  point = [0, -90, 0, -90, 90, 0]
  if conSuc:
    ret , result ,id=sendCMD(sock,"setSysVarP",{"addr":0,"pos":point })
    if ret :
      print ( result )
    else :
      print ("err_msg = ", result ["message"])
```

Note: This command is only supported in remote mode.

This command is applicable to v2.15.2 and above.

### 2.2.6.10 Set the scope of the V variable

```
{"jsonrpc":"2.0","method":"setSysVarV","params":{"addr":addr,"pose:":pose
   },"id":id}
```

Function: Set the value of the system V variable

Parameter: addr: variable address: range int[0,255]
pose: double pose[6], the range of rx,ry,rz is [-π,π]

Return: True for success, False for failure

Example:

```
if __name__ == "__main__":
  ip = "192.168.1.202"
  conSuc, sock = connectETController(ip)
  pos = [200, 125.5, -50, 1.57, -1.57, 3.14]
  if conSuc:
    ret , result , id=sendCMD(sock,"setSysVarV",{"addr":0,"pos":pos})
    if ret :
      print ( result )
    else :
      print ("err_msg = ", result ["message"])
```

Note: This command is only supported in remote mode.
This command is applicable to v2.15.2 and above.

### 2.2.6.11 Get the value of V variable

```
{"jsonrpc":"2.0","method":"getSysVarV","params":{"addr":addr},"id":id}
```

Function: Get the value of V variable

Parameter: addr: variable address, range: int [0,255]

Return: System V variable value double pose[6]

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController( robot_ip )
  if (conSuc)
    for i in range(0, 101 ,1) :
      # Get the value of V variable
      suc, result , id = sendCMD(sock, "getSysVarV", {"addr": i })
      print ( result )
```

### 2.2.6.12 Set the value of the V variable

```
{"jsonrpc":"2.0","method":"setSysVarV","params":{"addr:":addr,"pose:":
   pose},"id":id}
```

Function: Set the value of the system V variable

Parameter: addr: variable address, range int[0,255]
pose: the value of the V variable, double pose[6], the range of rx,ry,rz is [-π,π]

Return: True for success, False for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.205"
  conSuc,sock=connectETController(robot_ip)
  pose = [200, 125.5, -50, 0, 0, 0]
  if(conSuc):
    # Set the system V variable value
    suc, result, id = sendCMD(sock, "setSysVarV", {"addr": 0,"pose":
        [243.5,-219.4,169.578000,3.139376,-0.002601,0.106804]})
    print(suc, result)
  else:
    print("Connection failed")
  disconnectETController(sock)
```

Note: This command is only supported in remote mode.

### 2.2.6.13 Save variable data

```
{"jsonrpc":"2.0","method":"save_var_data","id":id}
```

Function: Save system variable data

Parameter: no

Return: True for success, False for failure

Example:

```
if __name__ == "__main__":
  ip = "192.168.1.202"
  conSuc, sock = connectETController(ip)
  pos = [200, 125.5, -50, 1.57, -1.57, 3.14]
  if conSuc:
    ret, result, id = sendCMD(sock, "save_var_data")
    if ret:
      print( result )
    else:
      print("err_msg = ", result ["message"])
```

Note: This command is only supported in remote mode.
This command is applicable to v2.15.2 and above.

### 2.2.7 Transparent Transmission Service

#### 2.2.7.1 Initialize transparent transmission service

```
{"jsonrpc":"2.0","method":"transparent_transmission_init","params":{"
   lookahead":lookahead,"t":t,"smoothness":smoothness,"response_enable":
   response_enable},"id":id}
```

Function: Initialize transparent transmission service of the robot

Parameter: lookahead: lookahead time, unit ms, range: int [10,1000]
t: sampling time, unit ms, range: int [2,100]
smoothness: gain, unit percentage, range: double [0,1].
response_enable: optional parameter, if not written, there will be a return value by default. int[0,1], whether the add point instruction returns a value, 0: no return value, 1: return value

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if (conSuc):
    # Initialize transparent transmission service
    suc, result , id = sendCMD(sock, "transparent_transmission_init", {"lookahead": 400, "t
        ": 10, "smoothness": 0.1,"response_enable": 0})
```

Note: This command is only supported in remote mode.

#### 2.2.7.2 Set the current transparent transmission servo target joint point

```
{"jsonrpc":"2.0","method":"tt_set_current_servo_joint","params":{"
   targetPos":targetPos},"id":id}
```

Function: Set the current transparent transmission servo target joint point

Parameter: targetpos: target joint point double pos[6], the range is [-360,360]

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  # Transparent transmission starting point
  P0 = [0, -90, 0, -90, 90, 0]
  if (conSuc):
    # Initialize transparent transmission service
    suc, result , id=sendCMD(sock,"transparent_transmission_init",{"lookahead":400,"t":10,"
        smoothness":0.1})
    # Set the current transparent transmission target joint point
    suc, result , id=sendCMD(sock,"tt_set_current_servo_joint",{"targetPos": P0})
```

Note: This command is only supported in remote mode. This command is deprecated.

### 2.2.7.3 Get whether the current robot is in a transparent transmission state

```
{"jsonrpc":"2.0","method":"get_transparent_transmission_state","id":id}
```

Function: Get whether the current robot is in a transparent transmission state

Parameter: None

Return: Current transparent transmission status. 0: non-transparent transmission state, 1: transparent transmission state

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if (conSuc):
    # Get whether the current robot is in a transparent transmission state
    suc, result , id=sendCMD(sock,"get_transparent_transmission_state")
```

### 2.2.7.4 Add transparent transmission of servo target joint point information to the cache

```
{"jsonrpc":"2.0","method":"tt_put_servo_joint_to_buf","params":{"
   targetPos":targetPos},"id":id} or {"jsonrpc":"2.0","method":"
   tt_put_servo_joint_to_buf","params":{"targetPose":targetPose},"id":
   id}
```

Function: Add transparent transmission of servo target joint point information to the cache

Parameter: targetpos: target joint point double pos[6], the range is [-360,360] or targetPose: target pose point is double pos[6]

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.202"
    conSuc,sock=connectETController(ip)
    i = 0
    if (conSuc):
        # Get whether the current robot is in a transparent transmission state
        suc, result , id=sendCMD(sock, "get_transparent_transmission_state")
        print (suc, result , id)
        if ( result == 1):
            # Clear the transparent transmission cache
            suc, result , id = sendCMD(sock,"tt_clear_servo_joint_buf")
            time.sleep (0.5)
        # open a file
            file_name ='D:\\ tttest8 . txt '
            fo = open(file_name, "r")
            while 1:
                # Read each line of the file in turn (point information)
                line = fo.readline ()
                if not line: break
                # Remove the blanks at the beginning and end of each line
                line_list = line.strip ()
                line_list = list (map(float, line_list .split (',')))
                if (i == 0):
                    # Joint movement to the starting point
                    suc, result , id=sendCMD(sock,"moveByJoint",{"targetPos": line_list ,"speed":30})
                    wait_stop () # Wait for the robot to stop
                    # Initialize the transparent transmission service
                    suc, result , id=sendCMD(sock,"transparent_transmission_init",{"lookahead":400,
                        "t":10,"smoothness":0.1,"response_enable":1})
                    print (suc, result , id)
                # Add transparent transmission of servo target joint point information to the
                    cache
                suc, result , id = sendCMD(sock, "tt_put_servo_joint_to_buf",{"targetPos":
                    line_list })
                time.sleep (0.01)
                i = i + 1
```

Note: This command is only supported in remote mode.
Choose one of the two parameters targetPos and targetPose, and only one parameter can be sent in a command.

### 2.2.7.5 Clear the transparent transmission cache

```
{"jsonrpc":"2.0","method":"tt_clear_servo_joint_buf","id":id}
```

Function: Clear the transparent transmission cache

Parameter: None

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc):
    # Get whether the current robot is in a transparent transmission state
    suc, result ,id =sendCMD(sock,"get_transparent_transmission_state")
    if( result == 1):
      # Clear the transparent transmission cache
      suc, result ,id=sendCMD(sock, "tt_clear_servo_joint_buf")
      time.sleep (0.5)
```

Note: This command is only supported in remote mode.

### 2.2.7.6 Example 1

```
1  import socket
2  import json
3  import time
4  import random
5
6  def connectETController(ip,port=8055):
7      sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
8      try:
9          sock.connect((ip,port))
10         return (True,sock)
11     except Exception as e:
12         sock.close()
13         return (False,None)
14
15 def disconnectETController(sock):
16     if(sock):
17         sock.close()
18         sock=None
19     else:
```

```
        sock=None

def sendCMD(sock,cmd,params=None,id=1):
    if(not params):
        params=[]
    else:
        params=json.dumps(params)
    sendStr="{{\"method\":\"{0}\",\"params\":{1},\"jsonrpc\":\"2.0\",\"
       id\":{2}}}".format(cmd,params,id)+"\n"
    try:
        sock.sendall(bytes(sendStr,"utf-8"))
        ret =sock.recv(1024)
        jdata=json.loads(str(ret,"utf-8"))
        if("result" in jdata.keys()):
            return (True,json.loads(jdata["result"]),jdata["id"])
        elif("error" in jdata.keys()):
            return (False,jdata["error"],jdata["id"])
        else:
            return (False,None,None)
    except Exception as e:
        return (False,None,None)

def wait_stop():
    while True:
        time.sleep(0.01)
        ret1, result1, id1 = sendCMD(sock, "getRobotState")
        if (ret1):
            if result1 == 0 or result1 == 4:
                break
        else:
            print("getRobotState failed")
            break

if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.202"
    conSuc,sock=connectETController(robot_ip)
    print(conSuc)
    if(conSuc):
    # Get robot status
        suc, result, id = sendCMD(sock, "getRobotState")
        if(result == 4):
            # Clear alarm
```

```
62              suc, result, id = sendCMD(sock, "clearAlarm")
63              time.sleep(0.5)
64          # Get synchronization status
65          suc, result, id = sendCMD(sock, "getMotorStatus")
66          if(result == 0):
67              # Synchronize servo encoder data
68              suc, result, id = sendCMD(sock, "syncMotorStatus")
69              time.sleep(0.5)
70          # Get the servo status of the robotic arm
71          suc, result, id = sendCMD(sock, "getServoStatus")
72          if (result == 0):
73              # Set the servo status of the robotic arm to ON
74              suc, result, id = sendCMD(sock, "set_servo_status",{"status
                    ":1})
75              time.sleep(1)
76      # Get whether the current robot is in a transparent transmission
           state
77          suc, result, id = sendCMD(sock, "
               get_transparent_transmission_state")
78          print(suc, result, id)
79          if(result == 1):
80              # Clear the transparent transmission cache
81              suc, result, id = sendCMD(sock,"tt_clear_servo_joint_buf")
82              time.sleep(0.5)
83        # open a file
84          file_name ='D:\\tttest8.txt'
85          fo = open(file_name, "r")
86          while 1:
87              # Read each line of the file in turn (point information)
88              line = fo.readline()
89              if not line: break
90              # Remove the blanks at the beginning and end of each line
91              line_list = line.strip()
92              line_list = list(map(float, line_list.split(',')))
93              print(i,line_list)
94              if (i == 0):
95                  # Joint movement to the starting point
96                  suc, result, id = sendCMD(sock, "moveByJoint", {"
                        targetPos": line_list, "speed": 30})
97                  wait_stop() # Wait for the robot to stop
98                  # Initialize the transparent transmission service
99                  suc, result, id = sendCMD(sock, "
                        transparent_transmission_init", {"lookahead": 400, "
```

```
                    t": 10, "smoothness": 0.1, "response_enable": 1})
                print(suc, result, id)
            # Add transparent transmission of servo target joint point
                information to the cache
            suc, result, id = sendCMD(sock, "tt_put_servo_joint_to_buf"
                ,{"targetPos": line_list})
            time.sleep(0.01)
            i = i + 1
        # Close file
        fo.close()
        # Clear the transparent transmission cache
        suc, result, id = sendCMD(sock, "tt_clear_servo_joint_buf")
        print("clear_ret = ", suc)
else:
        print("Connection failed")
    disconnectETController(sock)
```

### 2.2.7.7 Example 2

```
import socket
import json
import time

def connectETController(ip,port=8055):
    sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    try:
        sock.connect((ip,port))
        return (True,sock)
    except Exception as e:
        sock.close()
        return (False,None)

def disconnectETController(sock):
    if(sock):
        sock.close()
        sock=None
    else:
        sock=None

def sendCMD(sock,cmd,params=None,id=1):
    if(not params):
        params=[]
```

```
24      else:
25          params=json.dumps(params)
26      sendStr="{{\"method\":\"{0}\",\"params\":{1},\"jsonrpc\":\"2.0\",\"
            id\":{2}}}".format(cmd,params,id)+"\n"
27      try:
28          sock.sendall(bytes(sendStr,"utf-8"))
29          # print(sock.recv)
30          ret =sock.recv(1024)
31          jdata=json.loads(str(ret,"utf-8"))
32          if("result" in jdata.keys()):
33              return (True,json.loads(jdata["result"]),jdata["id"])
34          elif("error" in jdata.keys()):
35              return (False,jdata["error"],jdata["id"])
36          else:
37              return (False,None,None)
38      except Exception as e:
39          return (False,None,None)
40
41  def send_Point(sock,cmd,params=None,id=1):
42      if(not params):
43          params=[]
44      else:
45          params=json.dumps(params)
46      sendStr="{{\"method\":\"{0}\",\"params\":{1},\"jsonrpc\":\"2.0\",\"
            id\":{2}}}".format(cmd,params,id)+"\n"
47      sock.sendall(bytes(sendStr,"utf-8"))
48
49  def wait_stop():
50      while True:
51          time.sleep(0.01)
52          ret1, result1, id1 = sendCMD(sock, "getRobotState")  #
                getRobotstate
53          if (ret1):
54              if result1 == 0 or result1 == 4:
55                  break
56          else:
57              print("getRobotState failed")
58              break
59
60
61  if __name__ == "__main__":
62      # Robot IP address
63      robot_ip="192.168.1.202"
```

```
64      conSuc,sock=connectETController(robot_ip)
65
66      point = []
67      i = 0
68      if(conSuc):
69          # Get robot status
70          suc, result, id = sendCMD(sock, "getRobotState")
71          if(result == 4):
72              # Clear alarm
73              suc, result, id = sendCMD(sock, "clearAlarm")
74              time.sleep(0.5)
75          # Get synchronization status
76          suc, result, id = sendCMD(sock, "getMotorStatus")
77          if(result == 0):
78              # Synchronize servo encoder data
79              suc, result, id = sendCMD(sock, "syncMotorStatus")
80              time.sleep(0.5)
81          # Get the servo status of the robotic arm
82          suc, result, id = sendCMD(sock, "getServoStatus")
83          if (result == 0):
84              # Set the servo status of the robotic arm to ON
85              suc, result, id = sendCMD(sock, "set_servo_status",{"status
                    ":1})
86              time.sleep(1)
87          # Get whether the current robot is in a transparent
               transmission state
88          suc, result, id = sendCMD(sock, "
               get_transparent_transmission_state")
89          print(result)
90          if(result == 1):
91              # Clear the transparent transmission cache
92              suc, result, id = sendCMD(sock,"tt_clear_servo_joint_buf")
93              time.sleep(0.5)
94          # open a file
95          file_name ='D:\\tttest8.txt'
96          fo = open(file_name, "r")
97          while 1:
98              # Read each line of the file in turn (point information)
99              line = fo.readline()
100             if not line: break
101             # Remove the blanks at the beginning and end of each line
102             line_list = line.strip()
103             line_list = list(map(float, line_list.split(',')))
```

```
104
105             if (i == 0):
106                 # Joint movement to the starting point
107                 suc, result, id = sendCMD(sock, "moveByJoint", {"
                        targetPos": line_list, "speed": 30})
108                 print(result)
109                 wait_stop() # Wait for the robot to stop
110                 print(1)
111                 # Initialize the transparent transmission service
112                 suc, result, id = sendCMD(sock, "
                        transparent_transmission_init", {"lookahead": 400, "
                        t": 10, "smoothness": 0.1, "response_enable":0})
113                 print(result)
114             # Add transparent transmission of servo target joint point
                    information to the cache
115             send_Point(sock, "tt_put_servo_joint_to_buf",{"targetPos":
                    line_list})
116             print(result)
117             time.sleep(0.01)
118             i = i + 1
119         # Close file
120         fo.close()
121         # Clear the transparent transmission cache
122         suc, result, id = sendCMD(sock, "tt_clear_servo_joint_buf")
123         print("clear_ret = ", suc)
124     else:
125         print("Connection failed")
126     disconnectETController(sock)
```

### 2.2.8 System Service(SystemService)

#### 2.2.8.1 Get the controller software version number

```
{"jsonrpc":"2.0","method":"getSoftVersion","id":id}
```

Function: Get the controller software version number

Parameter: None

Return: The controller software version number

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController( robot_ip )
  if (conSuc):
    # Get the controller software version number
    suc, result , id = sendCMD(sock, "getSoftVersion")
    print ( result )
```

#### 2.2.8.2 Get servo version number

```
{" jsonrpc ":"2.0" ," method ":" getJointVersion ","params":{"axis":axis
   },"id":id}
```

Function: Get the servo version number

Parameter: axis: range int [0,7], corresponding to axis number 1~8

Return: Servo version number

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController( robot_ip )
  if (conSuc):
    # Get 1 axis servo version number
    ret , result , id = sendCMD(sock, "getJointVersion", {"axis":0})
    if ret :
      print (" result =", result )
    else :
      print ("err_msg=", result ["message"])
```

Note: The servo version for this function is 11 and above

### 2.2.9 TrajectoryService(TrajectoryService)

#### 2.2.9.1 Initial movement

```
{"jsonrpc":"2.0","method":"start_push_pos","params":{"path_lenth":
   path_lenth,"pos_type":pos_type,"ref_joint_pos":ref_joint_pos,"
   ref_frame":ref_frame,"ret_flag": ret_flag},"id":id}
```

Function: Initial movement

Parameter: path_lenth: the number of transmitted points, int, range: greater than or equal to 3
pos_type: point type, int[0,1], 0: joint, 1: pose
ref_joint_pos: reference point, double pos[6], if the input is a pose point, this reference point is the inverse solution reference point of the first point
ref_frame: coordinate system, double pose[6], if it is based on the base coordinate system, all 0; if the input coordinate is a pose point, this parameter is the coordinate system of the point.
ret_flag: int[0,1], whether the add point instruction returns a value, 0: no return value, 1: return value

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip = "192.168.1.202"
  conSuc, sock = connectETController(robot_ip)
  pos = [0, -90, 0, -90, 90, 0]
  frame = [0, 0, 0, 0, 0, 0]
    if (conSuc):
    suc, result ,id=sendCMD(sock,"start_push_pos",{"path_lenth":10,"pos_type":0,"
          ref_joint_pos ":pos,"ref_frame":frame," ref_flag ":1})
    print ( result )
```

Note: The timestamp of the first point of the transmission must be 0.

### 2.2.9.2 Add movement points

```
{"jsonrpc":"2.0","method":"push_pos","params":{"timestamp":timestamp,"pos
   ":pos},"id":id}
```

Function: Add exercise points

Parameter: timestamp: double, the timestamp of the point (at which time point in the point sequence), unit: s, range: greater than or equal to 0, and increasing
pos: double pose[6], point data

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    pos = [0, -90, 0, -90, 90, 0]
    frame = [0, 0, 0, 0, 0, 0]
    res = 0
    if (conSuc):
        ret, result, id = sendCMD(sock, "start_push_pos", {"path_lenth":10, "pos_type": 0,
            "ref_joint_pos": pos, "ref_frame": frame, "ret_flag": 1})
        print(result)
        ret, joint, id = sendCMD(sock, "get_joint_pos")
        time.sleep(0.2)
        for i in range(0, 10):
            ret, result, id = sendCMD(sock, "push_pos", {"timestamp": res, "pos": joint})
            print(result, i)
            joint[0] += 0.01
            res = res + 0.002
```

### 2.2.9.3 Stop adding movement points

```
{"jsonrpc":"2.0","method":"stop_push_pos","id":id}
```

Function: Stop adding timestamp points

Parameter: None

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    pos = [0, -90, 0, -90, 90, 0]
    frame = [0, 0, 0, 0, 0, 0]
    res = 0
    if (conSuc):
        ret, result, id = sendCMD(sock, "start_push_pos", {"path_lenth":10, "pos_type": 0,
            "ref_joint_pos": pos, "ref_frame": frame, "ret_flag": 1})
        print(result)
        ret, joint, id = sendCMD(sock, "get_joint_pos")
        time.sleep(0.2)
        for i in range(0, 10):
            ret, result, id = sendCMD(sock, "push_pos", {"timestamp": res, "pos": joint})
            print(result, i)
            joint[0] += 0.01
            res = res + 0.002
        ret, result, id = sendCMD(sock, "stop_push_pos")
        print(result)
```

Note: Stop_push_pos and push_pos are corresponding relations. Only when all the points sent by the corresponding push_pos are correct, will it return True, otherwise, it will return False.

### 2.2.9.4 Check execution status

```
{"jsonrpc":"2.0","method":"check_trajectory","id":id}
```

Function: Check the execution status

Parameter: None

Return: int[-3,0], 0: transmission point and time stamp are correct, -1: point length does not match, -2: point format error, -3: time stamp is not standardized.

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    pos = [0, -90, 0, -90, 90, 0]
    frame = [0, 0, 0, 0, 0, 0]
    res = 0
    if (conSuc):
        ret, result, id = sendCMD(sock, "start_push_pos", {"path_lenth":10, "pos_type": 0,
            "ref_joint_pos": pos, "ref_frame": frame, "ret_flag": 1})
        print(result)
        ret, joint, id = sendCMD(sock, "get_joint_pos")
        time.sleep(0.2)
        for i in range(0, 10):
            ret, result, id = sendCMD(sock, "push_pos", {"timestamp": res, "pos": joint})
            print(result, i)
            joint[0] += 0.01
            res = res + 0.002
        ret, result, id = sendCMD(sock, "stop_push_pos")
        print(result)
        ret, result, id = sendCMD(sock, "check_trajectory")
        print(result)
```

### 2.2.9.5 Start a time-stamped movement

```
{"jsonrpc":"2.0","method":"start_trajectory","params":{"speed_percent":
   speed_percent},"id":id}
```

Function: Start a time-stamped movement

Parameter: speed_percent: double, trajectory speed percentage, that is, the original speed multiplied by the percentage of speed movement, unit: %, range: greater than or equal to 0.1

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    pos = [0, -90, 0, -90, 90, 0]
    frame = [0, 0, 0, 0, 0, 0]
    res = 0
    if (conSuc):
        ret, result, id = sendCMD(sock, "start_push_pos", {"path_lenth":10, "pos_type": 0,
            "ref_joint_pos": pos, "ref_frame": frame, "ret_flag": 1})
        print(result)
        ret, joint, id = sendCMD(sock, "get_joint_pos")
        time.sleep(0.2)
        for i in range(0, 10):
            ret, result, id = sendCMD(sock, "push_pos", {"timestamp": res, "pos": joint})
            print(result, i)
            joint[0] += 0.01
            res = res + 0.002
        ret, result, id = sendCMD(sock, "stop_push_pos")
        print(result)
        ret, result, id = sendCMD(sock, "check_trajectory")
        print(result)
        ret, result, id = sendCMD(sock, "start_trajectory", {"speed_percent": 50})
        print(result)
```

Note: As long as flush_trajectory, or start_push_pos is not executed, the current trajectory can run in a loop without repeated transmission

### 2.2.9.6 Pause motion

```
{"jsonrpc":"2.0","method":"pause_trajectory","id":id}
```

Function: Pause motion

Parameter: None

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        ret , result , id=sendCMD(sock, "pause_trajectory")
        print ( result )
```

#### 2.2.9.7 Resume motion

```
{"jsonrpc":"2.0","method":"resume_trajectory","id":id}
```

Function: Resume motion

Parameter: None

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        ret , result , id=sendCMD(sock, "resume_trajectory")
        print ( result )
```

#### 2.2.9.8 Stop motion

```
{"jsonrpc":"2.0","method":"stop_trajectory","id":id}
```

Function: Stop motion

Parameter: None

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "192.168.1.202"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        ret , result , id=sendCMD(sock, "stop_trajectory")
        print ( result )
```

### 2.2.9.9 Empty the cache

```
{"jsonrpc":"2.0","method":"flush_trajectory","id":id}
```

Function: Empty the cache

Parameter: None

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip = "192.168.1.202"
  conSuc, sock = connectETController(robot_ip)
  if(conSuc):
    ret , result ,id=sendCMD(sock, "flush_trajectory")
    print ( result )
```

### 2.2.9.10 Example 1

```
1  import socket
2  import json
3  import time
4
5  def connectETController(ip,port=8055):
6      sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
7      try:
8          sock.connect((ip,port))
9          return (True,sock)
10     except Exception as e:
11         sock.close()
12         return (False,None)
13
14 def disconnectETController(sock):
15     if(sock):
16         sock.close()
17         sock=None
18     else:
19         sock=None
20
21 def sendCMD(sock,cmd,params=None,id=1):
22     if(not params):
23         params=[]
24     else:
```

```
25          params=json.dumps(params)
26      sendStr="{{\"method\":\"{0}\",\"params\":{1},\"jsonrpc\":\"2.0\",\"
            id\":{2} }}".format(cmd,params,id)+"\n"
27      try:
28          sock.sendall(bytes(sendStr,"utf-8"))
29          # print(sock.recv)
30          ret =sock.recv(1024)
31          jdata=json.loads(str(ret,"utf-8"))
32          if("result" in jdata.keys()):
33              return (True,json.loads(jdata["result"]),jdata["id"])
34          elif("error" in jdata.keys()):
35              return (False,jdata["error"],jdata["id"])
36          else:
37              return (False,None,None)
38      except Exception as e:
39          return (False,None,None)
40
41  def wait_stop():
42      while True:
43          time.sleep(0.01)
44          ret1, result1, id1 = sendCMD(sock, "getRobotState")
45          if (ret1):
46              if result1 == 0 or result1 == 4:
47                  break
48          else:
49              print("getRobotState failed")
50              break
51
52  if __name__ == "__main__":
53      ip = "192.168.1.200"
54      conSuc, sock = connectETController(ip)
55      start_pos = [0, -90, 0, -90, 90, 0]
56      ref_pos = [0, 0, 0, 0, 0, 0]
57      res = 0
58      if conSuc:
59          suc, result, id = sendCMD(sock, "getRobotState")
60          if (result == 4):
61              # Clear alarm
62              suc, result, id = sendCMD(sock, "clearAlarm")
63              time.sleep(0.5)
64          # Get synchronization status
65          suc, result, id = sendCMD(sock,"getMotorStatus")
66          if result != True:
```

```
67              # Synchronize servo encoder data
68              suc, result, id = sendCMD(sock,"syncMotorStatus")
69              time.sleep(0.5)
70          time.sleep(0.5)
71          # Get the servo status of the robotic arm
72          suc, result, id = sendCMD(sock, "getServoStatus")
73          if result == 0:
74              # Set the servo status of the robotic arm to ON
75              ret, result, id = sendCMD(sock, "set_servo_status", {"
                    status": 1})
76          suc, result, id = sendCMD(sock, "moveByJoint", {"targetPos":
                start_pos, "speed": 50})
77          wait_stop()
78          suc, result, id = sendCMD(sock, "start_push_pos", {"path_lenth"
                : 10, "pos_type": 0, "ref_joint_pos": start_pos, "ref_frame"
                : ref_pos, "ret_flag": 1})
79          print(result)
80          time.sleep(0.2)
81          for i in range(0, 10):
82              suc, result, id = sendCMD(sock, "push_pos", {"timestamp":
                    res, "pos": start_pos})
83              start_pos[0] += 0.02
84              res = res + 0.002
85          time.sleep(1)
86          suc, result, id = sendCMD(sock, "stop_push_pos")
87          print(result)
88          suc, result, id = sendCMD(sock, "check_trajectory")
89          print(result)
90          suc, result, id = sendCMD(sock, "start_trajectory", {"
                speed_percent": 50})
91          print(result)
92          time.sleep(5)
93          suc, result, id = sendCMD(sock, "flush_trajectory")
94          print(result)
95      else:
96          print("Connection failed")
97      disconnectETController(sock)
```

#### 2.2.9.11 Example 2

```python
import socket
import json
import time

def connectETController(ip,port=8055):
    sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    try:
        sock.connect((ip,port))
        return (True,sock)
    except Exception as e:
        sock.close()
        return (False,None)

def disconnectETController(sock):
    if(sock):
        sock.close()
        sock=None
    else:
        sock=None

def sendCMD(sock,cmd,params=None,id=1):
    if(not params):
        params=[]
    else:
        params=json.dumps(params)
    sendStr="{{\"method\":\"{0}\",\"params\":{1},\"jsonrpc\":\"2.0\",\"
       id\":{2} }}".format(cmd,params,id)+"\n"
    try:
        sock.sendall(bytes(sendStr,"utf-8"))
        # print(sock.recv)
        ret =sock.recv(1024)
        jdata=json.loads(str(ret,"utf-8"))
        if("result" in jdata.keys()):
            return (True,json.loads(jdata["result"]),jdata["id"])
        elif("error" in jdata.keys()):
            return (False,jdata["error"],jdata["id"])
        else:
            return (False,None,None)
    except Exception as e:
        return (False,None,None)

def send_Point(sock,cmd,params=None,id=1):
    if(not params):
```

```
        params=[]
    else:
        params=json.dumps(params)
    sendStr="{{\"method\":\"{0}\",\"params\":{1},\"jsonrpc\":\"2.0\",\"
       id\":{2} }}".format(cmd,params,id)+"\n"
    sock.sendall(bytes(sendStr,"utf-8"))

def wait_stop():
    while True:
        time.sleep(0.01)
        ret1, result1, id1 = sendCMD(sock, "getRobotState")
        if (ret1):
            if result1 == 0 or result1 == 4:
                break
        else:
            print("getRobotState failed")
            break

if __name__ == "__main__":
    ip = "192.168.1.200"
    conSuc, sock = connectETController(ip)
    start_pos = [0, -90, 0, -90, 90, 0]
    ref_pos = [0, 0, 0, 0, 0, 0]
    res = 0
    if conSuc:
        suc, result, id = sendCMD(sock, "getRobotState")
        if (result == 4):
            # Clear alarm
            suc, result, id = sendCMD(sock, "clearAlarm")
            time.sleep(0.5)
        # Get synchronization status
        suc, result, id = sendCMD(sock,"getMotorStatus")
        if result != True:
            # Synchronize servo encoder data
            suc, result, id = sendCMD(sock,"syncMotorStatus")
            time.sleep(0.5)
        time.sleep(0.5)
        # Get the servo status of the robotic arm
        suc, result, id = sendCMD(sock, "getServoStatus")
        if result == 0:
            # Set the servo status of the robotic arm to ON
            ret, result, id = sendCMD(sock, "set_servo_status", {"
               status": 1})
```

```
84          suc, result, id = sendCMD(sock, "moveByJoint", {"targetPos":
               start_pos, "speed": 50})
85          wait_stop()
86          suc, result, id = sendCMD(sock, "start_push_pos", {"path_lenth"
               : 10, "pos_type": 0, "ref_joint_pos": start_pos, "ref_frame"
               : ref_pos, "ret_flag": 0})
87          print(result)
88          time.sleep(0.2)
89          for i in range(0, 10):
90              send_Point(sock, "push_pos", {"timestamp": res, "pos":
                   start_pos})
91              start_pos[0] += 0.02
92              res = res + 0.002
93          time.sleep(1)
94          suc, result, id = sendCMD(sock, "stop_push_pos")
95          print(result)
96          suc, result, id = sendCMD(sock, "check_trajectory")
97          print(result)
98          suc, result, id = sendCMD(sock, "start_trajectory", {"
               speed_percent": 50})
99          print(result)
100         time.sleep(5)
101         suc, result, id = sendCMD(sock, "flush_trajectory")
102         print(result)
103     else:
104         print("Connection failed")
105     disconnectETController(sock)
```

### 2.2.10 ProfinetService(ProfinetService)

#### 2.2.10.1 Get the value of the profinet int input register

```
{"jsonrpc":"2.0","method":"get_profinet_int_input_registers","params":{"
   addr":addr,"length":length},"id":id}
```

Function: Get the value of the profinet int input register

Parameter: addr: register start address, range int[0,31]
length: number of registers, range int[1,32]
Note: the sum of addr and length should be less than or equal to 32

Return: list of register values int[length]

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip = "192.168.1.202"
  conSuc, sock = connectETController(robot_ip)
  if (conSuc):
    # Get the value of the profinet int input register
    suc, result , id = sendCMD(sock, "get_profinet_int_input_registers",{"addr":0,"length"
        :1})
    print ( result )
  else :
    print ("Connection failed")
  disconnectETController(sock)
```

### 2.2.10.2 Get the value of the profinet int output register

```
{"jsonrpc":"2.0","method":"get_profinet_int_output_registers","params":{"
   addr":addr,"length":length},"id":id}
```

Function: Get the value of the profinet int output register

Parameter: addr: register start address, range int[0,31]
length: number of registers, range int[1,32]
Note: the sum of addr and length should be less than or equal to 32

Return: list of register values int[length]

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip = "192.168.1.202"
  conSuc, sock = connectETController(robot_ip)
  if (conSuc):
    # Get the value of the profinet int output register
    suc, result , id = sendCMD(sock, "get_profinet_int_output_registers",{"addr":1,"length"
        :2})
    print ( result )
  else :
    print ("Connection failed")
  disconnectETController(sock)
```

### 2.2.10.3 Get the value of the profinet float type input register

```
{"jsonrpc":"2.0","method":"get_profinet_float_input_registers","params
   ":{"addr":addr,"length":length},"id":id}
```

Function: Get the value of the profinet float type input register

Parameter: addr: register start address, range int[0,31]
length: number of registers, range int[1,32]
Note: the sum of addr and length should be less than or equal to 32

Return: list of register values int[length]

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip = "192.168.1.202"
  conSuc, sock = connectETController(robot_ip)
  if (conSuc):
    # Get the value of the profinet float type input register
    suc, result , id = sendCMD(sock, "get_profinet_float_input_registers",{"addr":0,"length
        ":1})
    print ( result )
  else :
    print ("Connection failed")
  disconnectETController(sock)
```

### 2.2.10.4 Get the value of the profinet float type output register

```
{"jsonrpc":"2.0","method":"get_profinet_float_output_registers","params
   ":{"addr":addr,"length":length},"id":id}
```

Function: Get the value of the profinet float type output register

Parameter: addr: register start address, range int[0,31]
length: number of registers, range int[1,32]
Note: the sum of addr and length should be less than or equal to 32

Return: list of register values int[length]

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip = "192.168.1.202"
  conSuc, sock = connectETController(robot_ip)
  if (conSuc):
    # Get the value of the profinet float type output register
    suc, result , id = sendCMD(sock, "get_profinet_float_output_registers",{"addr":0,"
        length":1})
    print ( result )
  else :
    print ("Connection failed")
  disconnectETController(sock)
```

### 2.2.10.5 Set the value of the profinet int output register

```
{"jsonrpc":"2.0","method":"set_profinet_int_output_registers","params":{"
   addr":addr,"length":length,"value":value},"id":id}
```

Function: Set the value of the profinet int output register

Parameter: addr: register start address, range int[0,31]
length: number of registers, range int[1,32]
Note: the sum of addr and length should be less than or equal to 32
value: list of register values, type int[length], element range [-2147483648, 2147483647]

Return: Success True, Failure False

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip = "192.168.0.202"
  conSuc, sock = connectETController(robot_ip)
  if (conSuc):
    # Set the profinet int output register
    suc, result , id = sendCMD(sock, "set_profinet_int_output_registers", {"addr": 1, "
        length": 2, "value": [1,1]})
    print ( result )
  else :
    print ("Connection failed")
```

Note: This command is only supported in remote mode.

#### 2.2.10.6 Set the value of the profinet float output register

```
{"jsonrpc":"2.0","method":"set_profinet_int_output_registers","params":{"
   addr":addr,"length":length,"value":value},"id":id}
```

Function: Set the value of the profinet float type output register

Parameter: addr: register start address, range int[0,31]
length: number of registers, range int[1,32]
Note: the sum of addr and length should be less than or equal to 32
value: list of register values, type double[length], element range [-3.40E+38,3.40E+38]

Return: Success True, Failure False

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip = "192.168.0.202"
  conSuc, sock = connectETController(robot_ip)
  if (conSuc):
    # Set the profinet float type output register
    suc, result , id = sendCMD(sock, "set_profinet_float_output_registers", {"addr": 0, "
        length": 2, "value": [1,1]})
    print ( result )
  else :
    print ("Connection failed")
  disconnectETController(sock)
```

Note: This command is only supported in remote mode.

### 2.2.11 Backdrive service

#### 2.2.11.1 Get the opening of the servo brake

```
{"jsonrpc":"2.0","method":"get_servo_brake_off_status","id":id}
```

Function: Get the opening of the servo brake

Parameter: none

Return: Servo brake open condition array

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "172.16.11.240"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        suc, result , id = sendCMD(sock, "get_servo_brake_off_status")
        print(suc, result , id)
    else:
        print("Connection failed")
    disconnectETController(sock)
```

Note: This command is only supported in reverse drive mode.

### 2.2.11.2 Get whether it is in reverse drive mode

```
{"jsonrpc":"2.0","method":"get_backdrive_status","id":id}
```

Function: Get whether it is in backdrive mode

Parameter: none

Return: 1: Backdrive mode, 0: Not in backdrive mode

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "172.16.11.240"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        suc, result , id = sendCMD(sock, "get_backdrive_status")
        print(suc, result , id)
    else:
        print("Connection failed")
    disconnectETController(sock)
```

### 2.2.11.3 Enter backdrive mode

```
{"jsonrpc":"2.0","method":"enter_backdrive","id":id}
```

Function: Enter backdrive mode

Parameter: none

Return: Success True, Failure False

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "172.16.11.240"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        suc, result, id = sendCMD(sock, "enter_backdrive")
        print(suc, result, id)
    else:
        print("Connection failed")
    disconnectETController(sock)
```

Note: This command is only supported in remote mode, and the robot must be in the reset state.

### 2.2.11.4 Exit backdrive mode

```
{"jsonrpc":"2.0","method":"exit_backdrive","id":id}
```

Function: Exit backdrive mode

Parameter: none

Return: Success True, Failure False

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "172.16.11.240"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        suc, result, id = sendCMD(sock, "exit_backdrive")
        print(suc, result, id)
    else:
        print("Connection failed")
    disconnectETController(sock)
```

Note: This command is only supported in remote mode, and the robot must be in reverse drive mode.

### 2.2.11.5 Reset controller state

```
{"jsonrpc":"2.0","method":"reset_robot_status","id":id}
```

Function: Reset controller state

Parameter: none

Return: Success True, Failure False

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip = "172.16.11.240"
    conSuc, sock = connectETController(robot_ip)
    if (conSuc):
        suc, result , id = sendCMD(sock, "reset_robot_status")
        print(suc, result , id)
    else:
        print("Connection failed")
    disconnectETController(sock)
```

Note: The robot must be in a stopped or error state (excluding emergency stop alam).
This command is only supported in remote mode.

## 2.2.12 Ethernet/IP

### 2.2.12.1 Get the value of the Ethernet/IP int input register

```
{"jsonrpc":"2.0","method":"get_eip_int_input_registers","params":{
   "addr":addr,"length":length},"id":id}
```

Function: Get the value of the Ethernet/IP int input register

Parameter: addr: register start address, range int[0,31]
length: number of registers, range int[1,32]
Note: the sum of addr and length should be less than or equal to 32

Return: list of register values int[length]

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip = "192.168.1.202"
  conSuc, sock = connectETController(robot_ip)
  if (conSuc):
    # Get the value of the Ethernet/IP int input register
    suc, result , id = sendCMD(sock, " get_eip_int_input_registers ",{"addr":0,"length"
        :1})
    print ( result )
  else :
    print ("Connection failed")
  disconnectETController(sock)
```

Note: This command is applicable to v3.5.2 and above.

### 2.2.12.2 Get the value of the Ethernet/IP int output register

```
{"jsonrpc":"2.0","method":"get_eip_int_output_registers","params":{"
   addr":addr,"length":length},"id":id}
```

Function: Get the value of the Ethernet/IP int output register

Parameter: addr: register start address, range int[0,31]
length: number of registers, range int[1,32]
Note: the sum of addr and length should be less than or equal to 32

Return: list of register values int[length]

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip = "192.168.1.202"
  conSuc, sock = connectETController(robot_ip)
  if (conSuc):
    # Get the value of the Ethernet/IP int output register
    suc, result , id = sendCMD(sock, " get_eip_int_output_registers ",{"addr":1,"length"
        :2})
    print ( result )
  else :
    print ("Connection failed")
  disconnectETController(sock)
```

Note: This command is applicable to v3.5.2 and above.

### 2.2.12.3 Get the value of the Ethernet/IP float type input register

```
{"jsonrpc":"2.0","method":"get_eip_float_input_registers","params":{
   "addr":addr,"length":length},"id":id}
```

Function: Get the value of the Ethernet/IP float input register

Parameter: addr: register start address, range int[0,31]
length: number of registers, range int[1,32]
Note: the sum of addr and length should be less than or equal to 32

Return: list of register values float [length]

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip = "192.168.1.202"
  conSuc, sock = connectETController(robot_ip)
  if (conSuc):
    # Get the value of the Ethernet/IP float input register
    suc, result , id = sendCMD(sock, " get_eip_float_input_registers ",{"addr":0,"length"
        :1})
    print ( result )
  else :
    print ("Connection failed")
  disconnectETController(sock)
```

Note: This command is applicable to v3.5.2 and above.

### 2.2.12.4 Get the value of the Ethernet/IP float type output register

```
{"jsonrpc":"2.0","method":"get_eip_float_output_registers","params":{
   "addr":addr,"length":length},"id":id}
```

Function: Get the value of the Ethernet/IP float output register

Parameter: addr: register start address, range int[0,31]
length: number of registers, range int[1,32]
Note: the sum of addr and length should be less than or equal to 32

Return: list of register values float [length]

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip = "192.168.1.202"
  conSuc, sock = connectETController(robot_ip)
  if (conSuc):
    # Get the value of the Ethernet/IP float output register
    suc, result , id = sendCMD(sock, " get_eip_float_output_registers ",{"addr":0,"length"
        :1})
    print ( result )
  else :
    print ("Connection failed")
  disconnectETController(sock)
```

Note: This command is applicable to v3.5.2 and above.

### 2.2.12.5 Set the value of the Ethernet/IP int output register

```
{"jsonrpc":"2.0","method":"set_eip_int_output_registers","params":{"
   addr":addr,"length":length},"id":id}
```

Function: Set the value of the Ethernet/IP int output register

Parameter: addr: register start address, range int[0,31]
length: number of registers, range int[1,32]
Note: the sum of addr and length should be less than or equal to 32
value: list of register values, type int[length], element range [-2147483648, 2147483647]

Return: True for success, False for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip = "192.168.1.202"
  conSuc, sock = connectETController(robot_ip)
  if (conSuc):
    # Set the Ethernet/IP int output register
    suc, result , id = sendCMD(sock, " set_eip_int_output_registers ", {"addr": 1, "
        length": 2, "value": [1,1]})
    print( result )
  else :
    print ("Connection failed ")
  disconnectETController (sock)
```

Note: This command is applicable to v3.5.2 and above.

### 2.2.12.6 Set the value of the Ethernet/IP float output register

```
{"jsonrpc":"2.0","method":"set_eip_float_output_registers","params":{"
   addr":addr,"length":length},"id":id}
```

Function: Set the value of the Ethernet/IP float output register

Parameter: addr: register start address, range int[0,31]
length: number of registers, range int[1,32]
Note: the sum of addr and length should be less than or equal to 32
value: list of register values, type double[length], element range [-3.40E +38,3.40E+38]

Return: True for success, False for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip = "192.168.1.202"
  conSuc, sock = connectETController(robot_ip)
  if (conSuc):
    # Set the Ethernet/IP float output register
    suc, result , id = sendCMD(sock, " set_eip_float_output_registers ", {"addr": 1, "
        length": 2, "value": [1.1,1.2]})
    print( result )
  else:
    print ("Connection failed")
  disconnectETController(sock)
```

Note: This command can be used only in the remote mode and is applicable to v3.5.2 and above.

## 2.2.13 External force sensor service

### 2.2.13.1 Mark the start of the torque data transfer

```
{"jsonrpc":"2.0","method":"start_push_force","params":{"return_flag":
   rerurn_flag},"id":id}
```

Function: Mark the start of the torque data transfer

Parameter: return_flag: set if there is a return value, bool type. True means that there is a return value. False means that there isn't a return value.

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if (conSuc):
    # Mark the start of the torque data transfer
    suc, result , id=sendCMD(sock,"start_push_force",{"return_flag":True})
```

Note: The function can be used only when the data source of the force control is SDK.

### 2.2.13.2 Transfer the torque data

```
{"jsonrpc":"2.0","method":"push_external_force","params":{"index":index,
   "torque_arry":torque_arry},"id":id}
```

Function: Transfer the torque data

Parameter: index: serial no., indicates the transfer sequence, range is [0, 65535]

torque_arry*: arry of torque data, double torques[6]

* Note: The data names are given as below: the force in the X-axis direction, the force in the Y-axis direction, the force in the Z-axis direction, the torque on the X axis, the torque on the Y axis and the torque on the Z axis. When the data type is double, it indicates the force in the X-axis direction, the force in the Y-axis direction, the force in the Z-axis direction, the torque on the X axis, the torque on the Y axis and the torque on the Z axis under the output coordinate system of the force sensor. The unit of the force in the X, Y, Z-axis direction is kg and the unit of the torque is kgM. If the unit of the original data and the coordinate system are different from the defined ones, please change the parameter first and then pass in them.

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if (conSuc):
    # Mark the start of the torque data transfer
    suc, result , id=sendCMD(sock,"start_push_force",{"return_flag":True})
      if result==True:
        # Transfer the torque data
        suc, result , id=sendCMD(sock,"push_external_force",{"index":0,"torque_arry":[1,2,3,4,5,6]})
```

Note: The value of the parameter index shall be gradually increased from 0 to 65535 after a complete transfer process is started. When it reaches the maximum, the counter will restart from 0. The parameter torque_arry will be marked as the latest data by repeating this process.

Please call the parameter start_push_force first to mark the start of the external torque data first before using this function. The function can be used only when the data source of the force control is SDK.

### 2.2.13.3 Stop the transfer of the current torque data

```
{"jsonrpc":"2.0","method":"stop_push_force","id":id}
```

Function: End the transfer of the current torque data

Parameter: none

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController( robot_ip)
  if (conSuc):
    # Mark the start of the torque data transfer
    suc, result , id=sendCMD(sock,"start_push_force",{"return_flag":True})
      if result==True:
        # Transfer the torque data
        suc, result , id=sendCMD(sock,"push_external_force",{"index":0,"torque_arry":[1,2,3,4,5,6]})
        # Stop the transfer of the torque data
        suc, result , id=sendCMD(sock,"stop_push_force")
```

Note: The function can be used only when the data source of the force control is SDK.

### 2.2.13.4 Get the source of the current torque data

```
{"jsonrpc":"2.0","method":"get_force_ctrl_mode","id":id}
```

Function: Get the source of the current torque data

Parameter: none

Return: int[0,4], 0 and 1 indicate that the torque data is from the inside, 2 indicates that the torque data is from SDK, 3 indicates that the torque data is from LUA 4 indicates that the torque data is from the terminal end

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.200"
  conSuc,sock=connectETController(robot_ip)
  if (conSuc):
    # Get the source of the current torque data
    suc, result , id=sendCMD(sock,"get_force_ctrl_mode")
    print ( result )
```

# 2.2.13.5 Examples

## 2.2.13.5.1 Example 1 (of the data transfer: with the returned value)

```
import socket
import json
import os
import time import sleep

def connectETController(ip="192.168.1.200", port=8055):
    sock1 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        sock1.connect((Ip, port))
        return True, sock1
    except Exception as e:
        print('error is',e)
        sock1.close()
        return False, None

def disconnectETController(sock1):
    if (sock1):
        sock1.close()
        sock1 = None
    else:
        sock1 = None
def sendCMD(sock1, cmd, params=None, id=1):
    if (not params):
        params = []
    else:
        params = json.dumps(params)
    sendStr = "{{\"method\":\"{0}\",\"params\":{1},\"jsonrpc
       \":\"2.0\",\"id\":{2}}}".format(cmd, params, id) + "\n"
    try:
        # print(sendStr)
        sock1.sendall(bytes(sendStr, "utf-8"))
        ret = sock1.recv(1024)
        jData = json.loads(str(ret, "utf-8"))
        # print("raw data:",jData)
        if "result" in jData.keys():
```

```
            return True, json.loads(jData["result"]), jData["id"]
        elif "error" in jData.keys():
            return False, jData["error"], jData["id"]
        else:
            return False, None, None
    except Exception as e:

        return False, None, None

if __name__ == "__main__":
    robot_ip = "192.168.1.200"
    return_flag = True

    conSuc, sock = connectETController(robot_ip)
    if conSuc:
        ret, result, id = sendCMD(sock, "get_force_ctrl_mode")
        if result != 2:  # sdk mode = 2
            print("please change the mode to sdk.")
            exit()

        ret, result, id = sendCMD(sock, "start_push_force",
           {"return_flag":return_flag})
        if result != True:
            exit()

        # torque data array
        arry = [1.2,2.3,3.2,4.4,5.6,6.9]
        index = 0
        cycle_time = 0.005  #interval(ms)
        host_time = 5       #last time(s)
        loopCnt = host_time / cycle_time
        while loopCnt > 0:
            ret, result, id = sendCMD(sock, "push_external_force",)
               {"index":index,"torque_arry":arry})
            if result != True:
                print("push_external_force failed!", result)
                break

            sleep(cycle_time)
            loopCnt -= 1
```

```
73          arry[0] += 1
74          index += 1
75
76      ret, result, id = sendCMD(sock, "stop_push_force")
77      else:
78          print("Connection failed")
79      disconnectETController(sock)
```

#### 2.2.13.5.2 Example 2 (of the data transfer: with no returned value)

```
import socket
import json
import os
from time import sleep

def connectETController(ip="192.168.1.200", port=8055):
    sock1 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        sock1.connect((Ip, port))
        return True, sock1
    except Exception as e:
        print('error is',e)
        sock1.close()
        return False, None

def disconnectETController(sock1):
    if sock1:
        sock1.close()
         sock1 = None
     else:
         sock1 = None

def sendCMD(sock1, cmd, params=None, id=1):
    if not params:
        params = []
    else:
        params = json.dumps(params)
    sendStr = "{{\"method\":\"{0}\",\"params\":{1},\"jsonrpc
       \":\"2.0\",\"id\":{2}}}".format(cmd, params, id) + "\n"
    try:
        sock1.sendall(bytes(sendStr, "utf-8"))
        ret = sock1.recv(1024)
        jData = json.loads(str(ret, "utf-8"))
        if "result" in jData.keys():
          return True, json.loads(jData["result"]), jData["id"]
        elif "error" in jData.keys():
```

```
            return False, None, None
    except Exception as e:
        print('error is',e)
        return False, None, None

def push_external_force_without_return(index, arry):
    cmd = "push_external_force"
    params = {"index":index, "torque_arry":arry}
    if not params:
        params = []
    else:
        params = json.dumps(params)
    id = 1
    sendStr = "{{\"method\":\"{0}\",\"params\":{1},\"jsonrpc
       \":\"2.0\",\"id\":{2}}}".format(cmd, params, id) + "\n"
    sock1.sendall(bytes(sendStr, "utf-8"))

if __name__ == "__main__":
    robot_ip = "192.168.1.200"
    return_flag = False

    conSuc, sock = connectETController(robot_ip)
    if conSuc:
        ret, result, id = sendCMD(sock, "get_force_ctrl_mode")
        if result != 2:  # sdk mode = 2
            print("please change the mode to sdk.")
            exit()

        ret, result, id = sendCMD(sock, "start_push_force",
           {"return_flag":return_flag})
        if result != True:
            exit()

        # torque data array
        arry = [1.2,2.3,3.2,4.4,5.6,6.9]
        index = 0
```

```
72          cycle_time = 0.005  #interval(ms)
73          host_time = 5
74          loopCnt = host_time / cycle_time
75          while loopCnt > 0:
76              push_external_force_without_return(index, arry)
77              sleep(cycle_time)
78              loopCnt -= 1
79              arry[0] += 1
80              index += 1
81      else:
82          print("Connection failed")
83
84      disconnectETController(sock)
85      conSuc, sock = connectETController(robot_ip)
86      if conSuc:
87          ret, result, id = sendCMD(sock, "stop_push_force")
88          disconnectETController(sock)
```

#### 2.2.13.5.3 Example 3 (of the data parsing and transfer)

```
import serial
import time
import socket
import json
import struct
from time import sleep

def connectETController(Ip="172.16.11.199", port=8055):
    sock1 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        sock1.connect((Ip, port))
        return True, sock1
    except Exception as e:
        print('error is', e)
        sock1.close()
        return False, None


def disconnectETController(sock1):
    if sock1:
        sock1.close()

def sendCMD(sock1, cmd, params = None, id=1):
    if not params:
        params = []
    else:
        params = json.dumps(params)
    sendStr ="{{\"method\":\"{0}\",\"params\":{1},\"jsonrpc\":\"2.0\",\"id\":{2}}}".
      format(cmd, params, id) + "\n"
    try:
        sock1.sendall(bytes(sendStr, "utf-8"))
        ret = sock1.recv(1024)
        jData = json.loads(str(ret, "utf-8"))
        if "result" in jData.keys():
            return True, json.loads(jData["result"]), jData["id"]
        elif "error" in jData.keys():
            return False, jData["error"], jData["id"]
        else:
            return False, None, None
    except Exception as e:
        print('error is', e)
        return False, None, None
```

```
42
43  # Global variable
44  portDev = 'COM4'
45  baud = 460800 # 115200
46  parity = serial.PARITY_NONE # PARITY_ODD/PARITY_EVEN
47  stopbits = serial.STOPBITS_ONE
48  bytesize = serial.EIGHTBITS
49  tmout = 0.02 # unit sec
50
51  CMD_STOP_SEND = '43AA0D0A'
52  CMD_CLEAR_SEND = '47AA0D0A'
53  CMD_SEND_DATA = '49AA0D0A'
54  CMD_ACTIVE_SEND = '48AA0D0A'
55  DATA_HEAD = '48AA'
56  DATA_TAIL = '0D0A'
57
58  if __name__ == "__main__":
59      robot_ip = "172.16.11.200"
60      return_flag = True
61      index = 0
62      torque = [0, 0, 0, 0, 0, 0]
63      tm_begin = 0
64      call_tm = 0
65      err_cnt = 0
66
67      # Open the serial port
68      try:
69          ser = serial.Serial(port=portDev, baudrate=baud, parity=parity,
                stopbits=stopbits, bytesize=bytesize, timeout=tmout)
70      except serial.serialutil.SerialException as err:
71          print("OS error: {0}".format(err))
72          exit()
73
74      if not ser.isOpen:
75          print("serial open unsuccessful.")
76          exit()
77
78      # Clear the buffer zone
79      ser.flushInput()
80
81      # Connect the robot
82      conSuc, sock = connectETController(robot_ip)
83      if not conSuc:
84          print("Failed to connect to Robot.")
```

```
        exit()

    # Check if the data source of the force control mode is SDK
    ret, result, id = sendCMD(sock, "get_force_ctrl_mode")
    if result != 2:   # sdk mode = 2
        print("please change the mode to sdk.")
        exit()

    ret, result, id = sendCMD(sock, "stop_push_force")

    # The sensor sends the data actively
    while True:
        ser.flushInput()
        ser.write(bytes.fromhex(CMD_ACTIVE_SEND))
        count = ser.inWaiting() # Get the buffer zone data of the serial port
        if count > 0:  # It is configured successfully if the data is received
            break
        sleep(0.01)

    # Mark the start of the torque data transfer
    ret, result, id = sendCMD(sock, "start_push_force", {"return_flag" :
        return_flag})
    if result != True:
        exit()

    print("Start to transmission force torque data.")

    # Main loop
    while True:
        # A frame of the valid data is 28 bytes
        if count > 27:
            recvRaw = ser.read(count) # Read the serial port data, bytes type
            beginIdx = recvRaw.find(bytes.fromhex(DATA_HEAD))
            if beginIdx != -1:
                endIdx = recvRaw.find(bytes.fromhex(DATA_TAIL), beginIdx)
                if endIdx != -1 and endIdx - beginIdx == 26:
                    recv = recvRaw[beginIdx:endIdx + 2]
                    recv = list(recv)
                    f1_raw = recv[2:6]
                    f2_raw = recv[6:10]
                    f3_raw = recv[10:14]
                    f4_raw = recv[14:18]
                    f5_raw = recv[18:22]
                    f6_raw = recv[22:26]
```

```
                    f1 = struct.unpack('f', bytes(f1_raw))[0]
                    f2 = struct.unpack('f', bytes(f2_raw))[0]
                    f3 = struct.unpack('f', bytes(f3_raw))[0]
                    f4 = struct.unpack('f', bytes(f4_raw))[0]
                    f5 = struct.unpack('f', bytes(f5_raw))[0]
                    f6 = struct.unpack('f', bytes(f6_raw))[0]
                    torque = [f1, f2, f3, f4, f5, f6]
                    ser.flushInput()
                    err_cnt = 0
                else:
                    sleep(0.001)
            else:
                sleep(0.001)
        else:
            err_cnt += 1
            if err_cnt > 50 :
                print("sensor data receive length(%d) error, Transmission
                  quit!"%(count))
                break


        # Count the time to complete a transmission for the commissioning
        tm_now = time.perf_counter()
        tm_elapse = tm_now - tm_begin
        tm_begin = tm_now
        ret, result, id = sendCMD(sock, "push_external_force", {"index":index,
          "torque_arry":torque})
        if result != True:
            tm_now = time.perf_counter()
            curr_call_tm = tm_now - tm_begin  # Count the timing of calling the port
            print("last call time = %f, once loop elapse time = %f, current call
              time = %f." % (call_tm, tm_elapse, curr_call_tm))
            print("push_external_force failed!", result)
            break

        # Count the timing of calling the port last time
        call_tm = time.perf_counter() - tm_begin
        # Necessary steps
        if index >= 65535:
            index = 0
        else:
            index += 1

        sleep(0.001)
```

```
169         count = ser.inWaiting() # Get the buffer zone data of the serial port
170
171     ret, result, id = sendCMD(sock, "stop_push_force")
172     disconnectETController(sock)
173     ser.close()
```

## 2.2.14 Hardware Communication Control Service

### 2.2.14.1 Open the serial port of the hardware

```
{"jsonrpc":"2.0","method":"open_serial_port","params":{"device_type":
   2},"id":id}
```

Function: Open the serial port of the hardware

Parameter: device_type: data type, standard parameter, int[0,1], 0 means RS232 communication, 1 means RS485 communication

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="172.16.11.248"
  conSuc,sock=connectETController( robot_ip)
  if (conSuc):
      suc , result , id=sendCMD(sock,"open_serial_port",{ "device_type":0})
      print (suc , result , id )
  else:
      print ("Connection failed")
  disconnectETController (sock)
```

Note: After the serial port is successfully opened, the serial data will be automatically configured. The default baud rate is 9600. The default data length is 8. The default parity bit is N, The default stop bit is 1. To modify the settings, please refer to the section "Configure the serial port".
This command is only supported in the remote mode. This command is applicable to v3.6.2 and above.

### 2.2.14.2 Configure the serial port of the hardware

```
{"jsonrpc":"2.0","method":"setopt_serial_port","params":{"baud_rate":9600,
   "bits":8,"event":"N", "stop":1},"id":id}
```

Function: Configure the serial port of the hardware

Parameter: baud_rate: baud rate, int type, optional parameter.
RS232 communication: there are 10 choices of the baud rate, i.e. 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 460800 and 500000. When not passing in the parameter, the baud rate is 9600 by default
RS485 communication: there are 14 choices of the baud rate, i.e. 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 460800, 500000, 921600, 1000000, 2000000 and 3000000. When not passing in the parameter, the baud rate is 9600 by default
bits: data length, int type, optional parameter, there are 2 choices of the data length, i.e. 7 and 8. When not passing in the parameter, the data length is 8 by default
event: parity check, string data type, optional parameter, there are 3 choices of the parity check, i.e. O, N and E. When not passing in the parameter, the parity check is none by default
stop: stop bit, int type, optional parameter, there are 2 choices of the stop bit, i.e. 1 and 2. When not passing in the parameter, the stop bit is 1 by default

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="172.16.11.248"
  conSuc,sock=connectETController(robot_ip)
  if (conSuc):
      suc, result ,id=sendCMD(sock,"open_serial_port",{"device_type":0})
      print ( suc, result, id )
      suc, result ,id=sendCMD(sock,"setopt_serial_port",{"baud_rate":9600,"bits":8, "event":"N",
          "stop":1})
      print(suc, result ,id)
  else:
      print ("Connection failed")
  disconnectETController(sock)
```

Note: This command can be executed only when the serial port is opened.
This command is only supported in the remote mode. This command is applicable to v3.6.2 and above.

### 2.2.14.3 Receive the data

```
{"jsonrpc":"2.0","method":"recv_serial_port""params":{"count":5},"id":id}
```

Function: Receive the data

Parameter: count: received data length, standard parameter, int[0,256]
hex: received data mode, optional parameter, there are 2 choices of the data mode, int[0,1]. 0 means ASCII (text) mode, 1 means the hexadecimal mode. When not passing in the parameter, the default is the text mode
timeout: time-out period, optional parameter, int[0,2147483647], when not passing in the parameter, the timeout is 100ms by default

Return: result: true for success, false for failure
size: data length collected
buf: data collected

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="172.16.11.248"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc):
    suc, result ,id=sendCMD(sock,"open_serial_port",{"device_type":0})
    print(suc, result , id)
    suc, recv ,id=sendCMD(sock,"recv_serial_port",{"count":100,"hex":0, "timeout":10})
    print( suc, recv, id )
    time.sleep(1)
  else:
      print("Connection failed")
  disconnectETController(sock)
```

Note: The results returned will determine if the data is valid or invalid. When the result is true, the data is valid. This command can be executed only when the serial port is opened. This command is only supported in the remote mode. This command is applicable to v3.6.2 and above.

### 2.2.14.4 Send the data

```
{"jsonrpc":"2.0","method":"send_serial_port","params":{"send_buf":"hello
   world!"},"id":id}
```

Function: Send the data

Parameter: send_buf: data to be sent, string data type, length of the string data is max. 256
hex: data mode sent, optional parameter, there are 2 choices of the data mode, i.e. int[0,1]. 0 means ASCII (text) mode, 1 means the hexadecimal mode.

Return: result: true for success, false for failure
size: data length sent

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="172.16.11.248"
  conSuc,sock=connectETController(robot_ip)
  if (conSuc):
     suc, result ,id=sendCMD(sock,"open_serial_port",{"device_type":0})
     print ( suc, result, id )
     suc, result ,id=sendCMD(sock,"send_serial_port",{"send_buff":"1024","hex":0})
     print ( suc, result, id )
  else:
       print ("Connection failed")
  disconnectETController (sock)
```

Note: The command can be executed only when the serial port is opened. This command is only supported in the remote mode. This command is applic-able to v3.6.2 and above.

### 2.2.14.5 Clear the buffer zone of the hardware serial port

```
{"jsonrpc":"2.0","method":"flush_serial_port","id":id}
```

Function: Clear the buffer zone of the hardware serial port

Parameter: none

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="172.16.11.248"
  conSuc,sock=connectETController( robot_ip )
  if (conSuc):
      suc, result , id=sendCMD(sock,"flush_serial_port")
      print ( suc, result, id )
  else:
      print ("Connection failed")
  disconnectETController (sock)
```

Note: This command is only supported in the remote mode. This command is applicable to v3.6.2 and above.

### 2.2.14.6 Close the serial port of the hardware

```
{"jsonrpc":"2.0","method":"close_serial_port","id":id}
```

Function: Close the serial port of the hardware

Parameter: none

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="172.16.11.248"
  conSuc,sock=connectETController(robot_ip)
  if (conSuc):
        suc, result ,id=sendCMD(sock,"close_serial_port")
        print ( suc, result, id )
  else:
       print ("Connection failed")
  disconnectETController (sock)
```

Note: This command is only supported in the remote mode. This command is applicable to v3.6.2 and above.

## 2.2.15 TCI Communication Control Service

### 2.2.15.1 Open the TCI serial port

```
{"jsonrpc":"2.0","method":"open_tci","id":id}
```

Function: Open the TCI serial port

Parameter: none

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="172.16.11.248"
  conSuc,sock=connectETController( robot_ip)
  if (conSuc):
      suc, result , id=sendCMD(sock,"open_tci ")
      print (suc, result , id)
  else:
      print ("Connection failed")
  disconnectETController (sock)
```

Note: After the serial port is successfully opened, the serial data will be automatically configured. The default baud rate is 9600. The default data length is 8. The default parity bit is N, The default stop bit is 1. To modify the settings, please refer to the section "Configure the serial port".
This command is only supported in the remote mode. This command is applicable to v3.6.2 and above.

### 2.2.15.2 Configure the TCI serial port

```
{"jsonrpc":"2.0","method":"setopt_tci","params":{"baud_rate":9600, "bits":8,
   "event":"N", "stop":1},"id":id}
```

Function: Configure the TCI serial port

Parameter: baud_rate: baud rate, int type, optional parameter, there are 13 choices of the baud rate, i.e. 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 460800, 500000, 921600, 1000000 and 2000000. When not passing in the parameter, the baud rate is 9600 by default
bits: data length, int type, optional parameter, the data length is 8
event: parity check, string data type, optional parameter, there are 3 choices of the parity check, i.e. O, N and E. When not passing in the parameter, the parity check is none by default
stop: stop bit, int type, optional parameter, there are 2 choices of the stop bit, i.e. 1 and 2. When not passing in the parameter, the stop bit is 1 by default

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="172.16.11.248"
  conSuc,sock=connectETController(robot_ip)
  if (conSuc):
      suc, result ,id=sendCMD(sock,"open_tci")
      print ( suc, result, id )
      suc, result ,id=sendCMD(sock,"setopt_tci",{"baud_rate":9600,"bits":8, "event":"N",
          "stop":1})
      print (suc, result ,id)
  else:
      print ("Connection failed")
  disconnectETController (sock)
```

Note: This command can be executed only when the serial port is opened. This command is only supported in the remote mode. This command is applicable to v3.6.2 and above.

### 2.2.15.3 Receive the data

```
{"jsonrpc":"2.0","method":"recv_tci""params":{"count":5},"id":id}
```

Function: Receive the data

Parameter: count: received data length, standard parameter, int[0,256]
hex: received data mode, optional parameter, there are 2 choices of the data mode, int[0,1]. 0 means ASCII (text) mode, 1 means the hexadecimal mode. When not passing in the parameter, the default is the text mode
timeout: time-out period, optional parameter, int[0,2147483647], when not passing in the parameter, the timeout is 100ms by default

Return: result: true for success, false for failure
size: data length collected
buf: data collected

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="172.16.11.248"
  conSuc,sock=connectETController(robot_ip)
  if(conSuc):
    suc, result ,id=sendCMD(sock,"open_tci")
    print(suc, result , id)
    suc, recv ,id=sendCMD(sock,"recv_tci",{"count":100,"hex":0, "timeout":10})
    print( suc, recv, id )
    time.sleep(1)
  else:
      print("Connection failed")
  disconnectETController(sock)
```

Note: The results returned will determine if the data is valid or invalid. When the result is true, the data is valid. This command can be executed only when the serial port is opened. This command is only supported in the remote mode. This command is applicable to v3.6.2 and above.

### 2.2.15.4 Send the data

```
{"jsonrpc":"2.0","method":"send_tci","params":{"send_buf":"hello world!"},
   "id":id}
```

Function: Send the data

Parameter: send_buf: data to be sent, string type, length of the string data is max. 256
hex: data mode sent, optional parameter, there are 2 choices of the data mode, i.e. int[0,1]. 0 means ASCII (text) mode, 1 means the hexadecimal mode.

Return: result: true for success, false for failure
size: data length sent

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="172.16.11.248"
  conSuc,sock=connectETController(robot_ip)
  if (conSuc):
     suc, result ,id=sendCMD(sock,"open_tci")
     print ( suc, result, id )
     suc, result ,id=sendCMD(sock,"send_tci",{"send_buff":"1024","hex":0})
     print ( suc, result, id )
  else:
       print ("Connection failed")
  disconnectETController (sock)
```

Note: The command can be executed only when the serial port is opened. This command is only supported in the remote mode. This command is applic-able to v3.6.2 and above.

### 2.2.15.5 Clear the buffer zone of the TCI serial port

```
{"jsonrpc":"2.0","method":"flush_tci","id":id}
```

Function: Clear the buffer zone of the TCI serial port

Parameter: none

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="172.16.11.248"
  conSuc,sock=connectETController(robot_ip)
  if (conSuc):
      suc, result , id=sendCMD(sock,"flush_tci")
      print ( suc, result, id )
  else:
      print ("Connection failed")
  disconnectETController(sock)
```

Note: This command is only supported in the remote mode. This command is applicable to v3.6.2 and above.

### 2.2.15.6 Close the TCI serial port

```
{"jsonrpc":"2.0","method":"close_tci","id":id}
```

Function: Close the TCI serial port

Parameter: none

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="172.16.11.248"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        suc, result , id=sendCMD(sock,"close_tci")
        print ( suc, result, id )
    else:
        print ("Connection failed")
    disconnectETController (sock)
```

Note: This command is only supported in the remote mode. This command is applicable to v3.6.2 and above.

## 2.2.16 Force Control Service

### 2.2.16.1 Start the force control

```
{"jsonrpc":"2.0","method":"start_force_mode","params":{"mode":3,
   "arr_frame":frame,"arr_optional":[6,1,5,1,0,1],"arr_torque":
   [-10,-10,-10,1.3,1.30,1.30],"arr_speed":speed},"id":id}
```

Function: Start the force control mode

Parameter: mode: mode, optional parameter, int[0,4], The numbers mean the fixed mode, the point mode, the motion mode, the TCP mode and the pose mode. The default is 0.
arr_frame: the user-specified force control coordinate system, optional parameter, unit of x, y and z is mm, range is [-10000,10000], unit of Rx, Ry and Rz is radian, range is [-π,π]. The default is [0,0,0,0,0,0] and the the force control coordinate system will be concentric with the TCP coordinate system.
arr_optional: force control method (mask of the DOFs), optional parameter, the value range is int[0,6] and the numbers mean the motion control, the force tracking, the fixed mode, the floating mode, the spring mode, the floating mode & motion control, the spring mode & motion control. The default is [0,0,0,0,0,0,] and it indicates the motion control.
arr_torque: target torque, optional parameter, double arr_torque[6], the first three numbers mean the force, the torque range of EC63 is [-30,30]N, the torque range of EC66 is [-60,60]N, the torque range of EC612 is [-120,120]N and the torque range of EC616 is [-160,160]N; the last three numbers mean the torque and the range is [-1.5,1.5]N. The default is [0,0,0,0,0,0]
arr_speed: speed limit, optional parameter, unit is mm/s and °/s, double arr_speed[6], the first three numbers mean the linear speed, range is [0,200]; the last three numbers mean the angular speed, range is [0,11]. The default is [100, 100,100, 5.73, 5.73, 5.73]

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="172.16.11.248"
  conSuc,sock=connectETController( robot_ip)
  if (conSuc):
      suc, result , id=sendCMD(sock,"start_force_mode",{"mode":0, "arr_frame":[0,0,0,0,0,0],
          "arr_optional":[1,0,0,0,0,0], "arr_torque":[1,0,0,0,0,0], "arr_speed":[100,100,100,5.73,
          5.73,5.73]})
      print (suc, result , id)
  else:
      print ("Connection failed")
  disconnectETController (sock)
```

Note: The force control function can be enabled by using this command. In the process, users can dynamically configure the related parameters.
When the force control mode is started, the jog movement is executed and the force control mode will be automatically ended.
It is impossible to start the force control mode and configure the parameters when the robot is moving.
It is also impossible to start the force control mode when selecting "Unknown data source" from the drop-down list.
The drag function and the force control function are mutually exclusive. They cannot work together.
Currently, users can only select the fixed mode, the TCP mode, the motion control and the force tracking. This command is applicable to v3.6.2 and above.

### 2.2.16.2 End the force control mode

```
{"jsonrpc":"2.0","method":"end_force_mode","id":id}
```

Function: End the force control mode

Parameter: none

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="172.16.11.248"
  conSuc,sock=connectETController(robot_ip)
  if (conSuc):
      suc, result , id=sendCMD(sock,"end_force_mode")
      print (suc, result , id)
  else:
      print ("Connection failed")
  disconnectETController (sock)
```

Note: Once the force control mode is started, it is impossible to end the mode during the robotic movement and the robot will stop running after sending this command.
This command is applicable to v3.6.2 and above.

### 2.2.16.3 Get the state of the force control mode

```
{"jsonrpc":"2.0","method":"get_force_mode_state","id":id}
```

Function: End the force control mode

Parameter: none

Return: True means that the robot is in the force control mode, false means that it is not in the force control mode

Example:

```
if __name__ == "__main__":
    # Robot IP address
    robot_ip="172.16.11.248"
    conSuc,sock=connectETController(robot_ip)
    if (conSuc):
        suc, result ,id=sendCMD(sock,"get_force_mode_state")
        print(suc, result ,id)
    else:
        print ("Connection failed")
    disconnectETController(sock)
```

Note: This command is applicable to v3.6.2 and above.

### 2.2.16.4 Clear all values of the force sensor to zero

```
{"jsonrpc":"2.0","method":"zero_force_sensor","id":id}
```

Function: Clear all values of the force sensor to zero and record the current force sensor data

Parameter: none

Return: True for success, false for failure

Example:

```
if __name__ == "__main__":
  # Robot IP address
  robot_ip="192.168.1.240"
  conSuc,sock=connectETController( robot_ip )
  point = []
  point.append([0, -107.828, 119.349, -109.754, 83.351, -5.531, 0, 0])
  point.append([10, -98.342, 51.251, -92.155, 90, 0, 0, 0])
  point.append([-10.60024752475249, -80.04362623762376, 135.67295792079207,
  -193.44984567901236, 123.90856481481481, 24.090277777777775])
  point.append([47.40996287128712, -51.567450495049506, 49.92419554455446,
  -88.9988425925926, 79.45138888888889, 160.08603395061726])
  if (conSuc):
    # Get the servo status of the robotic arm
    suc, result , id=sendCMD(sock,"getServoStatus")
    print (suc, result, id )
    if (result = =0):
      # Set the servo status of the robotic arm to ON
      suc, result , id=sendCMD(sock,"set_servo_status",{"status":1})
      print (suc, result, id)
    # Joint movement to the starting point
    suc, result , id=sendCMD(sock,"moveByJoint",{"targetPos": point[0],"speed":10,"acc":20,"dec":
    20})
    print (result):
    time.sleep(1)
    while 1:
      ret, result, id=sendCMD(sock,"getRobotState"):
      if result==0:
        break
    # Start the force control mode
    suc, result , id=sendCMD(sock,"start_force_mode",{"mode":0,"arr_frame":[0,0,0,0,0,0],
    "arr_optional":[0,0,1,0,0,0], "arr_torque":[0,0,0,0,0,0],"arr_speed":[100,100,100,5.73,5.73,5.73]
    })
    print (suc, result, id)
    # Clear all values to zero
    ret, result , id=sendCMD(sock,"zero_force_sensor")
    print ( "Clear all values to zero:", result )
    suc, result , id=sendCMD(sock,"moveByLine",{"targetPos": point[1],"speed":100,"speed_type":
```

```
    0, "acc":80,"dec":80})
    print (result):
    while 1:
        ret,  result,  id=sendCMD(sock,"getRobotState"):
        if result==0:
            break
    # End the force control mode
    suc, result , id=sendCMD(sock,"end_force_mode")
    print (suc, result, id)
else :
    print ("Connection failed")
disconnectETController (sock)
```

Note: The current force sensor data will be read & saved when inserting the command after STARTFORCEMODE. In the process of the force control, the current readings will subtract what is previously saved. The command is only valid when selecting LUA, SDK and Back-end data source in the force control mode. The command is only supported in remote mode. In the force control mode, the command will be invalid after sending the command end_force_mode.

## 2.3 Examples

### 2.3.1 Example 1

```
1  import socket
2  import json
3  import time
4  import random
5
6  def connectETController(ip,port=8055):
7      sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
8      try:
9          sock.connect((ip,port))
10         return (True,sock)
11     except Exception as e:
12         sock.close()
13         return (False,None)
14
15 def disconnectETController(sock):
16     if(sock):
17         sock.close()
18         sock=None
```

```
        else:
            sock=None

def sendCMD(sock,cmd,params=None,id=1):
    if(not params):
        params=[]
    else:
        params=json.dumps(params)
    sendStr="{{\"method\":\"{0}\",\"params\":{1},\"jsonrpc\":\"2.0\",\"
       id\":{2}}}".format(cmd,params,id)+"\n"
    try:
        sock.sendall(bytes(sendStr,"utf-8"))
        ret =sock.recv(1024)
        jdata=json.loads(str(ret,"utf-8"))
        if("result" in jdata.keys()):
            return (True,json.loads(jdata["result"]),jdata["id"])
        elif("error" in jdata.keys()):
            return (False,jdata["error"],jdata["id"])
        else:
            return (False,None,None)
    except Exception as e:
        return (False,None,None)

if __name__ == "__main__":
    # Robot IP address
    robot_ip="192.168.1.202"
    conSuc,sock=connectETController(robot_ip)
    print(conSuc)
    if(conSuc):
        # Clear alarm
        ret, result, id = sendCMD(sock, "clearAlarm")
        print("Clear alarm")
        print("ret = ", ret, "", "id = ", id)
        if (ret == True):
            print("result = ", result)
            time.sleep(1)
        else:
            print("err_msg = ", result["message"])
        # Get synchronization status
        ret, result, id = sendCMD(sock, "getMotorStatus")
        print("Get synchronization status")
        print("ret = ", ret, "", "id = ", id)
        if (ret == True):
            print("result = ", result)
            if(result != 1):
```

```
                # Synchronize
                ret1, result1, id = sendCMD(sock, "syncMotorStatus")
                print("synchronization")
                print("ret = ", ret1, "", "id = ", id)
                if (ret1 == True):
                    print("result = ", result1)
                    time.sleep(0.5)
                else:
                    print("err_msg = ", result1["message"])
        else:
            print("err_msg = ", result["message"])


        # Turn on the servo
        ret, result, id = sendCMD(sock, "set_servo_status", {"status"
            :1})
        print("Turn on the servo")
        print("ret = ", ret, "", "id = ", id)
        if (ret == True):
            print("result = ", result)
            time.sleep(1)
        else:
            print("err_msg = ", result["message"])
        # Get servo status
        ret, result, id = sendCMD(sock, "getServoStatus")
        print("ret = ", ret, "", "id = ", id)
        if(ret == True):
            print("result = ", result)
        else:
            print("err_msg = ", result["message"])
    else:
        print("Connection failed")
    disconnectETController(sock)
```

### 2.3.2 Example 2

```
import socket
import json
import time


# v1.2
```

```
7
8  def connectETController(ip, port=8055):
9      sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
10     try:
11         sock.connect((ip, port))
12         return (True, sock)
13     except Exception as e:
14         sock.close()
15         return (False, None)
16
17
18 def disconnectETController(sock):
19     if (sock):
20         sock.close()
21         sock = None
22     else:
23         sock = None
24
25
26 def sendCMD(sock, cmd, params=None, id=1):
27     if (not params):
28         params = []
29     else:
30         params = json.dumps(params)
31     sendStr = "{{\"method\":\"{0}\",\"params\":{1},\"jsonrpc
       \":\"2.0\",\"id\":{2}}}".format(cmd, params, id) + "\n"
32     try:
33         # print(sendStr)
34         sock.sendall(bytes(sendStr, "utf-8"))
35         ret = sock.recv(1024)
36         jdata = json.loads(str(ret, "utf-8"))
37         if ("result" in jdata.keys()):
38             return (True, json.loads(jdata["result"]), jdata["id"])
39         elif ("error" in jdata.keys()):
40             return (False, jdata["error"], jdata["id"])
41         else:
42             return (False, None, None)
43     except Exception as e:
44         return (False, None, None)
45
46
47 if __name__ == "__main__":
48     ip = "192.168.1.205"
```

```
    conSuc, sock = connectETController(ip)
    # print(conSuc)
    if (conSuc):
        # Get robot status
        ret, result, id = sendCMD(sock, "getRobotState")
        print("Get robot status")
        print("ret = ", ret, " ", "id = ", id)
        if (ret == True):
            print("result = ", result)
        else:
            print("err_msg = ", result["message"])
        # Get robot mode
        ret, result, id = sendCMD(sock, "getMotorStatus")
        print("Get robot mode")
        print("ret = ", ret, " ", "id = ", id)
        if (ret == True):
            print("result = ", result)
        else:
            print("err_msg = ", result["message"])
        # Get the current position information of the robot
        ret, result, id = sendCMD(sock, "get_joint_pos")
        print("Get the current position information of the robot")
        print("ret = ", ret, " ", "id = ", id)
        if (ret == True):
            print("result = ", result)
        else:
            print("err_msg = ", result["message"])
        # Get the current pose information of the robot
        print("Get the current pose information of the robot")
        ret, result, id = sendCMD(sock, "get_tcp_pose")
        print("ret = ", ret, " ", "id = ", id)
        if (ret == True):
            print("result = ", result)
        else:
            print("err_msg = ", result["message"])
        # Get the value of analog input
        ret, result, id=sendCMD(sock, "getAnalogInput", {"addr":1})
        print("Get the value of analog input")
        print("ret = ", ret, " ", "id = ", id)
        if(ret == True):
            print("result = ", result)
        else:
            print("err_msg = ", result["message"])
```

```
92      else:
93          print("Connection failed")
94      disconnectETController(sock)
```

### 2.3.3 Example 3

```
1   import socket
2   import json
3   import time
4
5
6
7   def connectETController(ip, port=8055):
8       sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
9       try:
10          sock.connect((ip, port))
11          return (True, sock)
12      except Exception as e:
13          sock.close()
14          return (False, None)
15
16
17  def disconnectETController(sock):
18      if (sock):
19          sock.close()
20          sock = None
21      else:
22          sock = None
23
24
25  def sendCMD(sock, cmd, params=None, id=1):
26      if (not params):
27          params = []
28      else:
29          params = json.dumps(params)
30      sendStr = "{{\"method\":\"{0}\",\"params\":{1},\"jsonrpc
            \":\"2.0\",\"id\":{2}}}".format(cmd, params, id) + "\n"
31      try:
32          # print(sendStr)
33          sock.sendall(bytes(sendStr, "utf-8"))
34          ret = sock.recv(1024)
35          jdata = json.loads(str(ret, "utf-8"))
```

```
36          if ("result" in jdata.keys()):
37              return (True, json.loads(jdata["result"]), jdata["id"])
38          elif ("error" in jdata.keys()):
39              return (False, jdata["error"], jdata["id"])
40          else:
41              return (False, None, None)
42      except Exception as e:
43          return (False, None, None)
44
45
46  if __name__ == "__main__":
47      ip = "192.168.1.205"
48      conSuc, sock = connectETController(ip)
49      # print(conSuc)
50      if (conSuc):
51          # Switch the current tool number to 0
52          ret, result, id = sendCMD(sock, "setToolNumber", {"tool_num":
                0})
53          print("Switch the current tool number to 0")
54          print("ret = ", ret, " ", "id = ", id)
55          if (ret == True):
56              print("result = ", result)
57              time.sleep(3)
58          else:
59              print("err_msg = ", result["message"])
60          # Set the load and center of gravity of the robotic arm
61          ret, result, id = sendCMD(sock, "cmd_set_payload", {"tool_num":
                0, "m": 6, "cog": [20.2, 40, 30.5]})
62          print("Set the load and center of gravity of the robotic arm")
63          print("ret = ", ret, " ", "id = ", id)
64          if (ret == True):
65              print("result = ", result)
66          else:
67              print("err_msg = ", result["message"])
68          # Set up the tool center of the robotic arm
69          point1 = [1.002, -2.5, 5.0, 0.74, -1.57, 0]
70          ret, result, id = sendCMD(sock, "cmd_set_tcp", {"tool_num": 0,
                "point": point1})  # cmd_set_tcp
71          print("ret = ", ret, " ", "id = ", id)
72          if (ret == True):
73              print("result = ", result)
74          else:
75              print("err_msg = ", result["message"])
```

```
76      else:
77          print("Connection failed")
78      disconnectETController(sock)
```

# Chapter 3 Monitor Interface

Users can get robot information by connecting to monitor interface of the robot through the socket client.

**CAUTION**

This function is applicable to 2.13.1 and above versions.

**CAUTION**

8056 is a (tcp) long-connection interface. If the access time exceeds 8ms, the data will pile up in the buffer zone.

## 3.1 Data description list of monitor interface

| Name | Type | Byte | Description |
|---|---|---|---|
| Message Size | unsigned int32 | 4*1 | The current packet length is 1024, the effective field length is 800, the reserved field length is 220, and the parity field length is 4 |
| timestamp | unsigned int64 | 8*1 | timestamp, the number of milliseconds since January 1, 1970 |
| autorun_cycelMode | unsigned char | 1*1 | cycle mode, 0: single step, 1: single cycle, 2: continuous |
| machinePos | double[AXIS_COUNT] | 8*8 | joint angle, unit degree |
| machinePose | Double[6] | 8*6 | Base coordinates, the first three items are in millimeters, the last three items are in radians |

| Name | Type | Byte | Description |
|---|---|---|---|
| machineUserPose | Double[6] | 8*6 | Current user coordinates, the first three items are in millimeters, the last three items are in radians |
| torque | double[AXIS_COUNT] | 8*8 | The ratio per thousand of the rated torque of the joint, unit ‰, |
| robotState | int32_t | 4*1 | Robot state: 0: stop, 1: pause, 2: emergency stop, 3: run, 4: alarm |
| servoReady | int32_t | 4*1 | Brake status: 0: not open, 1: open. |
| can_motor_run | int32_t | 4*1 | Synchronization status: 0: not synchronized, 1: synchronized |
| motor_speed | int[AXIS_COUNT] | 4*8 | Motor speed, unit: rpm |
| robotMode | int32_t | 4*1 | Robot mode: 0: teaching mode, 1: automatic mode, 2: remote mode |
| analog_ioInput | double[ANALOG_IN_NUM] | 8*3 | Analog input port data, unit V |
| analog_ioOutput | double[ANALOG_OUT_NUM] | 8*5 | Analog output port data, unit V |
| digital_ioInput | unsigned int64 | 8*1 | digital input port data |
| digital_ioOutput | unsigned int64 | 8*1 | digital output port data |
| collision | unsigned char | 1*1 | collision alarm state, 0: non-collision alarm state, 1: collision alarm state. |
| machineFlangePose | Double[6] | 8*6 | The pose of the flange center in the Base coordinate system, the first three items are in millimeters, and the last three are in radians. |
| machineUserFlangePose | Double[6] | 8*6 | The pose of the flange center in the user coordinate system, the first three items are in millimeters, and the last three are in radians. v2.14.4 added |
| emergencyStopState | unsigned char | 1*1 | Whether it is an emergency stop state, v2.16.2 added |
| tcpSpeed | Double | 8*1 | tcp movement speed, unit mm/s, v2.16.2 added |
| jointSpeed | Double[AXIS_COUNT] | 8*8 | joint speed, unit degree/s, v2.16.2 added |

| Name | Type | Byte | Description |
|---|---|---|---|
| tcpAcc | Double | 8*1 | tcp acceleration, unit: mm/$s^2$ |
| jointAcc | Double[ANALOG_OUT_NUM] | 8*8 | joint acceleration, unit: degree/$s^2$ |
| jointTemperature | Double[6] | 8*6 | joint temperature, unit: °C |
| jointTorque | Double[6] | 8*6 | joint output torque, unit: Nm |
| extJointTorques | Double[6] | 8*6 | estimated external joint torque value, unit: Nm |
| exTcpForceInTool | Double[6] | 8*6 | estimated external end force/moment of force in the current tool coordinate system, unit of the first three items: N, unit of the last thee items: Nm |
| dragState | unsigned char | 1*1 | drag state, 1: enabled, 0: diabled |
| Reserved | unsigned char | 1*220 | The reserved length of the data packet, the length is 220 |
| matchingWord | unsigned int | 4*1 | parity bits, the last 4 bytes of the data packet, it is valid only when the value is 3967833836 (i.e. 0xec8056ec). Otherwise, the data will be invalid. |

## 3.2 Example

```
1  import socket
2  import struct
3  import collections
4  import time
5  import math
6  HOST = "192.168.1.202"
7  PORT = 8056
8
9  while 1:
10       s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
11       s.settimeout(8)
```

```
        s.connect((HOST,PORT))
        index = 0
        lost = 0

        while True:
                dic = collections.OrderedDict()
                dic['MessageSize'] = 'I'
                dic['TimeStamp'] = 'Q'
                dic['autorun_cycleMode'] = 'B'
                dic['machinePos01'] = 'd'
                dic['machinePos02'] = 'd'
                dic['machinePos03'] = 'd'
                dic['machinePos04'] = 'd'
                dic['machinePos05'] = 'd'
                dic['machinePos06'] = 'd'
                dic['machinePos07'] = 'd'
                dic['machinePos08'] = 'd'
                dic['machinePose01'] = 'd'
                dic['machinePose02'] = 'd'
                dic['machinePose03'] = 'd'
                dic['machinePose04'] = 'd'
                dic['machinePose05'] = 'd'
                dic['machinePose06'] = 'd'
                dic['machineUserPose01'] = 'd'
                dic['machineUserPose02'] = 'd'
                dic['machineUserPose03'] = 'd'
                dic['machineUserPose04'] = 'd'
                dic['machineUserPose05'] = 'd'
                dic['machineUserPose06'] = 'd'
                dic['torque01'] = 'd'
                dic['torque02'] = 'd'
                dic['torque03'] = 'd'
                dic['torque04'] = 'd'
                dic['torque05'] = 'd'
                dic['torque06'] = 'd'
                dic['torque07'] = 'd'
                dic['torque08'] = 'd'
                dic['robotState'] = 'i'
                dic['servoReady'] = 'i'
                dic['can_motor_run'] = 'i'
                dic['motor_speed01'] = 'i'
                dic['motor_speed02'] = 'i'
                dic['motor_speed03'] = 'i'
```

```
            dic['motor_speed04'] = 'i'
            dic['motor_speed05'] = 'i'
            dic['motor_speed06'] = 'i'
            dic['motor_speed07'] = 'i'
            dic['motor_speed08'] = 'i'
            dic['robotMode'] = 'i'
            dic['analog_ioInput01'] = 'd'
            dic['analog_ioInput02'] = 'd'
            dic['analog_ioInput03'] = 'd'
            dic['analog_ioOutput01'] = 'd'
            dic['analog_ioOutput02'] = 'd'
            dic['analog_ioOutput03'] = 'd'
            dic['analog_ioOutput04'] = 'd'
            dic['analog_ioOutput05'] = 'd'
            dic['digital_ioInput'] = 'Q'
            dic['digital_ioOutput'] = 'Q'
            dic['collision'] = 'B'
            dic['machineFlangePose01'] = 'd'
            dic['machineFlangePose02'] = 'd'
            dic['machineFlangePose03'] = 'd'
            dic['machineFlangePose04'] = 'd'
            dic['machineFlangePose05'] = 'd'
            dic['machineFlangePose06'] = 'd'
            dic['machineUserFlangePose01'] = 'd'
            dic['machineUserFlangePose02'] = 'd'
            dic['machineUserFlangePose03'] = 'd'
            dic['machineUserFlangePose04'] = 'd'
            dic['machineUserFlangePose05'] = 'd'
            dic['machineUserFlangePose06'] = 'd'
            dic["emergencyStopState"] = "B"
            dic["tcp_speed"] = "d"
            dic["joint_speed01"] = "d"
            dic["joint_speed02"] = "d"
            dic["joint_speed03"] = "d"
            dic["joint_speed04"] = "d"
            dic["joint_speed05"] = "d"
            dic["joint_speed06"] = "d"
            dic["joint_speed07"] = "d"
            dic["joint_speed08"] = "d"
            dic["tcpacc"] = "d"
            dic["jointacc01"] = "d"
            dic["jointacc02"] = "d"
            dic["jointacc03"] = "d"
```

```
98              dic["jointacc04"] = "d"
99              dic["jointacc05"] = "d"
100             dic["jointacc06"] = "d"
101             dic["jointacc07"] = "d"
102             dic["jointacc08"] = "d"
103             dic["joint_temperature01"] = "d"
104             dic["joint_temperature02"] = "d"
105             dic["joint_temperature03"] = "d"
106             dic["joint_temperature04"] = "d"
107             dic["joint_temperature05"] = "d"
108             dic["joint_temperature06"] = "d"
109             dic["joint_torque01"] = "d"
110             dic["joint_torque02"] = "d"
111             dic["joint_torque03"] = "d"
112             dic["joint_torque04"] = "d"
113             dic["joint_torque05"] = "d"
114             dic["joint_torque06"] = "d"
115             dic["extjoint_torques01"] = "d"
116             dic["extjoint_torques02"] = "d"
117             dic["extjoint_torques03"] = "d"
118             dic["extjoint_torques04"] = "d"
119             dic["extjoint_torques05"] = "d"
120             dic["extjoint_torques06"] = "d"
121             dic["exttcpforceintool01"] = "d"
122             dic["exttcpforceintool02"] = "d"
123             dic["exttcpforceintool03"] = "d"
124             dic["exttcpforceintool04"] = "d"
125             dic["exttcpforceintool05"] = "d"
126             dic["exttcpforceintool06"] = "d"
127             dic["dragState"] = "B"
128
129             print("index =", index)
130             data = s.recv (1024)
131             if len(data) != 1024:
132                   lost += 1
133                   print(str(lost))
134                   continue
135
136             names =[]
137             ii=range(len(dic))
138             for key ,i in zip(dic ,ii):
139                   fmtsize = struct. calcsize (dic[key ])
140                   data1 , data = data [0: fmtsize], data[fmtsize :]
```

```
                fmt="!" + dic[key]
                names.append(struct.unpack(fmt ,data1))
                dic[key] = dic[key], struct.unpack(fmt , data1)
        output = ""
        for key in dic.keys ():
                output += str(key) + ":" + str(dic[key ][1][0]) + ";\
                    n"
        output = "lost : " + str(lost) + " index : " + str(index) +
            ";" + output + "\n"
        if dic['MessageSize'] != ('I', (1024,)):
                s.close()
                break
        matchingWord = struct.unpack('!I', data[-4:])[0]
        if matchingWord != 3967833836:
                continue

        if index %10 == 0:
                # Print all information
                print(output)
                # Print packet length
                print(dic['MessageSize'])
                # Print timestamp
                timestamp01_value = dic['TimeStamp'][1][0] // 1000
                timeValue = time.gmtime(int(timestamp01_value))
                print(time.strftime("%Y-%m-%d %H:%M:%S", timeValue))
                # Print joint coordinates
                print(dic['machinePos01'][1][0], dic['machinePos02'
                    ][1][0],
                        dic['machinePos03'][1][0], dic['machinePos04'
                            ][1][0],
                        dic['machinePos05'][1][0], dic['machinePos06'
                            ][1][0],
                        dic['machinePos07'][1][0], dic['machinePos08'
                            ][1][0])
                # Print base coordinates
                print(dic['machinePose01'][1][0], dic['machinePose02'
                    ][1][0], dic['machinePose03'][1][0],
                        dic['machinePose04'][1][0],
                        dic['machinePose05'][1][0], dic['machinePose06'
                            ][1][0])
                # Print user coordinates
                print(dic['machineUserPose01'][1][0], dic['
                    machineUserPose02'][1][0], dic['machineUserPose03'
                    ][1][0],
```

```
175                     dic['machineUserPose04'][1][0],
176                     dic['machineUserPose05'][1][0], dic['
                            machineUserPose06'][1][0])
177                 # Print the percentage of joint rated torque
178                 print(dic['torque01'][1][0], dic['torque02'][1][0],
                        dic['torque03'][1][0], dic['torque04'][1] [0],
179                     dic['torque05'][1][0],
180                     dic['torque06'][1][0], dic['torque07'][1][0],
                            dic['torque08'][1][0])
181                 # Print robot status
182                 print(dic['robotState'][1][0])
183                 # Print servo enable status
184                 print(dic['servoReady'][1][0])
185                 # Print synchronization status
186                 print(dic['can_motor_run'][1][0])
187                 # Print the motor speed of each axis
188                 print(dic['motor_speed01'][1][0], dic['motor_speed02'
                        ][1][0], dic['motor_speed03'][1][0],
189                     dic['motor_speed04'][1][0], dic['motor_speed05'
                            ][1][0],
190                     dic['motor_speed06'] [1][0], dic['motor_speed07
                            '][1][0], dic['motor_speed08'][1][0])
191                 # Print robot mode
192                 print(dic['robotMode'][1][0])
193                 # Print analog input port data
194                 print(dic['analog_ioInput01'][1][0], dic['
                        analog_ioInput02'][1][0], dic['analog_ioInput03'
                        ][1][0])
195                 # Print analog output port data
196                 print(dic['analog_ioOutput01'][1][0], dic['
                        analog_ioOutput02'][1][0], dic['analog_ioOutput03'
                        ][1][0],
197                     dic['analog_ioOutput04'][1][0], dic['
                            analog_ioOutput05'][1][0])
198                 # Print the binary form of the digital input port
                        data
199                 print(bin(dic['digital_ioInput'][1][0])[2:].zfill(64)
                        )
200                 # Print the binary form of the digital output port
                        data
201                 print(bin(dic['digital_ioOutput'][1][0])[2:].zfill
                        (64))
202                 # Print collision alarm status
203                 print(dic["collision"][1][0])
```

```
204                     # Print the center pose of the flange in the base
                            coordinate system
205                     print(dic['machineFlangePose01'][1][0], dic['
                            machineFlangePose02'][1][0], dic['
                            machineFlangePose03'][1][0],
206                           dic['machineFlangePose04'][1][0],
207                           dic['machineFlangePose05'][1][0], dic['
                                  machineFlangePose06'][1][0])
208                     # Print the center pose of the flange in the user
                            coordinate system
209                     print(dic['machineUserFlangePose01'][1][0], dic['
                            machineUserFlangePose02'][1][0],
210                           dic['machineUserFlangePose03'][1][0], dic['
                                  machineUserFlangePose04'][1][0],
211                           dic['machineUserFlangePose05'][1][0], dic['
                                  machineUserFlangePose06'][1][0])
212                     # Print whether it is currently in emergency stop
                            state
213                     print(dic["emergencyStopState"][1][0])
214                     # Print tcp movement speed
215                     print(dic["tcp_speed"][1][0])
216                     # Print the speed of each joint under joint motion
217                     print(dic['joint_speed01'][1][0], dic['joint_speed02'
                            ][1][0],
218                           dic['joint_speed03'][1][0], dic['joint_speed04'
                                  ][1][0],
219                           dic['joint_speed05'][1][0], dic['joint_speed06'
                                  ][1][0],
220                           dic['joint_speed07'][1][0], dic['joint_speed08'
                                  ][1][0])
221                     # Print tcp acceleration
222                     print(dic["tcpacc"][1][0])
223                     # Print the acceleration of each joint under joint
                            movement
224                     print(dic['jointacc01'][1][0], dic['jointacc02'
                            ][1][0],
225                           dic['jointacc03'][1][0], dic['jointacc04'
                                  ][1][0],
226                           dic['jointacc05'][1][0], dic['jointacc06'
                                  ][1][0],
227                           dic['jointacc07'][1][0], dic['jointacc08'
                                  ][1][0])
228                     # Print temperature
```

```
229                     print(dic['joint_temperature01'][1][0], dic['
                            joint_temperature02'][1][0],
230                           dic['joint_temperature03'][1][0], dic['
                                  joint_temperature04'][1][0],
231                           dic['joint_temperature05'][1][0], dic['
                                  joint_temperature06'][1][0])
232                     # Print output torque
233                     print(dic['joint_torque01'][1][0], dic['
                            joint_torque02'][1][0],
234                           dic['joint_torque03'][1][0], dic['
                                  joint_torque04'][1][0],
235                     dic['joint_torque05'][1][0], dic['joint_torque06'
                            ][1][0])
236                     # Print the estimated external joint torque value
237                     print(dic['exjoint_torques01'][1][0], dic['
                            exjoint_torques02'][1][0],
238                           dic['exjoint_torques03'][1][0], dic['
                                  exjoint_torques04'][1][0],
239                           dic['exjoint_torques05'][1][0], dic['
                            exjoint_torques06'][1][0])
240                     # Print the estimated external end force/moment of force
                        in the current tool coordinate system
241                     print(dic['extcpforceintool01'][1][0], dic['
                            extcpforceintool02'][1][0],
242                           dic['extcpforceintool03'][1][0], dic['
                                  extcpforceintool04'][1][0],
243                           dic['extcpforceintool05'][1][0], dic['
                            extcpforceintool06'][1][0])
244                     # Print the drag-enabling state
245                     print(dic['dragState'][1][0])
246                 index = index +1
247                 output = ""
248                 dic = {}
249                 data = ""
250                 time.sleep(0.008)
251         s.close ()
```

# Chapter 4 Log Interface

Users can connect to the robot log interface through the socket client.

The log types are: Error, Warning, Info. If the Error type is entered, Error information will be obtained; If the Warning type is entered, the log information of Error and Warning types will be obtained; If you enter Info, all types of log information will be obtained.

After connecting, enter all to enter all logs; enter a number, such as 10, to output the last 10 lines of logs; enter exit to exit the connection.

## 4.1 Example

```python
import socket
HOST = "192.168.1.202"
PORT = 8058

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.settimeout(2)
s.connect((HOST,PORT))
# New file or empty file content
# file = open(r'D:\205\log\all_err_log.md','w').close()
# Get all log information
str1 = "Type=Info\n"
s.send(str1.encode())
str2 = "all\n"
s.send(str2.encode())
while True:
    try:
        data = s.recv(128000)
        # with open(r'D:\205\log\all_err_log.md','a+') as f:
        # f.write(data.decode())
        print(data.decode())
    except(Exception):
         break
s.close()
```

# Chapter 5 Raw Log Interface

Users can connect to the raw log interface of the robot through the socket client.

After connecting, enter all to enter all logs; enter a number, such as 10, to output the last 10 lines of logs; enter exit to exit the connection.

**CAUTION**

This function is applicable to 2.14.0 and above.

## 5.1 Example

```python
import socket
HOST = "192.168.1.205"
PORT = 8059

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.settimeout(2)
s.connect((HOST,PORT))
# New file or empty file content
file = open(r'D:\205\log\err_log_1.md','w').close()

# Get the last 10 log entries, the number sent corresponds to the number of log entries obtained
str1 = "10\n"
s.send(str1.encode())
while True:
    try:
        data = s.recv(1024)
        with open(r'D:\205\log\err_log_1.md','a+') as f:
            f.write(data.decode())
        print(data.decode())
    except(Exception):
        break
s.close()
```

ELITE ROBOTS
艾利特机器人

# ALWAYS EASIER THAN BEFORE

## Contact Us

Sales & Service: market@elibot.cn
Technical Support: tech@elibot.cn

### Shanghai

Building 18, Lane 36,
Xuelin Road, Shanghai

### Suzhou

1F, Building 4,
No 259 Changyang Street, Suzhou
+86-400-189-9358
+86-0512-83951898

### Beijing

Room 1102, Building 6, No. 2,
Ronghua South Road, Beijing

### Shenzhen

Room 202, Building 1A,
Hangkong Road, Shenzhen

WeChat
Official Account