AUBO C++ SDK function manual

Version 1.0.1

AUBO (Beijing) Robotics Technology Co, Ltd.

This manual will be checked and revised periodically, and the updated content will appear on the latest version.

The information in this manual is subject to change without notice and should not be regarded as a commitment by AUBO (Beijing) Robotics Technology Co., Ltd.

AUBO (Beijing) Robotics Technology Co., Ltd. assumes no responsibility for any errors or omissions in this document. AUBO (Beijing) Robotics Technology Co., Ltd. assumes no responsibility for incidental or consequential damages arising from use of this manual and products described herein.

Please read this manual before install or use.

Please keep this manual to read and as reference any time.

The pictures in this manual are for reference only, please refer to the actual product received.

Content

Introduction

The AUBO API is implemented based on the network and provides a large number of interfaces for operating the robotic arm, including login and logout, joint motion, linear motion, rotational motion, and more. This file defines the interface class for using the manipulator, which is developed based on C++, and the method in the class is the interface for operating the manipulator.

These interfaces can be applied to realize the use of 2D smart cameras, 3D vision integration, implementation of palletizing examples, software based force control mode, tool end control force control and other functions.

This manual contains five sections. The first chapter introduces data types, the second chapter introduces the definition of interface functions, the third chapter introduces the error codes, the fourth chapter introduces the use cases of the interface functions, and the fifth chapter introduces the configuration environment.

Note: The unit of length in the interface is meter, and the unit of angle is radian.

1 Data Types

1.1 Robotic arm motion Data types

```
/**
 * @brief Number of Arm Joints
 **/
enum {
    ARM_DOF = 6,
};
```

```
/**
  * @brief DH Parameters
  **/
typedef struct
{
    double A3;
    double A4;
    double D1;
    double D2;
    double D5;
    double D6;

    // general dh model
    double alpha[ARM_DOF];
    double a[ARM_DOF];
    double d[ARM_DOF];
    double theta[ARM_DOF];
}RobotDhPara;
```

```
/**
 * @brief Position info
 **/
struct Pos
{
    double x;
    double y;
    double z;
};
```

```
/**
 * @brief Common body description of position info
```



```
 **/
union cartesianPos_U
{
    Pos     position;
    double  positionVector[3];
};
```

```
/**
 * @brief Four element representation of pose
 **/
struct Ori
{
    double w;
    double x;
    double y;
    double z;
};
```

```
/**
 * @brief Euler angle representation of pose
 **/
struct Rpy
{
    double rx;
    double ry;
    double rz;
};
```

```
typedef struct
{
    double jointPos[ARM_DOF];
}JointParam;
```

```
/**
  * @brief Description of velocity and acceleration of the joint
  */
typedef struct
{
    double jointPara[ARM_DOF];
}JointVelcAccParam;
```

```
/**
 * @brirf Joint collision compensation（Range 0.00~0.51 unit）
```

```
 **/
typedef struct
{
    double jointOffset[ARM_DOF];
}RobotJointOffset;
```

```
/**
 * @brief Waypoint information of manipulator
 **/
typedef struct
{
    cartesianPos_U  cartPos;            // Manipulator position info (x,y,z)
    Ori             orientation;        // Pose info (w,x,y,z)
    double          jointpos[ARM_DOF];  // Joints angle info
}wayPoint_S;
```

```
/**
 * @brief Force sensor data
 **/
typedef struct
{
    double data[6];
}ForceSensorData;
```

```
/**
 * @brief Orientation and Position of manipulator
 **/
typedef struct
{
    Pos position;
    Ori quaternion;
}PositionAndQuaternion;
```

```
/**
 *  @brief Description of the offset attribute in the motion attribute
 */
typedef struct
{
    bool ena;                       // Enable offset
    float relativePosition[3];      // Offset x,y,z
    Ori  relativeOri;               // Orientation offset
}MoveRelative;
```

```
/**
 * @brief Teaching mode enumeration
 **/
enum teach_mode
{
    NO_TEACH = 0,
    JOINT1,
    JOINT2,
    JOINT3,
    JOINT4,
    JOINT5,
    JOINT6,
    MOV_X,
    MOV_Y,
    MOV_Z,
    ROT_X,
    ROT_Y,
    ROT_Z
};
```

```
/**
 * @brief Motion trajectory enumeration
 **/
enum move_track
{
    NO_TRACK = 0,

    //for moveJ and moveL
    TRACKING,

    //cartesian motion for moveP
    ARC_CIR,
    CARTESIAN_MOVEP,
    CARTESIAN_CUBICSPLINE,
    CARTESIAN_UBSPLINEINTP,
    CARTESIAN_GNUBSPLINEINTP,
    CARTESIAN_LOOKAHEAD,

    //joint motion for moveP
    JIONT_CUBICSPLINE,
    JOINT_UBSPLINEINTP,
    JOINT_GNUBSPLINEINTP,

    ARC,
```

```
    CIRCLE,
    ARC_ORI_ROTATED,
    CIRCLE_ORI_ROTATED,

    ORI_POSITION_ROTATE_CIRCUMFERENCE=101,
};
```

```
typedef struct
{
    double jointMaxAcc[ARM_DOF];        //Joints move maximum acceleration
    double jointMaxVelc[ARM_DOF];       //Joints move Maximum speed
    double endMaxLineAcc;               //TCP maximum acceleration
    double endMaxLineVelc;              //TCP maximum speed
    MoveRelative relative;              //Offset parameters
    CoordCalibrateByJointAngleAndTool relativeOnCoord;//Offset coordinates
    double blendRadius;                 //blend radius
    ToolInEndDesc toolInEndDesc;    //Tool properties
}MoveProfile_t;
```

1.2 Manipulator joint status

```
/**
 * Description of joint state of the manipulator
 */
typedef struct PACKED
{
    int     jointCurrentI;          // Current of joint
    int     jointSpeedMoto;         // Speed of join
    float   jointPosJ;      // Joint position in radian
    float   jointCurVol;    // Rated voltage of motor. Unit: mV
    float   jointCurTemp;       // Current temprature of joint
    int     jointTagCurrentI;   // Target current of motor
    float   jointTagSpeedMoto;// Target speed of motor
    float   jointTagPosJ;       // Target position of joint in radian
    uint16  jointErrorNum;      // Joint error num
}JointStatus;
```

1.3 Event Types

```
/**
 * Description of manipulator Event Types
 *
 * A lot of information about the robotic arm (such as faults, notifications) is notified to
```

```
customers through events, so when using the SDK, be sure to register a callback
function that receives events
*/
typedef enum{
    RobotEvent_armCanbusError,// Manipulator can bus error      Obsolete, not
recommended
    RobotEvent_remoteHalt,       //remote shutdown
    RobotEvent_remoteEmergencyStop,     //arm remote emergency stop
    RobotEvent_jointError,               // Joint error, PS: obsolete, not
recommended
    RobotEvent_forceControl,             //Force control
    RobotEvent_exitForceControl,         //Exit from the force control mode
    RobotEvent_softEmergency,            //Soft emergency stop
    RobotEvent_exitSoftEmergency,        //Exit from the soft emergency stop
    RobotEvent_collision,              // Collision, PS: obsolete, not recommended
                                       // Replaced with RobotEventJointCollision(2123)
    RobotEvent_collisionStatusChanged, //Collision state changed, obsolete, not
recommended
                                       //Repelced with RobotEventJointCollision(2123)
    RobotEvent_tcpParametersSucc,    // Tool dynamics parameters set successfully
                                          // System events, users can ignore
    RobotEvent_powerChanged,           // Arm power switch state change
    RobotEvent_ArmPowerOff,        //Arm power off, not recommended to use
                                       //Replaced with RobotEventArmPowerOff(2600)
    RobotEvent_mountingPoseChanged,     //Arm installation position change
    RobotEvent_encoderError,            //encoder error, not recommended
    RobotEvent_encoderLinesError, //encoder lines inconsistent, not recommended to
use
                                  //replaced with RobotEventEncoderLineError（2203）
    RobotEvent_singularityOverspeed,    //Singularity overspeed
    RobotEvent_currentAlarm,            //Arm abnormal current
    RobotEvent_toolioError,             // Manipulator tool end error
    RobotEvent_robotStartupPhase,       //Arm start-up phase
                                        // System events, users can ignore
    RobotEvent_robotStartupDoneResult,  // Arm Startup Completion Result
                                        // System events, users can ignore
    RobotEvent_robotShutdownDone,       //Arm shutdown result
                                        // System events, users can ignore
    RobotEvent_atTrackTargetPos,        //Arm track arrive target position signal
                                        // System events, users can ignore
    RobotSetPowerOnDone,                // Set power on status complete
    RobotReleaseBrakeDone,              //Arm brake release complete
                                        // System events, users can ignore
    RobotEvent_robotControllerStateChaned,     // Robot controller status change
```

```
                                                    // System events, users can ignore
    RobotEvent_robotControllerError,        //Arm controller error,
// It is generally returned when there is a problem with the algorithm planning
    RobotEvent_socketDisconnected,          //socket Disconnect
    RobotEvent_robotControlException,
    RobotEvent_trackPlayInterrupte,
    RobotEvent_staticCollisionStatusChanged,    // Not recommended, obsolete
    RobotEvent_MountingPoseWarning,
    RobotEvent_MacDataInterruptWarning,
    RobotEvent_ToolIoError,
    RobotEvent_InterfacBoardSafeIoEvent,        // Safety IO notification event
    RobotEvent_RobotHandShakeSucc,              // System events, users can ignore
    RobotEvent_RobotHandShakeFailed,            // System events, users can ignore
    RobotEvent_RobotErrorInfoNotify,            // Not recommended, obsolete
    RobotEvent_InterfacBoardDIChanged,          // Notification event DI status change
    RobotEvent_InterfacBoardDOChanged,          // Notification event DO status
change

    RobotEvent_InterfacBoardAIChanged,          // Notification event AI status change

    RobotEvent_InterfacBoardAOChanged,          // Notification event AO status
change

    RobotEvent_UpdateJoint6Rot360Flag,          // System events, users can ignore
    RobotEvent_RobotMoveControlDone,            // System events, users can ignore
    RobotEvent_RobotMoveControlStopDone,    // System events, users can ignore
    RobotEvent_RobotMoveControlPauseDone,  // System events, users can ignore
    RobotEvent_RobotMoveControlContinueDone,// System events, users can ignore

    // Master slave mode switching
    RobotEvent_RobotSwitchToOnlineMaster,      // Notification event, enter the
linkage main mode
    RobotEvent_RobotSwitchToOnlineSlave,       // Notification event, enter linkage
slave mode

    RobotEvent_ConveyorTrackRobotStartup,       // System events, users can ignore
    RobotEvent_ConveyorTrackRobotCatchup,      // System events, users can ignore

    RobotEvent_exceptEvent = 100,
    RobotEventInvalid = 1000,    // Invalid event

/**
 * RobotControllerErrorEvent    Controller exception event 1001~1499
 *
```

```
 * Event handling suggestions
 * Recommended action: stop the current movement
 *
 * PS:   These events can cause false returns of Arm movements
 * When using, try to use enumeration variables. Enumeration variable values
are only for the convenience of viewing logs
 *
 **/
    RobotEventMoveJConfigError                  = 1001,
    // moveJ configuration error 关节运动属性配置错误
    RobotEventMoveLConfigError                  = 1002,
    // moveL configuration error 直线运动属性配置错误
    RobotEventMovePConfigError                  = 1003,
    // moveP configuration error 轨迹运动属性配置错误
    RobotEventInvailConfigError                 = 1004,
    // invalid move configuration 无效的运动属性配置
    RobotEventWaitRobotStopped                  = 1005,
    // please wait robot stopped 等待机器人停止
    RobotEventJointOutRange                     = 1006,
    // joint out of range 超出关节运动范围
    RobotEventFirstWaypointSetError             = 1007,
    // please set first waypoint correctly in movep
    // 请正确设置 MOVEP 第一个路点
    RobotEventConveyorTrackConfigError          = 1008,
    // configuration error for conveyor tracking 传送带跟踪配置错误
    RobotEventConveyorTrackTrajectoryTypeError  = 1009,
    // unsupported conveyor tracking trajectory type 传送带轨迹类型错误
    RobotEventRelativeTransformIKFailed         = 1010,
    // inverse kinematics failure due to invalid relative transform
    // 相对坐标变换逆解失败
    RobotEventTeachModeCollision                = 1011,
    // collision in teach-mode 示教模式发生碰撞
    RobotEventextErnalToolConfigError           = 1012,
    // configuration error for external tool and hand workobject
    // 运动属性配置错误,外部工具或手持工件配置错误
    RobotEventTrajectoryAbnormal                = 1101,
    // Trajectory is abnormal 轨迹异常
    RobotEventOnlineTrajectoryPlanError         = 1102,
    // Trajectory is abnormal,online planning failed  轨迹规划错误
    RobotEventOnlineTrajectoryTypeIIError       = 1103,
    // Trajectory is abnormal, type II online planning failed
    // 二型在线轨迹规划失败
    RobotEventIKFailed                          = 1104,
    // Trajectory is abnormal,inverse kinematics failed 逆解失败
```

```
    RobotEventAbnormalLimitProtect                  = 1105,
    // Trajectory is abnormal, abnormal limit protection 动力学限制保护
    RobotEventConveyorTrackingFailed                = 1106,
    // Trajectory is abnormal, conveyor tracking failed   传送带跟踪失败
    RobotEventConveyorOutWorkingRange               = 1107,
    // Trajectory is abnormal, exceeding the conveyor working range
    // 超出传送带工作范围
    RobotEventTrajectoryJointOutOfRange             = 1108,
    // Trajectory is abnormal, joint out of range 关节超出范围
    RobotEventTrajectoryJointOverspeed              = 1109,
    // Trajectory is abnormal, joint overspeed 关节超速
    RobotEventOfflineTrajectoryPlanFailed           = 1110,
    // Trajectory is abnormal, Offline track planning failed 离线轨迹规划失败
    RobotEventTrajectoryJointAccOutOfRange          = 1111,
    // Trajectory is abnormal, joint acc out of range 轨迹异常,关节加速度超限

    RobotEventForceModeException                    = 1120,   // 力控模式异常
    RobotEventForceModeIKFailed                     = 1121,
    // Trajectory is abnormal,force control mode Ik failed
    // 轨迹异常，力控模式下失败
    RobotEventForceModeTrackJointverspeed           = 1122,
    // Trajectory is abnormal, joint overspeed 关节超速
    RobotEventControllerIKFailed                    = 1200,
    // The controller has an exception and the inverse kinematics failed
    // 控制器异常，逆解失败
    RobotEventControllerStatusException             = 1201,
    // The controller has an exception and the status is abnormal
    // 控制器异常，状态异常
    RobotEventControllerTrackingLost                = 1202,
    // Exception that joint tracking is lost, 关节跟踪误差过大.
    RobotEventMonitorErrTrackingLost                = 1203,
    // Exception that joint tracking is lost, 关节跟踪误差过大.
    RobotEventMonitorErrNoArrivalInTime             = 1204, // not used 预留
    RobotEventMonitorErrCurrentOverload             = 1205, // not used 预留
    RobotEventMonitorErrJointOutOfRange             = 1206,
    // Exception that joint out of range   机械臂关节超出限制范围
    RobotEventMonitorErrFifoDataTimeNotRead         = 1207,
    // Controller fifo data timeout was not read 队列中数据超时未被读取
    RobotEventMoveEnterStopState                    = 1300,
    // Movement enters the stop state 运动进入到 stop 阶段


/**
 * RobotHardwareErrorEvent    Abnormal hardware events 2001~2999
```

```
 *
 * Event handling suggestions
 * RobotEventJointEncoderPollustion        Recommended Action: Warning Notice
 * RobotEventDriveVersionError             Recommended Action: Warning Notice
 * RobotEventJointCollision                Suggested measures: If you need to use the
pause function to resume the current movement, call the collision recovery function
first, and then call the continue function; if you do not need to resume the current
movement, call the stop function, and call the collision recovery function when
recovering to suggest other events. Action to take: Stop the current motion **/

    RobotEventHardwareErrorNotify                = 2001,
    // Robot hardware error  机械臂硬件错误

    RobotEventJointError                                 = 2101,
    // Robot joint error  机械臂关节错误
    RobotEventJointOverCurrent                           = 2102,
    // Robot joint over current.   机械臂关节过流
    RobotEventJointOverVoltage                           = 2103,
    // Robot joint over voltage.  机械臂关节过压
    RobotEventJointLowVoltage                            = 2104,
    // Robot joint low voltage.   机械臂关节欠压
    RobotEventJointOverTemperature                       = 2105,
    // Robot joint over temperature.  机械臂关节过温
    RobotEventJointHallError                             = 2106,
    // Robot joint hall error.  机械臂关节霍尔错误
    RobotEventJointEncoderError                          = 2107,
    // Robot joint encoder error.  机械臂关节编码器错误
    RobotEventJointAbsoluteEncoderError                  = 2108,
    // Robot joint absolute encoder error.  机械臂关节绝对编码器错误
    RobotEventJointCurrentDetectError                    = 2109,
    // Robot joint current position error.  机械臂关节当前位置错误
    RobotEventJointEncoderPollustion                     = 2110,
    // Robot joint encoder pollution.
    //  机械臂关节编码器污染             建议采取措施:警告性通知
    RobotEventJointEncoderZSignalError                   = 2111,
    // Robot joint encoder Z signal error.  机械臂关节编码器 Z 信号错误
    RobotEventJointEncoderCalibrateInvalid               = 2112,
    // Robot joint encoder calibrate invalid.  机械臂关节编码器校准失效
    RobotEventJoint_IMU_SensorInvalid                    = 2113,
    // Robot joint IMU sensor invalid.  机械臂关节 IMU 传感器失效
    RobotEventJointTemperatureSensorError                = 2114,
    // Robot joint temperature sensor error.  机械臂关节温度传感器出错
    RobotEventJointCanBusError                           = 2115,
    // Robot joint CAN BUS error.  机械臂关节 CAN 总线出错
```

```
RobotEventJointCurrentError = 2116,
// Robot joint current error. 机械臂关节当前电流错误
RobotEventJointCurrentPositionError = 2117,
// Robot joint current position error. 机械臂关节当前位置错误
RobotEventJointOverSpeed = 2118,
// Robot joint over speed. 机械臂关节超速
RobotEventJointOverAccelerate = 2119,
// Robot joint over accelerate. 机械臂关节加速度过大错误
RobotEventJointTraceAccuracy = 2120,
// Robot joint trace accuracy. 机械臂关节跟踪精度错误
RobotEventJointTargetPositionOutOfRange = 2121,
// Robot joint target position out of range. 机械臂关节目标位置超范围
RobotEventJointTargetSpeedOutOfRange = 2122,
// Robot joint target speed out of range. 机械臂关节目标速度超范围
RobotEventJointCollision = 2123,
// Robot joint collision. 机械臂碰撞 建议采取措施:暂停当前运动

RobotEventDataAbnormal = 2200,
// Robot data abnormal 机械臂信息异常
RobotEventRobotTypeError = 2201,
// Robot type error 机械臂类型错误
RobotEventAccelerationSensorError = 2202,
// Robot acceleration sensor error 机械臂加速度计芯片错误
RobotEventEncoderLineError = 2203,
// Robot encoder line error 机械臂编码器线数错误
RobotEventEnterDragAndTeachModeError = 2204,
// Robot enter drag and teach mode error 机械臂进入拖动示教模式错误
RobotEventExitDragAndTeachModeError = 2205,
// Robot exit drag and teach mode error 机械臂退出拖动示教模式错误
RobotEventMACDataInterruptionError = 2206,
// Robot MAC data interruption error 机械臂 MAC 数据中断错误
RobotEventDriveVersionError = 2207,
// Drive version error 驱动器版本错误(关节固件版本不一致)

RobotEventInitAbnormal = 2300,
// Robot init abnormal 机械臂初始化异常
RobotEventDriverEnableFailed = 2301,
// Robot driver enable failed 机械臂驱动器使能失败
RobotEventDriverEnableAutoBackFailed = 2302,
// Robot driver enable auto back failed 机械臂驱动器使能自动回应失败
RobotEventDriverEnableCurrentLoopFailed = 2303,
// Robot driver enable current loop failed 机械臂驱动器使能电流环失败
RobotEventDriverSetTargetCurrentFailed = 2304,
// Robot driver set target current failed 机械臂驱动器设置目标电流失败
```

```
    RobotEventDriverReleaseBrakeFailed                  = 2305,
    // Robot driver release brake failed    机械臂释放刹车失败
    RobotEventDriverEnablePostionLoopFailed             = 2306,
    // Robot driver enable postion loop failed    机械臂使能位置环失败
    RobotEventSetMaxAccelerateFailed                    = 2307,
    // Robot set max accelerate failed    设置最大加速度失败

    RobotEventSafetyError                               = 2400,
    // Robot Safety error    机械臂安全出错
    RobotEventExternEmergencyStop                       = 2401,
    // Robot extern emergency stop    机械臂外部紧急停止
    RobotEventSystemEmergencyStop                       = 2402,
    // Robot system emergency stop    机械臂系统紧急停止
    RobotEventTeachpendantEmergencyStop                 = 2403,
    // Robot teachpendant emergency stop    机械臂示教器紧急停止
    RobotEventControlCabinetEmergencyStop               = 2404,
    // Robot control cabinet emergency stop    机械臂控制柜紧急停止
    RobotEventProtectionStopTimeout                     = 2405,
    // Robot protection stop timeout    机械臂保护停止超时
    RobotEventEeducedModeTimeout                        = 2406,
    // Robot reduced mode timeout    机械臂缩减模式超时

    RobotEventSystemAbnormal                            = 2500,
    // Robot systen abnormal    机械臂系统异常
    RobotEvent_MCU_CommunicationAbnormal                = 2501,
    // Robot mcu communication error    机械臂 mcu 通信异常
    RobotEvent485CommunicationAbnormal                  = 2502,
    // Robot RS485 communication error    机械臂 485 通信异常

//#if 0 // Non-real-time version
    RobotEventCurrentJointOutRange                      = 2550,
    // Joint out of Range
//#else
    RobotEventSoftEmergency                             = 2550, //Soft emergency
    RobotEventSoftEmergencyExit                         = 2551,     // Exit
//#endif

    RobotEventArmPowerOff                               = 2600,
    // Disconnecting the contactor causes the arm 48V power off
    //  控制柜接触器断开导致机械臂 48V 断电

    RobotEventHardwareErrorNotifyMaximumIndex           = 2999,    // index

    RobotEventNotifyEvent                               = 3000,
```

```
    // Robot notification event  机械臂通知性事件
    RobotEventNotifyCollisionLevelChange                = 3001,
    // Robot Collision level change  机械臂事件通知-碰撞等级被改变
    RobotEventNotifyEnterFlexibleControlMode            = 3010,
    // Notification of entering flexible control mode
    RobotEventNotifyExitFlexibleControlMode             = 3011,
    // Exit
    RobotEventNotifyEnterSpeedReducedMode               = 3015,
    // Enter reduced speed mode notification
    RobotEventNotifyExitSpeedReducedMode                = 3016,
    // Exit

    RobotEventNotifyStopCurrentMove                     = 3100,
    // Stop the current movement



    //unknown event
    robot_event_unknown                                 = 10000,

    //user event
    RobotEvent_User                                     = 9000,
    // first user event id
    RobotEvent_MaxUser                                  = 9999
    // last user event id

}RobotEventType;
```

```
/** Event type **/
typedef struct{
    RobotEventType      eventType;          // Event type number
    int                 eventCode;
    std::string         eventContent;       //Event content
}RobotEventInfo;
```

1.4 Troubleshooting Info

```
/**
 * Robot arm diagnostic information
 **/
typedef struct PACKED
{
    uint8    armCanbusStatus;
// CAN communication status: 0x01~0x80: joint CAN communication error
// (each joint occupies 1bit) 0x00: no error
```

```
    float     armPowerCurrent;
    // Current of 48V power supply of manipulator
    float     armPowerVoltage;
    // Current voltage of 48V power supply of manipulator
    bool      armPowerStatus;
    // Manipulator 48V power status (ON, Off)
    char      contorllerTemp;              // Control box temperature
    uint8     contorllerHumidity;          // Control box humidity
    bool      remoteHalt;                  // Remote shutdown signal
    bool      softEmergency;               // Arm soft emergency stop
    bool      remoteEmergency;             // Remote emergency stop signal
    bool      robotCollision;              // collision detection bit
    bool      forceControlMode;
    // Arm enters the force control mode flag
    bool      brakeStuats;                 // Brake status
    float     robotEndSpeed;               // TCP speed
    int       robotMaxAcc;                 // Maximum Acceleration
    bool      orpeStatus;                  // Software status bit
    bool      enableReadPose;              // Pose read enable bit
    bool      robotMountingPoseChanged;    // Installation position status
    bool      encoderErrorStatus;          // Magnetic encoder error status
    bool      staticCollisionDetect;       // Static Collision Detection Switch
    uint8     jointCollisionDetect;
    // Joint collision detection: each joint occupies 1 bit
    // 0-no collision, 1-collision present
    bool      encoderLinesError;
    // Photoelectric encoder inconsistent error: 0-no error, 1-error
    bool      jointErrorStatus;            // joint error status
    bool      singularityOverSpeedAlarm;   //Arm singularity overspeed alarm
    bool      robotCurrentAlarm;           // Arm current error alarm
    uint8     toolIoError;                 // tool error
    bool      robotMountingPoseWarning;    // Arm mounting position alarm
                                           //(only works in force control mode)
    uint16    macTargetPosBufferSize;      // Mac buffer length, reserved
    uint16    macTargetPosDataSize;
    // Mac buffer, effective data length, reserved
  uint8     macDataInterruptWarning;       // Mac data interrupt, reservation
uint8     controlBoardAbnormalStateFlag;
// Abnormal status flag of main control board (interface board)
}RobotDiagnosis;
```

1.5 Callback function types

```
/**
```

```
 * @brief                    Get real-time joint state callback function type
 * @param jointStatus  Current joint state;
 * @param size               Length of the previous parameter (jointStatus);
 * @param arg                The second parameter passed by the user in the registered
callback function;
 */
typedef void (*RealTimeJointStatusCallback) (const
aubo_robot_namespace::JointStatus *jointStatus, int size, void *arg);
```

```
/**
 * @brief
Callback function type for obtaining real-time waypoint information
 * @param Waypoint          Current waypoint information；
 * @param arg
The second parameter passed by the user in the registration callback function；
 */
typedef void (*RealTimeRoadPointCallback) (const
aubo_robot_namespace::Waypoint_S *Waypoint, void *arg);
```

```
/**
 *@brief                    Callback function type to get real-time TCP speed
 *@param speed              Current TCP speed;
 *@param arg
The second parameter passed by the user in the registered callback function;
 */
typedef void (*RealTimeEndSpeedCallback) (double speed, void *arg);
```

```
/**
 *@brief
Get the callback function type for real-time Movep execution progress
 *@param num                Current Movep execution progress;
 *@param arg                The second parameter passed by the user in the registered
callback function;
 */
typedef void (*RealTimeMovepStepNumNotifyCallback) (int num, void *arg);
```

```
/**
 * @brief                    Callback function type to get the event information of the
manipulator
 * @param arg           The second parameter passed by the user in the registered
callback function;
 */
typedef void (*RobotEventCallback) (const aubo_robot_namespace::RobotEventInfo
```



```
*eventInfo, void *arg);
```

```
/**
 * @brief                    Callback function type corresponding to log output
 * @param logLevel    Log level;
 * @param str             Log info;
 */
typedef void (*RobotLogPrintCallback) (aubo_robot_namespace::LOG_LEVEL
logLevel, const char *str, void *arg);
```

1.6 Tool parameter types

```
/** Tool description, Kinematic parameters of the tool
 *
 *   Used to describe the kinematic parameters of a tool or tool itself
 */
typedef struct
{
    Pos          toolInEndPosition;       // Tool position relative to the flange
    Ori          toolInEndOrientation;    // Tool orientation relative to the flange
}ToolInEndDesc;
```

```
typedef ToolInEndDesc ToolKinematicsParam; // Kinematic parameters
```

```
/**
 * Description of tool dynamics parameters
 *
 * Note：
 *          Before the robot arm is powered on, when the tool installed at the end of
the robot changes, the dynamic parameters of the tool need to be reset.
 *          In general, the dynamic parameters and kinematic parameters of the tool
need to be set together;
 * Remember:
 * If this parameter is not set correctly, it will affect the safety level and motion
trajectory of the robot arm.
 **/
typedef struct
{
    double       positionX;     // X coordinate of tool center of gravity
    double       positionY;     // Y coordinate of tool center of gravity
    double       positionZ;     // Z coordinate of tool center of gravity
    double       payload;       // Tool weight
    ToolInertia toolInertia;
```

```
    // Tool inertia, reserved and used are all set to 0
}ToolDynamicsParam;
```

```
/**
 * This structure describes the inertia of the tool
 *
 * Note: this structure belongs to redundant data type. When using, set all parameters to {0,0,0,0,0}.
 **/
typedef struct
{
    double xx;
    double xy;
    double xz;
    double yy;
    double yz;
    double zz;
}ToolInertia;
```

1.7 Tool Calibration

```
/**
 * @brief Enumeration of methods for tool pose calibration
 *
 */
enum ToolKinematicsOriCalibrateMathod
{
    ToolKinematicsOriCalibrateMathod_Invalid = -1,

    ToolKinematicsOriCalibrateMathod_xOxy,
    // First Point: origin, second point: any point at positive X-axis, Third point: any point on first quadrant of xOy plane.
    ToolKinematicsOriCalibrateMathod_yOyz,
    // First Point: origin, second point: any point at positive Y-axis, Third point: any point on first quadrant of yOz plane.
    ToolKinematicsOriCalibrateMathod_zOzx,
    // First Point: origin, second point: any point at positive Z-axis, Third point: any point on first quadrant of zOx plane.
    ToolKinematicsOriCalibrateMathod_TxRBz_TxyPBzAndTyABnz,
    // The tool x-axis is parallel to the base coordinate system z-axis;
// The tool xOy plane is parallel to the z-axis of the base coordinate system, and the angle between the tool y-axis and the negative z-axis of the base coordinate system is an acute angle
    ToolKinematicsOriCalibrateMathod_TyRBz_TyzPBzAndTzABnz,
```

```
    // The y-axis of the tool is parallel and opposite to the z-axis of the base coordinate system;
// With the yOz plane parallel to the z-axis of the base coordinate system, the angle between the tool z-axis and the negative z-axis of the base coordinate system is an acute angle
    ToolKinematicsOriCalibrateMathod_TzRBz_TzxPBzAndTxABnz,
    // The tool z-axis is parallel to the base coordinate system z-axis;
// The tool zOx plane is parallel to the z-axis of the base coordinate system, and the angle between the tool x-axis and the negative z-axis of the base coordinate system is an acute angle

    ToolKinematicsOriCalibrateMathodCount
};
```

```
typedef struct
{
    int posCalibrateNum ;
    // Number of points used for position calibration
    wayPoint_S posCalibrateWaypoint[4];  // Position calibration point
    int oriCalibrateNum;     // Number of points used for orientation calibration
    wayPoint_S oriCalibrateWaypoint[3];                    // Ori calibration point
    ToolKinematicsOriCalibrateMathod    CalibrateMathod; //Ori calibration method

}ToolCalibrate;
```

1.8 Coordinate system calibration

```
/**
 * Coordinate system description
 *
 * This structure describes a coordinate system. The system describes a coordinate system (base coordinate system, user coordinate system, end coordinate system or tool coordinate system).
 *
 * 3 types of coordinate systems:  Base coordinate system (BaseCoordinate);
 *                                 User coordinate system (WorldCoordinate);
 *                                 Tool coordinate system (EndCoordinate);
 *
 * Definition:
 *          The base coordinate system is a coordinate system established based on the base of the robot arm;
 *          The user coordinate system refers to the user coordinate system defined on the workpiece. At any position within the allowable range of robot action, set the X, y and
```

```
Z axes of any angle. The origin is located on the workpiece grasped by the robot. The direction of the coordinate system is arbitrarily defined according to the needs of customers.
 *          The end coordinate system is the tool coordinate system installed at the end of the robot. The origin and direction are constantly changing with the end position and angle. The coordinate system is actually obtained by changing the basic coordinate system through rotation and displacement; the flange is A special end coordinate system.
 *
 * Structure parameter description:
 *        coordType：Coordinate system type, which type the coordinate system belongs to
 *        methods：  The calibration method of the user coordinate system is only valid when the coordType is the user coordinate system (WorldCoordinate);
 *        wayPointArray[3]：The three waypoint information for calibrating the user coordinate system is valid only when the coordType is the user coordinate system (WorldCoordinate);
 *        toolDesc：Tool End description, when coordType=WorldCoordinate, it means the tool installed at the end of the robot when calibrating the user coordinate system; when coordType=EndCoordinate, it describes which tool coordinate system it is
 *
 * Usage description:
 *        Base coordinate system：coordType=BaseCoordinate，Other parameters default
 *        User coordinate system：  coordType=WorldCoordinate
 *        methods：for the calibration method
 *        wayPointArray[3]：3 waypoints for the calibration coordinate system
 *        toolDesc：When calibrating the user coordinate system, the tool installed at the end of the robot
 *        End coordinate system or tool coordinate system：  coordType=EndCoordinate
 *        methods：Default, no need to set
 *        wayPointArray[3]：Default, no need to set
 *        toolDesc：Tool at the end of the robot
 *
 * Remark:
 * Flange is a special tool, the position in the tool description is set to (0,0,0),
 * Attitude information is set to (1,0,0,0)
 * Its structure is defined as
 *          {
 *              pos{0,0,0},
 *              Ori{1,0,0,0}
 *           }
 *
 *        This structure is also used for the calibration of the user coordinate syste
```

```
m. Generally, it is realized by teaching 3 teaching points, the first
* The first teaching point is the origin of the user coordinate system; the selecti
on of the second and third teaching points is based on the calibration method
* to determine, follow the right-hand method. */
typedef struct
{
    coordinate_refer        coordType;          // Coordinate system type
    CoordCalibrateMathod    methods;  // Calibration method of user coordinate system
    // 3 points for calibrating the user coordinate system
    JointParam              wayPointArray[3];
                    // (joint angle)
    ToolInEndDesc           toolDesc;           // tool description
}CoordCalibrateByJointAngleAndTool;
```

```
/**
 * Coordinate system type enumeration
 **/
enum coordinate_refer
{
    BaseCoordinate = 0,     // base coordinate system
    EndCoordinate,          // End coordinate system or tool coordinate system
    WorldCoordinate,        // user coordinate system
};
```

```
/**
 * User coordinate system calibration method enumeration
 *
 * Description: 3-point calibration coordinate system The meaning of the 3 teaching
points.
 **/
enum CoordCalibrateMathod
{
    Origin_AnyPointOnPositiveXAxis_AnyPointOnPositiveYAxis,
    // Origin, positive half-axis of x-axis, positive half-axis of y-axis
    Origin_AnyPointOnPositiveYAxis_AnyPointOnPositiveZAxis,
    // Origin, positive half-axis of y-axis, positive half-axis of z-axis
    Origin_AnyPointOnPositiveZAxis_AnyPointOnPositiveXAxis,
    // Origin, positive half-axis of z-axis, positive half-axis of x-axis
    Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrantOfXOYPlane,
    // The origin, the positive half-axis of the x-axis, any point on the first quadrant of
the x- and y-axis planes
    Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrantOfXOZPlane,
    // The origin, the positive half-axis of the x-axis, any point on the first quadrant of
the x- and z-axis planes
```

```
    Origin_AnyPointOnPositiveYAxis_AnyPointOnFirstQuadrantOfYOZPlane,
    // The origin, the positive half-axis of the y-axis, any point on the first quadrant of the y- and z-axis planes
    Origin_AnyPointOnPositiveYAxis_AnyPointOnFirstQuadrantOfYOXPlane,
    // The origin, the positive half-axis of the y-axis, any point on the first quadrant of the y- and x-axis planes
    Origin_AnyPointOnPositiveZAxis_AnyPointOnFirstQuadrantOfZOXPlane,
    // The origin, the positive half-axis of the z-axis, any point on the first quadrant of the z- and x-axis planes
    Origin_AnyPointOnPositiveZAxis_AnyPointOnFirstQuadrantOfZOYPlane,
    // The origin, the positive half-axis of the z-axis, any point on the first quadrant of the z- and y-axis planes

    CoordTypeCount,
};
```

1.9 IO Related data types

```
/**
 * @brief IO type enumeration of
 *
 **/
typedef enum
{
    RobotBoardControllerDI,     // Interface board controller DI (digital input),
        // Read-only (generally used internally by the system)
    RobotBoardControllerDO,     // Interface board controller DO (digital output),
                                // Read-only (generally used internally by the system)
    RobotBoardControllerAI,     // Interface board controller AI (analog input),
    // Read-only (generally used internally by the system)
    RobotBoardControllerAO,     // Interface board controller AO (analog output),
                                // Read-only (generally used internally by the system)

    RobotBoardUserDI,           // Interface board user DI (digital input), readable and writable
    RobotBoardUserDO,           // Interface board user DO (digital output), readable and writable
    RobotBoardUserAI,           // Interface board user AI (analog input), readable and writable
    RobotBoardUserAO,           // Interface board user AO (analog output), readable and writable
```

```
    RobotToolDI,                // Tool end DI
    RobotToolDO,                // Tool end DO
    RobotToolAI,                // Tool end AI
    RobotToolAO,                // Tool end AO
}RobotIoType;
```

```
/**
 * IO Type
 **/
typedef enum
{
    IO_IN = 0,              //Input
    IO_OUT                  //Out
}ToolIOType;
```

```
/**
 * Power supply type of the tool
 **/
typedef enum
{
    OUT_0V   = 0,
    OUT_12V = 1,
    OUT_24V = 2
}ToolPowerType;
```

```
typedef enum
{
    RobotToolIoTypeDI=RobotToolDI,          //Tool end DI
    RobotToolIoTypeDO=RobotToolDO           //Tool end DO
}RobotToolIoType;
```

```
typedef enum                    //IO status
{
    IO_STATUS_INVALID = 0,      //Valid
    IO_STATUS_VALID             //Invalid
}IO_STATUS;
```

```
typedef enum
{
    TOOL_DIGITAL_IO_0 = 0,
    TOOL_DIGITAL_IO_1 = 1,
    TOOL_DIGITAL_IO_2 = 2,
    TOOL_DIGITAL_IO_3 = 3
```

```
}ToolDigitalIOAddr;
```

```
/**
   * Comprehensively describe an IO
   **/
typedef struct PACKED
{
    char            ioId[32];       //IO-ID Not currently in use
    RobotIoType     ioType;         //IO Type
    char            ioName[32];     //IO Name
    int             ioAddr;         //IO Address
    double          ioValue;        //IO Status
}RobotIoDesc;
```

```
//Interface board digital quantity data
typedef struct
{
    uint8 addr ;
    uint8 value;
    uint8 type;
}RobotDiagnosisIODesc;
```

```
//Interface board analog data
typedef struct
{
    uint8   addr ;
    float   value;
    uint8   type;
}RobotAnalogIODesc;
```

```
typedef struct PACKED
{
    ToolIOType  ioType;
    uint8       ioData;
}ToolDigitalStatus;
```

2 Interface function

2.1 Robot system interface

2.1.1 Login

int robotServiceLogin(const char *host, int port, const char *userName, const char *password);	
Description:	Log in and establish a network connection with the robotic arm server. The success of this interface is the premise of calling other interfaces, and other interfaces can only be used if the interface returns correctly.
Parameters:	1. host： The IP address of the robotic arm server, that is, the IP of the controller. 2. port：The port number of the robotic arm server, the default is 8899. 3. userName：Username, defaults to Aubo. 4. possword：Password, the default is 123456.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

int robotServiceLogin(const char* host, int port, const char *userName, const char* password, aubo_robot_namespace::RobotType &robotType, aubo_robot_namespace::RobotDhPara &robotDhPara);	
Description:	Log in and establish a network connection with the robotic arm server. The success of this interface is the premise of calling other interfaces, and other interfaces can only be used if the interface returns correctly.
Data:	1. host: IP address of the robotic arm server, that is, the IP of the controller. 2. port: the port number of the robotic arm server, the default is 8899. 3. userName: User name, the default is Aubo. 4. possword: password, the default is 123456. 5. robotType: Robot type. 6. robotDhPara: DH parameter.
Value:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.1.2 Sign out

int robotServiceLogout();	
Description:	Log out and disconnect from the robotic arm server.
Data:	No.
Return:	Successful: Returns ErrnoSucc.
	failed: return error

2.1.3 Robot startup

int rootServiceRobotStartup(const aubo_robot_namespace::ToolDynamicsParam &toolDynamicsParam, uint8 collisionClass, bool readPose, bool staticCollisionDetect, int boardBaxAcc, aubo_robot_namespace::ROBOT_SERVICE_STATE &result, bool IsBolck = true);	
Description:	Start the robotic arm, i.e. initialization-----This operation will complete the functions of powering on the robotic arm, releasing the brakes, setting the collision level, and setting the dynamic parameters. The function takes a long time to complete, so you can adjust the function return time by setting whether to block or not. When it is set to non-blocking, the function will return immediately after the function is called, and the call result will be notified through events. When set to blocking, the return value of the parameter indicates whether the interface is called successfully, and result indicates the start result of the robot arm.
Parameters:	1. toolDynamicsParam: Dynamics parameter, if the tool is clamped at the end, this parameter should be set according to the specifics; if there is no tool at the end, set each item of this parameter to 0. 2. collisionClass: Collision class. 3. readPose: Whether to allow reading the position, the default is true. 4. staticCollisionDetect: The default is true. 5. boardBaxAcc: The default is 1000. 6. result: Outgoing parameter, initialization result, refer to ROBOT_SERVICE_STATE type for details. The robot arm startup result is only result == ROBOT_SERVICE_WORKING, indicating that the robot arm is successfully started, otherwise it indicates that the startup fails. 7. IsBolck: Whether to block, the default is true.
Return:	Successful: Returns ErrnoSucc.
	Failed: Returns an error number.

2.1.4 Shutdown

int robotServiceRobotShutdown(bool IsBolck = true);	
Description:	Power off the robot
Parameter:	**IsBolck:** Whether to block, the default is true. When blocking is set, the function returns until the robotic arm is shut down; when non-blocking is set, it returns immediately, and the result is returned through event push.
Return:	Successful: Return ErrnoSucc。
	Failed: Return error code.

2.2 Getting the status

2.2.1 Set whether to allow real-time joint state push

int robotServiceSetRealTimeJointStatusPush(bool enable);	
Description:	Set whether to allow real-time joint state push.
Parameter:	enable: true means allowed, false means not allowed.
Return:	Successful: Return ErrnoSucc.
	Failed: Return error code.

2.2.2 Register a callback function for getting joint status

int robotServiceRegisterRealTimeJointStatusCallback(RealTimeJointStatusCallback ptr, void *arg);	
Description:	Register a callback function for getting joint state. After registering the callback function, the server pushes the current joint state information in real time. Frequency 30ms.
Parameter:	1. **ptr:** The callback function pointer for obtaining real-time joint status information. When ptr==NULL, it is equivalent to cancel the registration of the callback function. Cancelling the push information can also be done through the interface robotServiceSetRealTimeJointStatusPush. 2. **arg:** This parameter system does not do any processing, but just caches it. When the system calls the registered callback function, the parameter will be returned through the parameter of the callback function.
Return:	Successful: Return ErrnoSucc.
	Failed: Return error code.

2.2.3 Set whether to allow real-time waypoint information push

int robotServiceSetRealTimeRoadPointPush(bool enable);	
Description:	Set whether to allow real-time waypoint information push.
Parameter:	enable: true means allowed, false means not allowed.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.2.4 Register a callback function for getting real-time waypoints

int robotServiceRegisterRealTimeRoadPointCallback(const RealTimeRoadPointCallbac

k ptr, void *arg);	
Description:	Register a callback function for obtaining real-time waypoint information. After registering the callback function, the server pushes the current waypoint information in real time. Frequency 30ms.
Parameter:	1. ptr: The callback function pointer for obtaining real-time waypoint information. When ptr==NULL, it is equivalent to cancel the registration of the callback function. Cancellation of the push information can also be done through the interface robotServiceSetRealTimeRoadPointPush. 2. arg: This parameter system does not do any processing, but just caches it. When the system calls the registered callback function, the parameter will be returned through the parameter of the callback function.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.2.5 Set whether to allow real-time end speed push

int robotServiceSetRealTimeEndSpeedPush(bool enable);	
Description:	Set whether to allow real-time end speed push.
Parameter:	enable: true means allowed, false means not allowed.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.2.6 Register a callback function for getting real-time end velocity

int robotServiceRegisterRealTimeEndSpeedCallback(const RealTimeEndSpeedCallback ptr, void *arg);	
Description:	Register a callback function for getting real-time end velocity. After registering the callback function, the server will push the current terminal speed in real time. Frequency 30ms.
Parameter:	1. ptr: The callback function pointer for obtaining real-time terminal speed information. When ptr==NULL, it is equivalent to cancel the registration of the callback function. Cancelling the push information can also be done through the interface robotServiceSetRealTimeEndSpeedPush. 2. arg: This parameter system does not do any processing, but just caches it. When the system calls the registered callback function, the parameter will be returned through the parameter of the

	callback function.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.2.7 Register the callback function for obtaining the event information of the robot arm

int robotServiceRegisterRobotEventInfoCallback(RobotEventCallback ptr, void *arg);	
Description:	Register the callback function for obtaining the event information of the manipulator. After registering the callback function, the server will push the event information of the robotic arm in real time. Frequency 30ms. **Note: The push of event information does not provide an interface for whether to allow the push, because many important notifications of the robotic arm are implemented through event push, so the event information is pushed by the system by default and cannot be canceled.**
Parameter:	1. ptr: The function pointer to obtain the event information of the robot arm. When ptr==NULL, it is equivalent to cancel the registration of the callback function. 2. arg: This parameter system does not do any processing, but just caches it. When the system calls the registered callback function, the parameter will be returned through the parameter of the callback function.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.2.8 Register callback function for movep progress notification

int robotServiceRegisterMovepStepNumNotifyCallback(RealTimeMovepStepNumNotifyCallback ptr, void *arg);	
Description:	Register callback function for movep progress notification.
Parameter:	1. ptr: Get the function pointer of the movep progress notification of the robotic arm. When ptr==NULL, it is equivalent to cancel the registration of the callback function. 2. arg: This parameter system does not do any processing, but just caches it. When the callback function is triggered, the parameter will be returned through the parameters of the callback function.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.2.9 Register log output callback function

int robotServiceRegisterLogPrintCallback(RobotLogPrintCallback ptr, void *arg);	
Description:	Register the log output callback function.
Parameter:	1. ptr: Get the function pointer of the log output of the robotic arm. When ptr==NULL, it is equivalent to cancel the registration of the callback function. 2. arg: This parameter system does not do any processing, but just caches it. When the callback function is triggered, the parameter will be returned through the parameters of the callback function.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.3 Interfaces related to robot motion

This part of the interface is about motion-related interfaces, including the setting of motion attributes and various forms of motion.

Note: The corresponding motion properties should be set before motion.

The robotic arm interface supports the following motions:

1. Joint move: the corresponding interface function is robotServiceJointMove();

2. Line move: the corresponding interface function is robotServiceLineMove();

3. Track move: the corresponding interface function is robotServiceTrackMove();

4. Rotation: the corresponding interface function is robotServiceRotateMove();

An extended motions based on the above motion:

1. The joint moves to the target position: the corresponding interface functions are robotMoveJointToTargetPositionByRelative(), robotMoveJointToTargetPosition();
2. Straight line movement to the target position: the corresponding interface functions are robotMoveLineToTargetPositionByRelative(), robotMoveLineToTargetPosition();

Note: If you use an offset, you need to set the offset property. If a tool is used, the kinematics parameters (see interface robotServiceSetToolKinematicsParam) and dynamics parameters (see interface robotServiceSetToolDynamicsParam) of the tool need to be set before motion.

2.3.1 Initialize global move properties

int robotServiceInitGlobalMoveProfile();	
Description:	Initialize the global move properties. Initialize the motion properties and set each property to its initial value. When this function is called, the manipulator does not move, and this function initializes each move attribute to the default value. The default value of each property after initialization is: 0: The maximum speed and maximum acceleration properties of joint move: the default joint maximum speed is 25 degrees per second; the default joint maximum acceleration is 25 degrees per second. Valid when the motion type is joint move. 1: The maximum linear velocity and maximum linear acceleration properties of the end-type motion: the default maximum speed of the end is 0.03 meters per second; the default maximum acceleration of the end is 0.03 meters per second square; The maximum angular velocity and maximum angular acceleration properties of the end-type movement: the default maximum speed of the

	end is 100 degrees per second; the default maximum acceleration of the end is 100 degrees per second; Valid when the movement type is end type movement. 2: Waypoint information cache properties: The default waypoint cache is empty, which is used when the trajectory is moving. 3: Blend radius attribute: The default is 0.02 meters. Used when the trajectory motion subtype is MOVEP. 4: The number of loops attribute of the circular track in orbital move: the default is 0, and this property takes effect when the trajectory motion type is equal to ARC_CIR; and when the number of turns of the circular track is equal to zero, ARC_CIR represents an arc, and when the number of turns of the circular track is greater than zero, ARC_CIR stands for circle. 5: Offset attribute of move attribute: There is no offset by default, and all motions except the teaching move are valid. 6: Tool parameter attribute attribute: If there is no tool, the position of the tool is 000. 7: Teaching coordinate system attribute: The teaching coordinate system is the base frame, which is only applicable to the teaching movement.
Parameter:	N/a.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.3.2 Setting the maximum acceleration for manipulator

int robotServiceSetGlobalMoveJointMaxAcc(const aubo_robot_namespace::JointVelcAccParam &moveMaxAcc)；	
Description:	Sets the maximum acceleration for articulation. Articulation includes: 1. Joint movement; 2. Joint teaching in teaching motion (JOINT1, JOINT2, JOINT3, JOINT4, JOINT5, JOINT6) 3. Under trajectory motion (JIONT_CUBICSPLINE, JOINT_UBSPLINEINTP) Note: When setting the speed and acceleration, the user needs to set it according to the type of motion.
Parameter:	moveMaxAcc：Maximum acceleration of joint move, unit rad/s^2.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.3.3 Sets the maximum speed of the articulation

int robotServiceSetGlobalMoveJointMaxVelc(const aubo_robot_namespace::JointVelcAccParam &moveMaxVelc);	
Description:	Sets the maximum speed for articulation. **Note: When setting the speed and acceleration, the user needs to set it according to the type of motion.**
Parameter:	moveMaxVelc：Maximum speed of joint movement, unit rad/s.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.3.4 Get the maximum acceleration of an articulation

void robotServiceGetGlobalMoveJointMaxAcc(aubo_robot_namespace::JointVelcAccParam &moveMaxAcc);	
Description:	Gets the maximum acceleration for the articulation.
Parameter:	moveMaxAcc：Outgoing parameter, indicating the maximum acceleration of joint move, unit rad/s^2.
Return:	N/a

2.3.5 Get the maximum velocity of the articulation

void robotServiceGetGlobalMoveJointMaxVelc(aubo_robot_namespace::JointVelcAccParam &moveMaxVelc);	
Description:	Gets the maximum velocity of the articulation.
Parameter:	moveMaxVelc：Outgoing parameter, indicating the maximum speed of joint move, unit rad/s.
Return:	N/a

2.3.6 Setting the maximum linear acceleration for end-type moves

int robotServiceSetGlobalMoveEndMaxLineAcc(double moveMaxAcc);	
Description:	Sets the maximum acceleration for end-type movements. Terminal types include: 1. Line move (MODEL); 2. Position teaching and attitude teaching in the teaching movement (MOV_X, MOV_Y, MOV_Z, ROT_X, ROT_Y, ROT_Z); 3. Under trajectory move (ARC_CIR, CARTESIAN_MOVEP, CARTESIAN_CUBICSPLINE, CARTESIAN_UBSPLINEINTP) **Note: When setting the speed and acceleration, the user needs to set it according to the type of motion.**

Parameter:	moveMaxAcc：Maximum acceleration of end-type move, unit m/s^2.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.3.7 Sets the maximum linear velocity for end-type move

int robotServiceSetGlobalMoveEndMaxLineVelc(double moveMaxVelc);	
Description:	Sets the maximum linear velocity for end-type move. **Note: When setting the speed and acceleration, the user needs to set it according to the type of motion.**
Parameter:	moveMaxVelc: The maximum linear velocity of the end-type movement, in m/s.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.3.8 Get the maximum linear acceleration of the end-type move

void robotServiceGetGlobalMoveEndMaxLineAcc(double &moveMaxAcc);	
Description:	Get the maximum linear acceleration of end-type move.
Parameter:	moveMaxAcc: Outgoing parameter, the maximum linear acceleration of the end-type movement, Unit m/s^2
Return:	N/a.

2.3.9 moveMaxAcc: Outgoing parameter, the maximum linear acceleration of the end-type movement

void robotServiceGetGlobalMoveEndMaxLineVelc(double &moveMaxVelc);	
Description:	Gets the maximum linear velocity for end-type move.
Parameter:	moveMaxVelc：Outgoing parameter, the maximum velocity of the end-type movement, unit m/s.
Return:	N/a.

2.3.10 Sets the maximum angular acceleration for end-type Move

int robotServiceSetGlobalMoveEndMaxAngleAcc(double moveMaxAcc);	
Description:	Sets the maximum angular acceleration for end-type moves. **Note: When setting the speed and acceleration, the user needs to set it according to the type of move.**
Parameter:	moveMaxAcc: the maximum angular acceleration of the end-type

	movement, unit rad/s^2.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.3.11 Sets the maximum angular velocity for end-type move

int robotServiceSetGlobalMoveEndMaxAngleVelc(double moveMaxVelc);	
Description:	Sets the maximum angular velocity for end-type move. **Note: When setting the speed and acceleration, the user needs to set it according to the type of move.**
Parameter:	moveMaxVelc：Maximum angular velocity of end-type move, unit rad/s.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.3.12 Get the maximum angular acceleration of the end-type move

void robotServiceGetGlobalMoveEndMaxAngleAcc(double &moveMaxAcc);	
Description:	Gets the maximum angular acceleration for end-type move.
Parameter:	moveMaxAcc：Outgoing parameter, maximum angular acceleration for end-type move, unit rad/s^2.
Return:	N/a

2.3.13 Get the maximum angular velocity of end-type move

void robotServiceGetGlobalMoveEndMaxAngleVelc(double &moveMaxVelc);	
Description:	Gets the maximum angular velocity for end-type move.
Parameter:	moveMaxVelc：Outgoing parameter, the maximum angular velocity of the end-type movement, unit rad/s.
Return:	N/a

2.3.14 Set jerk

int robotServiceSetJerkAccRatio(double acc);	
Description:	Set the jerk.
Parameter:	acc：jerk.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.3.15 Get jerk

void robotServiceGetJerkAccRatio(double &acc);	
Description:	Get the jerk.
Parameter:	acc：Outgoing parameter, jerk.
Return:	N/a

2.3.16 Clear waypoint container

void robotServiceClearGlobalWayPointVector();	
Description:	Clears the waypoint container, usually before a new trajectory motion.
Parameter:	N/a
Return:	N/a

2.3.17 Add waypoints

int robotServiceAddGlobalWayPoint(const aubo_robot_namespace::wayPoint_S &wayPoint);	
Description:	Add waypoints for trajectory move robotServiceTrackMove.
Parameter:	wayPoint: The waypoint information of the flange center based on the base frame. Note: **wayPoint_S** can only pass **jointpos** (the system can calculate position and orientation); it cannot pass only position and orientation, otherwise it means zero.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

int robotServiceAddGlobalWayPoint(const double jointAngle[aubo_robot_namespace::ARM_DOF]);	
Description:	Add waypoints for trajectory move robotServiceTrackMove.
Parameter:	jointAngle: The joint angle.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.3.18 Get waypoint container

void robotServiceGetGlobalWayPointVector(std::vector<aubo_robot_namespace::wayPoint_S> &wayPointVector);	
Description:	Get the waypoint container.
Parameter:	wayPointVector: Outgoing parameter, representing the global waypoint container.

Return:	none.

2.3.19 Set blend radius

int robotServiceSetGlobalBlendRadius(float value);	
Description:	Sets the blend radius. **The range of blending radius: 0.0m~0.05m.** **Note: The blend radius must be greater than 0.0.**
Parameter:	Blend radius, unit m.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.3.20 Get blend radius

float robotServiceGetGlobalBlendRadius();	
Description:	Get the blend radius.
Parameter:	none.
Return:	Returns the blend radius value in m.

2.3.21 Getting number of loops of the circle track

int robotServiceGetGlobalCircularLoopTimes();	
Description:	The number of loops of the circle when the circle track of the move attribute. **Note: when the track type is arc_ Effective at cir.** When the circle number attribute **(circularlooptimes)** is 0, it indicates the arc track; When the circle's number of loops attribute (circularlooptimes) is greater than 0, it indicates the circle track.
Parameter:	N/a
Return:	Returns the number of loops of a circle. When times = 0, it indicates circular motion; When times > 0, it indicates the motion of the circle and the number of loops of the circle

2.3.22 Set the number of Loops of the circle when the circle track is set

void robotServiceSetGlobalCircularLoopTimes(int times);	
Description:	Sets the number of loops for circular move.

	Note: Valid when the track type is ARC_CIR.
Parameter:	times: The number of turns of the circle. When times = 0, it means circular move; When times > 0, it indicates circular move, and indicates the number of circles
Return:	N/a

2.3.23 Set the offset property in the move properties

int robotServiceSetMoveRelativeParam(const aubo_robot_namespace::MoveRelative &relativeMoveOnBase);	
Description:	Description: Sets the relative offset property based on the base coordinate system.
Parameter:	Parameter: relativeMoveOnBase: The offset based on the base coordinate system.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

int robotServiceSetMoveRelativeParam(const aubo_robot_namespace::MoveRelative &relativeMoveOnUserCoord, const aubo_robot_namespace::CoordCalibrateByJointAngleAndTool &userCoord);	
Description:	Sets the relative offset property based on a custom coordinate system.
Parameter:	1. **relativeMoveOnUserCoord:** Based on the offset under the following parameters userCoord, including enable, position offset and attitude offset. 2. **userCoord:** custom coordinate system.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.3.24 Settings no arrival ahead

int robotServiceSetNoArrivalAhead();	
Description:	The settings are for not setting the arrival ahead mode.
Parameter:	None.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

2.3.25 Setting the arrival ahead distance mode

int robotServiceSetArrivalAheadDistanceMode(double distance);	
Description:	The settings are for setting the arrival ahead by distance.

	Note: The arrival ahead of the follow mode is currently only applicable to joint move.
Parameter:	distance：distance, unit m.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

2.3.26 Setting the arrival ahead time mode

int robotServiceSetArrivalAheadTimeMode(double second);	
Description:	The settings are for setting the arrival ahead by time. **Note: The arrival ahead of the follow mode is currently only applicable to joint move.**
Parameter:	second：time, unit s.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

2.3.27 Setting the arrival ahead blend radius mode

int robotServiceSetArrivalAheadBlendDistanceMode(double distance);	
Description:	The settings are for setting the arrival ahead by blend radius. **Note: The arrival ahead of the follow mode is currently only applicable to joint move.**
Parameter:	distance：blend radius, unit m.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

2.3.28 Joint Move

int robotServiceJointMove(aubo_robot_namespace::wayPoint_S &wayPoint, bool IsBolck);	
Description:	The joint movement of the motion interface.
Parameter:	1. wayPoint：waypoint info. 2. IsBolck: Whether to block. IsBolck==true means blocking, the robot moves until it reaches the target position or returns after a failure. IsBolck==false means non-blocking, returns immediately, returns when the motion command is sent successfully, and the robotic arm starts to move after the function returns.
Return:	Success: Returns ErrnoSucc.

	Failed: Returns an error code.

int robotServiceJointMove(double jointAngle[aubo_robot_namespace::ARM_DOF], bool IsBolck);	
Description:	The joint movement of the move interface.
Parameter:	1. jointAngle: joint angle. 2. IsBlock： Whether to block. IsBolck==true means blocking, the robot moves until it reaches the target position or returns after a failure. IsBolck==false means non-blocking, returns immediately, returns when the motion command is sent successfully, and the robotic arm starts to move after the function returns.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

int robotServiceJointMove(aubo_robot_namespace::MoveProfile_t &moveProfile, double jointAngle[aubo_robot_namespace::ARM_DOF], bool IsBolck);	
Description:	The joint movement of the move interface.
Parameter:	1. moveProfile: offset property. 2. jointAngle: joint angle. 3. IsBlock: Whether to block. IsBolck==true means blocking, the robot moves until it reaches the target position or returns after a failure. IsBolck==false means non-blocking, returns immediately, returns when the motion command is sent successfully, and the robotic arm starts to move after the function returns.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

2.3.29 Maintain the current pose and move to the target position through joint move, where the target position is given by the offset from the current position

int robotMoveJointToTargetPositionByRelative(const aubo_robot_namespace::CoordCalibrateByJointAngleAndTool &userCoord, const aubo_robot_namespace::MoveRelative &relativeMoveOnUserCoord, bool IsBolck = false);	
Description:	Keep the current pose and move to the target position through joint movement, where the target position is given by the offset from the current position.

Parameter:	1. userCoord: the reference coordinate system. 2. relativeMoveOnUserCoord: Offset in the reference coordinate system. 3. IsBolck: Whether to block. 4. IsBolck==true means blocking, the robot moves until it reaches the target position or returns after a failure. 5. IsBolck==false means non-blocking, returns immediately, returns when the motion command is sent successfully, and the robotic arm starts to move after the function returns.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

2.3.30 Maintain the current pose and move to the target position through joint move

int robotMoveJointToTargetPosition(const aubo_robot_namespace::CoordCalibrateByJointAngleAndTool &userCoord, const aubo_robot_namespace::Pos &toolEndPositionOnUserCoord, const aubo_robot_namespace::ToolInEndDesc &toolInEndDesc, bool IsBolck = false);	
Description:	Maintain the current pose and move to the target position through joint move.
Parameter:	1. userCoord: the reference coordinate system. 2. toolEndPositionOnUserCoord: The position parameter of the tool coordinate system in the reference coordinate system. 3. toolInEndDesc: tool parameter, when no tool is used, set this parameter to 0. 4. IsBolck: whether to block. 5. IsBolck==true means blocking, the robot moves until it reaches the target position or returns after a failure. 6. IsBolck==false means non-blocking, returns immediately, returns when the move command is sent successfully, and the robotic arm starts to move after the function returns.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

2.3.31 Joint move based on follow mode

int robotServiceFollowModeJointMove(double jointAngle[aubo_robot_namespace::ARM_DOF]);	
Description:	Joint moving interface based on following mode.

Parameter:	jointAngle：joint angle
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

2.3.32 Line move

int robotServiceLineMove(aubo_robot_namespace::wayPoint_S &wayPoint, bool IsBolck);	
Description:	The line move of the motion interface belongs to the end-type move.
Parameter:	1. wayPoint: waypoint. 2. IsBlock: Whether to block. IsBolck==true means blocking, the robot moves until it reaches the target position or returns after a failure. IsBolck==false means non-blocking, returns immediately, returns when the move command is sent successfully, and the robotic arm starts to move after the function returns.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

int robotServiceLineMove(double jointAngle[aubo_robot_namespace::ARM_DOF], bool IsBolck);	
Description:	The line move of the move interface belongs to the end-type motion.
Parameter:	1. jointAngle: joint angle. 2. IsBlock: Whether to block. IsBolck==true means blocking, the robot moves until it reaches the target position or returns after a failure. IsBolck==false means non-blocking, returns immediately, returns when the move command is sent successfully, and the robotic arm starts to move after the function returns.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

int robotServiceLineMove(aubo_robot_namespace::MoveProfile_t &moveProfile, double jointAngle[aubo_robot_namespace::ARM_DOF], bool IsBolck);	
Description:	linear move of the move interface belongs to the end-type move.
Parameter:	1. moveProfile: Move offset property. 2. jointAngle: joint angle. IsBlock: Whether to block. IsBolck==true means blocking, the robot moves until it reaches the target position or returns after a failure. IsBolck==false means non-blocking, returns immediately, returns when the motion command is sent successfully, and the robotic arm

	starts to move after the function returns.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

2.3.33 Mountain the current pose and move to the target position by line move, where the target position is given by the offset from the current position

int robotMoveLineToTargetPositionByRelative(const aubo_robot_namespace::CoordCalibrateByJointAngleAndTool &userCoord, const aubo_robot_namespace::MoveRelative &relativeMoveOnUserCoord, bool IsBolck);	
Description:	Keep the current pose and move to the target position by line move, where the target position is given by the offset from the current position
Parameter:	1. userCoord: the reference coordinate system. 2. relativeMoveOnUserCoord: The offset under the reference coordinate system userCoord. 3. IsBlock: Whether to block. IsBolck==true means blocking, the robot moves until it reaches the target position or returns after a failure. IsBolck==false means non-blocking, returns immediately, returns when the move command is sent successfully, and the robotic arm starts to move after the function returns.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

2.3.34 Maintain the current pose and move to the target position through line move

int robotMoveLineToTargetPosition(const aubo_robot_namespace::CoordCalibrateByJointAngleAndTool &userCoord, const aubo_robot_namespace::Pos &toolEndPositionOnUserCoord, const aubo_robot_namespace::ToolInEndDesc &toolInEndDesc, bool IsBolck = false);	
Description:	Keep the current pose and move to the target position through line move.
Parameter:	1. userCoord: The reference coordinate system of the target position (positionOnUserCoord). The coordinate system parameter (userCoord) indicates that the following target position

	(positionOnUserCoord) is based on the plane. positionOnUserCoord: The position parameter of the tool coordinate system in the reference coordinate system. The target location based on the user plane representation. The position information of the tool end point (target position x, y, z), the tool parameters are given by the next parameter (toolInEndDesc); 2. toolInEndDesc: tool parameter, when the tool is not used, set this parameter to 0; IsBlock: Whether to block. IsBolck==true means blocking, the robot moves until it reaches the target position or returns after a failure. IsBolck==false means non-blocking, returns immediately, returns when the motion command is sent successfully, and the robotic arm starts to move after the function returns.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

2.3.35 Maintain the current position, change the orientation and do the rotation movement

int robotServiceRotateMove(const aubo_robot_namespace::CoordCalibrateByJointAngleAndTool &userCoord, const double rotateAxisOnUserCoord[3], double rotateAngle, bool IsBolck);	
Description:	The move interface maintains the current position and changes the attitude to perform rotation, that is, the end point of the manipulator rotates around the specified rotation axis (the position remains unchanged).
Parameter:	1. userCoord: is the reference coordinate system of the rotation axis (rotateAxisOnUserCoord). 2. rotateAxisOnUserCoord: Rotation axis [x, y, z]. 3. rotateAngle: The rotation angle around the rotation axis, in rad. 4. IsBlock: Whether to block. IsBolck==true means blocking, the robot moves until it reaches the target position or returns after a failure. IsBolck==false means non-blocking, returns immediately, returns when the motion command is sent successfully, and the robotic arm starts to move after the function returns.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

2.3.36 Obtain the target pose according to the current pose and the rotation axis and rotation angle represented in the base coordinate system

int robotServiceGetRotateTargetWaypiont(const aubo_robot_namespace::wayPoint_S &originateWayPointOnBaseCoord, const double rotateAxisOnBaseCoord[], double rotateAngle, aubo_robot_namespace::wayPoint_S &targetWayPointOnBaseCoord);	
Description:	Obtain the target pose according to the current pose and the rotation axis and rotation angle represented in the base coordinate system
Parameter:	1. originateWayPointOnBaseCoord: originate waypoint information. 2. rotateAxisOnBaseCoord: The rotation axis represented in the base coordinate system. 3. rotateAngle: the rotation angle. 4. targetWayPointOnBaseCoord: Outgoing parameter, target waypoint information.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

2.3.37 Transform the rotation axis described in the user coordinate system to the description under the base coordinate system

int robotServiceGetRotateAxisUserToBase(const aubo_robot_namespace::Ori &oriOnUserCoord, const double rotateAxisOnUserCoord[], double rotateAxisOnBaseCoord[]);	
Description:	Transforms the rotation axis described in the user coordinate system to the rotation axis described in the base coordinate system.
Parameter:	1. oriOnUserCoord: User coordinate system attitude. 2. rotateAxisOnUserCoord: the rotation axis described in the user coordinate system. 3. rotateAxisOnBaseCoord: outgoing parameter, the rotation axis described in the base coordinate system
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

2.3.38 Rotate move to target waypoint

int robotServiceRotateMoveToWaypoint(const aubo_robot_namespace::waypoint_S &targetWayPointOnBaseCoord, bool IsBolck);	
Description:	Rotate movement to the target waypoint.
Parameter:	1. targetWayPointOnBaseCoord: The target waypoint in the base coordinate system. 2. IsBlock: Whether to block. IsBolck==true means blocking, the robot moves until it reaches the target position or returns after a failure. IsBolck==false means non-blocking, returns immediately, returns when the motion command is sent successfully, and the robotic arm starts to move after the function returns.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

2.3.39 Track move

int robotServiceTrackMove(aubo_robot_namespace::move_track subMoveMode, bool IsBolck);	
Description:	Trajectory move
Parameter:	1. subMoveMode: trajectory motion type. 1) When subMoveMode==ARC_CIR, CARTESIAN_MOVEP, CARTESIAN_CUBICSPLINE, CARTESIAN_UBSPLINEINTP, the movement is end-type movement. 2) When subMoveMode==JIONT_CUBICSPLINE, JOINT_UBSPLINEINTP, the motion belongs to joint motion. 3) When subMoveMode==ARC_CIR, it means circle or arc. When the circle number attribute (CircularLoopTimes) is 0, it indicates a circular arc trajectory; when the circle circle number attribute (CircularLoopTimes) is greater than 0, it indicates a circular trajectory. 4) When subMoveMode==CARTESIAN_MOVEP, it means MOVEP trajectory, which requires the user's only blending radius attribute. 5) When subMoveMode==JOINT_UBSPLINEINTP, the track reproduces the interface. 2. IsBolck: Whether to block. IsBolck==true means blocking, the robot moves until it reaches the target position or returns after a failure. IsBolck==false means non-blocking, returns immediately, returns

	when the motion command is sent successfully, and the robotic arm starts to move after the function returns.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

2.3.40 Starting the teaching code

int robotServiceTeachStart(aubo_robot_namespace::teach_mode mode, bool direction);	
Description:	Starting the teaching move.
Parameter:	1. mode: Teaching mode. 1) Teaching joints: JOINT1, JOINT2, JOINT3, JOINT4, JOINT5, JOINT6. 2) Position teaching: MOV_X, MOV_Y, MOV_Z. 3) Orientation teaching: ROT_X, ROT_Y, ROT_Z. 2. direction：Movement direction, true for positive direction, false for negative direction.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

2.3.41 Setting the coordinate system of the teaching move

int robotServiceSetTeachCoordinateSystem(const aubo_robot_namespace::CoordCalibrateByJointAngleAndTool &coordSystem);	
Description:	Setting the coordinate system of the teaching move.
Parameter:	coordSystem: Determine a coordinate system through this parameter, using the usage instructions at the reference type definition.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

2.3.42 Stop teaching mode

int robotServiceTeachStop();	
Description:	Stop the Teaching.
Parameter:	None.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

2.3.43 Clear offline track waypoints

int robotServiceOfflineTrackWaypointClear ();

Description:	Clear waypoints for server offline tracks.
Parameter:	N/a.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

2.3.44 Add offline track waypoints

int robotServiceOfflineTrackWaypointAppend(const std::vector<aubo_robot_namespace ::wayPoint_S> &wayPointVector);	
Description:	Add offline trajectory waypoints to the server via the waypoint container.
Parameter:	wayPointVector: waypoint container, the container allows a maximum of 4000 waypoints.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

int robotServiceOfflineTrackWaypointAppend(const char *fileName);	
Description:	Add offline track waypoints to the server in the form of waypoint files.
Parameter:	fileName: waypoint file.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

2.3.45 Start offline track move

int robotServiceOfflineTrackMoveStartup (bool IsBolck);	
Description:	Starts the run of the offline track.
Parameter:	IsBolck: Whether to block when calling the interface.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

2.3.46 Stop the offline trajectory move

int robotServiceOfflineTrackMoveStop();	
Description:	End offline trajectory move.
Parameter:	N/a.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

2.3.47 Enter TCP to CAN transparent transmission mode

int robotServiceEnterTcp2CanbusMode();	
Description:	Enter TCP to CAN transparent transmission mode.
Parameter:	Parameter: None.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

2.3.48 Send coordinate data to the joint CAN bus

int robotServiceSetRobotPosData2Canbus(double jointAngle[aubo_robot_namespace::ARM_DOF]);	
Description:	Send coordinate data to the joint CAN bus. Send joint angle information to the driver through transparent transmission.
Parameter:	jointAngle：joint angle，unit *rad*.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

int robotServiceSetRobotPosData2Canbus(const std::vector<aubo_robot_namespace::wayPoint_S> &wayPointVector);	
Description:	Send coordinate data to the joint CAN bus. Send waypoint containers to the drive via passthrough.
Parameter:	wayPointVector：waypoint container
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

2.3.49 Exit TCP to CAN transparent transmission mode

int robotServiceLeaveTcp2CanbusMode();	
Description:	Exit the TCP to CAN transparent transmission mode.
Parameter:	none.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

2.4 Tool interface

2.4.1 Forward Kinematics

int robotServiceRobotFk(const double *jointAngle,

int size, aubo_robot_namespace::waypoint_S &wayPoint);	
Description:	FK, this function is a forward kinematics solution function, and the joint angle is known to find the position and orientation of the corresponding position.
Parameter:	1. jointAngle: The joint angle of the six joints, in rad. 2. size: The length of the joint angle array, which is specified as 6. 3. wayPoint: Outgoing parameters, the positive solution gets the waypoint.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

2.4.2 Inverse Kinematics

int robotServiceRobotIk(const double *startPointJointAngle, const aubo_robot_namespace::Pos &position, const aubo_robot_namespace::Ori &ori, aubo_robot_namespace::wayPoint_S &wayPoint);	
Description:	Inverse kinematics. The joint angle information of the corresponding position is obtained according to the position information (x, y, z) and the reference pose (w, x, y, z) of the corresponding position. Inverse kinematics problem: For a robot, when the position and posture of the manipulator (flange center) in the base frame are given, the corresponding joint angle information is obtained.
Parameter:	1. startPointJointAngle: The joint angle of the six joints at the reference point, usually the position of the current robotic arm, in rad. 2. position：The location of the target waypoint. 3. ori：The reference pose of the target waypoint. For example, the current position and pose can be obtained as this parameter, which is equivalent to maintaining the current pose. 4. wayPoint：Outgoing parameters, target waypoint information.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

int robotServiceRobotIk(const aubo_robot_namespace::Pos &position, const aubo_robot_namespace::Ori &ori, std::vector<aubo_robot_namespace::wayPoint_S> &wayPointVector);	
Description:	Inverse kinematics, this function is the inverse solution function of the manipulator. According to the position information (x, y, z) and the reference orientation (w, x, y, z) of the corresponding position, the joint angle information of the corresponding position is obtained.

Parameter:	1. 1. position: The position of the target waypoint. 2. 2. ori: the reference pose of the target waypoint. 3. 3. wayPointVector: Outgoing parameter, target waypoint container.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

2.4.3 Base coordinate system to user coordinate system

static int baseToUserCoordinate(const aubo_robot_namespace::Pos &flangeCenterPositionOnBase, const aubo_robot_namespace::Ori &flangeCenterOrientationOnBase, const aubo_robot_namespace::CoordCalibrateByJointAngleAndTool &userCoord, const aubo_robot_namespace::ToolInEndDesc &toolInEndDesc, aubo_robot_namespace::Pos &toolEndPositionOnUserCoord, aubo_robot_namespace::Ori &toolEndOrientationOnUserCoord);	
Description:	Convert the position and orientation of the flange center based on the base coordinate system to the position and orientation of the tool end based on the user coordinate system. Extension 1: The center of the flange can be regarded as a special tool, that is, the position of the tool is (0,0,0) and the orientation is (1,0,0,0). Therefore, when the tool is (0,0,0), it is equivalent to converting the flange center based on the position and orientation of the base coordinate system to the flange center based on the user coordinate system. Extension 2: The user coordinate system can also be selected as the base coordinate system, namely: userCoord.coordType = BaseCoordinate. Therefore, when the user plane is the base coordinate system, it is equivalent to converting the position and orientation of the flange center based on the base coordinate system into the position and orientation of the tool end based on the base coordinate system, that is, adding the tool in the base coordinate system.
Parameter:	1. flangeCenterPositionOnBase：The flange center is based on the position information (x, y, z) in the base coordinate system. 2. flangeCenterOrientationOnBase：The flange center is based on the attitude information (w, x, y, z) in the base coordinate system. 3. userCoord：User coordinate system. 4. toolInEndDesc：Tool parameters 5. toolEndPositionOnUserCoord：Outgoing parameter, the tool end is based on the position information in the user coordinate system. toolEndOrientationOnUserCoord：Outgoing parameter, the tool end is based on the orientation information in the user coordinate system.
Return:	Success: Returns ErrnoSucc.

	Failed: Returns an error code.

2.4.4 Convert the base coordinate system to the base coordinate to get the position and orientation of the end point of the tool

static int baseToBaseAdditionalTool(const aubo_robot_namespace::Pos &flangeCenterPositionOnBase, const aubo_robot_namespace::Ori &flangeCenterOrientationOnBase, const aubo_robot_namespace::ToolInEndDesc &toolInEndDesc, aubo_robot_namespace::Pos &toolEndPositionOnBase, aubo_robot_namespace::Ori &toolEndOrientationOnBase);	
Description:	The position and orientation of the flange center in the base coordinate system The position and attitude of the tool end in the base coordinate system.
Parameter:	1. flangeCenterPositionOnBase：The flange center is based on the position information in the base coordinate system. 2. flangeCenterOrientationOnBase：The flange center is based on the orientation information in the base coordinate system. 3. toolInEndDesc：Tool parameters 4. toolEndPositionOnUserCoord：Outgoing parameters, the tool end is based on the position information in the base coordinate system. 5. toolEndOrientationOnUserCoord： Outgoing parameters, the tool end is based on the orientation information in the base coordinate system.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

2.4.5 User coordinate system position and attitude information Turn the position and orientation of the base coordinate system

static int userToBaseCoordinate(const aubo_robot_namespace::Pos &toolEndPositionOnUserCoord, const aubo_robot_namespace::Ori &toolEndOrientationOnUserCoord, const aubo_robot_namespace::CoordCalibrateByJointAngleAndTool &userCoord, const aubo_robot_namespace::ToolInEndDesc &toolInEndDesc, aubo_robot_namespace::Pos &flangeCenterPositionOnBase, aubo_robot_namespace::Ori &flangeCenterOrientationOnBase);	
Description:	Turn the tool end based on the position and orientation of the user coordinate system into the position and orientation of the flange center based on the base coordinate system.

	Extension 1: The center of the flange can be regarded as a special tool, that is, the position of the tool is (0,0,0) and the orientation is (1,0,0,0). Therefore, when the position of the tool is (0,0,0) and the pose is (1,0,0,0), it means that toolEndPositionOnUserCoord and toolEndOrientationOnUserCoord are toolless. Extension 2: The user coordinate system can also be selected as the base coordinate system, namely: **userCoord.coordType = BaseCoordinate.** Therefore, when the user plane is the base coordinate system, it is equivalent to converting the position and orientation of the tool end based on the base coordinate system into the position and attitude of the flange center based on the base coordinate system. Extension 3: Use this function and inverse kinematics combination to achieve. When the user provides the position and attitude of the tool end in the custom coordinate system (especially the base coordinate system), the position and attitude of the flange center in the base coordinate system are obtained, and then the inverse kinematics is performed to obtain the target waypoint.
Parameter:	1. toolEndPositionOnUserCoord： The tool end is based on the position information in the user coordinate system. 2. toolEndOrientationOnUserCoord：The tool end is based on the attitude information in the user coordinate system. 3. userCoord：Reference coordinate system, a coordinate system is determined by this parameter. 4. toolInEndDesc：Tool parameters. 5. flangeCenterPositionOnBase：The outgoing parameter, the flange center is based on the position information in the base coordinate system. 6. flangeCenterOrientationOnBase：Outgoing parameters, the flange center is based on the orientation information in the base coordinate system. **Note: This coordinate system transformation does not support the end system, that is, does not supportuserCoord==EndCoordinate, If userCoord==EndCoordinate, a parameter error (ErrCode_ParamError) will be reported.**
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

2.4.6 Convert a position information (x, y, z) based on the user coordinate system in space to the position information (x, y, z) based on the base coordinate system

static int userCoordPointToBasePoint(const aubo_robot_namespace::Pos &userCoordPoint, const aubo_robot_namespace::CoordCalibrateByJointAngleAndTool &userCoordSystem, aubo_robot_namespace::Pos &basePoint);	
Description:	Convert a position information (x, y, z) based on the user coordinate system in the space to the position information (x, y, z) based on the base coordinate system. The same tool parameter.
Parameter:	1. userCoordPoint： Position information x, y, z in the user coordinate system (must be based on the coordinate system provided by the following parameters) 2. userCoordSystem：User coordinate system description, a coordinate system is determined by this parameter. 3. basePoint：Outgoing parameters, based on the position parameters in the base coordinate system. **Note: this coordinate system transformation does not support the end system, that is, it does not support userCoord==EndCoordinate, if userCoord==EndCoordinate, a parameter error (ErrCode_ParamError) will be reported.**
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error code.

2.4.7 Flange orientation to tool orientation

static int endOrientation2ToolOrientation(aubo_robot_namespace::Ori &tcpOriInEnd, const aubo_robot_namespace::Ori &endOri, aubo_robot_namespace::Ori &toolOri);	
Description:	Converts the pose of the center of the flange to the pose of the end of the tool.
Parameter:	1. tcpOriInEnd：Tool pose parameters. 2. endOri：Flange center pose. 3. toolOri：Outgoing parameters, tool pose.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.4.8 Convert the tool orientation to the flange orientation

static int toolOrientation2EndOrientation(aubo_robot_namespace::Ori &tcpOriInEnd, const aubo_robot_namespace::Ori &toolOri, aubo_robot_namespace::Ori &endOri);	
Description:	Convert the tool pose to the flange center pose.
Parameter:	1. tcpOriInEnd：Tool pose parameters 2. toolOri：Tool pose 3. endOri：Outgoing parameters, flange center pose
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.4.9 Get target waypoint information based on position

static int getTargetWaypointByPosition(const aubo_robot_namespace::wayPoint_S &sourceWayPointOnBaseCoord, const aubo_robot_namespace::CoordCalibrateByJointAngleAndTool &userCoordSystem, const aubo_robot_namespace::Pos &toolEndPosition, const aubo_robot_namespace::ToolInEndDesc &toolInEndDesc, aubo_robot_namespace::wayPoint_S &targetWayPointOnBaseCoord);	
Description:	Obtain the target waypoint information according to the position parameters (orientation homologous waypoint)
Parameter:	1. sourceWayPointOnBaseCoord：Source waypoint. 2. userCoordSystem：coordinate system. 3. toolEndPosition：tool end position 4. toolInEndDesc：tool end parameters 5. targetWayPointOnBaseCoord：Outgoing parameter, target waypoint.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.4.10 Quaternion to Euler Angles

int quaternionToRPY(const aubo_robot_namespace::Ori &ori, aubo_robot_namespace::Rpy &rpy);	
Description:	Quaternion to Euler angles.
Parameter:	1. ori：Quaternion representation of orientation. 2. rpy：Outgoing parameters, the Euler angle representation of the orientation.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.4.11 Euler angle to quaternion

int RPYToQuaternion(const aubo_robot_namespace::Rpy &rpy, aubo_robot_namespace::Ori &ori);	
Description:	Euler angles to quaternion.
Parameter:	1. rpy：Euler angle representation of orientation 2. ori：The outgoing parameter, the four-element representation of the pose.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.4.12 Get error information based on error code

std::string getErrDescByCode(aubo_robot_namespace::RobotErrorCode code);	
Description:	Get error information based on error code
Parameter:	code: Error code.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.5 Robot Control Interface

2.5.1 Robot motion control: stop, pause, resume

int rootServiceRobotMoveControl(aubo_robot_namespace::RobotMoveControlCommand cmd);	
Description:	Robot motion control: stop, pause, resume
Parameter:	cmd：control command. **Note: RobotMoveControl needs to be called in a different thread than move. And the move thread needs to be stopped after robotMoveStop is called, otherwise the next move instruction will be executed.**
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.5.2 Robot control

int rootServiceRobotControl(const aubo_robot_namespace::RobotControlCommand cmd);	
Description:	Arm control.
Parameter:	cmd：control command.

Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.5.3 Fast stop of the robotic arm

int robotMoveFastStop();	
Description:	Fast stop of the robotic arm. **Note: robotMoveFastStop needs to be called in a different thread than move. And the move thread needs to be stopped after robotMoveFastStop is called, otherwise the next move command will be executed.**
Parameter:	N/a
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.5.4 Robot motion stop

int robotMoveStop();	
Description:	Robot motion stop. **Note: robotMoveStop needs to be called in a different thread than move.**
Parameter:	N/a
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.5.5 Set the power state of the robot

int robotServicePowerControl(bool value);	
Description:	Description: Sets the power state of the robotic arm.
Parameter:	Parameter: value: Arm power status.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.5.6 Release the brakes

int robotServiceReleaseBrake();	
Description:	Releasing the brakes
Parameter:	N/a
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.6 Tool end interfaces

2.6.1 Set the dynamics parameters without tool

int robotServiceSetNoneToolDynamicsParam();	
Description:	Setting the dynamics parameters without tool
Parameter:	N/a
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.6.2 Set the tool dynamics parameters

int robotServiceSetToolDynamicsParam(const aubo_robot_namespace::ToolDynamicsParam &toolDynamicsParam);	
Description:	Set the tool dynamics parameters.
Parameter:	toolDynamicsParam：tool dynamics parameters.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.6.3 Get the kinetic parameters of the tool

int robotServiceGetToolDynamicsParam(aubo_robot_namespace::ToolDynamicsParam &toolDynamicsParam);	
Description:	Getting the kinetic parameters of the tool
Parameter:	toolDynamicsParam：Outgoing parameters, tool dynamics parameters.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.6.4 Setting kinematics parameters without tool

int robotServiceSetNoneToolKinematicsParam();	
Description:	Setting kinematics parameters without tool
Parameter:	N/a
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.6.5 Set the kinematic parameters of the tool

int robotServiceSetToolKinematicsParam(const aubo_robot_namespace::ToolKinematicsParam &toolKinematicsParam);	
Description:	Setting the kinematic parameters of the tool
Parameter:	toolKinematicsParam：kinematic parameters of the tool
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.6.6 Get the kinematic parameters of the tool

int robotServiceGetToolKinematicsParam(aubo_robot_namespace::ToolKinematicsParam &toolKinematicsParam);	
Description:	Getting the kinematic parameters of the tool
Parameter:	toolKinematicsParam：Outgoing parameters, tool kinematics parameters.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.7 Interface to set and get related parameters of the robot

2.7.1 Get the current connection status

void robotServiceGetConnectStatus(bool &connectStatus);	
Description:	Get the current connection status. This function is used to view the connection status with the robot server.
Parameter:	connectStatus：Outgoing parameters, the network is connected, return true; otherwise, return false.
Return:	N/a

2.7.2 Set the current robot mode: simulation or real

int robotServiceSetRobotWorkMode(aubo_robot_namespace::RobotWorkMode mode);	
Description:	Set the current robot mode: simulation or real.
Parameter:	mode: emulated or real enumeration type.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.7.3 Get the current robot mode

int robotServiceGetRobotWorkMode(aubo_robot_namespace::RobotWorkMode &mode);	
Description:	Get the current working mode of the robotic arm.
Parameter:	mode: Outgoing parameter, emulation or real enumeration type, indicating the current mode of the robotic arm.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.7.4 Get gravity component

int robotServiceGetRobotGravityComponent(aubo_robot_namespace::RobotGravityComponent &gravityComponent);	
Description:	Get the gravity component.
Parameter:	gravityComponent: Outgoing parameter, gravity component, which needs to be connected to the real robot.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.7.5 Get the current collision level

int robotServiceGetRobotCollisionCurrentService(int &collisionGrade);	
Description:	Get the collision level.
Parameter:	collisionGrade: Outgoing parameter, collision grade, which needs to be connected to a real robot.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.7.6 Set Collision Level

int robotServiceSetRobotCollisionClass(int grade);	
Description:	Set the collision level.
Parameter:	grade: Collision grade: 0~10.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

int robotServiceSetRobotCollisionClass(int grade, aubo_robot_namespace::CollisionMode collisionMode);	
Description:	Set the collision level.

Parameter:	1. grade: Collision grade: 0~10. 2. collisionMode: collision mode.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.7.7 Get device information

int robotServiceGetRobotDevInfoService(aubo_robot_namespace::RobotDevInfo &devInfo);	
Description:	To obtain robot device information, a real robot must be connected.
Parameter:	devInfo: Device information.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.7.8 Get whether there is a real robot

int robotServiceGetIsRealRobotExist(bool &value);	
Description:	Get whether there is a real robotic arm
Parameter:	value: outgoing parameter, true: there is a real robot arm; false: there is no real robot arm.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.7.9 Get 6-joint rotation 360 enable flag

int robotServiceGetJoint6Rotate360EnableFlag(bool &value);	
Description:	Getting the 6-joint rotation 360 enable flag
Parameter:	value: Outgoing parameter, whether J6 is the 360-degree version. true: 360-degree version; false: normal version.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.7.10 Get the joint state of the robot

int robotServiceGetRobotJointStatus(aubo_robot_namespace::JointStatus *jointStatus, int size);	
Description:	Getting the joint state of the robot
Parameter:	1. jointStatus: Outgoing parameters, joint status. 2. size: joint angle buffer length, 6.

Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.7.11 Get Robotic Arm Diagnostic Information

int robotServiceGetRobotDiagnosisInfo(aubo_robot_namespace::RobotDiagnosis &robotDiagnosisInfo);	
Description:	Get robotic arm diagnostic information.
Parameter:	robotDiagnosisInfo: Outgoing parameters, robot arm diagnostic information.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.7.12 Get the current joint angle information of the robot

int robotServiceGetJointAngleInfo(aubo_robot_namespace::JointParam &jointParam);	
Description:	Get the current joint angle information of the robotic arm.
Parameter:	jointParam: Outgoing parameters, joint angle information.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.7.13 Get the current waypoint information of the robot

int robotServiceGetCurrentWaypointInfo(aubo_robot_namespace::wayPoint_S &waypoint);	
Description:	Get the current waypoint information of the robotic arm.
Parameter:	wayPoint: Outgoing parameters, current waypoint information.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.7.14 Whether the robot arm is running in online mode

int robotServiceIsOnlineMode(bool &isOnlineMode);	
Description:	Returns whether the current robotic arm is running in online mode.
Parameter:	isOnlineMode: outgoing parameter, true: online false: not online.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.7.15 Whether the robot arm is running in online master mode

int robotServiceIsOnlineMasterMode(bool &isOnlineMasterMode);	
Description:	Returns whether the current robot is running in online master mode.
Parameter:	isOnlineMode: outgoing parameter, true: online master mode/manual mode false: online slave mode.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.7.16 Get the current operating status of the robot

int robotServiceGetRobotCurrentState(aubo_robot_namespace::RobotState &state);	
Description:	Get the current running state of the robotic arm. Note: need to be in a different thread from move.
Parameter:	state: outgoing parameter, running state.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.7.17 Get MAC communication status

int robotServiceGetMacCommunicationStatus(bool &value);	
Description:	Get MAC communication status.
Parameter:	value: Outgoing parameter, communication status. true: the connection succeeded false: the connection failed.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.7.18 Get Robot Security Configuration

int robotServiceGetRobotSafetyConfig(aubo_robot_namespace::RobotSafetyConfig &safetyConfig);	
Description:	Get the robot security configuration.
Parameter:	safetyConfig: safety configuration.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.8 Interface board IO related interface

The IO of the interface board is mainly divided into two parts, namely the controller IO and the user IO. Each IO has a corresponding name and address. In use, the status of the

IO can be obtained and set by the name or the address.

2.8.1 Get the configuration information of the specified IO set of the interface board

int robotServiceGetBoardIOConfig(const std::vector<aubo_robot_namespace::RobotIoType> &ioType, std::vector<aubo_robot_namespace::RobotIoDesc> &configVector);	
Description:	Get the configuration information of the specified IO set of the interface board.
Parameter:	1. ioType: A collection of IO types. 2. configVector: Outgoing parameters, a collection of IO configuration information.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.8.2 Get the status information of the specified IO set of the interface board

int robotServiceGetBoardIOStatus(const std::vector<aubo_robot_namespace::RobotIoType> ioType, std::vector<aubo_robot_namespace::RobotIoDesc> &statusVector);	
Description:	Get the status information of the specified IO set of the interface board.
Parameter:	1. ioType: A collection of IO types. 2. statusVector: Outgoing parameters, a collection of IO status information.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.8.3 Set the IO status of the interface board

int robotServiceSetBoardIOStatus(aubo_robot_namespace::RobotIoType type, std::string name, double value);	
Description:	Set the IO status according to the interface board IO type and name.
Parameter:	1. type: IO type. 2. name: IO name. 3. value: IO status.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

int robotServiceSetBoardIOStatus(aubo_robot_namespace::RobotIoType type, int addr, double value);	
Description:	Set the IO status according to the interface board IO type and address.
Parameter:	1. type: IO type. 2. addr: IO address. 3. value: IO status.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

int robotServiceGetBoardIOStatus(aubo_robot_namespace::RobotIoType type, std::string name, double &value);	
Description:	Get the IO status according to the IO type and name of the interface board.
Parameter:	1. type: IO type. 2. name: IO name. 3. value: outgoing parameter, IO status.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

int robotServiceGetBoardIOStatus(aubo_robot_namespace::RobotIoType type, int addr, double &value);	
Description:	Obtain the IO status according to the IO type and address of the interface board.
Parameter:	1. type: IO type. 2. addr: IO address. 3. value: outgoing parameter, IO status.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.9 Tool IO related interface

2.9.1 Set tool end power supply voltage type

int robotServiceSetToolPowerVoltageType(aubo_robot_namespace::ToolPowerType type);	
Description:	Description: Set the power voltage type of the tool end.
Parameter:	Parameter: type: Tool power supply voltage type.
Return:	Success: Returns ErrnoSucc.

	Failed: Returns an error number.

2.9.2 Get tool end power supply voltage type

int robotServiceGetToolPowerVoltageType(aubo_robot_namespace::ToolPowerType &type);	
Description:	Get the tool end power supply voltage type.
Parameter:	type: Outgoing parameter, tool power supply voltage type.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.9.3 Get the power supply voltage of the tool end

int robotServiceGetToolPowerVoltageStatus(double &value);	
Description:	Get the power supply voltage of the tool end.
Parameter:	value: Outgoing parameter, the power supply voltage of the tool.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.9.4 Set the power supply voltage type of the tool end and the type of all digital IOs

int robotServiceSetToolPowerTypeAndDigitalIOType(aubo_robot_namespace::ToolPowerType type, aubo_robot_namespace::ToolIOType io0, aubo_robot_namespace::ToolIOType io1, aubo_robot_namespace::ToolIOType io2);	
Description:	Set the power supply voltage type of the tool end and the type of all digital IOs.
Parameter:	1. type Power supply voltage type. 2. io0 Tool Type of IO 0. 3. io1 Tool Type of IO 1. 4. io2 Tool Type of IO 2. 5. io3 Tool Type of IO 3.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.9.5 Set the type of tool-side digital IO: input or output

int robotServiceSetToolDigitalIOType(aubo_robot_namespace::ToolDigitalIOAdd addr, aubo_robot_namespace::ToolIOType type);	
Description:	Set the type of tool-side digital IO: input or output.
Parameter:	1. addr: IO address. 2. type: IO type: input or output.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.9.6 Get the status of all digital IOs on the tool side

int robotServiceGetAllToolDigitalIOStatus(std::vector<aubo_robot_namespace::RobotIoDesc> &statusVector);	
Description:	Get the status of all digital IOs on the tool side.
Parameter:	statusVector: Outgoing parameter, IO status container.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.9.7 Set the status of the tool-side digital IO according to the address

int robotServiceSetToolDOStatus(aubo_robot_namespace::ToolDigitalIOAddr addr, aubo_robot_namespace::IO_STATUS value);	
Description:	Set the status of the tool-side digital IO according to the address.
Parameter:	1. addr: IO address. 2. value: IO status.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.9.8 Set the status of the tool-side digital IO according to the name

int robotServiceSetToolDOStatus(std::string name, aubo_robot_namespace::IO_STATUS value);	
Description:	Set the status of the tool-side digital IO according to the name.
Parameter:	1. name: IO name. 2. value: IO status value.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.9.9 Get the status of the tool-side IO according to the name

int robotServiceGetToolIoStatus(std::string name, double &value);	
Description:	Get the status of the tool-side IO according to the name.
Parameter:	1. name: IO name. 2. value: Outgoing parameter, IO status.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

2.9.10 Get the status of all AI on the tool side

int robotServiceGetAllToolAIStatus (std::vector<aubo_robot_namespace::RobotIoDesc> &statusVector);	
Description:	Get the status of all AI on the tool side.
Parameter:	statusVector: Outgoing parameter, tool-side AI container.
Return:	Success: Returns ErrnoSucc.
	Failed: Returns an error number.

3 Error codes

3.1 Interface function error code definition

Error no.	Error code	Error info
0	InterfaceCallSuccCode	Successful
10000	ErrCode_Base	
10001	ErrCode_Failed	generic failure
10002	ErrCode_ParamError	Parameter error
10003	ErrCode_ConnectSocketFailed	Socket connection failed
10004	ErrCode_SocketDisconnect	Socket disconnected
10005	ErrCode_CreateRequestFailed	Create request failed
10006	ErrCode_RequestRelatedVariableError	Request related internal variable error
10007	ErrCode_RequestTimeout	Request timed out
10008	ErrCode_SendRequestFailed	Failed to send request information
10009	ErrCode_ResponseInfoIsNULL	The response information is empty
10010	ErrCode_ResolveResponseFailed	Failed to parse response
10011	ErrCode_FkFailed	Forward kinematic error
10012	ErrCode_IkFailed	Inverse Kinematic error
10013	ErrCode_ToolCalibrateError	Tool calibration parameters are wrong
10014	ErrCode_ToolCalibrateParamError	Tool calibration parameters are wrong
10015	ErrCode_CoordinateSystemCalibrateError	Coordinate system calibration failed
10016	ErrCode_BaseToUserConvertFailed	Failed to convert base coordinate system to user

		coordinate system
10017	ErrCode_UserToBaseConvertFailed	User coordinate system to base coordinate system failed
10018	ErrCode_MotionRelatedVariableError	Motion related internal variable error
10019	ErrCode_MotionRequestFailed	motion request failed
10020	ErrCode_CreateMotionRequestFailed	Failed to generate motion request
10021	ErrCode_MotionInterruptedByEvent	Movement interrupted by event
10022	ErrCode_MotionWaypointVetorSizeError	Movement-related waypoint container length is not specified
10023	ErrCode_ResponseReturnError	The server responded with an error
10024	ErrCode_RealRobotNoExist	The real manipulator does not exist, because some interfaces can only be called if the real manipulator exists
10025	ErrCode_moveControlSlowStopFailed	Failed to call slow stop interface
10026	ErrCode_moveControlFastStopFailed	Failed to call emergency stop interface
10027	ErrCode_moveControlPauseFailed	Failed to call the pause interface
10028	ErrCode_moveControlContinueFailed	Failed to call continue interface

3.2 Error code due to controller exception event

Error number	Error code	Error message

21001	ErrCodeMoveJConfigError	Incorrect configuration of joint move properties
21002	ErrCodeMoveLConfigError	Line motion properties are misconfigured
21003	ErrCodeMovePConfigError	Incorrect configuration of trajectory motion properties
21004	ErrCodeInvailConfigError	Invalid motion attribute configuration
21005	ErrCodeWaitRobotStopped	Wait for the robot to stop
21006	ErrCodeJointOutRange	Joint out of range
21007	ErrCodeFirstWaypointSetError	Please set the MOVEP first waypoint correctly
21008	ErrCodeConveyorTrackConfigError	Conveyor Tracking Configuration Error
21009	ErrCodeConveyorTrackTrajectoryTypeError	conveyor track type error
21010	ErrCodeRelativeTransformIKFailed	Relative coordinate transformation inverse solution failed
21011	ErrCodeTeachModeCollision	Collision in teach mode
21012	ErrCodeextErnalToolConfigError	Misconfigured kinematic properties, misconfigured external tool or hand-held workpiece
21101	ErrCodeTrajectoryAbnormal	Trajectory abnormality
21102	ErrCodeOnlineTrajectoryPlanError	Trajectory planning error

21103	ErrCodeOnlineTrajectoryTypeIIError	Type II online trajectory planning failed
21104	ErrCodeIKFailed	Inverse solution failed
21105	ErrCodeAbnormalLimitProtect	Kinetic limit protection
21106	ErrCodeConveyorTrackingFailed	Conveyor tracking failed
21107	ErrCodeConveyorOutWorkingRange	beyond the working range of the conveyor belt
21108	ErrCodeTrajectoryJointOutOfRange	Joint out of range
21109	ErrCodeTrajectoryJointOverspeed	joint overspeed
21110	ErrCodeOfflineTrajectoryPlanFailed	Offline trajectory planning failed
21111	ErrCodeTrajectoryJointAccOutOfRange	The trajectory is abnormal, the joint acceleration exceeds the limit
21120	ErrCodeForceModeException	Abnormal force control mode
21121	ErrCodeForceModeIKFailed	The trajectory is abnormal, and it fails in force control mode
21122	ErrCodeForceModeTrackJointverspeed	joint overspeed
21200	ErrCodeControllerIKFailed	The controller is abnormal and the inverse solution fails
21201	ErrCodeControllerStatusException	The controller is abnormal, the state is abnormal
21202	ErrCodeControllerTrackingLost	Joint tracking error is too large
21203	ErrCodeMonitorErrTrackingLost	Joint tracking error is too large
21204	ErrCodeMonitorErrNoArrivalInTime	reserved
21205	ErrCodeMonitorErrCurrentOverload	reserved
21206	ErrCodeMonitorErrJointOutOfRange	The joint of the exceeds the limit
21207	ErrCodeFifoDataTimeNotRead	Cache timed out not updated

21300	ErrCodeMoveEnterStopState	The movement enters the stop stage
21301	ErrCodeMoveInterruptedByEvent	Movement interrupted by unknown event

3.3 Error codes due to abnormal events at the hardware layer

Error number	Error code	Error message
22001	ErrCodeHardwareErrorNotify	Robot arm hardware error, Cannot distinguish which hardware exception will return this error
22101	ErrCodeJointError	Robot arm joint error
22102	ErrCodeJointOverCurrent	Robot arm joint overcurrent
22103	ErrCodeJointOverVoltage	Robot arm joint over voltage
22104	ErrCodeJointLowVoltage	Robot arm joint undervoltage
22105	ErrCodeJointOverTemperature	Robot arm joint over temperature
22106	ErrCodeJointHallError	Robot arm joint Hall error
22107	ErrCodeJointEncoderError	Robot arm joint encoder error
22108	ErrCodeJointAbsoluteEncoderError	Robot arm joint absolute encoder error
22109	ErrCodeJointCurrentDetectError	The current position of the robot arm joint is wrong
22110	ErrCodeJointEncoderPollustion	The robot arm joint encoder is polluted. Recommended Action: Warning Notice

22111	ErrCodeJointEncoderZSignalError	Robot arm joint encoder Z signal error
22112	ErrCodeJointEncoderCalibrateInvalid	Robot arm joint encoder calibration failure
22113	ErrCodeJoint_IMU_SensorInvalid	Robot arm joint IMU sensor failure
22114	ErrCodeJointTemperatureSensorError	Robot arm joint temperature sensor error
22115	ErrCodeJointCanBusError	Robot arm joint CAN bus error
22116	ErrCodeJointCurrentError	The Joint current error
22117	ErrCodeJointCurrentPositionError	The current position of the robot arm joint is wrong
22118	ErrCodeJointOverSpeed	Robot arm joint overspeed
22119	ErrCodeJointOverAccelerate	Robot arm joint acceleration is too large error
22120	ErrCodeJointTraceAccuracy	Robot arm joint tracking accuracy error
22121	ErrCodeJointTargetPositionOutOfRange	The target position of the robot arm joint is out of range
22122	ErrCodeJointTargetSpeedOutOfRange	The target speed of the robot arm joint is out of range
22123	ErrCodeJointCollision	Suggested Action: Pause current exercise
22200	ErrCodeDataAbnormal	Robot arm information is abnormal
22201	ErrCodeRobotTypeError	Wrong type of arm

22202	ErrCodeAccelerationSensorError	Robot arm accelerometer chip error
22203	ErrCodeEncoderLineError	Robot arm encoder line number error
22204	ErrCodeEnterDragAndTeachModeError	The robot arm enters the drag teaching mode error
22205	ErrCodeExitDragAndTeachModeError	The robot arm exits the drag teaching mode error
22206	ErrCodeMACDataInterruptionError	Robotic arm MAC data interruption error
22207	ErrCodeDriveVersionError	Wrong drive version (inconsistent joint firmware version)
22300	ErrCodeInitAbnormal	Robot arm initialization is abnormal
22301	ErrCodeDriverEnableFailed	Robot arm driver enable failed
22302	ErrCodeDriverEnableAutoBackFailed	The robot arm driver fails to enable automatic response
22303	ErrCodeDriverEnableCurrentLoopFailed	Robot arm driver failed to enable current loop
22304	ErrCodeDriverSetTargetCurrentFailed	The robot arm driver failed to set the target current
22305	ErrCodeDriverReleaseBrakeFailed	Robot arm failed to release brake
22306	ErrCodeDriverEnablePostionLoopFailed	The robot arm fails to enable the position loop
22307	ErrCodeSetMaxAccelerateFailed	Failed to set max acceleration
22400	ErrCodeSafetyError	Robot arm safety

		error
22401	ErrCodeExternEmergencyStop	External emergency stop of the robotic arm
22402	ErrCodeSystemEmergencyStop	Emergency stop of the robotic arm system
22403	ErrCodeTeachpendantEmergencyStop	Robot arm teach pendant emergency stop
22404	ErrCodeControlCabinetEmergencyStop	Robot arm control cabinet emergency stop
22405	ErrCodeProtectionStopTimeout	Robot arm protection stop timeout
22406	ErrCodeEeducedModeTimeout	Robot Reduced Mode Timeout
22500	ErrCodeSystemAbnormal	The robot arm system is abnormal
22501	ErrCode_MCU_CommunicationAbnormal	The communication of the robot arm mcu is abnormal
22502	ErrCode485CommunicationAbnormal	The communication of the robot arm 485 is abnormal
22550	ErrCodeSoftEmergency	soft stop
22600	ErrCodeArmPowerOff	The contactor of the control box is disconnected, causing the 48V power failure of the robotic arm

4 Use Cases

4.1 Using SDK to build the simplest control project of manipulator

This case is to use SDK to build the simplest control project of manipulator. The main flow of the program is: manipulator login; initialization; Simulation service; Arm shutdown; Log out.

Note: change the IP address (server_host) in the following code to the IP address of the corresponding server.

Code of **example_0.h** as follow

```
#ifndef EXAMPLE_0_H
#define EXAMPLE_0_H

class Example_0
{
public:
    Example_0();

    /**
     * @brief demo
     *
     * Using SDK to build the simplest control project of manipulator
     */
    static void demo();
};

#endif // EXAMPLE_0_H
```

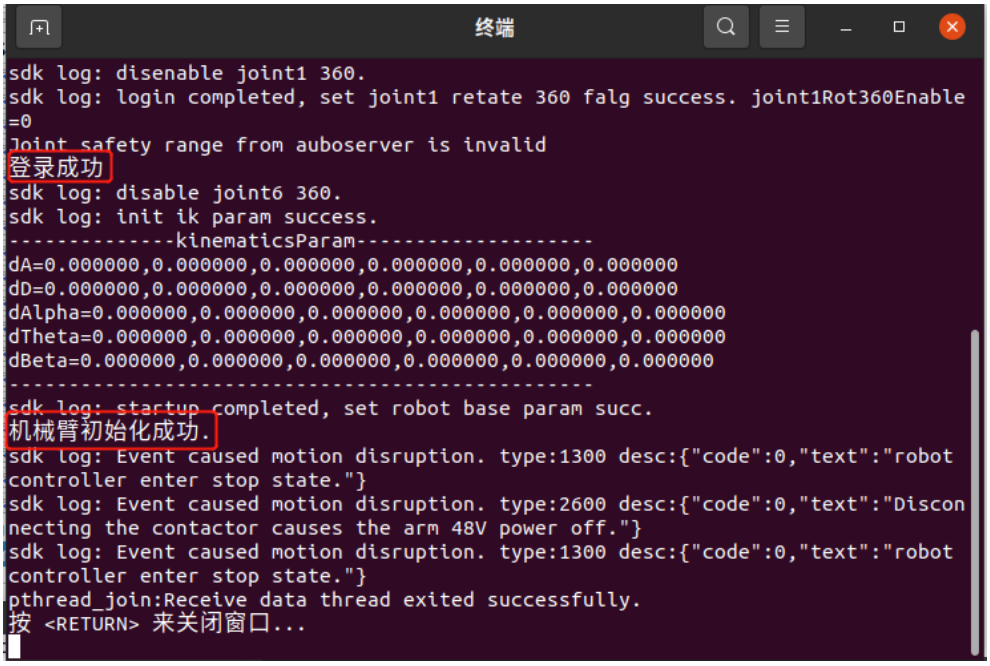
Code of **example_0.cpp** as follow:

```
#include "example_0.h"

#include <unistd.h>
#include <math.h>
#include <string.h>
#include <stdio.h>
#include <sstream>
#include <fstream>

#include "AuboRobotMetaType.h"
#include "serviceinterface.h"
```

```
#define SERVER_HOST "127.0.0.1"
#define SERVER_PORT 8899

Example_0::Example_0()
{
}

void Example_0::demo()
{

    ServiceInterface robotService;

    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Interface call: Login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login fail" << std::endl;
    }

    /** If a real manipulator is connected, the manipulator needs to be initialized **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    // Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam
    /** Tool dynamics parameters **/,
                        6          /*Collision Level*/,
                        true        /* Whether to allow reading pose, is true by default */,
                        true,       /* Leave the default to true */
                        1000,       /* Leave the default as 1000*/
                                                result); /* Robot initialization */
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cerr<<" Initialized successfully."<<std::endl;
    }
    else
```

```
    {
        std::cerr<<" Initialization failed."<<std::endl;
    }

    /** Simulated service **/
    sleep(5);

    /** Robot Shutdown **/
    robotService.robotServiceRobotShutdown();

    /** Interface call: exit login **/
    robotService.robotServiceLogout();
}
```

If the communication is successful, the statements of "login successful" and "robot arm initialization successful" will be printed.



4.2 Using callback functions to obtain real-time waypoints, end speeds, robotic arm events, and joint states

In this case, the callback function is used to obtain real-time information, including real-time waypoint information, real-time end speed, real-time arm events, and real-time joint status.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

Code of **example_1.h** as fellow:

```
#ifndef EXAMPLE_1_H
#define EXAMPLE_1_H
#include "AuboRobotMetaType.h"
#include "serviceinterface.h"


class Example_1
{
public:
    Example_1();

public:
   // Callback function for obtaining real-time waypoints
    static void RealTimeWaypointCallback (const
aubo_robot_namespace::wayPoint_S *wayPointPtr, void *arg);

  //Get real-time end speed callback function
    static void RealTimeEndSpeedCallback (double speed, void *arg);

  // Get real-time arm event callback function
    static void RealTimeEventInfoCallback(const
aubo_robot_namespace::RobotEventInfo *pEventInfo, void *arg);

  // Get real-time robotic arm joint status
    static void RealTimeJointStatusCallback(const aubo_robot_namespace::JointStatus
*jointStatusPtr, int size, void *arg);

    /**
     * @brief demo
     *
     * Get real-time waypoints, end speed, and events of the robotic arm by means of a
callback function
     */
    static void demo();
};

#endif // EXAMPLE_1_H
```

Code of **example_1.cpp** as follow:

Note: The functions of the Util class are used in the code, please refer to section 4.15.

```
#include "example_1.h"

#include <unistd.h>
#include <math.h>
```

```
#include <string.h>
#include <stdio.h>
#include <sstream>
#include <fstream>

#include "util.h"

#define SERVER_HOST "127.0.0.1"
#define SERVER_PORT 8899


Example_1::Example_1()
{

}

void Example_1::RealTimeWaypointCallback(const
aubo_robot_namespace::wayPoint_S *wayPointPtr, void *arg)
{
    (void)arg;
    aubo_robot_namespace::wayPoint_S waypoint = *wayPointPtr;
    Util::printWaypoint(waypoint);
}

void Example_1::RealTimeEndSpeedCallback(double speed, void *arg)
{
    (void)arg;
    std::cout << " real-time end speed:" << speed << std::endl;
}

void Example_1::RealTimeEventInfoCallback(const
aubo_robot_namespace::RobotEventInfo *pEventInfo, void *arg)
{
    (void)arg;
    Util::printEventInfo(*pEventInfo);
}

void Example_1::RealTimeJointStatusCallback(const
aubo_robot_namespace::JointStatus *jointStatusPtr, int size, void *arg)
{
    (void)arg;
    Util::printJointStatus(jointStatusPtr, size);
}
```

```
void Example_1::demo()
{
    ServiceInterface robotService;

    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Interface call: Login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cerr << "Login successful." << std::endl;
    }
    else
    {
        std::cerr << "Login failed." << std::endl;
    }


/**If a real manipulator is connected, the manipulator needs to be initialized**/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    // Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));

    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/**tool dynamic
parameters**/,
                                        6              /*collision level*/,
                    true        /* Whether to allow reading pose, true by default*/,
                                        true,       /* Leave the default as true */
                                        1000,       /* Leave the default as 1000*/
                                                result); /*Arm initialization */
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << "Robot initialization failed." << std::endl;
    }

    // Get real-time waypoint information
```

```
robotService.robotServiceRegisterRealTimeRoadPointCallback(Example_1::RealTime
WaypointCallback, NULL);

    // Get real-time end speed information

robotService.robotServiceRegisterRealTimeEndSpeedCallback(Example_1::RealTimeE
ndSpeedCallback, NULL);

    // Get real-time event information

robotService.robotServiceRegisterRobotEventInfoCallback(Example_1::RealTimeEven
tInfoCallback, NULL);

    // Get real-time joint state information

robotService.robotServiceRegisterRealTimeJointStatusCallback(Example_1::RealTime
JointStatusCallback, NULL);

    sleep(10);

}
```

4.3 Forward kinematics

This case uses SDK to realize the function of forward and forward kinematics of robot.

The main flow of the program is: (1) robot login (2) initialization (3) obtain the current waypoint information (4) obtain the target position and orientation according to the joint angle of the current waypoint (5) call the inverse solution function to obtain the inverse solution set and the optimal inverse solution according to the position and attitude obtained by the forward solution.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

Code of **example_2.h** as follow:

```
#ifndef EXAMPLE_2_H
#define EXAMPLE_2_H


class Example_2
{
public:
    Example_2();
```

```
    /**
     * @brief demo
     *
     * Inverse Kinematics
     */
public:
    static void demo();
};

#endif // EXAMPLE_2_H
```

Code of **example_2.cpp** as follow:

```
#include "example_2.h"
#include "AuboRobotMetaType.h"
#include "serviceinterface.h"

#include <unistd.h>
#include <math.h>
#include <string.h>
#include <stdio.h>
#include <sstream>
#include <fstream>
#include <vector>

using namespace std;

#define SERVER_HOST "192.168.221.13"
#define SERVER_PORT 8899

Example_2::Example_2()
{

}

void Example_2::demo()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Interface call: Login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
```

```
        std::cout << "Login successful" << std::endl;
    }
    else
    {
        std::cerr << "Login failed" << std::endl;
    }

    /** If a real manipulator is connected, the manipulator needs to be initialized**/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    // Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6             /*collision level*/,
                                    true        /* Whether to allow reading pose, true by default */,
                                               true,       /* Leave the default as true */
                                               1000,       /* Leave the default as 1000*/
                                                     result); /* Robot initialization */
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << "Robot initialization failed." << std::endl;
    }

    // Get the current waypoint information of the arm
    aubo_robot_namespace::wayPoint_S currentWaypoint;
    robotService.robotServiceGetCurrentWaypointInfo(currentWaypoint);

    //According to the joint angle of the current waypoint, the forward kinematics
obtains the target waypoint
    aubo_robot_namespace::wayPoint_S targetWaypoint;
    ret = robotService.robotServiceRobotFk(currentWaypoint.jointpos,
aubo_robot_namespace::ARM_DOF, targetWaypoint);
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "--------------------Forward kinematics --------------------" <<
std::endl;
        std::cout << " Path waypoint position obtained by FK：  "
                  << " x = " << targetWaypoint.cartPos.position.x
```

```
                        << " y = " << targetWaypoint.cartPos.position.y
                        << " z = " << targetWaypoint.cartPos.position.z
                        << std::endl;
            std::cout << " The pose of the target waypoint obtained from the FK
(quaternion)：  "
                        << " w = " << targetWaypoint.orientation.w
                        << " x = " << targetWaypoint.orientation.x
                        << " y = " << targetWaypoint.orientation.y
                        << " z = " << targetWaypoint.orientation.z
                        << std::endl;
            // Quaternion to Euler Angles
            aubo_robot_namespace::Rpy rpy;
            robotService.quaternionToRPY(targetWaypoint.orientation, rpy);
            std::cout << " The pose of the target waypoint obtained from the FK (Euler
angle):"
                        << " RX = " << rpy.rx*180/M_PI
                        << " RY = " << rpy.ry*180/M_PI
                        << " RZ = " << rpy.rz*180/M_PI
                        << std::endl;
        }
        else
        {
            std::cerr << " Call FK function failed " << std::endl;
        }

        // According to the position and pose obtained from the forward, the inverse
solution set is obtained
        // The position of the path waypoint obtained by the forward solution
        aubo_robot_namespace::Pos position = targetWaypoint.cartPos.position;

        // The position of the path waypoint obtained by the forward solution
        aubo_robot_namespace::Ori orientation = targetWaypoint.orientation
        std::vector<aubo_robot_namespace::wayPoint_S> wayPointVector;
        ret = robotService.robotServiceRobotIk(position, orientation, wayPointVector);

        std::cout << std::endl;
        std::cout << "---------- Inverse solution set ----------" <<
std::endl;
        std::cout << " Size of inverse solution set：  " << wayPointVector.size() <<
std::endl;
        for(int i = 0; i < wayPointVector.size(); i++)
        {
            std::cout << "first" << i+1 << " Group solution：" << std::endl;
            for(int j = 0; j < 6; j++)
```

```
            {
                std::cout << "joint" << j+ 1 << ": " <<
wayPointVector[i].jointpos[j]*180/M_PI << std::endl;
            }
        }



        // Obtain the optimal inverse solution according to the current waypoint
        aubo_robot_namespace::wayPoint_S wayPoint;
        robotService.robotServiceGetCurrentWaypointInfo(currentWaypoint);
        ret = robotService.robotServiceRobotIk(currentWaypoint.jointpos, position,
orientation, wayPoint);
        std::cout << std::endl;
        if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
        {
            std::cout << "---------------------- optimal inverse solution ----------------------"
<< std::endl;
            for(int i = 0; i < 6; i++)
            {
                std::cout << "joint" << i+ 1 << ": " << wayPoint.jointpos[i]*180/M_PI
<< std::endl;
            }

        }
        else
        {
            std::cerr << " Call inverse solution function failed " << std::endl;
        }
}
```

4.4 Coordinate system transformation

4.4.1 BaseToUserCoordinate()

baseToUserCoordinate Function example 1

This example is to convert the position and pose of the flange center in the base coordinate system into the position and position of the tool in the user coordinate system.

The main flow of this program is: (1) Manipulator login (2) Initialization (3) Setting the position and pose of the flange center in the base coordinate system (3) Setting the user coordinate system and tool parameters of the coordinate system (4) Set the tool parameters (5) call the baseToUserCoordinate function to obtain the position and pose of the tool in

the user coordinate system

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The code of the function baseToUserCoordinate() in example_3.cpp is as follows:

```
void Example_3::baseToUserCoordinate1()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Call interface: Login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "Login failed" << std::endl;
    }

    /** If a real manipulator is connected, the manipulator needs to be initialized **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    // Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6              /*collision level*/,
                                    true        /* Whether to allow reading pose, true by default */,
                                                true,       /* Leave the default as true */
                                                1000,       /* Leave the default as 1000 */
                                                    result); /* Arm initialization */
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }
//Position of flange center in base coordinate system
```

```
    aubo_robot_namespace::Pos flangeCenterPosOnBase;
    flangeCenterPosOnBase.x = 0.247248;
    flangeCenterPosOnBase.y = 0.280197;
    flangeCenterPosOnBase.z = 0.322796;

// Pose of flange center in base coordinate system (Euler angle)
    aubo_robot_namespace::Rpy flangeCenterRpyOnBase;
    flangeCenterRpyOnBase.rx = -101.335228*M_PI/180;
    flangeCenterRpyOnBase.ry = 8.768851*M_PI/180;
    flangeCenterRpyOnBase.rz = 93.718178*M_PI/180;

//Pose of flange center in base coordinate system (quaternion)
aubo_robot_namespace::Ori flangeCenterOriOnBase;
    robotService.RPYToQuaternion(flangeCenterRpyOnBase,
flangeCenterOriOnBase);

// User coordinate system
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool userCoord;
    userCoord.coordType = aubo_robot_namespace::WorldCoordinate;
    userCoord.methods =
aubo_robot_namespace::Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrant
OfXOYPlane;

    userCoord.wayPointArray[0].jointPos[0] = -75.093279*M_PI/180;
    userCoord.wayPointArray[0].jointPos[1] = 28.544643*M_PI/180;
    userCoord.wayPointArray[0].jointPos[2] = -114.313905*M_PI/180;
    userCoord.wayPointArray[0].jointPos[3] = -62.769247*M_PI/180;
    userCoord.wayPointArray[0].jointPos[4] = -87.343517*M_PI/180;
    userCoord.wayPointArray[0].jointPos[5] = -27.888262*M_PI/180;

    userCoord.wayPointArray[1].jointPos[0] = -89.239837*M_PI/180;
    userCoord.wayPointArray[1].jointPos[1] = 23.936171*M_PI/180;
    userCoord.wayPointArray[1].jointPos[2] = -122.299277*M_PI/180;
    userCoord.wayPointArray[1].jointPos[3] = -65.208902*M_PI/180;
    userCoord.wayPointArray[1].jointPos[4] = -85.011123*M_PI/180;
    userCoord.wayPointArray[1].jointPos[5] = -41.87417*M_PI/180;

    userCoord.wayPointArray[2].jointPos[0] = -77.059212*M_PI/180;
    userCoord.wayPointArray[2].jointPos[1] = 35.509518*M_PI/180;
    userCoord.wayPointArray[2].jointPos[2] = -101.108547*M_PI/180;
    userCoord.wayPointArray[2].jointPos[3] = -56.433133*M_PI/180;
    userCoord.wayPointArray[2].jointPos[4] = -87.006734*M_PI/180;
    userCoord.wayPointArray[2].jointPos[5] = -29.827440*M_PI/180;
```

```
        aubo_robot_namespace::ToolInEndDesc toolUserCoord;
        toolUserCoord.toolInEndPosition.x = 0;
        toolUserCoord.toolInEndPosition.y = 0;
        toolUserCoord.toolInEndPosition.z = 0.45;
        toolUserCoord.toolInEndOrientation.w = 1;
        toolUserCoord.toolInEndOrientation.x = 0;
        toolUserCoord.toolInEndOrientation.y = 0;
        toolUserCoord.toolInEndOrientation.z = 0;
        userCoord.toolDesc = toolUserCoord;

        aubo_robot_namespace::ToolInEndDesc toolInEndDesc;
        toolInEndDesc.toolInEndPosition.x = 0;
        toolInEndDesc.toolInEndPosition.y = 0;
        toolInEndDesc.toolInEndPosition.z = 0.45;
        toolInEndDesc.toolInEndOrientation.w = 1;
        toolInEndDesc.toolInEndOrientation.x = 0;
        toolInEndDesc.toolInEndOrientation.y = 0;
        toolInEndDesc.toolInEndOrientation.z = 0;

//The position of the end of the tool in the user coordinate system
        aubo_robot_namespace::Pos toolEndPosOnUser;
        // The pose of the tool end in the user coordinate system
        aubo_robot_namespace::Ori toolEndOriOnUser;

//Conversion the position and attitude of the flange center in the base coordinate system
into the position and attitude of the tool end in the user coordinate system
        robotService.baseToUserCoordinate(flangeCenterPosOnBase,
flangeCenterOriOnBase, userCoord, toolInEndDesc, toolEndPosOnUser,
toolEndOriOnUser);

        std::cout << " The position of the tool end in the user coordinate system: ";
        std::cout << "(" << toolEndPosOnUser.x << ", " << toolEndPosOnUser.y << ", "
<< toolEndPosOnUser.z << ")";
        std::cout << std::endl;

        std::cout << " The pose of the tool end in the user coordinate system (quaternion):
";
        std::cout << "(" << toolEndOriOnUser.w << ", " << toolEndOriOnUser.x << ", "
<< toolEndOriOnUser.y << ", " << toolEndOriOnUser.z << ")";
        std::cout << std::endl;

        aubo_robot_namespace::Rpy toolEndRpyOnUser;
        robotService.quaternionToRPY(toolEndOriOnUser, toolEndRpyOnUser);
        std::cout << "The pose of the tool tip in the user coordinate system (Eulerian
```

```
angles): ";
    std::cout << "(" << toolEndRpyOnUser.rx*180/M_PI << ", " <<
toolEndRpyOnUser.ry*180/M_PI << ", " << toolEndRpyOnUser.rz*180/M_PI << ")";
    std::cout << std::endl;

}
```

baseToUserCoordinate Function example 2

This example is to convert the position and pose of the flange center in the base coordinate system into the position and pose of the flange center in the user coordinate system. The center of the flange can be regarded as a special tool with a position (0,0,0) and an attitude (1,0,0,0).

The main flow of this program is: (1) Manipulator login (2) Initialization (3) Setting the position and pose of the flange center in the base coordinate system (3) Setting the user coordinate system and tool parameters of the coordinate system (4) Set the tool parameter to the center of the flange (5) call the baseToUserCoordinate function to obtain the position and pose of the center of the flange in the user coordinate system.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The code of the function baseToUserCoordinate2() in example_3.cpp is as follows:

```
void Example_3::baseToUserCoordinate2()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If a real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    // Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
```

```
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6              /*collision level*/,
                                    true        /* Whether to allow reading pose, true by default */,
                                                 true,       /* Leave the default as true */
                                                1000,        /* Leave the default as 1000 */
                                                     result); /* Arm initialization */
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    //Position of flange center in base coordinate system
    aubo_robot_namespace::Pos flangeCenterPosOnBase;
    flangeCenterPosOnBase.x = 0.247248;
    flangeCenterPosOnBase.y = 0.280197;
    flangeCenterPosOnBase.z = 0.322796;

    // Pose of flange center in base coordinate system (Euler angle)
    aubo_robot_namespace::Rpy flangeCenterRpyOnBase;
    flangeCenterRpyOnBase.rx = -101.335228*M_PI/180;
    flangeCenterRpyOnBase.ry = 8.768851*M_PI/180;
    flangeCenterRpyOnBase.rz = 93.718178*M_PI/180;

    //Pose of flange center in base coordinate system (quaternion)
aubo_robot_namespace::Ori flangeCenterOriOnBase;
    robotService.RPYToQuaternion(flangeCenterRpyOnBase,
flangeCenterOriOnBase);

// User coordinate system
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool userCoord;
    userCoord.coordType = aubo_robot_namespace::WorldCoordinate;
    userCoord.methods =
aubo_robot_namespace::Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrant
OfXOYPlane;

    userCoord.wayPointArray[0].jointPos[0] = -75.093279*M_PI/180;
    userCoord.wayPointArray[0].jointPos[1] = 28.544643*M_PI/180;
    userCoord.wayPointArray[0].jointPos[2] = -114.313905*M_PI/180;
```

```
    userCoord.wayPointArray[0].jointPos[3] = -62.769247*M_PI/180;
    userCoord.wayPointArray[0].jointPos[4] = -87.343517*M_PI/180;
    userCoord.wayPointArray[0].jointPos[5] = -27.888262*M_PI/180;

    userCoord.wayPointArray[1].jointPos[0] = -89.239837*M_PI/180;
    userCoord.wayPointArray[1].jointPos[1] = 23.936171*M_PI/180;
    userCoord.wayPointArray[1].jointPos[2] = -122.299277*M_PI/180;
    userCoord.wayPointArray[1].jointPos[3] = -65.208902*M_PI/180;
    userCoord.wayPointArray[1].jointPos[4] = -85.011123*M_PI/180;
    userCoord.wayPointArray[1].jointPos[5] = -41.87417*M_PI/180;

    userCoord.wayPointArray[2].jointPos[0] = -77.059212*M_PI/180;
    userCoord.wayPointArray[2].jointPos[1] = 35.509518*M_PI/180;
    userCoord.wayPointArray[2].jointPos[2] = -101.108547*M_PI/180;
    userCoord.wayPointArray[2].jointPos[3] = -56.433133*M_PI/180;
    userCoord.wayPointArray[2].jointPos[4] = -87.006734*M_PI/180;
    userCoord.wayPointArray[2].jointPos[5] = -29.827440*M_PI/180;

    aubo_robot_namespace::ToolInEndDesc toolUserCoord;
    toolUserCoord.toolInEndPosition.x = 0;
    toolUserCoord.toolInEndPosition.y = 0;
    toolUserCoord.toolInEndPosition.z = 0.45;
    toolUserCoord.toolInEndOrientation.w = 1;
    toolUserCoord.toolInEndOrientation.x = 0;
    toolUserCoord.toolInEndOrientation.y = 0;
    toolUserCoord.toolInEndOrientation.z = 0;
    userCoord.toolDesc = toolUserCoord;

    aubo_robot_namespace::ToolInEndDesc toolInEndDesc;
    toolInEndDesc.toolInEndPosition.x = 0;
    toolInEndDesc.toolInEndPosition.y = 0;
    toolInEndDesc.toolInEndPosition.z = 0;
    toolInEndDesc.toolInEndOrientation.w = 1;
    toolInEndDesc.toolInEndOrientation.x = 0;
    toolInEndDesc.toolInEndOrientation.y = 0;
    toolInEndDesc.toolInEndOrientation.z = 0;

    aubo_robot_namespace::Pos flangeCenterPosOnUser;
    aubo_robot_namespace::Ori flangeCenterOriOnUser;

    // Conversion of the position and attitude of the flange center in the base coordinate
system to the position and attitude of the flange center in the user coordinate system
    robotService.baseToUserCoordinate(flangeCenterPosOnBase,
flangeCenterOriOnBase, userCoord, toolInEndDesc, flangeCenterPosOnUser,
```

```
flangeCenterOriOnUser);

    std::cout << " The position of the flange center in the user coordinate system: ";
    std::cout << "(" << flangeCenterPosOnUser.x << ", " << flangeCenterPosOnUser.y
<< ", " << flangeCenterPosOnUser.z << ")";
    std::cout << std::endl;

    std::cout << " The pose of the flange center in the user coordinate system
(quaternion): ";
    std::cout << "(" << flangeCenterOriOnUser.w << ", " << flangeCenterOriOnUser.x
<< ", " << flangeCenterOriOnUser.y << ", " << flangeCenterOriOnUser.z << ")";
    std::cout << std::endl;

    aubo_robot_namespace::Rpy flangeCenterRpyOnUser;
    robotService.quaternionToRPY(flangeCenterOriOnUser,
flangeCenterRpyOnUser);
    std::cout << " The pose of the flange center in the user coordinate system (Eulerian
angle): ";
    std::cout << "(" << flangeCenterRpyOnUser.rx*180/M_PI << ", " <<
flangeCenterRpyOnUser.ry*180/M_PI << ", " <<
flangeCenterRpyOnUser.rz*180/M_PI << ")";
    std::cout << std::endl;

}
```

baseToUserCoordinate Function example 3

This example is to convert the position and pose of the flange center in the base coordinate system into the position and pose of the tool in the base coordinate system.

The main process of this program is: (1) robot login (2) initialization (3) setting the position and pose of flange center in the base coordinate system (3) setting the base coordinate system (4) setting tool parameters (5) calling the baseToUserCoordinate function to obtain the position and pose of the tool in the base coordinate system.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

Code of the function baseToUserCoordinate3() in example_3.cpp is as follows:

```
void Example_3::baseToUserCoordinate3()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
```

```
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If it is connected to a real robot arm, the robot arm needs to be initialized **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    // Tool Dynamics Parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool
Dynamics Parameters **/,
                                               6          /*collision level*/,
                        true        /* Whether to allow reading pose, true by default */,
                                          true,     /* Leave the default as true */
                                          1000,     /* Leave the default as 1000 */
                                              result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    //Position of flange center in base coordinate system
    aubo_robot_namespace::Pos flangeCenterPosOnBase;
    flangeCenterPosOnBase.x = -0.374741;
    flangeCenterPosOnBase.y = -0.534099;
    flangeCenterPosOnBase.z = 0.198738;

// Pose of flange center in base coordinate system (Euler angle)
    aubo_robot_namespace::Rpy flangeCenterRpyOnBase;
    flangeCenterRpyOnBase.rx = 179.956696*M_PI/180;
    flangeCenterRpyOnBase.ry = -5.516838*M_PI/180*M_PI/180;
    flangeCenterRpyOnBase.rz = -91.337303*M_PI/180;
```

```
        //Pose of flange center in base coordinate system (quaternion)
aubo_robot_namespace::Ori flangeCenterOriOnBase;
        // Euler angle to quaternion
robotService.RPYToQuaternion(flangeCenterRpyOnBase, flangeCenterOriOnBase);

        // User coordinate system
        aubo_robot_namespace::CoordCalibrateByJointAngleAndTool userCoord;
        baseCoord.coordType = aubo_robot_namespace::BaseCoordinate;
      // Tool parameters
        aubo_robot_namespace::ToolInEndDesc toolInEndDesc;
        toolInEndDesc.toolInEndPosition.x = 0;
        toolInEndDesc.toolInEndPosition.y = 0;
        toolInEndDesc.toolInEndPosition.z = 0.45;
        toolInEndDesc.toolInEndOrientation.w = 1;
        toolInEndDesc.toolInEndOrientation.x = 0;
        toolInEndDesc.toolInEndOrientation.y = 0;
        toolInEndDesc.toolInEndOrientation.z = 0;

//The position of the tool end in the base coordinate system
        aubo_robot_namespace::Pos toolEndPosOnBase;
        // The pose of the tool end in the base coordinate system
        aubo_robot_namespace::Ori toolEndOriOnBase;

   //Conversion of the position and pose of the flange center in the base coordinate
system into the position and pose of the tool under the base coordinate system
        robotService.baseToUserCoordinate(flangeCenterPosOnBase,
flangeCenterOriOnBase, baseCoord, toolInEndDesc, toolEndPosOnBase,
toolEndOriOnBase);

        std::cout << " The position of the tool end in the base coordinate system: ";
        std::cout << "(" << toolEndPosOnBase.x << ", " << toolEndPosOnBase.y << ", "
<< toolEndPosOnBase.z << ")";
        std::cout << std::endl;

      std::cout << " The pose of the tool end in the base coordinate system (quaternion): ";
        std::cout << "(" << toolEndOriOnBase.w << ", " << toolEndOriOnBase.x << ", "
<< toolEndOriOnBase.y << ", " << toolEndOriOnBase.z << ")";
        std::cout << std::endl;

        aubo_robot_namespace::Rpy toolEndRpyOnBase;
        robotService.quaternionToRPY(toolEndOriOnBase, toolEndRpyOnBase);
   std::cout << " The pose of the tool end in the base coordinate system (Euler angles): ";
        std::cout << "(" << toolEndRpyOnBase.rx*180/M_PI << ", " <<
```

```
toolEndRpyOnBase.ry*180/M_PI << ", " << toolEndRpyOnBase.rz*180/M_PI << ")";
    std::cout << std::endl;
}
```

4.4.2 Conversion of the base coordinate system to the base coordinate to get the position and pose of the end point of the tool: baseToBaseAdditionalTool()

baseToBaseAdditionalTool Function example

This example is to convert the position and pose of the flange center in the base coordinate system into the position and pose of the tool under the base coordinate system.

The main flow of this program is: (1) robot login (2) initialization (3) setting the position and attitude of flange center in the base coordinate system (4) setting tool parameters (5) calling the baseToBaseAdditionalTool function to obtain the position and attitude of the tool in the base coordinate system.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

Code of the function baseToBaseAdditionalTool1() in example_3.cpp is as follows:

```
void Example_3::baseToBaseAdditionalTool1()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    // Tool dynamics parameters
```

```
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                                6             /*collision level*/,
                            true        /* Whether to allow reading pose, true by default */,
                                              true,       /* Leave the default as true */
                                             1000,       /* Leave the default as 1000 */
                                                result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }


    //Position of flange center in base coordinate system
    aubo_robot_namespace::Pos flangeCenterPosOnBase;
    flangeCenterPosOnBase.x = 0.247248;
    flangeCenterPosOnBase.y = 0.280197;
    flangeCenterPosOnBase.z = 0.322796;

    // Pose of flange center in base coordinate system (Euler angle)
    aubo_robot_namespace::Rpy flangeCenterRpyOnBase;
    flangeCenterRpyOnBase.rx = -101.335228*M_PI/180;
    flangeCenterRpyOnBase.ry = 8.768851*M_PI/180;
    flangeCenterRpyOnBase.rz = 93.718178*M_PI/180;

    //Pose of flange center in base coordinate system (quaternion)
aubo_robot_namespace::Ori flangeCenterOriOnBase;
    robotService.RPYToQuaternion(flangeCenterRpyOnBase,
flangeCenterOriOnBase);

    // Tool parameters
    aubo_robot_namespace::ToolInEndDesc toolInEndDesc;
    toolInEndDesc.toolInEndPosition.x = 0;
    toolInEndDesc.toolInEndPosition.y = 0;
    toolInEndDesc.toolInEndPosition.z = 0.45;
    toolInEndDesc.toolInEndOrientation.w = 1;
    toolInEndDesc.toolInEndOrientation.x = 0;
```

```
    toolInEndDesc.toolInEndOrientation.y = 0;
    toolInEndDesc.toolInEndOrientation.z = 0;

    //The position of the tool end in the base coordinate system
    aubo_robot_namespace::Pos toolEndPosOnBase;
    // The pose of the tool end in the base coordinate system
    aubo_robot_namespace::Ori toolEndOriOnBase;

  //Conversion of the position and pose of the flange center in the base coordinate system to the position and pose of the tool in the base coordinate system
    robotService.baseToBaseAdditionalTool(flangeCenterPosOnBase, flangeCenterOriOnBase, toolInEndDesc, toolEndPosOnBase, toolEndOriOnBase);

    std::cout << " Position of the tool end in the base coordinate system: ";
    std::cout << "(" << toolEndPosOnBase.x << ", " << toolEndPosOnBase.y << ", " << toolEndPosOnBase.z << ")";
    std::cout << std::endl;

    std::cout << " Pose of the tool end in the base coordinate system (quaternion): ";
    std::cout << "(" << toolEndOriOnBase.w << ", " << toolEndOriOnBase.x << ", " << toolEndOriOnBase.y << ", " << toolEndOriOnBase.z << ")";
    std::cout << std::endl;

    aubo_robot_namespace::Rpy toolEndRpyOnBase;
    robotService.quaternionToRPY(toolEndOriOnBase, toolEndRpyOnBase);
    std::cout << " Pose of tool end in base coordinate system (Euler angle): ";
    std::cout << "(" << toolEndRpyOnBase.rx*180/M_PI << ", " << toolEndRpyOnBase.ry*180/M_PI << ", " << toolEndRpyOnBase.rz*180/M_PI << ")";
    std::cout << std::endl;

}
```

4.4.3 User coordinate system to base coordinate system

userToBaseCoordinate()

userToBaseCoordinate Function example 1

This example is to convert the position and pose of the tool end in the user coordinate system into the position and pose of the flange center in the base coordinate system.

The main flow of this program is: (1) Robot login (2) Initialization (3) Setting the position and pose of the tool end in the user coordinate system (4) Setting the user coordinate system and tool parameters of the coordinate system (5) Setting the tool Parameter (6) call the userToBaseCoordinate function to obtain the position and pose of the flange center

in the base coordinate system.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

Code of the function userToBaseCoordinate1() in example_3.cpp is as follows:

```
void Example_3::userToBaseCoordinate1()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6              /*collision level*/,
                        true        /* Whether to allow reading pose, true by default */,
                                          true,       /* Leave the default as true */
                                          1000,       /* Leave the default as 1000 */
                                              result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }
```

```
    //Position of the tool end in the user coordinate system
    aubo_robot_namespace::Pos toolEndPosOnUser;
    toolEndPosOnUser.x = 0.108191;
    toolEndPosOnUser.y = -0.319869;
    toolEndPosOnUser.z = 0.595867;

    //Pose of tool end in user coordinate system (quaternion)
    aubo_robot_namespace::Ori toolEndOriOnUser;
    //Pose of tool end in user coordinate system (Euler angle)
    aubo_robot_namespace::Rpy toolEndRpyOnUser;
    toolEndRpyOnUser.rx = -101.335*M_PI/180;
    toolEndRpyOnUser.ry = 8.76846*M_PI/180;
    toolEndRpyOnUser.rz = 93.7183*M_PI/180;
    robotService.RPYToQuaternion(toolEndRpyOnUser,toolEndOriOnUser);

    // Use coordinate system
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool userCoord;
    userCoord.coordType = aubo_robot_namespace::WorldCoordinate;
    userCoord.methods =
aubo_robot_namespace::Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrant
OfXOYPlane;

    userCoord.wayPointArray[0].jointPos[0] = -75.093279*M_PI/180;
    userCoord.wayPointArray[0].jointPos[1] = 28.544643*M_PI/180;
    userCoord.wayPointArray[0].jointPos[2] = -114.313905*M_PI/180;
    userCoord.wayPointArray[0].jointPos[3] = -62.769247*M_PI/180;
    userCoord.wayPointArray[0].jointPos[4] = -87.343517*M_PI/180;
    userCoord.wayPointArray[0].jointPos[5] = -27.888262*M_PI/180;

    userCoord.wayPointArray[1].jointPos[0] = -89.239837*M_PI/180;
    userCoord.wayPointArray[1].jointPos[1] = 23.936171*M_PI/180;
    userCoord.wayPointArray[1].jointPos[2] = -122.299277*M_PI/180;
    userCoord.wayPointArray[1].jointPos[3] = -65.208902*M_PI/180;
    userCoord.wayPointArray[1].jointPos[4] = -85.011123*M_PI/180;
    userCoord.wayPointArray[1].jointPos[5] = -41.87417*M_PI/180;

    userCoord.wayPointArray[2].jointPos[0] = -77.059212*M_PI/180;
    userCoord.wayPointArray[2].jointPos[1] = 35.509518*M_PI/180;
    userCoord.wayPointArray[2].jointPos[2] = -101.108547*M_PI/180;
    userCoord.wayPointArray[2].jointPos[3] = -56.433133*M_PI/180;
    userCoord.wayPointArray[2].jointPos[4] = -87.006734*M_PI/180;
    userCoord.wayPointArray[2].jointPos[5] = -29.827440*M_PI/180;
```

```
    aubo_robot_namespace::ToolInEndDesc toolUserCoord;
    toolUserCoord.toolInEndPosition.x = 0;
    toolUserCoord.toolInEndPosition.y = 0;
    toolUserCoord.toolInEndPosition.z = 0.45;
    toolUserCoord.toolInEndOrientation.w = 1;
    toolUserCoord.toolInEndOrientation.x = 0;
    toolUserCoord.toolInEndOrientation.y = 0;
    toolUserCoord.toolInEndOrientation.z = 0;
    userCoord.toolDesc = toolUserCoord;

    aubo_robot_namespace::ToolInEndDesc toolInEndDesc;
    toolInEndDesc.toolInEndPosition.x = 0;
    toolInEndDesc.toolInEndPosition.y = 0;
    toolInEndDesc.toolInEndPosition.z = 0.45;
    toolInEndDesc.toolInEndOrientation.w = 1;
    toolInEndDesc.toolInEndOrientation.x = 0;
    toolInEndDesc.toolInEndOrientation.y = 0;
    toolInEndDesc.toolInEndOrientation.z = 0;

    //Position of flange center in base coordinate system
    aubo_robot_namespace::Pos flangeCenterPosOnBase;
    //Pose of flange center in base coordinate system (quaternion)
    aubo_robot_namespace::Ori flangeCenterOriOnBase;

  //Conversion of the position and pose of the tool end in the user coordinate system to
the position and attitude of the flange center in the base coordinate system
    robotService.userToBaseCoordinate(toolEndPosOnUser, toolEndOriOnUser,
userCoord, toolInEndDesc, flangeCenterPosOnBase, flangeCenterOriOnBase);

    std::cout << " Position of flange center in base coordinate system: ";
    std::cout << "(" << flangeCenterPosOnBase.x << ", " <<
flangeCenterPosOnBase.y << ", " << flangeCenterPosOnBase.z << ")";
    std::cout << std::endl;

    std::cout << " Pose of flange center in base coordinate system (quaternion): ";
    std::cout << "(" << flangeCenterOriOnBase.w << ", " <<
flangeCenterOriOnBase.x << ", " << flangeCenterOriOnBase.y << ", " <<
flangeCenterOriOnBase.z << ")";
    std::cout << std::endl;

    //Pose of flange center in base coordinate system (Euler angle)
    aubo_robot_namespace::Rpy flangeCenterRpyOnBase;
    robotService.quaternionToRPY(flangeCenterOriOnBase,
flangeCenterRpyOnBase);
```

```
    std::cout << " Pose of flange center in base coordinate system (Euler angle): ";
    std::cout << "(" << flangeCenterRpyOnBase.rx*180/M_PI << ", " <<
flangeCenterRpyOnBase.ry*180/M_PI << ", " <<
flangeCenterRpyOnBase.rz*180/M_PI << ")";
    std::cout << std::endl;

}
```

userToBaseCoordinate Function example 2

This example is to convert the position and attitude of the flange center in the user coordinate system to the position and attitude of the flange center in the base coordinate system.

The main flow of this program is: (1) Robot login (2) Initialization (3) Setting the position and pose of the flange center in the user coordinate system (4) Setting the user coordinate system and its tool parameters (5) Setting the tool Parameters, call the userToBaseCoordinate function for the flange center (6) to obtain the position and pose of the flange center in the base coordinate system.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The code of the function userToBaseCoordinate2() in example_3.cpp is as follows:

```
void Example_3::userToBaseCoordinate2()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

        /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
```

```
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                                6             /*collision level*/,
                            true        /* Whether to allow reading pose, true by default */,
                                            true,       /* Leave the default as true */
                                            1000,       /* Leave the default as 1000 */
                                                result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    //Position of flange center in user coordinate system
    aubo_robot_namespace::Pos flangeCenterPosOnUser;
    flangeCenterPosOnUser.x = 0.547609;
    flangeCenterPosOnUser.y = -0.2778;
    flangeCenterPosOnUser.z = 0.683283;

    //Pose of flange center in user coordinate system (quaternion)
    aubo_robot_namespace::Ori flangeCenterOriOnUser;
    //Pose of flange center in user coordinate system (Euler angle)
    aubo_robot_namespace::Rpy flangeCenterRpyOnUser;
    flangeCenterRpyOnUser.rx = -101.335*M_PI/180;
    flangeCenterRpyOnUser.ry = 8.76846*M_PI/180;
    flangeCenterRpyOnUser.rz = 93.7183*M_PI/180;

robotService.RPYToQuaternion(flangeCenterRpyOnUser,flangeCenterOriOnUser);
    // User coordinate system
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool userCoord;
    userCoord.coordType = aubo_robot_namespace::WorldCoordinate;
    userCoord.methods =
aubo_robot_namespace::Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrant
OfXOYPlane;

    userCoord.wayPointArray[0].jointPos[0] = -75.093279*M_PI/180;
    userCoord.wayPointArray[0].jointPos[1] = 28.544643*M_PI/180;
    userCoord.wayPointArray[0].jointPos[2] = -114.313905*M_PI/180;
```

```
    userCoord.wayPointArray[0].jointPos[3] = -62.769247*M_PI/180;
    userCoord.wayPointArray[0].jointPos[4] = -87.343517*M_PI/180;
    userCoord.wayPointArray[0].jointPos[5] = -27.888262*M_PI/180;

    userCoord.wayPointArray[1].jointPos[0] = -89.239837*M_PI/180;
    userCoord.wayPointArray[1].jointPos[1] = 23.936171*M_PI/180;
    userCoord.wayPointArray[1].jointPos[2] = -122.299277*M_PI/180;
    userCoord.wayPointArray[1].jointPos[3] = -65.208902*M_PI/180;
    userCoord.wayPointArray[1].jointPos[4] = -85.011123*M_PI/180;
    userCoord.wayPointArray[1].jointPos[5] = -41.87417*M_PI/180;

    userCoord.wayPointArray[2].jointPos[0] = -77.059212*M_PI/180;
    userCoord.wayPointArray[2].jointPos[1] = 35.509518*M_PI/180;
    userCoord.wayPointArray[2].jointPos[2] = -101.108547*M_PI/180;
    userCoord.wayPointArray[2].jointPos[3] = -56.433133*M_PI/180;
    userCoord.wayPointArray[2].jointPos[4] = -87.006734*M_PI/180;
    userCoord.wayPointArray[2].jointPos[5] = -29.827440*M_PI/180;

    aubo_robot_namespace::ToolInEndDesc toolUserCoord;
    toolUserCoord.toolInEndPosition.x = 0;
    toolUserCoord.toolInEndPosition.y = 0;
    toolUserCoord.toolInEndPosition.z = 0.45;
    toolUserCoord.toolInEndOrientation.w = 1;
    toolUserCoord.toolInEndOrientation.x = 0;
    toolUserCoord.toolInEndOrientation.y = 0;
    toolUserCoord.toolInEndOrientation.z = 0;
    userCoord.toolDesc = toolUserCoord;

    aubo_robot_namespace::ToolInEndDesc toolInEndDesc;
    toolInEndDesc.toolInEndPosition.x = 0;
    toolInEndDesc.toolInEndPosition.y = 0;
    toolInEndDesc.toolInEndPosition.z = 0;
    toolInEndDesc.toolInEndOrientation.w = 1;
    toolInEndDesc.toolInEndOrientation.x = 0;
    toolInEndDesc.toolInEndOrientation.y = 0;
    toolInEndDesc.toolInEndOrientation.z = 0;
    //Position of flange center in base coordinate system
    aubo_robot_namespace::Pos flangeCenterPosOnBase;
    //Pose of flange center in base coordinate system (quaternion)
    aubo_robot_namespace::Ori flangeCenterOriOnBase;//

    //Conversion of the position and pose of the flange center in the user coordinate
system to the position and pose of the flange center in the base coordinate system
    robotService.userToBaseCoordinate(flangeCenterPosOnUser,
```

```
flangeCenterOriOnUser, userCoord, toolInEndDesc, flangeCenterPosOnBase,
flangeCenterOriOnBase);

    std::cout << " Position of flange center in base coordinate system: ";
    std::cout << "(" << flangeCenterPosOnBase.x << ", " <<
flangeCenterPosOnBase.y << ", " << flangeCenterPosOnBase.z << ")";
    std::cout << std::endl;

    std::cout << " Pose of flange center in base coordinate system (quaternion): ";
    std::cout << "(" << flangeCenterOriOnBase.w << ", " <<
flangeCenterOriOnBase.x << ", " << flangeCenterOriOnBase.y << ", " <<
flangeCenterOriOnBase.z << ")";
    std::cout << std::endl;

    //Pose of flange center in base coordinate system (Euler angle)
    aubo_robot_namespace::Rpy flangeCenterRpyOnBase;
    robotService.quaternionToRPY(flangeCenterOriOnBase,
flangeCenterRpyOnBase);
    std::cout << " Pose of flange center in base coordinate system (Euler angle): ";
    std::cout << "(" << flangeCenterRpyOnBase.rx*180/M_PI << ", " <<
flangeCenterRpyOnBase.ry*180/M_PI << ", " <<
flangeCenterRpyOnBase.rz*180/M_PI << ")";
    std::cout << std::endl;

}
```

userToBaseCoordinate Function example 3

This example is to convert the position and pose of the tool end in the base coordinate system to the position and pose of the flange center in the base coordinate system.

The main flow of this program is: (1) Robot login (2) initialization (3) setting the position and pose of the tool end in the base coordinate system (4) setting the base coordinate system (5) setting tool parameters (6) calling userToBaseCoordinate function to obtain the position and pose of the flange center in the base coordinate system.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

Code of the function userToBaseCoordinate3() in example_3.cpp is as follows

```
void Example_3::userToBaseCoordinate3()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Call interface: login ***/
```

```
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6              /*collision level*/,
                            true        /* Whether to allow reading pose, true by default */,
                                        true,       /* Leave the default as true */
                                        1000,       /* Leave the default as 1000 */
                                        result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    //The position of the tool end in the base coordinate system
    aubo_robot_namespace::Pos toolEndPosOnBase;
    toolEndPosOnBase.x = -0.375099;
    toolEndPosOnBase.y = -0.534847;
    toolEndPosOnBase.z = -0.251261;

    //Pose of the tool end in the base coordinate system (quaternion)
    aubo_robot_namespace::Ori toolEndOriOnBase;
    // Pose of the tool end in the base coordinate system (Euler angle)
    aubo_robot_namespace::Rpy toolEndRpyOnBase;
```

```
    toolEndRpyOnBase.rx = 179.957*M_PI/180;
    toolEndRpyOnBase.ry = -0.096287*M_PI/180;
    toolEndRpyOnBase.rz = -91.3373*M_PI/180;
    robotService.RPYToQuaternion(toolEndRpyOnBase, toolEndOriOnBase);

    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool baseCoord;
    baseCoord.coordType = aubo_robot_namespace::BaseCoordinate;

    aubo_robot_namespace::ToolInEndDesc toolInEndDesc;
    toolInEndDesc.toolInEndPosition.x = 0;
    toolInEndDesc.toolInEndPosition.y = 0;
    toolInEndDesc.toolInEndPosition.z = 0.45;
    toolInEndDesc.toolInEndOrientation.w = 1;
    toolInEndDesc.toolInEndOrientation.x = 0;
    toolInEndDesc.toolInEndOrientation.y = 0;
    toolInEndDesc.toolInEndOrientation.z = 0;

    //Position of flange center in base coordinate system
    aubo_robot_namespace::Pos flangeCenterPosOnBase;
    //Pose of flange center in base coordinate system (quaternion)
    aubo_robot_namespace::Ori flangeCenterOriOnBase;

    Conversion of the position and pose of the tool end under the base coordinate
system to the position and pose of the flange center under the base coordinate system
    robotService.userToBaseCoordinate(toolEndPosOnBase, toolEndOriOnBase,
baseCoord, toolInEndDesc, flangeCenterPosOnBase, flangeCenterOriOnBase);

    std::cout << " The position of the flange center in the base coordinate system: ";
    std::cout << "(" << flangeCenterPosOnBase.x << ", " <<
flangeCenterPosOnBase.y << ", " << flangeCenterPosOnBase.z << ")";
    std::cout << std::endl;

    std::cout << " The pose of the flange center in the base coordinate system
(quaternion): ";
    std::cout << "(" << flangeCenterOriOnBase.w << ", " <<
flangeCenterOriOnBase.x << ", " << flangeCenterOriOnBase.y << ", " <<
flangeCenterOriOnBase.z << ")";
    std::cout << std::endl;

    //The pose of the flange center in the base coordinate system (Eulerian angle)
    aubo_robot_namespace::Rpy flangeCenterRpyOnBase;
    robotService.quaternionToRPY(flangeCenterOriOnBase,
flangeCenterRpyOnBase);
    std::cout << " Pose of flange center in base coordinate system (Euler angle): ";
```

```
    std::cout << "(" << flangeCenterRpyOnBase.rx*180/M_PI << ", " <<
flangeCenterRpyOnBase.ry*180/M_PI << ", " <<
flangeCenterRpyOnBase.rz*180/M_PI << ")";
    std::cout << std::endl;

}
```

4.4.4 Position parameter base conversion coordinate system under user coordinate system userCoordPointToBasePoint()

userCoordPointToBasePoint Function example

This example converts the position of the tool end in the user coordinate system into the position of the tool end in the base coordinate system.

The main flow of this program is: (1) Robot login (2) Initialization (3) Setting the position of the tool end in the user coordinate system (4) Setting the user coordinate system and its tool parameters (5) Calling the userCoordPointToBasePoint function to obtain the tool The position of the end in the base coordinate system.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

Code of the function userCoordPointToBasePoint1() in example_3.cpp is as follows:

```
void Example_3::userCoordPointToBasePoint1()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }
```

```
    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                                6            /*collision level*/,
                        true        /* Whether to allow reading pose, true by default */,
                                        true,       /* Leave the default as true */
                                        1000,       /* Leave the default as 1000 */
                                            result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    //Position of the tool end in the user coordinate system
    aubo_robot_namespace::Pos toolEndPosOnUser;
    toolEndPosOnUser.x = -0.074733;
    toolEndPosOnUser.y = -1.092842;
    toolEndPosOnUser.z = 0.109217;

    //User coordinate system
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool userCoord;
    userCoord.coordType = aubo_robot_namespace::WorldCoordinate;
    userCoord.methods =
aubo_robot_namespace::Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrant
OfXOYPlane;

    userCoord.wayPointArray[0].jointPos[0] = -75.093279*M_PI/180;
    userCoord.wayPointArray[0].jointPos[1] = 28.544643*M_PI/180;
    userCoord.wayPointArray[0].jointPos[2] = -114.313905*M_PI/180;
    userCoord.wayPointArray[0].jointPos[3] = -62.769247*M_PI/180;
    userCoord.wayPointArray[0].jointPos[4] = -87.343517*M_PI/180;
    userCoord.wayPointArray[0].jointPos[5] = -27.888262*M_PI/180;

    userCoord.wayPointArray[1].jointPos[0] = -89.239837*M_PI/180;
```

```
    userCoord.wayPointArray[1].jointPos[1] = 23.936171*M_PI/180;
    userCoord.wayPointArray[1].jointPos[2] = -122.299277*M_PI/180;
    userCoord.wayPointArray[1].jointPos[3] = -65.208902*M_PI/180;
    userCoord.wayPointArray[1].jointPos[4] = -85.011123*M_PI/180;
    userCoord.wayPointArray[1].jointPos[5] = -41.87417*M_PI/180;

    userCoord.wayPointArray[2].jointPos[0] = -77.059212*M_PI/180;
    userCoord.wayPointArray[2].jointPos[1] = 35.509518*M_PI/180;
    userCoord.wayPointArray[2].jointPos[2] = -101.108547*M_PI/180;
    userCoord.wayPointArray[2].jointPos[3] = -56.433133*M_PI/180;
    userCoord.wayPointArray[2].jointPos[4] = -87.006734*M_PI/180;
    userCoord.wayPointArray[2].jointPos[5] = -29.827440*M_PI/180;

    aubo_robot_namespace::ToolInEndDesc toolUserCoord;
    toolUserCoord.toolInEndPosition.x = 0;
    toolUserCoord.toolInEndPosition.y = 0;
    toolUserCoord.toolInEndPosition.z = 0.45;
    toolUserCoord.toolInEndOrientation.w = 1;
    toolUserCoord.toolInEndOrientation.x = 0;
    toolUserCoord.toolInEndOrientation.y = 0;
    toolUserCoord.toolInEndOrientation.z = 0;
    userCoord.toolDesc = toolUserCoord;

    //Position of the tool end in the base coordinate system
    aubo_robot_namespace::Pos toolEndPosOnBase;
    // Conversion the pose of the flange center in the base coordinate system to the
attitude of the tool end in the base coordinate system
    robotService.userCoordPointToBasePoint(toolEndPosOnUser, userCoord,
toolEndPosOnBase);

    std::cout << " Position of the tool end in the base coordinate system: ";
    std::cout << "(" << toolEndPosOnBase.x << ", " << toolEndPosOnBase.y << ", "
<< toolEndPosOnBase.z << ")";
    std::cout << std::endl;
}
```

4.4.5 Flange pose to tool pose endOrientation2ToolOrientation()

endOrientation2ToolOrientation Function example

This example is to convert the pose of the flange center in the base coordinate system into the pose of the tool end in the base coordinate system.

The main flow of this program is: (1) Robot login (2) Initialization (3) Setting the pose of the flange center in the base coordinate system (4) Setting the tool pose parameters (5)

Calling the endOrientation2ToolOrientation function to get the tool end at The pose in the base coordinate system.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The code of the function endOrientation2ToolOrientation1() in example_3.cpp is as follows:

```
void Example_3::endOrientation2ToolOrientation1()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6              /*collision level*/,
                            true        /* Whether to allow reading pose, true by default */,
                                    true,       /* Leave the default as true */
                                    1000,       /* Leave the default as 1000 */
                                        result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
```

```
            std::cerr << " Robot initialization failed." << std::endl;
        }

        //Pose of flange center in base coordinate system (quaternion)
        aubo_robot_namespace::Ori flangeCenterOriOnBase;
        //Pose of flange center in base coordinate system (Euler angle)
        aubo_robot_namespace::Rpy flangeCenterRpyOnBase;
        flangeCenterRpyOnBase.rx = -176.317383*M_PI/180;
        flangeCenterRpyOnBase.ry = 19.773378*M_PI/180;
        flangeCenterRpyOnBase.rz = -169.362442*M_PI/180;
        robotService.RPYToQuaternion(flangeCenterRpyOnBase,
flangeCenterOriOnBase);

        aubo_robot_namespace::Ori toolOriParam;// Tool pose parameters (quaternion)
        aubo_robot_namespace::Rpy toolRpyParam;// Tool pose parameters (Euler angle)
        toolRpyParam.rx = -162.014703*M_PI/180;
        toolRpyParam.ry = -25.569674*M_PI/180;
        toolRpyParam.rz = -36.962539*M_PI/180;
        robotService.RPYToQuaternion(toolRpyParam, toolOriParam);
        //Pose of the tool end in the base coordinate system (quaternion)
        aubo_robot_namespace::Ori toolEndOriOnBase;
        // Conversion the attitude of the flange center in the base coordinate system to the
attitude of the tool end in the base coordinate system
        robotService.endOrientation2ToolOrientation(toolOriParam,
flangeCenterOriOnBase, toolEndOriOnBase);

        std::cout << " Pose of the tool end in the base coordinate system (quaternion): ";
        std::cout << "(" << toolEndOriOnBase.w << ", " << toolEndOriOnBase.x << ", "
<< toolEndOriOnBase.y << ", " << toolEndOriOnBase.z << ")";
        std::cout << std::endl;

        //Pose of tool end in base coordinate system (Euler angle)
        aubo_robot_namespace::Rpy toolEndRpyOnBase;
        robotService.quaternionToRPY(toolEndOriOnBase, toolEndRpyOnBase);
        std::cout << " Pose of tool end in base coordinate system (Euler angle): ";
        std::cout << "(" << toolEndRpyOnBase.rx*180/M_PI << ", " <<
toolEndRpyOnBase.ry*180/M_PI << ", " << toolEndRpyOnBase.rz*180/M_PI << ")";
        std::cout << std::endl;
}
```

4.4.6 Transformation of tool pose into end pose

toolOrientation2EndOrientation()

toolOrientation2EndOrientation Function example

This example is to convert the pose of the tool end in the base coordinate system to the pose of the flange center in the base coordinate system.

The main process of this program is: (1) robot login (2) initialization (3) setting the attitude of the pose end in the base coordinate system (4) setting the tool pose parameters (5) calling the **toolOrientation2EndOrientation** function to obtain the attitude of the flange center in the base coordinate system.

Note: Modify the IP address (SERVER_HOST) in the following code to the IP address of the corresponding server.

The code of the function toolOrientation2EndOrientation1() in example_3.cpp is as follows:

```
void Example_3::toolOrientation2EndOrientation1()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6            /*collision level*/,
```

```
                                true        /* Whether to allow reading pose, true by default */,
                                                    true,       /* Leave the default as true */
                                                   1000,       /* Leave the default as 1000 */
                                                        result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    //Pose of the tool end in the base coordinate system (quaternion)
    aubo_robot_namespace::Ori toolEndOriOnBase;
    //Attitude of tool end in base coordinate system (Euler angle)
    aubo_robot_namespace::Rpy toolEndRpyOnBase;
    toolEndRpyOnBase.rx = 36.627361*M_PI/180;
    toolEndRpyOnBase.ry = 38.051857*M_PI/180;
    toolEndRpyOnBase.rz = -123.093803*M_PI/180;
    robotService.RPYToQuaternion(toolEndRpyOnBase, toolEndOriOnBase);
    //Tool pose parameters (quaternion)
    aubo_robot_namespace::Ori toolOriParam;
    //Tool pose parameters (Euler angle)
    aubo_robot_namespace::Rpy toolRpyParam;
    toolRpyParam.rx = -162.014703*M_PI/180;
    toolRpyParam.ry = -25.569674*M_PI/180;
    toolRpyParam.rz = -36.962539*M_PI/180;
    robotService.RPYToQuaternion(toolRpyParam, toolOriParam);
    //The pose of the flange center in the base coordinate system (quaternion)
    aubo_robot_namespace::Ori flangeCenterOriOnBase;
    //Transfer of the pose of the tool end in the base coordinate system to the pose of
the flange center in the base coordinate system
    robotService.toolOrientation2EndOrientation(toolOriParam, toolEndOriOnBase,
flangeCenterOriOnBase);
    std::cout << " Pose of flange center in base coordinate system (quaternion): ";
    std::cout << "(" << flangeCenterOriOnBase.w << ", " <<
flangeCenterOriOnBase.x << ", " << flangeCenterOriOnBase.y << ", " <<
flangeCenterOriOnBase.z << ")";
    std::cout << std::endl;
    //Pose of flange center in base coordinate system (Euler angle)
    aubo_robot_namespace::Rpy flangeCenterRpyOnBase;
    robotService.quaternionToRPY(flangeCenterOriOnBase,
```

```
flangeCenterRpyOnBase);
    std::cout << " Pose of flange center in base coordinate system (Euler angle): ";
    std::cout << "(" << flangeCenterRpyOnBase.rx*180/M_PI << ", " <<
flangeCenterRpyOnBase.ry*180/M_PI << ", " <<
flangeCenterRpyOnBase.rz*180/M_PI << ")";
    std::cout << std::endl;
}
```

4.5 Set and obtain relevant parameters of the manipulator

This example is to set and get the parameters related to the arm.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) Obtaining the joint status of the robot arm (4) Obtaining whether the real robot exists (5) Obtaining the diagnostic information of the robot (6) Obtaining the connection status (7) Set the current working mode of the robot (8) Obtain the current working mode of the robot (9) Obtain the gravity component (10) Set the collision level of the robotic arm (11) Obtain the collision level of the robotic arm (12) Obtain device information (13) Obtain the current operating status of the robotic arm (14) Obtain the MAC communication status (15) Obtain the 6-joint rotation 360 enable flag (16) Obtain the current joint angle information of the robotic arm (17) Obtain the current waypoint information (18)) Whether the robot arm is in server mode (19) Whether the robot arm is in online master mode (20) Get the safety configuration of the robot arm.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The example_4.cpp code is as follows:

Note: The functions of the Util class are used in the code, please refer to chapter 4.15.

```
#include "example_4.h"
#include "AuboRobotMetaType.h"
#include "serviceinterface.h"
#include "util.h"

#include <unistd.h>
#include <math.h>
#include <string.h>
#include <stdio.h>
#include <sstream>
#include <fstream>

#define SERVER_HOST "192.168.221.13"
#define SERVER_PORT 8899
```

```
Example_4::Example_4()
{

}

void Example_4::demo()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6            /*collision level*/,
                        true        /* Whether to allow reading pose, true by default */,
                                    true,       /* Leave the default as true */
                                    1000,       /* Leave the default as 1000 */
                                        result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
```

```
    }

    //1. Get the joint status of the arm
    aubo_robot_namespace::JointStatus jointStatus[6];
    ret = robotService.robotServiceGetRobotJointStatus(jointStatus, 6);
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << " Getting joint status successful." << std::endl;

        Util::printJointStatus(jointStatus, 6);

    }
    else
    {
        std::cerr << " Getting joint status failed." << std::endl;
    }

    //2. Get whether the real arm exists
    bool IsRealRobotExist = false;
    ret = robotService.robotServiceGetIsRealRobotExist(IsRealRobotExist);

    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "Real robot exists：" << IsRealRobotExist << std::endl;
    }
    else
    {
        std::cerr << " ERROR: Failed to get the real robot existence." << std::endl;
    }

    //3.Robot Diagnostic Information
    aubo_robot_namespace::RobotDiagnosis robotDiagnosisInfo;
    ret = robotService.robotServiceGetRobotDiagnosisInfo(robotDiagnosisInfo);
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        Util::printRobotDiagnosis(robotDiagnosisInfo);
    }
    else
    {
        std::cerr << "ERROR: Failed to get robot diagnostic information." <<
std::endl;
    }

    //4. Get connection status
```

```
    bool connectStatus;
    robotService.robotServiceGetConnectStatus(connectStatus);
    std::cout << " Connection status of the robot arm：  " << connectStatus <<
std::endl;

    //5. Setting the current working mode of the robot
    aubo_robot_namespace::RobotWorkMode workmode =
aubo_robot_namespace::RobotModeReal;
    ret = robotService.robotServiceSetRobotWorkMode(workmode);
    std::cout << " Set the current working mode of the robot: " << workmode <<
std::endl;

    //6. Getting the current working mode of the robotic arm
    aubo_robot_namespace::RobotWorkMode workmode2;
    robotService.robotServiceGetRobotWorkMode(workmode2);
    std::cout << "Set the current working mode of the robot: " << workmode2 <<
std::endl;

    //7. Getting the gravity component
    aubo_robot_namespace::RobotGravityComponent gravity;
    robotService.robotServiceGetRobotGravityComponent(gravity);
    std::cout << "**** gravity component ****" << std::endl;
    std::cout << "x = " << gravity.x << std::endl;
    std::cout << "y = " << gravity.y << std::endl;
    std::cout << "z = " << gravity.z << std::endl;

    //8. Setting the collision level of the manipulator
    int collisionClass = 5;
    ret = robotService.robotServiceSetRobotCollisionClass(collisionClass);
    std::cout << " Set the collision level of the manipulator: " << collisionClass <<
std::endl;

    //9. Getting the collision level
    int collisiongrade;
    robotService.robotServiceGetRobotCollisionCurrentService(collisiongrade);
    std::cout << " Get collision level: " << collisiongrade << std::endl;

    //10. Getting the device information
    aubo_robot_namespace::RobotDevInfo robotdevinfo;
    robotService.robotServiceGetRobotDevInfoService(robotdevinfo);
    std::cout << "**** Manipulator device information ****" << std::endl;
    std::cout << " revision : " << robotdevinfo.revision << std::endl;
    std::cout << " slave version : " << robotdevinfo.slave_version << std::endl;
    std::cout << " external IO version : " << robotdevinfo.extio_version << std::endl;
```

```
    std::cout << " ID manu_id : " << robotdevinfo.manu_id << std::endl;
    std::cout << "joint_type : " << robotdevinfo.joint_type << std::endl;
    for(int i = 0; i < 8; i++)
    {
        if(i == 6)
        {
           std::cout <<"Tool" << " hardware version is : " <<
robotdevinfo.joint_ver[i].hw_version << std::endl;
           std::cout <<"Tool" << " software version is : " <<
robotdevinfo.joint_ver[i].sw_version << std::endl;
        } else if(i == 7)
        {
            std::cout <<"Base" << " hardware version is : " <<
robotdevinfo.joint_ver[i].hw_version << std::endl;
            std::cout <<"Base" << " software version is : " <<
robotdevinfo.joint_ver[i].sw_version << std::endl;
        } else
        {
            std::cout <<"joint[" << i+1 << "] hardware version is : " <<
robotdevinfo.joint_ver[i].hw_version << std::endl;
            std::cout <<"joint[" << i+1 << "] software version is : " <<
robotdevinfo.joint_ver[i].sw_version << std::endl;
        }
    }
    for(int i = 0; i < 8; i++)
    {
        if(i == 0)
        {
            std::cout <<"Interface Board ID is : " <<
robotdevinfo.jointProductID[i].productID << std::endl;
        } else if(i == 7)
        {
            std::cout <<"Tool ID is : " << robotdevinfo.jointProductID[i].productID
<< std::endl;
        } else
        {
        std::cout <<" joint[" << i << "] ID is : " <<
robotdevinfo.jointProductID[i].productID << std::endl;
        }
    }


    //11. Obtain the current operating state of the manipulator
    aubo_robot_namespace::RobotState robotstate;
```

```
    robotService.robotServiceGetRobotCurrentState(robotstate);
    std::cout << " Current operating state of manipulator: " << robotstate << std::endl;

    //12. Get MAC communication status
    bool macconnectstate;
    robotService.robotServiceGetMacCommunicationStatus(macconnectstate);
    std::cout << "MAC communication status: " << macconnectstate << std::endl;

    //13. Get the 6-joints rotation 360 enable flag
    bool j6_360_flag;
    robotService.robotServiceGetJoint6Rotate360EnableFlag(j6_360_flag);
    std::cout << "joint 6 360 flag : " << j6_360_flag << std::endl;

    //14. Obtain the current joint angle information of the manipulator
    aubo_robot_namespace::JointParam jointangle;
    robotService.robotServiceGetJointAngleInfo(jointangle);
    //Joint info
    std::cout<<"Joint angle: "<<std::endl;
    for(int i=0;i<aubo_robot_namespace::ARM_DOF;i++)
    {
        std::cout << "joint" << i+1 << ": "<< jointangle.jointPos[i] << " ~ " <<
        jointangle.jointPos[i]*180.0/M_PI << std::endl;
    }

    //15. Get current waypoint information
    aubo_robot_namespace::wayPoint_S wayPoint;
    robotService.robotServiceGetCurrentWaypointInfo(wayPoint);
    std::cout<<std::endl<<"---------- Current waypoint information ----------
"<<std::endl;
    //Position info
    std::cout<<"Pos: ";
    std::cout<<"x:"<<wayPoint.cartPos.position.x<<" ";
    std::cout<<"y:"<<wayPoint.cartPos.position.y<<" ";
    std::cout<<"z:"<<wayPoint.cartPos.position.z<<std::endl;
    //Pose info
    std::cout<<" Ori: ";
    std::cout<<"w:"<<wayPoint.orientation.w<<" ";
    std::cout<<"x:"<<wayPoint.orientation.x<<" ";
    std::cout<<"y:"<<wayPoint.orientation.y<<" ";
    std::cout<<"z:"<<wayPoint.orientation.z<<std::endl;
    aubo_robot_namespace::Rpy tempRpy;
    robotService.quaternionToRPY(wayPoint.orientation,tempRpy);
    std::cout<<" Euler angles Rpy: ";
    std::cout<<"RX:"<<tempRpy.rx*180.0/M_PI<<" RY:"<<tempRpy.ry*180.0/M_PI
```

```
            <<" RZ:"<<tempRpy.rz*180.0/M_PI<<std::endl;
        //Joints info
        std::cout<<"jont position: "<<std::endl;
        for(int i=0;i<aubo_robot_namespace::ARM_DOF;i++)
        {
            std::cout<<"joint"<<i+1<<": "<<wayPoint.jointpos[i]<<" ~
"<<wayPoint.jointpos[i]*180.0/M_PI<<std::endl;
        }

        //16. Whether in online mode
        bool isonlinemode;
        robotService.robotServiceIsOnlineMode(isonlinemode);
        std::cout << " Whether the robot is in online mode : " << isonlinemode <<
std::endl;

        //17. Whether in online main mode
        bool isonlinemastermode;
        robotService.robotServiceIsOnlineMasterMode(isonlinemastermode);
        std::cout << " Whether the robot is in online master mode: " <<
isonlinemastermode << std::endl;

        //18. Get Robot Arm Safety Configuration
        aubo_robot_namespace::RobotSafetyConfig safeconfig;
        robotService.robotServiceGetRobotSafetyConfig(safeconfig);
        std::cout << " Joint speed Limits for Reduced Mode：" << std::endl;
        for(int i = 0; i < 6 ;i++)
        {
            std::cout << "joint[" << i+1 << "] speed = " <<
safeconfig.robotReducedConfigJointSpeed[i] << std::endl;
        }
        std::cout << " Reduced mode TCP speed limit = " <<
safeconfig.robotReducedConfigTcpSpeed << std::endl;
        std::cout << " Reduced mode TCP force= " <<
safeconfig.robotReducedConfigTcpForce << std::endl;
        std::cout << " Momentum in reduced mode = " <<
safeconfig.robotReducedConfigMomentum << std::endl;
        std::cout << " Power in reduced mode = " <<
safeconfig.robotReducedConfigPower << std::endl;
}
```

4.6 IO

4.6.1 Tool IO

This example is about tool IO.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) Setting the tool-side power voltage type (4) Obtaining the tool-side power supply voltage type (5) Obtaining the tool-side power supply voltage status (6) Obtaining the status of all digital IOs on the tool side (7) Set the type of digital IO on the tool side: input or output (8) Set the status of the digital IO on the tool side according to the name (9) Set the status of the digital IO on the tool side according to the address (10) Obtain the status of all AIs on the tool side (11) Set the power supply voltage type of the tool side and the type of all digital IOs.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The toolio() function code in example_5.cpp is as follows:

```
#include "example_5.h"
#include "AuboRobotMetaType.h"
#include "serviceinterface.h"

#include <unistd.h>
#include <math.h>
#include <string.h>
#include <stdio.h>
#include <sstream>
#include <fstream>

#define SERVER_HOST "192.168.221.13"
#define SERVER_PORT 8899

Example_5::Example_5()
{

}

void Example_5::toolio()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

        /** Call interface: login ***/
```

```
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6              /*collision level*/,
                            true        /* Whether to allow reading pose, true by default */,
                                        true,       /* Leave the default as true */
                                        1000,       /* Leave the default as 1000 */
                                        result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    //1. Set tool end power supply voltage type
    aubo_robot_namespace::ToolPowerType type =
aubo_robot_namespace::OUT_24V;
    ret = robotService.robotServiceSetToolPowerVoltageType(type);

    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout<<"INFO: The tool end voltage type is set successfully. The currently
set type:"<<type<<std::endl;
    }
```

```
    else
    {
        std::cerr<<" ERROR: Failed to set tool end voltage type."<<std::endl;
    }

    //2. Get tool end power supply voltage type
    aubo_robot_namespace::ToolPowerType type2;
    ret = robotService.robotServiceGetToolPowerVoltageType(type2);

    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout<<"INFO: Getting the supply voltage type succeeded. The result
is:"<<type2<<std::endl;
    }
    else
    {
        std::cerr<<" ERROR: Failed to set tool end voltage type."<<std::endl;
    }


    //3. Getting the tool-end power supply voltage status
    double ToolPowerVoltageStatus;

    ret =
robotService.robotServiceGetToolPowerVoltageStatus(ToolPowerVoltageStatus);

    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout<<" INFO: getting the power supply voltage of the tool end
successfully Power supply voltage:"<<ToolPowerVoltageStatus<<std::endl;
    }
    else
    {
        std::cout<<" ERROR: Failed to get the power supply voltage of the
tool."<<std::endl;
    }

    //4. Get the status of all digital IOs on the tool end
    std::vector<aubo_robot_namespace::RobotIoDesc> statusVector;
    ret = robotService.robotServiceGetAllToolDigitalIOStatus(statusVector);
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout<<" INFO: the status of the digital IO of the tool end:"<<std::endl;
```

```
            for(int i=0;i<(int)statusVector.size();i++)
            {
                std::cout <<"Name:"<<statusVector[i].ioName <<"
type:"<<statusVector[i].ioType    <<"    address:"<<statusVector[i].ioAddr    <<"
status:"<<statusVector[i].ioValue <<std::endl;
            }

            std::cout<<" INFO: getting the status of the digital IO of the tool end
succeeded."<<std::endl;
        }
        else
        {
            std::cerr<<" ERROR: Failed to get the status of digital IO on the tool
end."<<std::endl;
        }

        //5. Set the type of tool-end digital IO: input or output
        aubo_robot_namespace::ToolDigitalIOAddr addr;
        aubo_robot_namespace::ToolIOType value;

        addr = aubo_robot_namespace::TOOL_DIGITAL_IO_1;
        value = aubo_robot_namespace::IO_OUT;

        ret = robotService.robotServiceSetToolDigitalIOType(addr, value);
        if( ret == aubo_robot_namespace::InterfaceCallSuccCode)
        {
            std::cout<<"INFO: Successfully set the type of digital IO on the tool end.
Current setting addr:"<<addr<<"    type:"<<value<<std::endl;
        }
        else
        {
            std::cerr<<"ERROR: Failed to set the type of digital IO on the tool side.
addr:"<<addr<<std::endl;
        }

        //6. Set the status of the tool-end digital IO according to the name
        std::string name = "T_DI/O_01";
        aubo_robot_namespace::IO_STATUS value2 =
aubo_robot_namespace::IO_STATUS_VALID;
        robotService.robotServiceSetToolDOStatus(name, value2);
        if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
        {
            std::cout<<"INFO: Set the status of the digital IO of the tool side successfully.
The current setting name:"<<name<<"    status:"<<value2<<std::endl;
```

```
    }
    else
    {
        std::cerr<<"ERROR: Failed to set the status of the digital IO of the tool side.
name:"<<name<<std::endl;
    }

    sleep(1);

    //7. Set the status of the tool-end digital IO according to the address
    aubo_robot_namespace::ToolDigitalIOAddr addr2;
    aubo_robot_namespace::IO_STATUS value3;

    addr2 = aubo_robot_namespace::TOOL_DIGITAL_IO_2;
    value3 = aubo_robot_namespace::IO_STATUS_INVALID;

    robotService.robotServiceSetToolDOStatus(addr2, value3);
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout<<"INFO: Set the status of the digital IO of the tool end successfully.
The current setting addr:"<<addr2<<"    status:"<<value3<<std::endl;
    }
    else
    {
        std::cerr<<"ERROR: Failed to set the state of the digital IO of the tool.
addr:"<<addr2<<std::endl;
    }

    sleep(1);

    //8. Get the status of all AIs on the tool end
    std::vector<aubo_robot_namespace::RobotIoDesc> toolAIStatusVector;
    ret = robotService.robotServiceGetAllToolAIStatus(toolAIStatusVector);
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout<<" INFO: The status of all AI on the tool end:"<<std::endl;
        for(int i=0;i<(int)toolAIStatusVector.size();i++)
        {
            std::cout <<"Name:"<<toolAIStatusVector[i].ioName <<"
name:"<<toolAIStatusVector[i].ioType    <<"
address:"<<toolAIStatusVector[i].ioAddr    <<"
status:"<<toolAIStatusVector[i].ioValue <<std::endl;
        }
```

```
            std::cout<<"INFO: Get the status of all AI on the tool end
successfully."<<std::endl;
        }
        else
        {
            std::cerr<<" ERROR: Failed to get the status of all AIs on the tool
end."<<std::endl;
        }

        //9. Set the power supply voltage type of the tool end and the type of all digital IOs
        ret =
robotService.robotServiceSetToolPowerTypeAndDigitalIOType(aubo_robot_namespac
e::OUT_12V,

aubo_robot_namespace::IO_OUT, aubo_robot_namespace::IO_OUT,

aubo_robot_namespace::IO_OUT, aubo_robot_namespace::IO_IN);
        if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
        {
            std::cerr<<" Set the tool end IO type SUCC."<<std::endl;
        }
        else
        {
        std::cerr<<" Set the tool end IO type Failed."<<std::endl;
        }

}
```

4.6.2 User IO

This example is about tool IO.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) Obtaining the IO configuration of the interface board (4) Obtaining the IO status of the interface board (5) Setting the IO status according to the IO type and name of the interface board (6)) Set the IO status according to the IO type and address of the interface board (7) Obtain the IO status according to the IO type and name of the interface board (8) Obtain the IO status according to the IO type and address of the interface board.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The userio() function code in example_5.cpp is as follows:

```
void Example_5::userio()
{
```

```
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

            /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6            /*collision level*/,
                        true        /* Whether to allow reading pose, true by default */,
                                    true,       /* Leave the default as true */
                                    1000,       /* Leave the default as 1000 */
                                        result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    //1. Getting the IO configuration of the interface board
    ret = aubo_robot_namespace::InterfaceCallSuccCode;
    std::vector<aubo_robot_namespace::RobotIoType> ioType;
    std::vector<aubo_robot_namespace::RobotIoDesc> configVector;
    ioType.push_back(aubo_robot_namespace::RobotBoardControllerDI);
    ioType.push_back(aubo_robot_namespace::RobotBoardControllerDO);
```

```
    ioType.push_back(aubo_robot_namespace::RobotBoardControllerAI);
    ioType.push_back(aubo_robot_namespace::RobotBoardControllerAO);
    ioType.push_back(aubo_robot_namespace::RobotBoardUserDI);
    ioType.push_back(aubo_robot_namespace::RobotBoardUserDO);
    ioType.push_back(aubo_robot_namespace::RobotBoardUserAI);
    ioType.push_back(aubo_robot_namespace::RobotBoardUserAO);

    ret = robotService.robotServiceGetBoardIOConfig(ioType, configVector);

    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout<<" The configuration of the interface version IO (including the
control box IO and user IO) is successful "<<std::endl;
    }
    else
    {
        std::cerr<<"ERROR:getBoardIOConfigAPI failed."<<std::endl;
    }

    std::cout << "ioTypeVector lenth = " << ioType.size() << std::endl;
    std::cout << "ioDescVector lenth = " << configVector.size() << std::endl;
    for(unsigned int i = 0; i < configVector.size(); i++)
    {
        std::cout << "U_DO_" << i << std::endl;
        std::cout << "ioID = " << configVector[i].ioId << " | ";
        std::cout << "ioType = " << configVector[i].ioType << " | ";
        std::cout << "ioName = " << configVector[i].ioName << " | ";
        std::cout << "ioAddr = " << configVector[i].ioAddr << " | ";
        std::cout << "ioValue = " << configVector[i].ioValue << std::endl;
    }

    //2. Getting the IO status of the interface board
    ret = aubo_robot_namespace::InterfaceCallSuccCode;
    std::vector<aubo_robot_namespace::RobotIoType> ioType2;
    std::vector<aubo_robot_namespace::RobotIoDesc> statusVector;
    ioType2.push_back(aubo_robot_namespace::RobotBoardControllerDI);
    ioType2.push_back(aubo_robot_namespace::RobotBoardControllerDO);
    ioType2.push_back(aubo_robot_namespace::RobotBoardControllerAI);
    ioType2.push_back(aubo_robot_namespace::RobotBoardControllerAO);
    ioType2.push_back(aubo_robot_namespace::RobotBoardUserDI);
    ioType2.push_back(aubo_robot_namespace::RobotBoardUserDO);
    ioType2.push_back(aubo_robot_namespace::RobotBoardUserAI);
    ioType2.push_back(aubo_robot_namespace::RobotBoardUserAO);
```

```
    std::cout << std::endl;

    ret = robotService.robotServiceGetBoardIOStatus(ioType2, statusVector);
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout<<" Get interface version IO (including control box IO and user IO)
status successfully "<<std::endl;
    }
    else
    {
        std::cerr<<"ERROR:getBoardIOStatusAPI failed."<<std::endl;
    }

    std::cout << "ioTypeVector length = " << ioType2.size() << std::endl;
    std::cout << "ioDescVector length = " << statusVector.size() << std::endl;
    for(int i = 0; i < statusVector.size(); i++)
    {
        std::cout << "U_DO_" << i << std::endl;
        std::cout << "ioID = " << statusVector[i].ioId << " | ";
        std::cout << "ioType = " << statusVector[i].ioType << " | ";
        std::cout << "ioName = " << statusVector[i].ioName << " | ";
        std::cout << "ioAddr = " << statusVector[i].ioAddr << " | ";
        std::cout << "ioValue = " << statusVector[i].ioValue << std::endl;
    }

    //3. Setting IO status according to interface board IO type and name
    ret =
robotService.robotServiceSetBoardIOStatus(aubo_robot_namespace::RobotBoardUser
DO, "U_DO_17", 1);
    sleep(1);
    if(ret != aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cerr << " Failed to set IO status based on interface board IO type and
name!" << std::endl;
    }
    else
    {
        std::cout << " Set the IO status according to the IO type and name of the
interface version successfully!" << std::endl;
    }

    //5. Set IO status according to interface board IO type and address
    aubo_robot_namespace::RobotIoType iotype3 =
aubo_robot_namespace::RobotBoardUserDO;
```

```
    int ioaddr2 = 40;
    double iovalue2 = 1;
    ret = robotService.robotServiceSetBoardIOStatus(iotype3,ioaddr2,iovalue2);
    if(ret != aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cerr << " Failed to set IO status according to interface board IO type and
address!" << std::endl;
    }
    else
    {
        std::cout << " Set the IO status according to the IO type and address of the
interface version successfully!" << std::endl;
    }
    sleep(1);
    std::cout << std::endl;


    //4. Getting the IO status based on the IO type and name of the interface board
    double value;

robotService.robotServiceGetBoardIOStatus(aubo_robot_namespace::RobotBoardUser
DO, "U_DO_02", value);
    std::cerr<<"DO_02 status:"<<value<<std::endl;

    //6. Getting the IO status according to the IO type and address of the interface
board
    // Get the status of U_DI_00
    aubo_robot_namespace::RobotIoType iotype4 =
aubo_robot_namespace::RobotBoardUserDI;
    int ioaddr3 = 36;
    double iovalue3;
    robotService.robotServiceGetBoardIOStatus(iotype4,ioaddr3,iovalue3);
    std::cout << "addr " << ioaddr3 << " = " << iovalue3 << std::endl;
    //Getting the status of U_DO_00
    iotype4 = aubo_robot_namespace::RobotBoardUserDO;
    ioaddr3 = 40;
    robotService.robotServiceGetBoardIOStatus(iotype4,ioaddr3,iovalue3);
    std::cout << "addr " << ioaddr3 << " = " << iovalue3 << std::endl;
}
```

4.6.3 Safety IO

This example is about entering and exiting reduced mode for safety IO.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) set the safety configuration of the robot (4) enter the reduction mode (5) move the joint to the target waypoint (6) exit the reduction mode.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The reduceMode() function code in example_5.cpp is as follows:

```
void Example_5::reduceMode()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6              /*collision level*/,
                        true        /* Whether to allow reading pose, true by default */,
                                    true,       /* Leave the default as true */
                                    1000,       /* Leave the default as 1000 */
                                    result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
```

```
        std::cerr << " Robot initialization failed." << std::endl;
    }

    aubo_robot_namespace::RobotSafetyConfig speed_config;
    speed_config.robotReducedConfigJointSpeed[0] = 15 / 180.0*M_PI;
    speed_config.robotReducedConfigJointSpeed[1] = 15 / 180.0*M_PI;
    speed_config.robotReducedConfigJointSpeed[2] = 15 / 180.0*M_PI;
    speed_config.robotReducedConfigJointSpeed[3] = 15 / 180.0*M_PI;
    speed_config.robotReducedConfigJointSpeed[4] = 15 / 180.0*M_PI;
    speed_config.robotReducedConfigJointSpeed[5] = 15 / 180.0*M_PI;
    speed_config.robotReducedConfigTcpSpeed = 10;
    robotService.robotServiceSetRobotSafetyConfig(speed_config);

    // Enter reduced mode
    robotService.robotServiceEnterRobotReduceMode();

    // Initialize motion properties
    robotService.robotServiceInitGlobalMoveProfile();
    // Set the maximum acceleration of joint motion
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30*M_PI/180;
    jointMaxAcc.jointPara[1] = 30*M_PI/180;
    jointMaxAcc.jointPara[2] = 30*M_PI/180;
    jointMaxAcc.jointPara[3] = 30*M_PI/180;
    jointMaxAcc.jointPara[4] = 30*M_PI/180;
    jointMaxAcc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);
    // Set the joint movement maximum speed
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30*M_PI/180;
    jointMaxVelc.jointPara[1] = 30*M_PI/180;
    jointMaxVelc.jointPara[2] = 30*M_PI/180;
    jointMaxVelc.jointPara[3] = 30*M_PI/180;
    jointMaxVelc.jointPara[4] = 30*M_PI/180;
    jointMaxVelc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);
    //JointAngle
    double jointAngle[6] = {};
    jointAngle[0] = 60.443151*M_PI/180;
    jointAngle[1] = 42.275463*M_PI/180;
    jointAngle[2] = -97.679737*M_PI/180;
    jointAngle[3] = -49.990510*M_PI/180;
    jointAngle[4] = -90.007372*M_PI/180;
    jointAngle[5] = 62.567046*M_PI/180;
```

```
    //JointMove
    robotService.robotServiceJointMove(jointAngle, true);

    // Exit Reduced Mode
    robotService.robotServiceExitRobotReduceMode();
}
```

4.7 TCP to CAN transparent transmission mode

In this example, the joint state is obtained in CAN transparent transmission mode.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) Enter TCP to CAN transparent transmission mode (4) Obtain the state of the robot arm joints (5) Exit TCP to CAN transparent transmission mode.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The demo1() function code in example_6.cpp is as follows:

Note: The functions of the Util class are used in the code, please refer to chapter 4.15.

```
#include "example_6.h"
#include "AuboRobotMetaType.h"
#include "serviceinterface.h"
#include "util.h"

#include <unistd.h>
#include <math.h>
#include <string.h>
#include <stdio.h>
#include <sstream>
#include <fstream>

#define SERVER_HOST "192.168.221.13"
#define SERVER_PORT 8899

Example_6::Example_6()
{

}

void Example_6::demo1()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;
```

```
    /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6              /*collision level*/,
                        true        /* Whether to allow reading pose, true by default */,
                                    true,       /* Leave the default as true */
                                    1000,       /* Leave the default as 1000 */
                                        result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    // Enter TCP to CAN transparent transmission mode
    ret = robotService.robotServiceEnterTcp2CanbusMode();
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << " Enter the TCP to CAN transparent transmission mode
successfully. " << std::endl;
    }
    else
```

```
    {
        std::cerr << " Failed to enter TCP to CAN transparent transmission mode.
Error code is " << ret << std::endl;
    }

    aubo_robot_namespace::JointStatus jointStatus[6];
    // Get the joint status of the arm
    ret = robotService.robotServiceGetRobotJointStatus(jointStatus, 6);
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << " Getting joint status succeeded." << std::endl;
        // Print joint status information
        Util::printJointStatus(jointStatus, 6);
    }
    else
    {
        std::cerr << " Failed to get joint state. Error code is " << ret << std::endl;
    }

    //Exit TCP to CAN transparent transmission mode
    robotService.robotServiceLeaveTcp2CanbusMode();

}
```

4.8 Joint Move

4.8.1 robotServiceJointMove Function

In this example, the joint is moved to the specified joint angle.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) Initialize global move properties (4) Set the maximum speed and acceleration of joint move (5) Joint move to the specified joint angle.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The code of the movej() function in example_movej.cpp is as follows:

```
void Example_MoveJ::movej()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

/** Call interface: login ***/
```

```
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6              /*collision level*/,
                            true        /* Whether to allow reading pose, true by default */,
                                        true,       /* Leave the default as true */
                                        1000,       /* Leave the default as 1000 */
                                        result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    //Initialize move properties
    robotService.robotServiceInitGlobalMoveProfile();

    // Set the maximum acceleration of joint move
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30*M_PI/180;
    jointMaxAcc.jointPara[1] = 30*M_PI/180;
    jointMaxAcc.jointPara[2] = 30*M_PI/180;
    jointMaxAcc.jointPara[3] = 30*M_PI/180;
    jointMaxAcc.jointPara[4] = 30*M_PI/180;
```

```
    jointMaxAcc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);

    // Set the joint movement maximum speed
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30*M_PI/180;
    jointMaxVelc.jointPara[1] = 30*M_PI/180;
    jointMaxVelc.jointPara[2] = 30*M_PI/180;
    jointMaxVelc.jointPara[3] = 30*M_PI/180;
    jointMaxVelc.jointPara[4] = 30*M_PI/180;
    jointMaxVelc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

    //jointAngle
    double jointAngle[6] = {};
    jointAngle[0] = 173.108713*M_PI/180;
    jointAngle[1] = -12.075005*M_PI/180;
    jointAngle[2] = -83.663342*M_PI/180;
    jointAngle[3] = -15.641249*M_PI/180;
    jointAngle[4] = -89.140000*M_PI/180;
    jointAngle[5] = -28.328713*M_PI/180;

    //Joint Move
    robotService.robotServiceJointMove(jointAngle, true);

}
```

4.8.2 robotMoveJointToTargetPositionByRelative Function

Example 1: Flange center offset in base coordinate system

This example is a joint move to a target position, where the target position is given by the center of the flange in the base coordinate system by offsetting 0.001m along the X axis relative to the current position.

The main flow of this program is: (1) Robot (2) Robot initialization (3) Joint movement to the starting waypoint (4) Setting the base coordinate system (5) Setting the offset (6) Calling the function robotMoveJointToTargetPositionByRelative to move to the target location.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The code of the movejToTargetPositionByRelative1 () function in example_movej.cpp is as follows:

```
void Example_MoveJ::movejToTargetPositionByRelative1()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6              /*collision level*/,
                        true        /* Whether to allow reading pose, true by default */,
                                            true,       /* Leave the default as true */
                                            1000,       /* Leave the default as 1000 */
                                                result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }


    // Initialize move properties
    robotService.robotServiceInitGlobalMoveProfile();
```

```
// Set the maximum acceleration of joint motion
aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
jointMaxAcc.jointPara[0] = 30*M_PI/180;
jointMaxAcc.jointPara[1] = 30*M_PI/180;
jointMaxAcc.jointPara[2] = 30*M_PI/180;
jointMaxAcc.jointPara[3] = 30*M_PI/180;
jointMaxAcc.jointPara[4] = 30*M_PI/180;
jointMaxAcc.jointPara[5] = 30*M_PI/180;
robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);

// Set the joint movement maximum speed
aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
jointMaxVelc.jointPara[0] = 30*M_PI/180;
jointMaxVelc.jointPara[1] = 30*M_PI/180;
jointMaxVelc.jointPara[2] = 30*M_PI/180;
jointMaxVelc.jointPara[3] = 30*M_PI/180;
jointMaxVelc.jointPara[4] = 30*M_PI/180;
jointMaxVelc.jointPara[5] = 30*M_PI/180;
robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

// Starting waypoint
double initAngle[6] = {};
initAngle[0] = 173.108713*M_PI/180;
initAngle[1] = -12.075005*M_PI/180;
initAngle[2] = -83.663342*M_PI/180;
initAngle[3] = -15.641249*M_PI/180;
initAngle[4] = -89.140000*M_PI/180;
initAngle[5] = -28.328713*M_PI/180;

// Joint movement to starting waypoint
robotService.robotServiceJointMove(initAngle, true);

// Set the base coordinate system
aubo_robot_namespace::CoordCalibrateByJointAngleAndTool baseCoord;
baseCoord.coordType = aubo_robot_namespace::BaseCoordinate;

// Set offset
aubo_robot_namespace::MoveRelative moveRelative;
moveRelative.relativePosition[0] = 0.001;
moveRelative.relativePosition[1] = 0;
moveRelative.relativePosition[2] = 0;

// The joint moves to the target position, where the target position is given by the
center of the flange in the base coordinate system by offsetting 0.001m along the X axis
```

```
relative to the current position.
    robotService.robotMoveJointToTargetPositionByRelative(baseCoord,
moveRelative, true);

}
```

Example 2: Flange center offset in user coordinate system

This example is to articulate to a target position, where the target position is given by the center of the flange in the user coordinate system by offsetting 0.001m along the X axis relative to the current position.

The main flow of this program is: (1) Robot login (2) Robot Initialization (3) joint movement to the starting waypoint (4) setting the user coordinate system and its tool parameters (5) setting the offset (6) Call the function robotMoveJointToTargetPositionByRelative to move to the target position.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The code of the movejToTargetPositionByRelative2 () function in example_movej.cpp is as follows:

```
void Example_MoveJ::movejToTargetPositionByRelative2()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
```

```
parameters **/,
                                                    6               /*collision level*/,
                            true        /* Whether to allow reading pose, true by default */,
                                                true,       /* Leave the default as true */
                                               1000,       /* Leave the default as 1000 */
                                                    result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    // Initialize move properties
    robotService.robotServiceInitGlobalMoveProfile();

    // Set the maximum acceleration of joint motion
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30*M_PI/180;
    jointMaxAcc.jointPara[1] = 30*M_PI/180;
    jointMaxAcc.jointPara[2] = 30*M_PI/180;
    jointMaxAcc.jointPara[3] = 30*M_PI/180;
    jointMaxAcc.jointPara[4] = 30*M_PI/180;
    jointMaxAcc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);

    // Set the joint movement maximum speed
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30*M_PI/180;
    jointMaxVelc.jointPara[1] = 30*M_PI/180;
    jointMaxVelc.jointPara[2] = 30*M_PI/180;
    jointMaxVelc.jointPara[3] = 30*M_PI/180;
    jointMaxVelc.jointPara[4] = 30*M_PI/180;
    jointMaxVelc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

    // Starting waypoint
    double initAngle[6] = {};
    initAngle[0] = 173.108713*M_PI/180;
    initAngle[1] = -12.075005*M_PI/180;
    initAngle[2] = -83.663342*M_PI/180;
```

```
    initAngle[3] = -15.641249*M_PI/180;
    initAngle[4] = -89.140000*M_PI/180;
    initAngle[5] = -28.328713*M_PI/180;

    // Joint movement to starting waypoint
    robotService.robotServiceJointMove(initAngle, true);

    // Set the user coordinate system
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool userCoord;
    userCoord.coordType = aubo_robot_namespace::WorldCoordinate;
    userCoord.methods =
aubo_robot_namespace::Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrant
OfXOYPlane;

    userCoord.wayPointArray[0].jointPos[0] = -75.093279*M_PI/180;
    userCoord.wayPointArray[0].jointPos[1] = 28.544643*M_PI/180;
    userCoord.wayPointArray[0].jointPos[2] = -114.313905*M_PI/180;
    userCoord.wayPointArray[0].jointPos[3] = -62.769247*M_PI/180;
    userCoord.wayPointArray[0].jointPos[4] = -87.343517*M_PI/180;
    userCoord.wayPointArray[0].jointPos[5] = -27.888262*M_PI/180;

    userCoord.wayPointArray[1].jointPos[0] = -89.239837*M_PI/180;
    userCoord.wayPointArray[1].jointPos[1] = 23.936171*M_PI/180;
    userCoord.wayPointArray[1].jointPos[2] = -122.299277*M_PI/180;
    userCoord.wayPointArray[1].jointPos[3] = -65.208902*M_PI/180;
    userCoord.wayPointArray[1].jointPos[4] = -85.011123*M_PI/180;
    userCoord.wayPointArray[1].jointPos[5] = -41.87417*M_PI/180;

    userCoord.wayPointArray[2].jointPos[0] = -77.059212*M_PI/180;
    userCoord.wayPointArray[2].jointPos[1] = 35.509518*M_PI/180;
    userCoord.wayPointArray[2].jointPos[2] = -101.108547*M_PI/180;
    userCoord.wayPointArray[2].jointPos[3] = -56.433133*M_PI/180;
    userCoord.wayPointArray[2].jointPos[4] = -87.006734*M_PI/180;
    userCoord.wayPointArray[2].jointPos[5] = -29.827440*M_PI/180;

    aubo_robot_namespace::ToolInEndDesc toolUserCoord;
    toolUserCoord.toolInEndPosition.x =    -0.177341;
    toolUserCoord.toolInEndPosition.y = 0.002327;
    toolUserCoord.toolInEndPosition.z = 0.146822;
    toolUserCoord.toolInEndOrientation.w = 1;
    toolUserCoord.toolInEndOrientation.x = 0;
    toolUserCoord.toolInEndOrientation.y = 0;
    toolUserCoord.toolInEndOrientation.z = 0;
    userCoord.toolDesc = toolUserCoord;
```

```
    // Set offset
    aubo_robot_namespace::MoveRelative moveRelative;
    moveRelative.relativePosition[0] = 0.001;
    moveRelative.relativePosition[1] = 0;
    moveRelative.relativePosition[2] = 0;

    //The joint moves to the target position, where the target position is given by the center of the flange in the user coordinate system by offsetting 0.001m along the X axis relative to the current position
    robotService.robotMoveJointToTargetPositionByRelative(userCoord, moveRelative, true);

}
```

Example 3: Tool end offset in tool coordinate system

This example is articulating to a target position, where the target position is given by the tool end in the tool coordinate system by offsetting 0.001m along the X axis from the current position.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) Joint movement to the starting waypoint (4) Setting the kinematic parameters of the tool (5) Setting the end coordinate system and its tool parameters (6) Set the offset (7) Call the function robotMoveJointToTargetPositionByRelative to move to the target position.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The code of the movejToTargetPositionByRelative3 () function in example_movej.cpp is as follows:

```
void Example_MoveJ::movejToTargetPositionByRelative3()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo", "123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
```

```
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics parameters **/,
                                               6            /*collision level*/,
                        true        /* Whether to allow reading pose, true by default */,
                                            true,       /* Leave the default as true */
                                           1000,        /* Leave the default as 1000 */
                                                result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    //Initialize move properties
    robotService.robotServiceInitGlobalMoveProfile();

    // Set the maximum acceleration of joint motion
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30*M_PI/180;
    jointMaxAcc.jointPara[1] = 30*M_PI/180;
    jointMaxAcc.jointPara[2] = 30*M_PI/180;
    jointMaxAcc.jointPara[3] = 30*M_PI/180;
    jointMaxAcc.jointPara[4] = 30*M_PI/180;
    jointMaxAcc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);

    //Set the joint movement maximum speed
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30*M_PI/180;
    jointMaxVelc.jointPara[1] = 30*M_PI/180;
```

```
    jointMaxVelc.jointPara[2] = 30*M_PI/180;
    jointMaxVelc.jointPara[3] = 30*M_PI/180;
    jointMaxVelc.jointPara[4] = 30*M_PI/180;
    jointMaxVelc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

    // Starting waypoint
    double initAngle[6] = {};
    initAngle[0] = 173.108713*M_PI/180;
    initAngle[1] = -12.075005*M_PI/180;
    initAngle[2] = -83.663342*M_PI/180;
    initAngle[3] = -15.641249*M_PI/180;
    initAngle[4] = -89.140000*M_PI/180;
    initAngle[5] = -28.328713*M_PI/180;

    // Joint movement to starting waypoint
    robotService.robotServiceJointMove(initAngle, true);

    // Setup tool
    aubo_robot_namespace::ToolInEndDesc toolEnd;
    toolEnd.toolInEndPosition.x =    -0.177341;
    toolEnd.toolInEndPosition.y = 0.002327;
    toolEnd.toolInEndPosition.z = 0.146822;
    toolEnd.toolInEndOrientation.w = 1;
    toolEnd.toolInEndOrientation.x = 0;
    toolEnd.toolInEndOrientation.y = 0;
    toolEnd.toolInEndOrientation.z = 0;
    robotService.robotServiceSetToolKinematicsParam(toolEnd);

    // Set tool coordinate system
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool endCoord;
    endCoord.coordType = aubo_robot_namespace::EndCoordinate;
    endCoord.toolDesc = toolEnd;

    // Set offset
    aubo_robot_namespace::MoveRelative moveRelative;
    moveRelative.relativePosition[0] = 0.001;
    moveRelative.relativePosition[1] = 0;
    moveRelative.relativePosition[2] = 0;

    // The joint moves to the target position, where the target position is given by the
tool end in the tool coordinate system by offsetting 0.001m along the X axis relative to
the current position
    robotService.robotMoveJointToTargetPositionByRelative(endCoord,
```

```
moveRelative, true);

}
```

Example 4: Tool end offset in base coordinate system

This example is articulating to a target position, where the target position is given by the tool tip in the base coordinate system by offsetting 0.001m along the X-axis relative to the current position.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) Joint movement to the starting waypoint (4) Setting tool kinematic parameters (5) Setting base coordinate system (6) Setting offset The shift amount (7) calls the function robotMoveJointToTargetPositionByRelative to move to the target position.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The code of the movejToTargetPositionByRelative4 () function in example_movej.cpp is as follows:

```
void Example_MoveJ::movejToTargetPositionByRelative4()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

        /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6            /*collision level*/,
```

```
                                true        /* Whether to allow reading pose, true by default */,
                                                    true,       /* Leave the default as true */
                                                   1000,        /* Leave the default as 1000 */
                                                        result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    //Initialize move properties
    robotService.robotServiceInitGlobalMoveProfile();

    // Set the maximum acceleration of joint motion
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30*M_PI/180;
    jointMaxAcc.jointPara[1] = 30*M_PI/180;
    jointMaxAcc.jointPara[2] = 30*M_PI/180;
    jointMaxAcc.jointPara[3] = 30*M_PI/180;
    jointMaxAcc.jointPara[4] = 30*M_PI/180;
    jointMaxAcc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);

    //Set the joint movement maximum speed
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30*M_PI/180;
    jointMaxVelc.jointPara[1] = 30*M_PI/180;
    jointMaxVelc.jointPara[2] = 30*M_PI/180;
    jointMaxVelc.jointPara[3] = 30*M_PI/180;
    jointMaxVelc.jointPara[4] = 30*M_PI/180;
    jointMaxVelc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

    //Starting waypoint
    double initAngle[6] = {};
    initAngle[0] = 173.108713*M_PI/180;
    initAngle[1] = -12.075005*M_PI/180;
    initAngle[2] = -83.663342*M_PI/180;
    initAngle[3] = -15.641249*M_PI/180;
    initAngle[4] = -89.140000*M_PI/180;
```

```
    initAngle[5] = -28.328713*M_PI/180;

    // Joint movement to starting waypoint
    robotService.robotServiceJointMove(initAngle, true);

    //Setup tool
    aubo_robot_namespace::ToolInEndDesc toolEnd;
    toolEnd.toolInEndPosition.x =    -0.177341;
    toolEnd.toolInEndPosition.y = 0.002327;
    toolEnd.toolInEndPosition.z = 0.146822;
    toolEnd.toolInEndOrientation.w = 1;
    toolEnd.toolInEndOrientation.x = 0;
    toolEnd.toolInEndOrientation.y = 0;
    toolEnd.toolInEndOrientation.z = 0;
    robotService.robotServiceSetToolKinematicsParam(toolEnd);

    // Set the base coordinate system
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool baseCoord;
    baseCoord.coordType = aubo_robot_namespace::BaseCoordinate;

    // Set offset
    aubo_robot_namespace::MoveRelative moveRelative;
    moveRelative.relativePosition[0] = 0.001;
    moveRelative.relativePosition[1] = 0;
    moveRelative.relativePosition[2] = 0;

    // The joint moves to the target position, where the target position is given by the
tool tip in the base coordinate system by offsetting 0.001m along the X axis relative to
the current position.
    robotService.robotMoveJointToTargetPositionByRelative(baseCoord,
moveRelative, true);
}
```

Example 5: Tool end offset in user coordinate system

This example is articulating to a target position, where the target position is given by the tool tip in the user coordinate system by offsetting 0.001m along the X-axis relative to the current position.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) joint move to the starting waypoint (4) setting the kinematic parameters of the tool (5) setting the user coordinate system and its tool parameters (6) Set the offset (7) Call the function robotMoveJointToTargetPositionByRelative to move to the target position.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The code of the movejToTargetPositionByRelative5 () function in example_movej.cpp is as follows:

```
void Example_MoveJ::movejToTargetPositionByRelative5()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6             /*collision level*/,
                        true        /* Whether to allow reading pose, true by default */,
                                    true,       /* Leave the default as true */
                                   1000,        /* Leave the default as 1000 */
                                        result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    // Initialize move properties
    robotService.robotServiceInitGlobalMoveProfile();
```

```
// Set the maximum acceleration of joint move
aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
jointMaxAcc.jointPara[0] = 30*M_PI/180;
jointMaxAcc.jointPara[1] = 30*M_PI/180;
jointMaxAcc.jointPara[2] = 30*M_PI/180;
jointMaxAcc.jointPara[3] = 30*M_PI/180;
jointMaxAcc.jointPara[4] = 30*M_PI/180;
jointMaxAcc.jointPara[5] = 30*M_PI/180;
robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);

//Set the joint movement maximum speed
aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
jointMaxVelc.jointPara[0] = 30*M_PI/180;
jointMaxVelc.jointPara[1] = 30*M_PI/180;
jointMaxVelc.jointPara[2] = 30*M_PI/180;
jointMaxVelc.jointPara[3] = 30*M_PI/180;
jointMaxVelc.jointPara[4] = 30*M_PI/180;
jointMaxVelc.jointPara[5] = 30*M_PI/180;
robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

//Starting waypoint
double initAngle[6] = {};
initAngle[0] = 173.108713*M_PI/180;
initAngle[1] = -12.075005*M_PI/180;
initAngle[2] = -83.663342*M_PI/180;
initAngle[3] = -15.641249*M_PI/180;
initAngle[4] = -89.140000*M_PI/180;
initAngle[5] = -28.328713*M_PI/180;

//Joint movement to starting waypoint
robotService.robotServiceJointMove(initAngle, true);

//Setup tool
aubo_robot_namespace::ToolInEndDesc toolEnd;
toolEnd.toolInEndPosition.x =    -0.177341;
toolEnd.toolInEndPosition.y = 0.002327;
toolEnd.toolInEndPosition.z = 0.146822;
toolEnd.toolInEndOrientation.w = 1;
toolEnd.toolInEndOrientation.x = 0;
toolEnd.toolInEndOrientation.y = 0;
toolEnd.toolInEndOrientation.z = 0;
robotService.robotServiceSetToolKinematicsParam(toolEnd);
```

```
    //Set the user coordinate system
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool userCoord;
    userCoord.coordType = aubo_robot_namespace::WorldCoordinate;
    userCoord.methods =
aubo_robot_namespace::Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrant
OfXOYPlane;

    userCoord.wayPointArray[0].jointPos[0] = -75.093279*M_PI/180;
    userCoord.wayPointArray[0].jointPos[1] = 28.544643*M_PI/180;
    userCoord.wayPointArray[0].jointPos[2] = -114.313905*M_PI/180;
    userCoord.wayPointArray[0].jointPos[3] = -62.769247*M_PI/180;
    userCoord.wayPointArray[0].jointPos[4] = -87.343517*M_PI/180;
    userCoord.wayPointArray[0].jointPos[5] = -27.888262*M_PI/180;

    userCoord.wayPointArray[1].jointPos[0] = -89.239837*M_PI/180;
    userCoord.wayPointArray[1].jointPos[1] = 23.936171*M_PI/180;
    userCoord.wayPointArray[1].jointPos[2] = -122.299277*M_PI/180;
    userCoord.wayPointArray[1].jointPos[3] = -65.208902*M_PI/180;
    userCoord.wayPointArray[1].jointPos[4] = -85.011123*M_PI/180;
    userCoord.wayPointArray[1].jointPos[5] = -41.87417*M_PI/180;

    userCoord.wayPointArray[2].jointPos[0] = -77.059212*M_PI/180;
    userCoord.wayPointArray[2].jointPos[1] = 35.509518*M_PI/180;
    userCoord.wayPointArray[2].jointPos[2] = -101.108547*M_PI/180;
    userCoord.wayPointArray[2].jointPos[3] = -56.433133*M_PI/180;
    userCoord.wayPointArray[2].jointPos[4] = -87.006734*M_PI/180;
    userCoord.wayPointArray[2].jointPos[5] = -29.827440*M_PI/180;

    userCoord.toolDesc = toolEnd;

    // Set offset
    aubo_robot_namespace::MoveRelative moveRelative;
    moveRelative.relativePosition[0] = 0.001;
    moveRelative.relativePosition[1] = 0;
    moveRelative.relativePosition[2] = 0;

    // The joint moves to the target position, where the target position is given by the
end of the tool in the user coordinate system by offsetting 0.001m along the X axis from
the current position.
    robotService.robotMoveJointToTargetPositionByRelative(userCoord,
moveRelative, true);

}
```

4.8.3 robotMoveJointToTargetPosition Function

Example 1: The position of the flange center in the base coordinate system

This example is a joint move to the target position.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) Joint movement to the starting waypoint (4) Setting tool parameters and setting the base coordinate system for the flange center (5) Setting the base coordinate system (6)) set the target position (7) call the function robotMoveJointToTargetPosition to move to the target position.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The code of the movejToTargetPosition1() function in example_movej.cpp is as follows:

```
void Example_MoveJ::movejToTargetPosition1()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6            /*collision level*/,
                        true        /* Whether to allow reading pose, true by default */,
                                    true,      /* Leave the default as true */
                                    1000,      /* Leave the default as 1000 */
```

```
                                                result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
        {
std::cerr << " Robot initialized successfully." << std::endl;
        }
        else
        {
                std::cerr << " Robot initialization failed." << std::endl;
        }

        //Initialize move properties
        robotService.robotServiceInitGlobalMoveProfile();

        //Set the maximum speed of joint motion
        aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
        jointMaxAcc.jointPara[0] = 30*M_PI/180;
        jointMaxAcc.jointPara[1] = 30*M_PI/180;
        jointMaxAcc.jointPara[2] = 30*M_PI/180;
        jointMaxAcc.jointPara[3] = 30*M_PI/180;
        jointMaxAcc.jointPara[4] = 30*M_PI/180;
        jointMaxAcc.jointPara[5] = 30*M_PI/180;
        robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);

        // Set the maximum speed of joint move
        aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
        jointMaxVelc.jointPara[0] = 30*M_PI/180;
        jointMaxVelc.jointPara[1] = 30*M_PI/180;
        jointMaxVelc.jointPara[2] = 30*M_PI/180;
        jointMaxVelc.jointPara[3] = 30*M_PI/180;
        jointMaxVelc.jointPara[4] = 30*M_PI/180;
        jointMaxVelc.jointPara[5] = 30*M_PI/180;
        robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

        //Starting waypiont
        double initAngle[6] = {};
        initAngle[0] = 173.108713*M_PI/180;
        initAngle[1] = -12.075005*M_PI/180;
        initAngle[2] = -83.663342*M_PI/180;
        initAngle[3] = -15.641249*M_PI/180;
        initAngle[4] = -89.140000*M_PI/180;
        initAngle[5] = -28.328713*M_PI/180;

        // Joint movement to starting waypoint
```

```
    robotService.robotServiceJointMove(initAngle, true);

    // Set the tool parameters for the center of the flange
    aubo_robot_namespace::ToolInEndDesc toolEnd;
    toolEnd.toolInEndPosition.x = 0;
    toolEnd.toolInEndPosition.y = 0;
    toolEnd.toolInEndPosition.z = 0;
    toolEnd.toolInEndOrientation.w = 1;
    toolEnd.toolInEndOrientation.x = 0;
    toolEnd.toolInEndOrientation.y = 0;
    toolEnd.toolInEndOrientation.z = 0;

    // Set the base coordinate system
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool baseCoord;
    baseCoord.coordType = aubo_robot_namespace::BaseCoordinate;

    // Set the target position, the position of the flange center in the base coordinate
system
    aubo_robot_namespace::Pos posOnBase;
    posOnBase.x = -0.2;
    posOnBase.y = -0.6;
    posOnBase.z = 0;

    // Joint movement to target position
    robotService.robotMoveJointToTargetPosition(baseCoord, posOnBase, toolEnd,
true);


}
```

Example 2: The position of the flange center in the user coordinate system

This example is a joint move to the target position.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) Joint movement to the starting waypoint (4) Setting tool parameters for the center of the flange (5) Setting the user coordinate system and its Tool parameters (6) Set the target position (7) Call the function robotMoveJointToTargetPosition to move to the target position.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The code of the movejToTargetPosition2() function in example_movej.cpp is as follows:

```
void Example_MoveJ::movejToTargetPosition2()
{
```

```
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6              /*collision level*/,
                        true        /* Whether to allow reading pose, true by default */,
                                          true,       /* Leave the default as true */
                                          1000,       /* Leave the default as 1000 */
                                              result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    //Initialize move properties
    robotService.robotServiceInitGlobalMoveProfile();

    // Set the maximum acceleration of joint motion
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30*M_PI/180;
```

```
    jointMaxAcc.jointPara[1] = 30*M_PI/180;
    jointMaxAcc.jointPara[2] = 30*M_PI/180;
    jointMaxAcc.jointPara[3] = 30*M_PI/180;
    jointMaxAcc.jointPara[4] = 30*M_PI/180;
    jointMaxAcc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);

    // Set the maximum speed of joint motion
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30*M_PI/180;
    jointMaxVelc.jointPara[1] = 30*M_PI/180;
    jointMaxVelc.jointPara[2] = 30*M_PI/180;
    jointMaxVelc.jointPara[3] = 30*M_PI/180;
    jointMaxVelc.jointPara[4] = 30*M_PI/180;
    jointMaxVelc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

    //Sarting waypoint
    double initAngle[6] = {};
    initAngle[0] = 173.108713*M_PI/180;
    initAngle[1] = -12.075005*M_PI/180;
    initAngle[2] = -83.663342*M_PI/180;
    initAngle[3] = -15.641249*M_PI/180;
    initAngle[4] = -89.140000*M_PI/180;
    initAngle[5] = -28.328713*M_PI/180;

    // Joint movement to starting waypoint
    robotService.robotServiceJointMove(initAngle, true);

    // Set the tool parameters for the center of the flange
    aubo_robot_namespace::ToolInEndDesc toolEnd;
    toolEnd.toolInEndPosition.x = 0;
    toolEnd.toolInEndPosition.y = 0;
    toolEnd.toolInEndPosition.z = 0;
    toolEnd.toolInEndOrientation.w = 1;
    toolEnd.toolInEndOrientation.x = 0;
    toolEnd.toolInEndOrientation.y = 0;
    toolEnd.toolInEndOrientation.z = 0;

    //Set the user coordinate system
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool userCoord;
    userCoord.coordType = aubo_robot_namespace::WorldCoordinate;
    userCoord.methods =
aubo_robot_namespace::Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrant
```

```
OfXOYPlane;

    userCoord.wayPointArray[0].jointPos[0] = -75.093279*M_PI/180;
    userCoord.wayPointArray[0].jointPos[1] = 28.544643*M_PI/180;
    userCoord.wayPointArray[0].jointPos[2] = -114.313905*M_PI/180;
    userCoord.wayPointArray[0].jointPos[3] = -62.769247*M_PI/180;
    userCoord.wayPointArray[0].jointPos[4] = -87.343517*M_PI/180;
    userCoord.wayPointArray[0].jointPos[5] = -27.888262*M_PI/180;

    userCoord.wayPointArray[1].jointPos[0] = -89.239837*M_PI/180;
    userCoord.wayPointArray[1].jointPos[1] = 23.936171*M_PI/180;
    userCoord.wayPointArray[1].jointPos[2] = -122.299277*M_PI/180;
    userCoord.wayPointArray[1].jointPos[3] = -65.208902*M_PI/180;
    userCoord.wayPointArray[1].jointPos[4] = -85.011123*M_PI/180;
    userCoord.wayPointArray[1].jointPos[5] = -41.87417*M_PI/180;

    userCoord.wayPointArray[2].jointPos[0] = -77.059212*M_PI/180;
    userCoord.wayPointArray[2].jointPos[1] = 35.509518*M_PI/180;
    userCoord.wayPointArray[2].jointPos[2] = -101.108547*M_PI/180;
    userCoord.wayPointArray[2].jointPos[3] = -56.433133*M_PI/180;
    userCoord.wayPointArray[2].jointPos[4] = -87.006734*M_PI/180;
    userCoord.wayPointArray[2].jointPos[5] = -29.827440*M_PI/180;

    // Tool Parameters for Coordinate Systems
    aubo_robot_namespace::ToolInEndDesc toolDesc;
    toolDesc.toolInEndPosition.x =    -0.177341;
    toolDesc.toolInEndPosition.y = 0.002327;
    toolDesc.toolInEndPosition.z = 0.146822;
    toolDesc.toolInEndOrientation.w = 1;
    toolDesc.toolInEndOrientation.x = 0;
    toolDesc.toolInEndOrientation.y = 0;
    toolDesc.toolInEndOrientation.z = 0;
    userCoord.toolDesc = toolDesc;

    // Set the target position, the position of the flange center in the user coordinate
system
    aubo_robot_namespace::Pos posOnUser;
    posOnUser.x = -0.220305;
    posOnUser.y = -1.152177;
    posOnUser.z = 0.216184;

    //Axial movement to target position
    robotService.robotMoveJointToTargetPosition(userCoord, posOnUser, toolEnd,
true);
```

```
}
```

Example 3: The position of the tool end in the base coordinate system

This example is a joint motion to the target position.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) Joint movement to the starting waypoint (4) Setting tool parameters (5) Setting base coordinate system (6) Setting target position (7)) call the function robotMoveJointToTargetPosition to move to the target position.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The code of the movejToTargetPosition3() function in example_movej.cpp is as follows:

```
void Example_MoveJ::movejToTargetPosition3()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

/** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6              /*collision level*/,
                        true        /* Whether to allow reading pose, true by default */,
                                    true,       /* Leave the default as true */
                                    1000,       /* Leave the default as 1000 */
```

```
                                                result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    //Initialize move properties
    robotService.robotServiceInitGlobalMoveProfile();

    // Set the maximum acceleration of joint motion
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30*M_PI/180;
    jointMaxAcc.jointPara[1] = 30*M_PI/180;
    jointMaxAcc.jointPara[2] = 30*M_PI/180;
    jointMaxAcc.jointPara[3] = 30*M_PI/180;
    jointMaxAcc.jointPara[4] = 30*M_PI/180;
    jointMaxAcc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);

    // Set the maximum speed of joint motion
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30*M_PI/180;
    jointMaxVelc.jointPara[1] = 30*M_PI/180;
    jointMaxVelc.jointPara[2] = 30*M_PI/180;
    jointMaxVelc.jointPara[3] = 30*M_PI/180;
    jointMaxVelc.jointPara[4] = 30*M_PI/180;
    jointMaxVelc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

    //Starting waypoint
    double initAngle[6] = {};
    initAngle[0] = 173.108713*M_PI/180;
    initAngle[1] = -12.075005*M_PI/180;
    initAngle[2] = -83.663342*M_PI/180;
    initAngle[3] = -15.641249*M_PI/180;
    initAngle[4] = -89.140000*M_PI/180;
    initAngle[5] = -28.328713*M_PI/180;

    // Joint movement to starting waypoint
```

```
    robotService.robotServiceJointMove(initAngle, true);

    // Set tool parameters for tool end
    aubo_robot_namespace::ToolInEndDesc toolEnd;
    toolEnd.toolInEndPosition.x = -0.177341;
    toolEnd.toolInEndPosition.y = 0.002327;
    toolEnd.toolInEndPosition.z = 0.146822;
    toolEnd.toolInEndOrientation.w = 1;
    toolEnd.toolInEndOrientation.x = 0;
    toolEnd.toolInEndOrientation.y = 0;
    toolEnd.toolInEndOrientation.z = 0;

    // Set the base coordinate system
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool baseCoord;
    baseCoord.coordType = aubo_robot_namespace::BaseCoordinate;

    // Set the target position, the position of the tool end in the base coordinate system
    aubo_robot_namespace::Pos posOnBase;
    posOnBase.x = 0.6;
    posOnBase.y = 0.0;
    posOnBase.z = 0.5;

    // Axial movement to target position
    robotService.robotMoveJointToTargetPosition(baseCoord, posOnBase, toolEnd,
true);
}
```

Example 4: The position of the tool end in the user coordinate system

This example is a joint move to the target position.

The main flow of this program is: (1) Robot login (2) Robot arm initialization (3) Joint movement to the starting waypoint (4) Setting tool parameters (5) Setting the user coordinate system and its tool parameters (6) Setting The target position (7) calls the function robotMoveJointToTargetPosition to move to the target position.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The code of the movejToTargetPosition4() function in example_movej.cpp is as follows:

```
void Example_MoveJ::movejToTargetPosition4()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;
```

```
    /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6              /*collision level*/,
                            true        /* Whether to allow reading pose, true by default */,
                                                true,       /* Leave the default as true */
                                               1000,       /* Leave the default as 1000 */
                                                    result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    //Initialize move properties
    robotService.robotServiceInitGlobalMoveProfile();

    // Set the maximum acceleration of joint motion
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30*M_PI/180;
    jointMaxAcc.jointPara[1] = 30*M_PI/180;
    jointMaxAcc.jointPara[2] = 30*M_PI/180;
    jointMaxAcc.jointPara[3] = 30*M_PI/180;
```

```
    jointMaxAcc.jointPara[4] = 30*M_PI/180;
    jointMaxAcc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);

    // Set the maximum acceleration of joint motion
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30*M_PI/180;
    jointMaxVelc.jointPara[1] = 30*M_PI/180;
    jointMaxVelc.jointPara[2] = 30*M_PI/180;
    jointMaxVelc.jointPara[3] = 30*M_PI/180;
    jointMaxVelc.jointPara[4] = 30*M_PI/180;
    jointMaxVelc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

    //Starting waypoint
    double initAngle[6] = {};
    initAngle[0] = 173.108713*M_PI/180;
    initAngle[1] = -12.075005*M_PI/180;
    initAngle[2] = -83.663342*M_PI/180;
    initAngle[3] = -15.641249*M_PI/180;
    initAngle[4] = -89.140000*M_PI/180;
    initAngle[5] = -28.328713*M_PI/180;

    // Joint movement to starting waypoint
    robotService.robotServiceJointMove(initAngle, true);

    // Set tool parameters for tool end
    aubo_robot_namespace::ToolInEndDesc toolEnd;
    toolEnd.toolInEndPosition.x = -0.177341;
    toolEnd.toolInEndPosition.y = 0.002327;
    toolEnd.toolInEndPosition.z = 0.146822;
    toolEnd.toolInEndOrientation.w = 1;
    toolEnd.toolInEndOrientation.x = 0;
    toolEnd.toolInEndOrientation.y = 0;
    toolEnd.toolInEndOrientation.z = 0;

    // Set the user coordinate system
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool userCoord;
    userCoord.coordType = aubo_robot_namespace::WorldCoordinate;
    userCoord.methods =
aubo_robot_namespace::Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrant
OfXOYPlane;

    userCoord.wayPointArray[0].jointPos[0] = -75.093279*M_PI/180;
```

```
    userCoord.wayPointArray[0].jointPos[1] = 28.544643*M_PI/180;
    userCoord.wayPointArray[0].jointPos[2] = -114.313905*M_PI/180;
    userCoord.wayPointArray[0].jointPos[3] = -62.769247*M_PI/180;
    userCoord.wayPointArray[0].jointPos[4] = -87.343517*M_PI/180;
    userCoord.wayPointArray[0].jointPos[5] = -27.888262*M_PI/180;

    userCoord.wayPointArray[1].jointPos[0] = -89.239837*M_PI/180;
    userCoord.wayPointArray[1].jointPos[1] = 23.936171*M_PI/180;
    userCoord.wayPointArray[1].jointPos[2] = -122.299277*M_PI/180;
    userCoord.wayPointArray[1].jointPos[3] = -65.208902*M_PI/180;
    userCoord.wayPointArray[1].jointPos[4] = -85.011123*M_PI/180;
    userCoord.wayPointArray[1].jointPos[5] = -41.87417*M_PI/180;

    userCoord.wayPointArray[2].jointPos[0] = -77.059212*M_PI/180;
    userCoord.wayPointArray[2].jointPos[1] = 35.509518*M_PI/180;
    userCoord.wayPointArray[2].jointPos[2] = -101.108547*M_PI/180;
    userCoord.wayPointArray[2].jointPos[3] = -56.433133*M_PI/180;
    userCoord.wayPointArray[2].jointPos[4] = -87.006734*M_PI/180;
    userCoord.wayPointArray[2].jointPos[5] = -29.827440*M_PI/180;

    // Tool Parameters for Coordinate Systems
    aubo_robot_namespace::ToolInEndDesc toolDesc;
    toolDesc.toolInEndPosition.x =    -0.177341;
    toolDesc.toolInEndPosition.y = 0.002327;
    toolDesc.toolInEndPosition.z = 0.146822;
    toolDesc.toolInEndOrientation.w = 1;
    toolDesc.toolInEndOrientation.x = 0;
    toolDesc.toolInEndOrientation.y = 0;
    toolDesc.toolInEndOrientation.z = 0;
    userCoord.toolDesc = toolDesc;

    // Set the target position, the position of the tool end in the user coordinate system
    aubo_robot_namespace::Pos posOnUser;
    posOnUser.x = -0.368243;
    posOnUser.y = -1.167162;
    posOnUser.z = 0.195709;

    // Axial movement to target position
    robotService.robotMoveJointToTargetPosition(userCoord, posOnUser, toolEnd,
true);

}
```

4.9 Follow Mode

4.9.1 Axis movement in follow mode

robotServiceFollowModeJointMove

This example is axial move in follow mode.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) Joint movement to initial position (4) Joint movement based on follow mode.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The code of the followModeJointMove() function in example_followmode.cpp is as follows:

```
void Example_FollowMode::followModeJointMove()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

        /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6              /*collision level*/,
                        true        /* Whether to allow reading pose, true by default */,
                                            true,       /* Leave the default as true */
```

```
                                                1000,       /* Leave the default as 1000 */
                                                    result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
        {
std::cerr << " Robot initialized successfully." << std::endl;
        }
        else
        {
              std::cerr << " Robot initialization failed." << std::endl;
        }

        //Initialize move properties
        robotService.robotServiceInitGlobalMoveProfile();

        // Set the maximum acceleration of joint move
        aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
        jointMaxAcc.jointPara[0] = 30*M_PI/180;
        jointMaxAcc.jointPara[1] = 30*M_PI/180;
        jointMaxAcc.jointPara[2] = 30*M_PI/180;
        jointMaxAcc.jointPara[3] = 30*M_PI/180;
        jointMaxAcc.jointPara[4] = 30*M_PI/180;
        jointMaxAcc.jointPara[5] = 30*M_PI/180;
        robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);

        // Set the maximum speed of joint move
        aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
        jointMaxVelc.jointPara[0] = 30*M_PI/180;
        jointMaxVelc.jointPara[1] = 30*M_PI/180;
        jointMaxVelc.jointPara[2] = 30*M_PI/180;
        jointMaxVelc.jointPara[3] = 30*M_PI/180;
        jointMaxVelc.jointPara[4] = 30*M_PI/180;
        jointMaxVelc.jointPara[5] = 30*M_PI/180;
        robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

        //Starting position
        double jointAngle[6] = {};
        jointAngle[0] = 173.108713*M_PI/180;
        jointAngle[1] = -12.075005*M_PI/180;
        jointAngle[2] = -83.663342*M_PI/180;
        jointAngle[3] = -15.641249*M_PI/180;
        jointAngle[4] = -89.140000*M_PI/180;
        jointAngle[5] = -28.328713*M_PI/180;
```

```
    // Joint movement to initial position
    robotService.robotServiceJointMove(jointAngle, true);

    // Joint move based on follow mode
    for(int i = 0; i < 1000; i++)
    {
        jointAngle[0] = jointAngle[0] + 0.0001;
        std::cout << jointAngle[0] << std::endl;
        robotService.robotServiceFollowModeJointMove(jointAngle);
        usleep(1000*5);
    }
}
```

4.9.2 Arrival ahead of follow mode

This example is the arrival ahead to position in the follow mode.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) Joint movement to zero position (4) 5 cycles: when i=even, it is in the distance mode in arrival ahead Articulate to waypoint 1 and waypoint 2; when i=odd, it is in no arrival ahead mode, then articulate to waypoint 1 and waypoint 2.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The arrivalAhead() function code in example_movefollowmode.cpp is as follows:

```
void Example_FollowMode::arrivalAhead()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;
```

```
    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                                6             /*collision level*/,
                            true        /* Whether to allow reading pose, true by default */,
                                            true,       /* Leave the default as true */
                                            1000,       /* Leave the default as 1000 */
                                                result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    //Initialize move properties
    robotService.robotServiceInitGlobalMoveProfile();

    // Set the maximum acceleration of joint move
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30*M_PI/180;
    jointMaxAcc.jointPara[1] = 30*M_PI/180;
    jointMaxAcc.jointPara[2] = 30*M_PI/180;
    jointMaxAcc.jointPara[3] = 30*M_PI/180;
    jointMaxAcc.jointPara[4] = 30*M_PI/180;
    jointMaxAcc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);

    // Set the maximum speed of joint move
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30*M_PI/180;
    jointMaxVelc.jointPara[1] = 30*M_PI/180;
    jointMaxVelc.jointPara[2] = 30*M_PI/180;
    jointMaxVelc.jointPara[3] = 30*M_PI/180;
    jointMaxVelc.jointPara[4] = 30*M_PI/180;
    jointMaxVelc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);
```

```
// The robot moves to the zero position
double jointAngle[aubo_robot_namespace::ARM_DOF] = {0};
jointAngle[0] = 0.0/180.0*M_PI;
jointAngle[1] = 0.0/180.0*M_PI;
jointAngle[2] = 0.0/180.0*M_PI;
jointAngle[3] = 0.0/180.0*M_PI;
jointAngle[4] = 0.0/180.0*M_PI;
jointAngle[5] = 0.0/180.0*M_PI;

robotService.robotServiceJointMove(jointAngle, true);


for(int i=0; i<5; i++)
{
    if(i%2==0)
    {
        // Arrival ahead in follow mode currently only applicable to joint move
        // Set the distance mode of arrival ahead
        robotService.robotServiceSetArrivalAheadDistanceMode(0.2);
    }
    else
    {
        robotService.robotServiceSetNoArrivalAhead();
    }

    jointAngle[0] = 20.0/180.0*M_PI;
    jointAngle[1] = 0.0/180.0*M_PI;
    jointAngle[2] = 90.0/180.0*M_PI;
    jointAngle[3] = 0.0/180.0*M_PI;
    jointAngle[4] = 90.0/180.0*M_PI;
    jointAngle[5] = 0.0/180.0*M_PI;
    robotService.robotServiceJointMove(jointAngle, true);
    if(ret != aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cerr << " Move 1 failed. Error code is：" << ret << std::endl;
        break;
    }
    else
    {
        std::cerr << " Move 1 successful. i = " << i << std::endl;
    }

    jointAngle[0] = 50.0/180.0*M_PI;
    jointAngle[1] = 40.0/180.0*M_PI;
```

```
            jointAngle[2] = 78.0/180.0*M_PI;
            jointAngle[3] = 20.0/180.0*M_PI;
            jointAngle[4] = 66.0/180.0*M_PI;
            jointAngle[5] = 0.0/180.0*M_PI;
            robotService.robotServiceJointMove(jointAngle, true);
            if(ret != aubo_robot_namespace::InterfaceCallSuccCode)
            {
                std::cerr << " Move 2 failed. The error code is: " << ret << std::endl;
                 break;
            }
            else
            {
                std::cerr << " Move 2 successful .i = " << i << std::endl;
            }
        }

}
```

4.10 Line Move

4.10.1 robotServiceLineMove Function

This example is a linear move to the specified joint angle.

The main flow of this program is: The main flow of this program is: (1) Robot login (2) Robot initialization (3) Joint movement to the initial position (4) Linear move.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The code of the moveL() function in example_movel.cpp is as follows:

```
void Example_MoveL::moveL()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
```



```
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics parameters **/,
                                               6            /*collision level*/,
                        true        /* Whether to allow reading pose, true by default */,
                                            true,       /* Leave the default as true */
                                           1000,        /* Leave the default as 1000 */
                                                result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    // Initialize move properties
    robotService.robotServiceInitGlobalMoveProfile();

    // Set the maximum acceleration of joint move
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30*M_PI/180;
    jointMaxAcc.jointPara[1] = 30*M_PI/180;
    jointMaxAcc.jointPara[2] = 30*M_PI/180;
    jointMaxAcc.jointPara[3] = 30*M_PI/180;
    jointMaxAcc.jointPara[4] = 30*M_PI/180;
    jointMaxAcc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);

    // Set the joint movement maximum speed
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30*M_PI/180;
    jointMaxVelc.jointPara[1] = 30*M_PI/180;
```

```
    jointMaxVelc.jointPara[2] = 30*M_PI/180;
    jointMaxVelc.jointPara[3] = 30*M_PI/180;
    jointMaxVelc.jointPara[4] = 30*M_PI/180;
    jointMaxVelc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

    // Set the TCP maximum acceleration
    double lineMaxAcc = 0.2;
    robotService.robotServiceSetGlobalMoveEndMaxLineAcc(lineMaxAcc);

    // Set the maximum speed of the TCP
    double lineMaxVelc = 0.2;
    robotService.robotServiceSetGlobalMoveEndMaxLineVelc(lineMaxVelc);

    //Starting point
    double jointAngle[6] = {};
    jointAngle[0] = 173.108713*M_PI/180;
    jointAngle[1] = -12.075005*M_PI/180;
    jointAngle[2] = -83.663342*M_PI/180;
    jointAngle[3] = -15.641249*M_PI/180;
    jointAngle[4] = -89.140000*M_PI/180;
    jointAngle[5] = -28.328713*M_PI/180;

    // Joint movement to initial position
    robotService.robotServiceJointMove(jointAngle, true);

    jointAngle[0] = 206.372431*M_PI/180;
    jointAngle[1] = -8.979195*M_PI/180;
    jointAngle[2] = -99.242567*M_PI/180;
    jointAngle[3] = -30.164649*M_PI/180;
    jointAngle[4] = -107.130486*M_PI/180;
    jointAngle[5] = 0.065458*M_PI/180;

    // Line move
    robotService.robotServiceLineMove(jointAngle, true);

}
```

4.10.2 robotMoveLineToTargetPositionByRelative Funtion

Example 1: Flange center offset in base coordinate system

This example is a straight line movement to the target position, where the target position is given by the center of the flange in the base coordinate system by offsetting 0.001m

along the X-axis relative to the current position.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) Joint movement to the starting waypoint (4) Setting the base coordinate system (5) Setting the offset (6) Calling the function **robotMoveLineToTargetPositionByRelative** to move to the target location.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The code of the moveToTargetPositionByRelative1 () function in example_movel.cpp is as follows:

```
void Example_MoveL::movelToTargetPositionByRelative1()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6              /*collision level*/,
                        true        /* Whether to allow reading pose, true by default */,
                                            true,       /* Leave the default as true */
                                            1000,       /* Leave the default as 1000 */
                                                result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
```

```
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    // Initialize move properties
    robotService.robotServiceInitGlobalMoveProfile();

    // Set the maximum acceleration of joint move
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30*M_PI/180;
    jointMaxAcc.jointPara[1] = 30*M_PI/180;
    jointMaxAcc.jointPara[2] = 30*M_PI/180;
    jointMaxAcc.jointPara[3] = 30*M_PI/180;
    jointMaxAcc.jointPara[4] = 30*M_PI/180;
    jointMaxAcc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);

    // Set the maximum speed of joint move
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30*M_PI/180;
    jointMaxVelc.jointPara[1] = 30*M_PI/180;
    jointMaxVelc.jointPara[2] = 30*M_PI/180;
    jointMaxVelc.jointPara[3] = 30*M_PI/180;
    jointMaxVelc.jointPara[4] = 30*M_PI/180;
    jointMaxVelc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

    //Starting point
    double initAngle[6] = {};
    initAngle[0] = 173.108713*M_PI/180;
    initAngle[1] = -12.075005*M_PI/180;
    initAngle[2] = -83.663342*M_PI/180;
    initAngle[3] = -15.641249*M_PI/180;
    initAngle[4] = -89.140000*M_PI/180;
    initAngle[5] = -28.328713*M_PI/180;

    // Joint movement to starting waypoint
    robotService.robotServiceJointMove(initAngle, true);

    // Set the base coordinate system
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool baseCoord;
```

```
    baseCoord.coordType = aubo_robot_namespace::BaseCoordinate;

    // Set offset
    aubo_robot_namespace::MoveRelative moveRelative;
    moveRelative.relativePosition[0] = 0.001;
    moveRelative.relativePosition[1] = 0;
    moveRelative.relativePosition[2] = 0;

    // Straight line movement to the target position, where the target position is given by the center of the flange in the base coordinate system by offsetting 0.001m along the X-axis relative to the current position
    robotService.robotMoveLineToTargetPositionByRelative(baseCoord, moveRelative, true);

}
```

Example 2: Flange center offset in user coordinate system

This example is a linear movement to the target position, where the target position is given by the center of the flange in the user coordinate system by offsetting 0.001m along the X axis relative to the current position.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) Joint movement to the starting waypoint (4) setting the user coordinate system and its tool parameters (5) Setting the offset (6) Call the function **robotMoveLineToTargetPositionByRelative** to move to the target position.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The code of the moveToTargetPositionByRelative2 () function in example_movel.cpp is as follows:

```
void Example_MoveL::movelToTargetPositionByRelative2()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

/** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo", "123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
```

```
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6              /*collision level*/,
                        true        /* Whether to allow reading pose, true by default */,
                                          true,       /* Leave the default as true */
                                          1000,       /* Leave the default as 1000 */
                                              result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    // Initialize move properties
    robotService.robotServiceInitGlobalMoveProfile();

    // Set the maximum acceleration of joint move
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30*M_PI/180;
    jointMaxAcc.jointPara[1] = 30*M_PI/180;
    jointMaxAcc.jointPara[2] = 30*M_PI/180;
    jointMaxAcc.jointPara[3] = 30*M_PI/180;
    jointMaxAcc.jointPara[4] = 30*M_PI/180;
    jointMaxAcc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);

    // Set the joint movement maximum speed
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30*M_PI/180;
```

```
    jointMaxVelc.jointPara[1] = 30*M_PI/180;
    jointMaxVelc.jointPara[2] = 30*M_PI/180;
    jointMaxVelc.jointPara[3] = 30*M_PI/180;
    jointMaxVelc.jointPara[4] = 30*M_PI/180;
    jointMaxVelc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

    //Starting position
    double initAngle[6] = {};
    initAngle[0] = 173.108713*M_PI/180;
    initAngle[1] = -12.075005*M_PI/180;
    initAngle[2] = -83.663342*M_PI/180;
    initAngle[3] = -15.641249*M_PI/180;
    initAngle[4] = -89.140000*M_PI/180;
    initAngle[5] = -28.328713*M_PI/180;

    // Joint movement to starting waypoint
    robotService.robotServiceJointMove(initAngle, true);

    // Set the user coordinate system
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool userCoord;
    userCoord.coordType = aubo_robot_namespace::WorldCoordinate;
    userCoord.methods =
aubo_robot_namespace::Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrant
OfXOYPlane;

    userCoord.wayPointArray[0].jointPos[0] = -75.093279*M_PI/180;
    userCoord.wayPointArray[0].jointPos[1] = 28.544643*M_PI/180;
    userCoord.wayPointArray[0].jointPos[2] = -114.313905*M_PI/180;
    userCoord.wayPointArray[0].jointPos[3] = -62.769247*M_PI/180;
    userCoord.wayPointArray[0].jointPos[4] = -87.343517*M_PI/180;
    userCoord.wayPointArray[0].jointPos[5] = -27.888262*M_PI/180;

    userCoord.wayPointArray[1].jointPos[0] = -89.239837*M_PI/180;
    userCoord.wayPointArray[1].jointPos[1] = 23.936171*M_PI/180;
    userCoord.wayPointArray[1].jointPos[2] = -122.299277*M_PI/180;
    userCoord.wayPointArray[1].jointPos[3] = -65.208902*M_PI/180;
    userCoord.wayPointArray[1].jointPos[4] = -85.011123*M_PI/180;
    userCoord.wayPointArray[1].jointPos[5] = -41.87417*M_PI/180;

    userCoord.wayPointArray[2].jointPos[0] = -77.059212*M_PI/180;
    userCoord.wayPointArray[2].jointPos[1] = 35.509518*M_PI/180;
    userCoord.wayPointArray[2].jointPos[2] = -101.108547*M_PI/180;
    userCoord.wayPointArray[2].jointPos[3] = -56.433133*M_PI/180;
```

```
    userCoord.wayPointArray[2].jointPos[4] = -87.006734*M_PI/180;
    userCoord.wayPointArray[2].jointPos[5] = -29.827440*M_PI/180;

    aubo_robot_namespace::ToolInEndDesc toolUserCoord;
    toolUserCoord.toolInEndPosition.x =    -0.177341;
    toolUserCoord.toolInEndPosition.y = 0.002327;
    toolUserCoord.toolInEndPosition.z = 0.146822;
    toolUserCoord.toolInEndOrientation.w = 1;
    toolUserCoord.toolInEndOrientation.x = 0;
    toolUserCoord.toolInEndOrientation.y = 0;
    toolUserCoord.toolInEndOrientation.z = 0;
    userCoord.toolDesc = toolUserCoord;


    // Set offset
    aubo_robot_namespace::MoveRelative moveRelative;
    moveRelative.relativePosition[0] = 0.001;
    moveRelative.relativePosition[1] = 0;
    moveRelative.relativePosition[2] = 0;

    // Straight line movement to the target position, where the target position is given
by the center of the flange in the user coordinate system by offsetting 0.001m along the
X axis relative to the current position
    robotService.robotMoveLineToTargetPositionByRelative(userCoord,
moveRelative, true);

}
```

Example 3: Tool end offset in tool coordinate system

This example is a linear movement to the target position, where the target position is given by the tool tip in the tool coordinate system by offsetting 0.001m along the X axis relative to the current position.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) Joint movement to the starting waypoint (4) Setting the kinematic parameters of the tool (5) Setting the end coordinate system and its tool parameters (6) Set the offset (7) Call the function **robotMoveLineToTargetPositionByRelative** to move to the target position.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The code of the moveToTargetPositionByRelative3 () function in example_movel.cpp is as follows:

```
void Example_MoveL::movelToTargetPositionByRelative3()
```

```
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6              /*collision level*/,
                        true        /* Whether to allow reading pose, true by default */,
                                        true,       /* Leave the default as true */
                                        1000,       /* Leave the default as 1000 */
                                            result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    // Initialize move properties
    robotService.robotServiceInitGlobalMoveProfile();

    // Set the maximum acceleration of joint move
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
```

```
jointMaxAcc.jointPara[0] = 30*M_PI/180;
jointMaxAcc.jointPara[1] = 30*M_PI/180;
jointMaxAcc.jointPara[2] = 30*M_PI/180;
jointMaxAcc.jointPara[3] = 30*M_PI/180;
jointMaxAcc.jointPara[4] = 30*M_PI/180;
jointMaxAcc.jointPara[5] = 30*M_PI/180;
robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);

// Set the maximum speed of joint move
aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
jointMaxVelc.jointPara[0] = 30*M_PI/180;
jointMaxVelc.jointPara[1] = 30*M_PI/180;
jointMaxVelc.jointPara[2] = 30*M_PI/180;
jointMaxVelc.jointPara[3] = 30*M_PI/180;
jointMaxVelc.jointPara[4] = 30*M_PI/180;
jointMaxVelc.jointPara[5] = 30*M_PI/180;
robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

//Starting position
double initAngle[6] = {};
initAngle[0] = 173.108713*M_PI/180;
initAngle[1] = -12.075005*M_PI/180;
initAngle[2] = -83.663342*M_PI/180;
initAngle[3] = -15.641249*M_PI/180;
initAngle[4] = -89.140000*M_PI/180;
initAngle[5] = -28.328713*M_PI/180;

// Joint movement to starting waypoint
robotService.robotServiceJointMove(initAngle, true);

//Setup tool
aubo_robot_namespace::ToolInEndDesc toolEnd;
toolEnd.toolInEndPosition.x =    -0.177341;
toolEnd.toolInEndPosition.y = 0.002327;
toolEnd.toolInEndPosition.z = 0.146822;
toolEnd.toolInEndOrientation.w = 1;
toolEnd.toolInEndOrientation.x = 0;
toolEnd.toolInEndOrientation.y = 0;
toolEnd.toolInEndOrientation.z = 0;
robotService.robotServiceSetToolKinematicsParam(toolEnd);

// Set tool coordinate system
aubo_robot_namespace::CoordCalibrateByJointAngleAndTool endCoord;
endCoord.coordType = aubo_robot_namespace::EndCoordinate;
```

```
    endCoord.toolDesc = toolEnd;

    // Set offset
    aubo_robot_namespace::MoveRelative moveRelative;
    moveRelative.relativePosition[0] = 0.001;
    moveRelative.relativePosition[1] = 0;
    moveRelative.relativePosition[2] = 0;

    // Linear movement to the target position, where the target position is given by the
tool end in the tool coordinate system by offsetting 0.001m along the X axis relative to
the current position
    robotService.robotMoveLineToTargetPositionByRelative(endCoord,
moveRelative, true);
}
```

Example 4: Tool end offset in base coordinate system

This example is a linear movement to the target position, where the target position is given by the tool tip in the base coordinate system by offsetting 0.001m along the X axis relative to the current position.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) joint movement to the starting waypoint (4) setting the kinematic parameters of the tool (5) setting the base coordinate system (6) setting the offset Shift amount (7) call the function **robotMoveLineToTargetPositionByRelative** to move to the target position.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The moveToTargetPositionByRelative4 () function code in example_movel.cpp is as follows:

```
void Example_MoveL::movelToTargetPositionByRelative4()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

        /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
```

```
            std::cerr << "login failed" << std::endl;
        }

        /** If real manipulator is connected, the manipulator needs to be initialized    **/
        aubo_robot_namespace::ROBOT_SERVICE_STATE result;

        //Tool dynamics parameters
        aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
        memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
        ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                                   6              /*collision level*/,
                                true        /* Whether to allow reading pose, true by default */,
                                                     true,       /* Leave the default as true */
                                                    1000,        /* Leave the default as 1000 */
                                                        result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
        {
std::cerr << " Robot initialized successfully." << std::endl;
        }
        else
        {
            std::cerr << " Robot initialization failed." << std::endl;
        }

        // Initialize move properties
        robotService.robotServiceInitGlobalMoveProfile();

        // Set the maximum acceleration of joint move
        aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
        jointMaxAcc.jointPara[0] = 30*M_PI/180;
        jointMaxAcc.jointPara[1] = 30*M_PI/180;
        jointMaxAcc.jointPara[2] = 30*M_PI/180;
        jointMaxAcc.jointPara[3] = 30*M_PI/180;
        jointMaxAcc.jointPara[4] = 30*M_PI/180;
        jointMaxAcc.jointPara[5] = 30*M_PI/180;
        robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);

        // Set the joint movement maximum speed
        aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
        jointMaxVelc.jointPara[0] = 30*M_PI/180;
        jointMaxVelc.jointPara[1] = 30*M_PI/180;
        jointMaxVelc.jointPara[2] = 30*M_PI/180;
```

```
    jointMaxVelc.jointPara[3] = 30*M_PI/180;
    jointMaxVelc.jointPara[4] = 30*M_PI/180;
    jointMaxVelc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

    // starting waypoint
    double initAngle[6] = {};
    initAngle[0] = 173.108713*M_PI/180;
    initAngle[1] = -12.075005*M_PI/180;
    initAngle[2] = -83.663342*M_PI/180;
    initAngle[3] = -15.641249*M_PI/180;
    initAngle[4] = -89.140000*M_PI/180;
    initAngle[5] = -28.328713*M_PI/180;

    // Joint movement to starting waypoint
    robotService.robotServiceJointMove(initAngle, true);

    // setup tool
    aubo_robot_namespace::ToolInEndDesc toolEnd;
    toolEnd.toolInEndPosition.x =    -0.177341;
    toolEnd.toolInEndPosition.y = 0.002327;
    toolEnd.toolInEndPosition.z = 0.146822;
    toolEnd.toolInEndOrientation.w = 1;
    toolEnd.toolInEndOrientation.x = 0;
    toolEnd.toolInEndOrientation.y = 0;
    toolEnd.toolInEndOrientation.z = 0;
    robotService.robotServiceSetToolKinematicsParam(toolEnd);

    // Set the base coordinate system
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool baseCoord;
    baseCoord.coordType = aubo_robot_namespace::BaseCoordinate;

    // set offset
    aubo_robot_namespace::MoveRelative moveRelative;
    moveRelative.relativePosition[0] = 0.001;
    moveRelative.relativePosition[1] = 0;
    moveRelative.relativePosition[2] = 0;

    // Line movement to the target position, where the target position is given by the
tool end in the base coordinate system by offsetting 0.001m along the X axis relative to
the current position
    robotService.robotMoveLineToTargetPositionByRelative(baseCoord,
moveRelative, true);
}
```

Example 5: Tool end offset in user coordinate system

This example is a straight line movement to the target position, where the target position is given by the tool tip in the user coordinate system by offsetting 0.001m along the X axis relative to the current position.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) joint move to the starting waypoint (4) setting the kinematic parameters of the tool (5) setting the user coordinate system and its tool parameters (6) Set the offset (7) Call the function **robotMoveLineToTargetPositionByRelative** to move to the target position.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The code of moveToTargetPositionByRelative5 () function in example_movel.cpp is as follows

```
void Example_MoveL::movelToTargetPositionByRelative5()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

/** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6             /*collision level*/,
                        true        /* Whether to allow reading pose, true by default */,
                                            true,       /* Leave the default as true */
                                            1000,       /* Leave the default as 1000 */
```

```
                                                result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    // Initialize move properties
    robotService.robotServiceInitGlobalMoveProfile();

    // Set the maximum acceleration of joint move
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30*M_PI/180;
    jointMaxAcc.jointPara[1] = 30*M_PI/180;
    jointMaxAcc.jointPara[2] = 30*M_PI/180;
    jointMaxAcc.jointPara[3] = 30*M_PI/180;
    jointMaxAcc.jointPara[4] = 30*M_PI/180;
    jointMaxAcc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);

    // Set the joint movement maximum speed
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30*M_PI/180;
    jointMaxVelc.jointPara[1] = 30*M_PI/180;
    jointMaxVelc.jointPara[2] = 30*M_PI/180;
    jointMaxVelc.jointPara[3] = 30*M_PI/180;
    jointMaxVelc.jointPara[4] = 30*M_PI/180;
    jointMaxVelc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

    // starting waypoint
    double initAngle[6] = {};
    initAngle[0] = 173.108713*M_PI/180;
    initAngle[1] = -12.075005*M_PI/180;
    initAngle[2] = -83.663342*M_PI/180;
    initAngle[3] = -15.641249*M_PI/180;
    initAngle[4] = -89.140000*M_PI/180;
    initAngle[5] = -28.328713*M_PI/180;

    // Joint movement to starting waypoint
```

```
    robotService.robotServiceJointMove(initAngle, true);

    // setup tool
    aubo_robot_namespace::ToolInEndDesc toolEnd;
    toolEnd.toolInEndPosition.x =    -0.177341;
    toolEnd.toolInEndPosition.y = 0.002327;
    toolEnd.toolInEndPosition.z = 0.146822;
    toolEnd.toolInEndOrientation.w = 1;
    toolEnd.toolInEndOrientation.x = 0;
    toolEnd.toolInEndOrientation.y = 0;
    toolEnd.toolInEndOrientation.z = 0;
    robotService.robotServiceSetToolKinematicsParam(toolEnd);

    // Set the user coordinate system
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool userCoord;
    userCoord.coordType = aubo_robot_namespace::WorldCoordinate;
    userCoord.methods =
aubo_robot_namespace::Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrant
OfXOYPlane;

    userCoord.wayPointArray[0].jointPos[0] = -75.093279*M_PI/180;
    userCoord.wayPointArray[0].jointPos[1] = 28.544643*M_PI/180;
    userCoord.wayPointArray[0].jointPos[2] = -114.313905*M_PI/180;
    userCoord.wayPointArray[0].jointPos[3] = -62.769247*M_PI/180;
    userCoord.wayPointArray[0].jointPos[4] = -87.343517*M_PI/180;
    userCoord.wayPointArray[0].jointPos[5] = -27.888262*M_PI/180;

    userCoord.wayPointArray[1].jointPos[0] = -89.239837*M_PI/180;
    userCoord.wayPointArray[1].jointPos[1] = 23.936171*M_PI/180;
    userCoord.wayPointArray[1].jointPos[2] = -122.299277*M_PI/180;
    userCoord.wayPointArray[1].jointPos[3] = -65.208902*M_PI/180;
    userCoord.wayPointArray[1].jointPos[4] = -85.011123*M_PI/180;
    userCoord.wayPointArray[1].jointPos[5] = -41.87417*M_PI/180;

    userCoord.wayPointArray[2].jointPos[0] = -77.059212*M_PI/180;
    userCoord.wayPointArray[2].jointPos[1] = 35.509518*M_PI/180;
    userCoord.wayPointArray[2].jointPos[2] = -101.108547*M_PI/180;
    userCoord.wayPointArray[2].jointPos[3] = -56.433133*M_PI/180;
    userCoord.wayPointArray[2].jointPos[4] = -87.006734*M_PI/180;
    userCoord.wayPointArray[2].jointPos[5] = -29.827440*M_PI/180;

    userCoord.toolDesc = toolEnd;

    // set offset
```

```
    aubo_robot_namespace::MoveRelative moveRelative;
    moveRelative.relativePosition[0] = 0.001;
    moveRelative.relativePosition[1] = 0;
    moveRelative.relativePosition[2] = 0;

    // Line movement to the target position, where the target position is given by the tool end in the user coordinate system by offsetting 0.001m along the X axis relative to the current position
    robotService.robotMoveLineToTargetPositionByRelative(userCoord, moveRelative, true);
}
```

4.10.3 robotMoveLineToTargetPosition Function

Example 1: The position of the flange center in the base coordinate system

This example is a straight line movement to the target position.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) Joint movement to the starting waypoint (4) Setting tool parameters and setting the base coordinate system for the flange center (5) Setting the base coordinate system (6)) set the target position (7) call the function **robotMoveLineToTargetPosition** to move to the target position.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The moveToTargetPosition1() function code in example_movel.cpp is as follows:

```
void Example_MoveL::movelToTargetPosition1()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo", "123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }
```

```
    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                                6             /*collision level*/,
                        true        /* Whether to allow reading pose, true by default */,
                                        true,       /* Leave the default as true */
                                        1000,       /* Leave the default as 1000 */
                                            result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    //Initialize move properties
    robotService.robotServiceInitGlobalMoveProfile();

    // Set the maximum acceleration of joint move
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30*M_PI/180;
    jointMaxAcc.jointPara[1] = 30*M_PI/180;
    jointMaxAcc.jointPara[2] = 30*M_PI/180;
    jointMaxAcc.jointPara[3] = 30*M_PI/180;
    jointMaxAcc.jointPara[4] = 30*M_PI/180;
    jointMaxAcc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);

    // Set the joint movement maximum speed
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30*M_PI/180;
    jointMaxVelc.jointPara[1] = 30*M_PI/180;
    jointMaxVelc.jointPara[2] = 30*M_PI/180;
    jointMaxVelc.jointPara[3] = 30*M_PI/180;
    jointMaxVelc.jointPara[4] = 30*M_PI/180;
    jointMaxVelc.jointPara[5] = 30*M_PI/180;
```

```
    robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

    // starting waypoint
    double initAngle[6] = {};
    initAngle[0] = 173.108713*M_PI/180;
    initAngle[1] = -12.075005*M_PI/180;
    initAngle[2] = -83.663342*M_PI/180;
    initAngle[3] = -15.641249*M_PI/180;
    initAngle[4] = -89.140000*M_PI/180;
    initAngle[5] = -28.328713*M_PI/180;

    // Joint movement to starting waypoint
    robotService.robotServiceJointMove(initAngle, true);

    // Set the tool parameters for the center of the flange
    aubo_robot_namespace::ToolInEndDesc toolEnd;
    toolEnd.toolInEndPosition.x = 0;
    toolEnd.toolInEndPosition.y = 0;
    toolEnd.toolInEndPosition.z = 0;
    toolEnd.toolInEndOrientation.w = 1;
    toolEnd.toolInEndOrientation.x = 0;
    toolEnd.toolInEndOrientation.y = 0;
    toolEnd.toolInEndOrientation.z = 0;

    // Set the base coordinate system
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool baseCoord;
    baseCoord.coordType = aubo_robot_namespace::BaseCoordinate;

    // Set the target position, the position of the flange center in the base coordinate
system
    aubo_robot_namespace::Pos posOnBase;
    posOnBase.x = -0.2;
    posOnBase.y = -0.6;
    posOnBase.z = 0;

    // Line move to target position
    robotService.robotMoveLineToTargetPosition(baseCoord, posOnBase, toolEnd,
true);

}
```

Example 2: The position of the flange center in the user coordinate system

This example is a straight line movement to the target position.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) Joint movement to the starting waypoint (4) Setting tool parameters for the center of the flange (5) Setting the user coordinate system and its Tool parameters (6) set the target position (7) call the function **robotMoveLineToTargetPosition** to move to the target position.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The code of the moveToTargetPosition2() function in example_movel.cpp is as follows:

```
void Example_MoveL::movelToTargetPosition2()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6              /*collision level*/,
                        true        /* Whether to allow reading pose, true by default */,
                                          true,       /* Leave the default as true */
                                          1000,       /* Leave the default as 1000 */
                                              result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
```

```
{
    std::cerr << " Robot initialization failed." << std::endl;
}

// Initialize move properties
robotService.robotServiceInitGlobalMoveProfile();

// Set the maximum acceleration of joint move
aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
jointMaxAcc.jointPara[0] = 30*M_PI/180;
jointMaxAcc.jointPara[1] = 30*M_PI/180;
jointMaxAcc.jointPara[2] = 30*M_PI/180;
jointMaxAcc.jointPara[3] = 30*M_PI/180;
jointMaxAcc.jointPara[4] = 30*M_PI/180;
jointMaxAcc.jointPara[5] = 30*M_PI/180;
robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);

// Set the maximum speed of joint move
aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
jointMaxVelc.jointPara[0] = 30*M_PI/180;
jointMaxVelc.jointPara[1] = 30*M_PI/180;
jointMaxVelc.jointPara[2] = 30*M_PI/180;
jointMaxVelc.jointPara[3] = 30*M_PI/180;
jointMaxVelc.jointPara[4] = 30*M_PI/180;
jointMaxVelc.jointPara[5] = 30*M_PI/180;
robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

// starting waypoint
double initAngle[6] = {};
initAngle[0] = 173.108713*M_PI/180;
initAngle[1] = -12.075005*M_PI/180;
initAngle[2] = -83.663342*M_PI/180;
initAngle[3] = -15.641249*M_PI/180;
initAngle[4] = -89.140000*M_PI/180;
initAngle[5] = -28.328713*M_PI/180;

// Joint movement to starting waypoint
robotService.robotServiceJointMove(initAngle, true);

// Set the tool parameters for the center of the flange
aubo_robot_namespace::ToolInEndDesc toolEnd;
toolEnd.toolInEndPosition.x = 0;
toolEnd.toolInEndPosition.y = 0;
toolEnd.toolInEndPosition.z = 0;
```

```
    toolEnd.toolInEndOrientation.w = 1;
    toolEnd.toolInEndOrientation.x = 0;
    toolEnd.toolInEndOrientation.y = 0;
    toolEnd.toolInEndOrientation.z = 0;

    // Set the user coordinate system
aubo_robot_namespace::CoordCalibrateByJointAngleAndTool userCoord;
    userCoord.coordType = aubo_robot_namespace::WorldCoordinate;
    userCoord.methods =
aubo_robot_namespace::Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrant
OfXOYPlane;

    userCoord.wayPointArray[0].jointPos[0] = -75.093279*M_PI/180;
    userCoord.wayPointArray[0].jointPos[1] = 28.544643*M_PI/180;
    userCoord.wayPointArray[0].jointPos[2] = -114.313905*M_PI/180;
    userCoord.wayPointArray[0].jointPos[3] = -62.769247*M_PI/180;
    userCoord.wayPointArray[0].jointPos[4] = -87.343517*M_PI/180;
    userCoord.wayPointArray[0].jointPos[5] = -27.888262*M_PI/180;

    userCoord.wayPointArray[1].jointPos[0] = -89.239837*M_PI/180;
    userCoord.wayPointArray[1].jointPos[1] = 23.936171*M_PI/180;
    userCoord.wayPointArray[1].jointPos[2] = -122.299277*M_PI/180;
    userCoord.wayPointArray[1].jointPos[3] = -65.208902*M_PI/180;
    userCoord.wayPointArray[1].jointPos[4] = -85.011123*M_PI/180;
    userCoord.wayPointArray[1].jointPos[5] = -41.87417*M_PI/180;

    userCoord.wayPointArray[2].jointPos[0] = -77.059212*M_PI/180;
    userCoord.wayPointArray[2].jointPos[1] = 35.509518*M_PI/180;
    userCoord.wayPointArray[2].jointPos[2] = -101.108547*M_PI/180;
    userCoord.wayPointArray[2].jointPos[3] = -56.433133*M_PI/180;
    userCoord.wayPointArray[2].jointPos[4] = -87.006734*M_PI/180;
    userCoord.wayPointArray[2].jointPos[5] = -29.827440*M_PI/180;

    // Tool Parameters for Coordinate Systems
    aubo_robot_namespace::ToolInEndDesc toolDesc;
    toolDesc.toolInEndPosition.x =    -0.177341;
    toolDesc.toolInEndPosition.y = 0.002327;
    toolDesc.toolInEndPosition.z = 0.146822;
    toolDesc.toolInEndOrientation.w = 1;
    toolDesc.toolInEndOrientation.x = 0;
    toolDesc.toolInEndOrientation.y = 0;
    toolDesc.toolInEndOrientation.z = 0;
    userCoord.toolDesc = toolDesc;
```

```
    // Set the target position, the position of the flange center in the user coordinate
system
    aubo_robot_namespace::Pos posOnUser;
    posOnUser.x = -0.220305;
    posOnUser.y = -1.152177;
    posOnUser.z = 0.216184;

    // Line move to target position
    robotService.robotMoveLineToTargetPosition(userCoord, posOnUser, toolEnd,
true);
}
```

3: The position of the tool end in the base coordinate system

This example is a straight line movement to the target position.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) Joint movement to the starting waypoint (4) Setting tool parameters (5) Setting base coordinate system (6) Setting target position (7)) call the function robotMoveLineToTargetPosition to move to the target position.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The code of the moveToTargetPosition3() function in example_movel.cpp is as follows:

```
void Example_MoveL::movelToTargetPosition3()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

/** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;
```

```
    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                                6            /*collision level*/,
                            true        /* Whether to allow reading pose, true by default */,
                                                true,       /* Leave the default as true */
                                                1000,       /* Leave the default as 1000 */
                                                    result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    // Initialize move properties
    robotService.robotServiceInitGlobalMoveProfile();

    // Set the maximum acceleration of joint move
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30*M_PI/180;
    jointMaxAcc.jointPara[1] = 30*M_PI/180;
    jointMaxAcc.jointPara[2] = 30*M_PI/180;
    jointMaxAcc.jointPara[3] = 30*M_PI/180;
    jointMaxAcc.jointPara[4] = 30*M_PI/180;
    jointMaxAcc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);

    // Set the maximum speed of joint move
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30*M_PI/180;
    jointMaxVelc.jointPara[1] = 30*M_PI/180;
    jointMaxVelc.jointPara[2] = 30*M_PI/180;
    jointMaxVelc.jointPara[3] = 30*M_PI/180;
    jointMaxVelc.jointPara[4] = 30*M_PI/180;
    jointMaxVelc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

    //Starting waypoint
```

```
    double initAngle[6] = {};
    initAngle[0] = 173.108713*M_PI/180;
    initAngle[1] = -12.075005*M_PI/180;
    initAngle[2] = -83.663342*M_PI/180;
    initAngle[3] = -15.641249*M_PI/180;
    initAngle[4] = -89.140000*M_PI/180;
    initAngle[5] = -28.328713*M_PI/180;

    // Joint movement to starting waypoint
    robotService.robotServiceJointMove(initAngle, true);

    // Set tool parameters for tool end
    aubo_robot_namespace::ToolInEndDesc toolEnd;
    toolEnd.toolInEndPosition.x = -0.177341;
    toolEnd.toolInEndPosition.y = 0.002327;
    toolEnd.toolInEndPosition.z = 0.146822;
    toolEnd.toolInEndOrientation.w = 1;
    toolEnd.toolInEndOrientation.x = 0;
    toolEnd.toolInEndOrientation.y = 0;
    toolEnd.toolInEndOrientation.z = 0;

    // Set the base coordinate system
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool baseCoord;
    baseCoord.coordType = aubo_robot_namespace::BaseCoordinate;

    // Set the target position, the position of the tool end in the base coordinate system
    aubo_robot_namespace::Pos posOnBase;
    posOnBase.x = 0.56;
    posOnBase.y = -0.10;
    posOnBase.z = 0.33;

    // Linear move to target position
    robotService.robotMoveLineToTargetPosition(baseCoord, posOnBase, toolEnd,
true);

}
```

Example 4: The position of the tool end in the user coordinate system

This example is a straight line movement to the target position.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) Joint movement to the starting waypoint (4) Setting tool parameters (5) Setting the user coordinate system and its tool parameters (6) Setting The target position (7) calls the function robotMoveLineToTargetPosition to move to the target position.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The code of the moveToTargetPosition2() function in example_movel.cpp is as follows:

```
void Example_MoveL::movelToTargetPosition4()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

/** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6              /*collision level*/,
                           true         /* Whether to allow reading pose, true by default */,
                                        true,       /* Leave the default as true */
                                        1000,       /* Leave the default as 1000 */
                                        result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }
```

```
// Initialize move properties
robotService.robotServiceInitGlobalMoveProfile();

// Set the maximum acceleration of joint move
aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
jointMaxAcc.jointPara[0] = 30*M_PI/180;
jointMaxAcc.jointPara[1] = 30*M_PI/180;
jointMaxAcc.jointPara[2] = 30*M_PI/180;
jointMaxAcc.jointPara[3] = 30*M_PI/180;
jointMaxAcc.jointPara[4] = 30*M_PI/180;
jointMaxAcc.jointPara[5] = 30*M_PI/180;
robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);

// Set the maximum speed of joint move
aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
jointMaxVelc.jointPara[0] = 30*M_PI/180;
jointMaxVelc.jointPara[1] = 30*M_PI/180;
jointMaxVelc.jointPara[2] = 30*M_PI/180;
jointMaxVelc.jointPara[3] = 30*M_PI/180;
jointMaxVelc.jointPara[4] = 30*M_PI/180;
jointMaxVelc.jointPara[5] = 30*M_PI/180;
robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

// starting waypoint
double initAngle[6] = {};
initAngle[0] = 173.108713*M_PI/180;
initAngle[1] = -12.075005*M_PI/180;
initAngle[2] = -83.663342*M_PI/180;
initAngle[3] = -15.641249*M_PI/180;
initAngle[4] = -89.140000*M_PI/180;
initAngle[5] = -28.328713*M_PI/180;

//关节运动到起始路点
robotService.robotServiceJointMove(initAngle, true);

// Set tool parameters for tool end
aubo_robot_namespace::ToolInEndDesc toolEnd;
toolEnd.toolInEndPosition.x = -0.177341;
toolEnd.toolInEndPosition.y = 0.002327;
toolEnd.toolInEndPosition.z = 0.146822;
toolEnd.toolInEndOrientation.w = 1;
toolEnd.toolInEndOrientation.x = 0;
toolEnd.toolInEndOrientation.y = 0;
toolEnd.toolInEndOrientation.z = 0;
```

```
    // Set the user coordinate system
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool userCoord;
    userCoord.coordType = aubo_robot_namespace::WorldCoordinate;
    userCoord.methods =
aubo_robot_namespace::Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrant
OfXOYPlane;

    userCoord.wayPointArray[0].jointPos[0] = -75.093279*M_PI/180;
    userCoord.wayPointArray[0].jointPos[1] = 28.544643*M_PI/180;
    userCoord.wayPointArray[0].jointPos[2] = -114.313905*M_PI/180;
    userCoord.wayPointArray[0].jointPos[3] = -62.769247*M_PI/180;
    userCoord.wayPointArray[0].jointPos[4] = -87.343517*M_PI/180;
    userCoord.wayPointArray[0].jointPos[5] = -27.888262*M_PI/180;

    userCoord.wayPointArray[1].jointPos[0] = -89.239837*M_PI/180;
    userCoord.wayPointArray[1].jointPos[1] = 23.936171*M_PI/180;
    userCoord.wayPointArray[1].jointPos[2] = -122.299277*M_PI/180;
    userCoord.wayPointArray[1].jointPos[3] = -65.208902*M_PI/180;
    userCoord.wayPointArray[1].jointPos[4] = -85.011123*M_PI/180;
    userCoord.wayPointArray[1].jointPos[5] = -41.87417*M_PI/180;

    userCoord.wayPointArray[2].jointPos[0] = -77.059212*M_PI/180;
    userCoord.wayPointArray[2].jointPos[1] = 35.509518*M_PI/180;
    userCoord.wayPointArray[2].jointPos[2] = -101.108547*M_PI/180;
    userCoord.wayPointArray[2].jointPos[3] = -56.433133*M_PI/180;
    userCoord.wayPointArray[2].jointPos[4] = -87.006734*M_PI/180;
    userCoord.wayPointArray[2].jointPos[5] = -29.827440*M_PI/180;

    // Tool parameters for coordinate systems
    aubo_robot_namespace::ToolInEndDesc toolDesc;
    toolDesc.toolInEndPosition.x =    -0.177341;
    toolDesc.toolInEndPosition.y = 0.002327;
    toolDesc.toolInEndPosition.z = 0.146822;
    toolDesc.toolInEndOrientation.w = 1;
    toolDesc.toolInEndOrientation.x = 0;
    toolDesc.toolInEndOrientation.y = 0;
    toolDesc.toolInEndOrientation.z = 0;
    userCoord.toolDesc = toolDesc;

    // Set the target position, the position of the tool end in the user coordinate system
    aubo_robot_namespace::Pos posOnUser;
    posOnUser.x = 0.701194;
    posOnUser.y = -0.730656;
```

```
    posOnUser.z = 0.386717;

    // Line move to target position
    robotService.robotMoveLineToTargetPosition(userCoord, posOnUser, toolEnd,
true);
}
```

4.11 Offset move

4.11.1 robotServiceSetMoveRelativeParam Function

Example 1: Flange center is in the base coordinate system

In this example, the robot arm does the pose offset movement, and the flange center is offset by 10 degrees along the X-axis in the base coordinate system.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) Setting the base coordinate system (4) Setting the offset (5) Joint move.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The demo1() function code in example_moverelative.cpp is as follows:

```
void Example_MoveRotate::demo1()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

/** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;
```

```
    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                                6            /*collision level*/,
                            true        /* Whether to allow reading pose, true by default */,
                                                true,       /* Leave the default as true */
                                                1000,       /* Leave the default as 1000 */
                                                    result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    // Initialize move properties
    robotService.robotServiceInitGlobalMoveProfile();

    // Set the maximum acceleration of joint move
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30*M_PI/180;
    jointMaxAcc.jointPara[1] = 30*M_PI/180;
    jointMaxAcc.jointPara[2] = 30*M_PI/180;
    jointMaxAcc.jointPara[3] = 30*M_PI/180;
    jointMaxAcc.jointPara[4] = 30*M_PI/180;
    jointMaxAcc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);

    // Set the maximum speed of joint move
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30*M_PI/180;
    jointMaxVelc.jointPara[1] = 30*M_PI/180;
    jointMaxVelc.jointPara[2] = 30*M_PI/180;
    jointMaxVelc.jointPara[3] = 30*M_PI/180;
    jointMaxVelc.jointPara[4] = 30*M_PI/180;
    jointMaxVelc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

    //Joint angle
```

```
    double jointAngle[6] = {};
    jointAngle[0] = 173.108713*M_PI/180;
    jointAngle[1] = -12.075005*M_PI/180;
    jointAngle[2] = -83.663342*M_PI/180;
    jointAngle[3] = -15.641249*M_PI/180;
    jointAngle[4] = -89.140000*M_PI/180;
    jointAngle[5] = -28.328713*M_PI/180;

    // Joint movement to starting position
    robotService.robotServiceJointMove(jointAngle, true);

    //Set the base coordinate system
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool baseCoord;
    baseCoord.coordType = aubo_robot_namespace::BaseCoordinate;

    // Rotation axis and rotation angle
    double rotateAxis[3] = {1, 0, 0};
    double rotateAngle = 10.0*M_PI/180;

    // Rotation movement: the flange center rotates 10 degrees along the x+ direction
in the base coordinate system, and the current position remains unchanged
    robotService.robotServiceRotateMove(baseCoord, rotateAxis, rotateAngle, true);

}
```

Example 2: Flange center in the user coordinate system

In this example, the robot arm does an pose offset movement, and the flange center is offset by 10 degrees along the X-axis in the user coordinate system.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) Setting the user coordinate system and its tool parameters (4) Setting the offset (5) Joint move.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The demo2() function code in example_moverelative.cpp is as follows:

```
void Example_MoveRotate::demo2()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

/** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
```

```
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6              /*collision level*/,
                        true        /* Whether to allow reading pose, true by default */,
                                          true,       /* Leave the default as true */
                                          1000,       /* Leave the default as 1000 */
                                              result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    // Initialize move properties
    robotService.robotServiceInitGlobalMoveProfile();

    // Set the maximum acceleration of joint move
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30*M_PI/180;
    jointMaxAcc.jointPara[1] = 30*M_PI/180;
    jointMaxAcc.jointPara[2] = 30*M_PI/180;
    jointMaxAcc.jointPara[3] = 30*M_PI/180;
    jointMaxAcc.jointPara[4] = 30*M_PI/180;
    jointMaxAcc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);
```

```
    // Set the maximum speed of joint move
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30*M_PI/180;
    jointMaxVelc.jointPara[1] = 30*M_PI/180;
    jointMaxVelc.jointPara[2] = 30*M_PI/180;
    jointMaxVelc.jointPara[3] = 30*M_PI/180;
    jointMaxVelc.jointPara[4] = 30*M_PI/180;
    jointMaxVelc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

    //Joint angle
    double jointAngle[6] = {};
    jointAngle[0] = 173.108713*M_PI/180;
    jointAngle[1] = -12.075005*M_PI/180;
    jointAngle[2] = -83.663342*M_PI/180;
    jointAngle[3] = -15.641249*M_PI/180;
    jointAngle[4] = -89.140000*M_PI/180;
    jointAngle[5] = -28.328713*M_PI/180;

    // Joint movement to starting position
    robotService.robotServiceJointMove(jointAngle, true);

    // Set the user coordinate system
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool userCoord;
    userCoord.coordType = aubo_robot_namespace::WorldCoordinate;
    userCoord.methods =
aubo_robot_namespace::Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrant
OfXOYPlane;

    userCoord.wayPointArray[0].jointPos[0] = -75.093279*M_PI/180;
    userCoord.wayPointArray[0].jointPos[1] = 28.544643*M_PI/180;
    userCoord.wayPointArray[0].jointPos[2] = -114.313905*M_PI/180;
    userCoord.wayPointArray[0].jointPos[3] = -62.769247*M_PI/180;
    userCoord.wayPointArray[0].jointPos[4] = -87.343517*M_PI/180;
    userCoord.wayPointArray[0].jointPos[5] = -27.888262*M_PI/180;

    userCoord.wayPointArray[1].jointPos[0] = -89.239837*M_PI/180;
    userCoord.wayPointArray[1].jointPos[1] = 23.936171*M_PI/180;
    userCoord.wayPointArray[1].jointPos[2] = -122.299277*M_PI/180;
    userCoord.wayPointArray[1].jointPos[3] = -65.208902*M_PI/180;
    userCoord.wayPointArray[1].jointPos[4] = -85.011123*M_PI/180;
    userCoord.wayPointArray[1].jointPos[5] = -41.87417*M_PI/180;
```

```
    userCoord.wayPointArray[2].jointPos[0] = -77.059212*M_PI/180;
    userCoord.wayPointArray[2].jointPos[1] = 35.509518*M_PI/180;
    userCoord.wayPointArray[2].jointPos[2] = -101.108547*M_PI/180;
    userCoord.wayPointArray[2].jointPos[3] = -56.433133*M_PI/180;
    userCoord.wayPointArray[2].jointPos[4] = -87.006734*M_PI/180;
    userCoord.wayPointArray[2].jointPos[5] = -29.827440*M_PI/180;

    aubo_robot_namespace::ToolInEndDesc toolUserCoord;
    toolUserCoord.toolInEndPosition.x =    -0.177341;
    toolUserCoord.toolInEndPosition.y = 0.002327;
    toolUserCoord.toolInEndPosition.z = 0.146822;
    toolUserCoord.toolInEndOrientation.w = 1;
    toolUserCoord.toolInEndOrientation.x = 0;
    toolUserCoord.toolInEndOrientation.y = 0;
    toolUserCoord.toolInEndOrientation.z = 0;
    userCoord.toolDesc = toolUserCoord;

    // Rotation axis and rotation angle
    double rotateAxis[3] = {1, 0, 0};
    double rotateAngle = 10.0*M_PI/180;

    // Rotation movement: the flange center rotates 10 degrees along the x+ direction
in the user coordinate system, and the current position remains unchanged
    robotService.robotServiceRotateMove(userCoord, rotateAxis, rotateAngle, true);
}
```

Example 3: Tool end in tool coordinate system

In this example, the robot arm does an pose offset move, and the tool is offset by 10 degrees along the X axis in the tool coordinate system.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) Setting the kinematic parameters of the tool (4) Setting the end coordinate system and its tool parameters (5) Setting the offset (6) Joints sports.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The demo3() function code in example_moverelative.cpp is as follows:

```
void Example_MoveRotate::demo3()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

/** Call interface: login ***/
```

```
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6              /*collision level*/,
                        true        /* Whether to allow reading pose, true by default */,
                                          true,       /* Leave the default as true */
                                          1000,       /* Leave the default as 1000 */
                                              result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    // Initialize move properties
    robotService.robotServiceInitGlobalMoveProfile();

    // Set the maximum acceleration of joint move
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30*M_PI/180;
    jointMaxAcc.jointPara[1] = 30*M_PI/180;
    jointMaxAcc.jointPara[2] = 30*M_PI/180;
    jointMaxAcc.jointPara[3] = 30*M_PI/180;
    jointMaxAcc.jointPara[4] = 30*M_PI/180;
```

```
jointMaxAcc.jointPara[5] = 30*M_PI/180;
robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);

// Set the maximum speed of joint move
aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
jointMaxVelc.jointPara[0] = 30*M_PI/180;
jointMaxVelc.jointPara[1] = 30*M_PI/180;
jointMaxVelc.jointPara[2] = 30*M_PI/180;
jointMaxVelc.jointPara[3] = 30*M_PI/180;
jointMaxVelc.jointPara[4] = 30*M_PI/180;
jointMaxVelc.jointPara[5] = 30*M_PI/180;
robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

//Joint angle
double jointAngle[6] = {};
jointAngle[0] = 173.108713*M_PI/180;
jointAngle[1] = -12.075005*M_PI/180;
jointAngle[2] = -83.663342*M_PI/180;
jointAngle[3] = -15.641249*M_PI/180;
jointAngle[4] = -89.140000*M_PI/180;
jointAngle[5] = -28.328713*M_PI/180;

// Starting waypoint
double initAngle[6] = {};
initAngle[0] = 173.108713*M_PI/180;
initAngle[1] = -12.075005*M_PI/180;
initAngle[2] = -83.663342*M_PI/180;
initAngle[3] = -15.641249*M_PI/180;
initAngle[4] = -89.140000*M_PI/180;
initAngle[5] = -28.328713*M_PI/180;

// Joint movement to starting waypoint
robotService.robotServiceJointMove(initAngle, true);

// Setup tool
aubo_robot_namespace::ToolInEndDesc toolEnd;
toolEnd.toolInEndPosition.x =    -0.177341;
toolEnd.toolInEndPosition.y = 0.002327;
toolEnd.toolInEndPosition.z = 0.146822;
toolEnd.toolInEndOrientation.w = 1;
toolEnd.toolInEndOrientation.x = 0;
toolEnd.toolInEndOrientation.y = 0;
toolEnd.toolInEndOrientation.z = 0;
robotService.robotServiceSetToolKinematicsParam(toolEnd);
```

```
    // Set tool coordinate system
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool endCoord;
    endCoord.coordType = aubo_robot_namespace::EndCoordinate;
    endCoord.toolDesc = toolEnd;

    // Rotation axis and rotation angle
    double rotateAxis[3] = {1, 0, 0};
    double rotateAngle = 10.0*M_PI/180;

    // Rotation movement: the tool end rotates 10 degrees in the x+ direction under the tool coordinate system, and the current position remains unchanged
    robotService.robotServiceRotateMove(endCoord, rotateAxis, rotateAngle, true);

}
```

Example 4: Tool end in base coordinate system

In this example, the robot arm does an attitude offset move, and the tool is offset by 10 degrees along the X axis in the base coordinate system.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) Setting the kinematic parameters of the tool (4) Setting the base coordinate system (5) Setting the offset (6) Joint motion.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The demo4() function code in example_moverelative.cpp is as follows:

```
void Example_MoveRotate::demo4()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo", "123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }
```

```
    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics parameters **/,
                                                    6              /*collision level*/,
                            true        /* Whether to allow reading pose, true by default */,
                                                true,       /* Leave the default as true */
                                               1000,        /* Leave the default as 1000 */
                                                    result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    // Initialize move properties
    robotService.robotServiceInitGlobalMoveProfile();

    // Set the maximum acceleration of joint move
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30*M_PI/180;
    jointMaxAcc.jointPara[1] = 30*M_PI/180;
    jointMaxAcc.jointPara[2] = 30*M_PI/180;
    jointMaxAcc.jointPara[3] = 30*M_PI/180;
    jointMaxAcc.jointPara[4] = 30*M_PI/180;
    jointMaxAcc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);

    // Set the maximum speed of joint move
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30*M_PI/180;
    jointMaxVelc.jointPara[1] = 30*M_PI/180;
    jointMaxVelc.jointPara[2] = 30*M_PI/180;
    jointMaxVelc.jointPara[3] = 30*M_PI/180;
    jointMaxVelc.jointPara[4] = 30*M_PI/180;
```

```
    jointMaxVelc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

    //Starting waypoint
    double initAngle[6] = {};
    initAngle[0] = 173.108713*M_PI/180;
    initAngle[1] = -12.075005*M_PI/180;
    initAngle[2] = -83.663342*M_PI/180;
    initAngle[3] = -15.641249*M_PI/180;
    initAngle[4] = -89.140000*M_PI/180;
    initAngle[5] = -28.328713*M_PI/180;

    // Joint movement to starting waypoint
    robotService.robotServiceJointMove(initAngle, true);

    // Setup tool
    aubo_robot_namespace::ToolInEndDesc toolEnd;
    toolEnd.toolInEndPosition.x =    -0.177341;
    toolEnd.toolInEndPosition.y = 0.002327;
    toolEnd.toolInEndPosition.z = 0.146822;
    toolEnd.toolInEndOrientation.w = 1;
    toolEnd.toolInEndOrientation.x = 0;
    toolEnd.toolInEndOrientation.y = 0;
    toolEnd.toolInEndOrientation.z = 0;
    robotService.robotServiceSetToolKinematicsParam(toolEnd);

    // Set the base coordinate system
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool baseCoord;
    baseCoord.coordType = aubo_robot_namespace::BaseCoordinate;

    // Rotation axis and rotation angle
    double rotateAxis[3] = {1, 0, 0};
    double rotateAngle = 10.0*M_PI/180;

    // Rotation movement: the tool end rotates 10 degrees along the x+ direction in the
base coordinate system, and the current position remains unchanged
    robotService.robotServiceRotateMove(baseCoord, rotateAxis, rotateAngle, true);

}
```

Example 5: Tool end in user coordinate system

In this example, the robot arm does an offset move, and the tool is offset by 10 degrees along the X axis in the user coordinate system.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) Setting the kinematic parameters of the tool (4) Setting the user coordinate system and its tool parameters (5) Setting the offset (6) Joints sports.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The demo5() function code in example_moverelative.cpp is as follows:

```
void Example_MoveRotate::demo5()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6              /*collision level*/,
                        true        /* Whether to allow reading pose, true by default */,
                                    true,       /* Leave the default as true */
                                    1000,       /* Leave the default as 1000 */
                                        result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
```

```
        std::cerr << " Robot initialization failed." << std::endl;
}

// Initialize move properties
robotService.robotServiceInitGlobalMoveProfile();

// Set the maximum acceleration of joint move
aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
jointMaxAcc.jointPara[0] = 30*M_PI/180;
jointMaxAcc.jointPara[1] = 30*M_PI/180;
jointMaxAcc.jointPara[2] = 30*M_PI/180;
jointMaxAcc.jointPara[3] = 30*M_PI/180;
jointMaxAcc.jointPara[4] = 30*M_PI/180;
jointMaxAcc.jointPara[5] = 30*M_PI/180;
robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);

// Set the maximum speed of joint move
aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
jointMaxVelc.jointPara[0] = 30*M_PI/180;
jointMaxVelc.jointPara[1] = 30*M_PI/180;
jointMaxVelc.jointPara[2] = 30*M_PI/180;
jointMaxVelc.jointPara[3] = 30*M_PI/180;
jointMaxVelc.jointPara[4] = 30*M_PI/180;
jointMaxVelc.jointPara[5] = 30*M_PI/180;
robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

//Starting waypoint
double initAngle[6] = {};
initAngle[0] = 173.108713*M_PI/180;
initAngle[1] = -12.075005*M_PI/180;
initAngle[2] = -83.663342*M_PI/180;
initAngle[3] = -15.641249*M_PI/180;
initAngle[4] = -89.140000*M_PI/180;
initAngle[5] = -28.328713*M_PI/180;

// Joint movement to starting waypoint
robotService.robotServiceJointMove(initAngle, true);

//Set the tool
aubo_robot_namespace::ToolInEndDesc toolEnd;
toolEnd.toolInEndPosition.x =    -0.177341;
toolEnd.toolInEndPosition.y = 0.002327;
toolEnd.toolInEndPosition.z = 0.146822;
toolEnd.toolInEndOrientation.w = 1;
```

```
    toolEnd.toolInEndOrientation.x = 0;
    toolEnd.toolInEndOrientation.y = 0;
    toolEnd.toolInEndOrientation.z = 0;
    robotService.robotServiceSetToolKinematicsParam(toolEnd);

    // Set the user coordinate system
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool userCoord;
    userCoord.coordType = aubo_robot_namespace::WorldCoordinate;
    userCoord.methods =
aubo_robot_namespace::Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrant
OfXOYPlane;

    userCoord.wayPointArray[0].jointPos[0] = -75.093279*M_PI/180;
    userCoord.wayPointArray[0].jointPos[1] = 28.544643*M_PI/180;
    userCoord.wayPointArray[0].jointPos[2] = -114.313905*M_PI/180;
    userCoord.wayPointArray[0].jointPos[3] = -62.769247*M_PI/180;
    userCoord.wayPointArray[0].jointPos[4] = -87.343517*M_PI/180;
    userCoord.wayPointArray[0].jointPos[5] = -27.888262*M_PI/180;

    userCoord.wayPointArray[1].jointPos[0] = -89.239837*M_PI/180;
    userCoord.wayPointArray[1].jointPos[1] = 23.936171*M_PI/180;
    userCoord.wayPointArray[1].jointPos[2] = -122.299277*M_PI/180;
    userCoord.wayPointArray[1].jointPos[3] = -65.208902*M_PI/180;
    userCoord.wayPointArray[1].jointPos[4] = -85.011123*M_PI/180;
    userCoord.wayPointArray[1].jointPos[5] = -41.87417*M_PI/180;

    userCoord.wayPointArray[2].jointPos[0] = -77.059212*M_PI/180;
    userCoord.wayPointArray[2].jointPos[1] = 35.509518*M_PI/180;
    userCoord.wayPointArray[2].jointPos[2] = -101.108547*M_PI/180;
    userCoord.wayPointArray[2].jointPos[3] = -56.433133*M_PI/180;
    userCoord.wayPointArray[2].jointPos[4] = -87.006734*M_PI/180;
    userCoord.wayPointArray[2].jointPos[5] = -29.827440*M_PI/180;

    userCoord.toolDesc = toolEnd;

    // Rotation axis and rotation angle
    double rotateAxis[3] = {1, 0, 0};
    double rotateAngle = 10.0*M_PI/180;

    // Rotation movement: the tool end rotates 10 degrees in the x+ direction in the
user coordinate system, and the current position remains unchanged
    robotService.robotServiceRotateMove(userCoord, rotateAxis, rotateAngle, true);

}
```

4.12 Rotational move

4.12.1 robotServiceRotateMove Function

Example 1: Flange center in the base coordinate system

In this example, the flange center is rotated 10 degrees along the X+ direction in the user coordinate system, and the current position remains unchanged.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) Joint movement to the starting position (5) Setting the base coordinate system (6) Setting the rotation axis and rotation angle (7) Rotational movement.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The demo1() function code in example_moverotate.cpp is as follows:

```
void Example_MoveRotate::demo1()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

/** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
```

```
                                                    6               /*collision level*/,
                            true        /* Whether to allow reading pose, true by default */,
                                                true,       /* Leave the default as true */
                                               1000,       /* Leave the default as 1000 */
                                                    result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    // Initialize move properties
    robotService.robotServiceInitGlobalMoveProfile();

    // Set the maximum acceleration of joint move
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30*M_PI/180;
    jointMaxAcc.jointPara[1] = 30*M_PI/180;
    jointMaxAcc.jointPara[2] = 30*M_PI/180;
    jointMaxAcc.jointPara[3] = 30*M_PI/180;
    jointMaxAcc.jointPara[4] = 30*M_PI/180;
    jointMaxAcc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);

    // Set the maximum speed of joint move
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30*M_PI/180;
    jointMaxVelc.jointPara[1] = 30*M_PI/180;
    jointMaxVelc.jointPara[2] = 30*M_PI/180;
    jointMaxVelc.jointPara[3] = 30*M_PI/180;
    jointMaxVelc.jointPara[4] = 30*M_PI/180;
    jointMaxVelc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

    //Joint angle
    double jointAngle[6] = {};
    jointAngle[0] = 173.108713*M_PI/180;
    jointAngle[1] = -12.075005*M_PI/180;
    jointAngle[2] = -83.663342*M_PI/180;
    jointAngle[3] = -15.641249*M_PI/180;
```

```
    jointAngle[4] = -89.140000*M_PI/180;
    jointAngle[5] = -28.328713*M_PI/180;

    // Joint movement to starting position
    robotService.robotServiceJointMove(jointAngle, true);

    // Set the base coordinate system
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool baseCoord;
    baseCoord.coordType = aubo_robot_namespace::BaseCoordinate;

    // Rotation axis and rotation angle
    double rotateAxis[3] = {1, 0, 0};
    double rotateAngle = 10.0*M_PI/180;

    // Rotation movement: the flange center rotates 10 degrees along the x+ direction
in the base coordinate system, and the current position remains unchanged
    robotService.robotServiceRotateMove(baseCoord, rotateAxis, rotateAngle, true);

}
```

Example 2: Flange center in the user coordinate system

This example shows that the flange center rotates 10 degrees in the x+ direction in the user coordinate system, and the current position remains unchanged.

The main process of this example is: (1) robot login (2) robot initialization (3) joint movement to the starting position (5) setting the user coordinate system and its tool parameters (6) setting the rotation axis and rotation angle (7) rotation movement.

Note: change the IP address (server_host) in the following code to the IP address of the corresponding server.

The demo2() function code in example_moverotate.cpp is as follows:

```
void Example_MoveRotate::demo2()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

/** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
```

```
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics parameters **/,
                                                6             /*collision level*/,
                            true        /* Whether to allow reading pose, true by default */,
                                        true,       /* Leave the default as true */
                                        1000,       /* Leave the default as 1000 */
                                        result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    // Initialize move properties
    robotService.robotServiceInitGlobalMoveProfile();

    // Set the maximum acceleration of joint move
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30*M_PI/180;
    jointMaxAcc.jointPara[1] = 30*M_PI/180;
    jointMaxAcc.jointPara[2] = 30*M_PI/180;
    jointMaxAcc.jointPara[3] = 30*M_PI/180;
    jointMaxAcc.jointPara[4] = 30*M_PI/180;
    jointMaxAcc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);

    // Set the maximum speed of joint move
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30*M_PI/180;
```

```
    jointMaxVelc.jointPara[1] = 30*M_PI/180;
    jointMaxVelc.jointPara[2] = 30*M_PI/180;
    jointMaxVelc.jointPara[3] = 30*M_PI/180;
    jointMaxVelc.jointPara[4] = 30*M_PI/180;
    jointMaxVelc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

    //Joint angle
    double jointAngle[6] = {};
    jointAngle[0] = 173.108713*M_PI/180;
    jointAngle[1] = -12.075005*M_PI/180;
    jointAngle[2] = -83.663342*M_PI/180;
    jointAngle[3] = -15.641249*M_PI/180;
    jointAngle[4] = -89.140000*M_PI/180;
    jointAngle[5] = -28.328713*M_PI/180;

    // Joint movement to starting position
    robotService.robotServiceJointMove(jointAngle, true);

    // Set the user coordinate system
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool userCoord;
    userCoord.coordType = aubo_robot_namespace::WorldCoordinate;
    userCoord.methods =
aubo_robot_namespace::Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrant
OfXOYPlane;

    userCoord.wayPointArray[0].jointPos[0] = -75.093279*M_PI/180;
    userCoord.wayPointArray[0].jointPos[1] = 28.544643*M_PI/180;
    userCoord.wayPointArray[0].jointPos[2] = -114.313905*M_PI/180;
    userCoord.wayPointArray[0].jointPos[3] = -62.769247*M_PI/180;
    userCoord.wayPointArray[0].jointPos[4] = -87.343517*M_PI/180;
    userCoord.wayPointArray[0].jointPos[5] = -27.888262*M_PI/180;

    userCoord.wayPointArray[1].jointPos[0] = -89.239837*M_PI/180;
    userCoord.wayPointArray[1].jointPos[1] = 23.936171*M_PI/180;
    userCoord.wayPointArray[1].jointPos[2] = -122.299277*M_PI/180;
    userCoord.wayPointArray[1].jointPos[3] = -65.208902*M_PI/180;
    userCoord.wayPointArray[1].jointPos[4] = -85.011123*M_PI/180;
    userCoord.wayPointArray[1].jointPos[5] = -41.87417*M_PI/180;

    userCoord.wayPointArray[2].jointPos[0] = -77.059212*M_PI/180;
    userCoord.wayPointArray[2].jointPos[1] = 35.509518*M_PI/180;
    userCoord.wayPointArray[2].jointPos[2] = -101.108547*M_PI/180;
    userCoord.wayPointArray[2].jointPos[3] = -56.433133*M_PI/180;
```

```
    userCoord.wayPointArray[2].jointPos[4] = -87.006734*M_PI/180;
    userCoord.wayPointArray[2].jointPos[5] = -29.827440*M_PI/180;

    aubo_robot_namespace::ToolInEndDesc toolUserCoord;
    toolUserCoord.toolInEndPosition.x =    -0.177341;
    toolUserCoord.toolInEndPosition.y = 0.002327;
    toolUserCoord.toolInEndPosition.z = 0.146822;
    toolUserCoord.toolInEndOrientation.w = 1;
    toolUserCoord.toolInEndOrientation.x = 0;
    toolUserCoord.toolInEndOrientation.y = 0;
    toolUserCoord.toolInEndOrientation.z = 0;
    userCoord.toolDesc = toolUserCoord;

    // Rotation axis and rotation angle
    double rotateAxis[3] = {1, 0, 0};
    double rotateAngle = 10.0*M_PI/180;

    // Rotational movement: The center of the flange is rotated 10 degrees along the
X+ direction in the user coordinate system, and the current position remains unchanged.
    robotService.robotServiceRotateMove(userCoord, rotateAxis, rotateAngle, true);
}
```

Example 3: Tool end in the tool coordinate system

In this example, the tool end is rotated 10 degrees along the X+ direction in the tool coordinate system, and the current position remains unchanged.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) Joint movement to the starting position (5) Setting tools (6) Setting the end coordinate system and its tool parameters (7) Setting the rotation axis and the rotation angle (8) for the rotational movement.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The demo3() function code in example_moverotate.cpp is as follows:

```
void Example_MoveRotate::demo3()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
```

```
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6              /*collision level*/,
                        true        /* Whether to allow reading pose, true by default */,
                                          true,       /* Leave the default as true */
                                          1000,       /* Leave the default as 1000 */
                                              result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    // Initialize move properties
    robotService.robotServiceInitGlobalMoveProfile();

    // Set the maximum acceleration of joint move
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30*M_PI/180;
    jointMaxAcc.jointPara[1] = 30*M_PI/180;
    jointMaxAcc.jointPara[2] = 30*M_PI/180;
    jointMaxAcc.jointPara[3] = 30*M_PI/180;
    jointMaxAcc.jointPara[4] = 30*M_PI/180;
    jointMaxAcc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);
```

```
// Set the maximum speed of joint move
aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
jointMaxVelc.jointPara[0] = 30*M_PI/180;
jointMaxVelc.jointPara[1] = 30*M_PI/180;
jointMaxVelc.jointPara[2] = 30*M_PI/180;
jointMaxVelc.jointPara[3] = 30*M_PI/180;
jointMaxVelc.jointPara[4] = 30*M_PI/180;
jointMaxVelc.jointPara[5] = 30*M_PI/180;
robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

//Starting waypoint
double initAngle[6] = {};
initAngle[0] = 173.108713*M_PI/180;
initAngle[1] = -12.075005*M_PI/180;
initAngle[2] = -83.663342*M_PI/180;
initAngle[3] = -15.641249*M_PI/180;
initAngle[4] = -89.140000*M_PI/180;
initAngle[5] = -28.328713*M_PI/180;

// Joint movement to starting waypoint
robotService.robotServiceJointMove(initAngle, true);

// setup tool
aubo_robot_namespace::ToolInEndDesc toolEnd;
toolEnd.toolInEndPosition.x =    -0.177341;
toolEnd.toolInEndPosition.y = 0.002327;
toolEnd.toolInEndPosition.z = 0.146822;
toolEnd.toolInEndOrientation.w = 1;
toolEnd.toolInEndOrientation.x = 0;
toolEnd.toolInEndOrientation.y = 0;
toolEnd.toolInEndOrientation.z = 0;
robotService.robotServiceSetToolKinematicsParam(toolEnd);

// Set tool coordinate system
aubo_robot_namespace::CoordCalibrateByJointAngleAndTool endCoord;
endCoord.coordType = aubo_robot_namespace::EndCoordinate;
endCoord.toolDesc = toolEnd;

// Rotation axis and rotation angle
double rotateAxis[3] = {1, 0, 0};
double rotateAngle = 10.0*M_PI/180;

// Rotation movement: the tool end rotates 10 degrees in the x+ direction under the
tool coordinate system, and the current position remains unchanged
```

```
    robotService.robotServiceRotateMove(endCoord, rotateAxis, rotateAngle, true);


}
```

Example 4: Tool end in base coordinate system

This example shows that the tool end rotates 10 degrees along the x+ direction in the base coordinate system, and the current position remains unchanged.

The main process of this procedure is: (1) robot login (2) robot initialization (3) joint movement to the starting position (5) setting tool (6) setting base coordinate system (7) setting rotation axis and rotation angle (8) rotation movement.

Note: change the IP address (server_host) in the following code to the IP address of the corresponding server.

The demo4() function code in example_moverotate.cpp is as follows:

```
void Example_MoveRotate::demo4()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6              /*collision level*/,
                               true         /* Whether to allow reading pose, true by default */,
                                                    true,       /* Leave the default as true */
```

```
                                                1000,       /* Leave the default as 1000 */
                                                    result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
        {
std::cerr << " Robot initialized successfully." << std::endl;
        }
        else
        {
              std::cerr << " Robot initialization failed." << std::endl;
        }

        // Initialize move properties
        robotService.robotServiceInitGlobalMoveProfile();

        // Set the maximum acceleration of joint move
        aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
        jointMaxAcc.jointPara[0] = 30*M_PI/180;
        jointMaxAcc.jointPara[1] = 30*M_PI/180;
        jointMaxAcc.jointPara[2] = 30*M_PI/180;
        jointMaxAcc.jointPara[3] = 30*M_PI/180;
        jointMaxAcc.jointPara[4] = 30*M_PI/180;
        jointMaxAcc.jointPara[5] = 30*M_PI/180;
        robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);

        // Set the maximum speed of joint move
        aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
        jointMaxVelc.jointPara[0] = 30*M_PI/180;
        jointMaxVelc.jointPara[1] = 30*M_PI/180;
        jointMaxVelc.jointPara[2] = 30*M_PI/180;
        jointMaxVelc.jointPara[3] = 30*M_PI/180;
        jointMaxVelc.jointPara[4] = 30*M_PI/180;
        jointMaxVelc.jointPara[5] = 30*M_PI/180;
        robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

        //Starting waypoint
        double initAngle[6] = {};
        initAngle[0] = 173.108713*M_PI/180;
        initAngle[1] = -12.075005*M_PI/180;
        initAngle[2] = -83.663342*M_PI/180;
        initAngle[3] = -15.641249*M_PI/180;
        initAngle[4] = -89.140000*M_PI/180;
        initAngle[5] = -28.328713*M_PI/180;
```

```
    // Joint movement to starting waypoint
    robotService.robotServiceJointMove(initAngle, true);

    // setup tool
    aubo_robot_namespace::ToolInEndDesc toolEnd;
    toolEnd.toolInEndPosition.x =    -0.177341;
    toolEnd.toolInEndPosition.y = 0.002327;
    toolEnd.toolInEndPosition.z = 0.146822;
    toolEnd.toolInEndOrientation.w = 1;
    toolEnd.toolInEndOrientation.x = 0;
    toolEnd.toolInEndOrientation.y = 0;
    toolEnd.toolInEndOrientation.z = 0;
    robotService.robotServiceSetToolKinematicsParam(toolEnd);

    // Set the base coordinate system
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool baseCoord;
    baseCoord.coordType = aubo_robot_namespace::BaseCoordinate;

    // Rotation axis and rotation angle
    double rotateAxis[3] = {1, 0, 0};
    double rotateAngle = 10.0*M_PI/180;

    // Rotation movement: the tool end rotates 10 degrees along the x+ direction in the
base coordinate system, and the current position remains unchanged
robotService.robotServiceRotateMove(baseCoord, rotateAxis, rotateAngle, true);

}
```

Example 5: Tool end in user coordinate system

In this example, the tool end is rotated 10 degrees along the X+ direction in the user coordinate system, and the current position remains unchanged.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) Joint movement to the starting position (5) Setting tools (6) Setting the user coordinate system and its tool parameters (7) Setting the rotation axis and the rotation angle (8) for the rotational movement.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The demo5() function code in example_moverotate.cpp is as follows:

```
void Example_MoveRotate::demo5()
{
    ServiceInterface robotService;
```

```
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6              /*collision level*/,
                            true        /* Whether to allow reading pose, true by default */,
                                                true,       /* Leave the default as true */
                                               1000,       /* Leave the default as 1000 */
                                                    result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    // Initialize move properties
    robotService.robotServiceInitGlobalMoveProfile();

    // Set the maximum acceleration of joint move
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30*M_PI/180;
    jointMaxAcc.jointPara[1] = 30*M_PI/180;
```

```
    jointMaxAcc.jointPara[2] = 30*M_PI/180;
    jointMaxAcc.jointPara[3] = 30*M_PI/180;
    jointMaxAcc.jointPara[4] = 30*M_PI/180;
    jointMaxAcc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);

    // Set the maximum speed of joint move
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30*M_PI/180;
    jointMaxVelc.jointPara[1] = 30*M_PI/180;
    jointMaxVelc.jointPara[2] = 30*M_PI/180;
    jointMaxVelc.jointPara[3] = 30*M_PI/180;
    jointMaxVelc.jointPara[4] = 30*M_PI/180;
    jointMaxVelc.jointPara[5] = 30*M_PI/180;
    robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

    //Starting waypoint
    double initAngle[6] = {};
    initAngle[0] = 173.108713*M_PI/180;
    initAngle[1] = -12.075005*M_PI/180;
    initAngle[2] = -83.663342*M_PI/180;
    initAngle[3] = -15.641249*M_PI/180;
    initAngle[4] = -89.140000*M_PI/180;
    initAngle[5] = -28.328713*M_PI/180;

    // Joint movement to starting waypoint
    robotService.robotServiceJointMove(initAngle, true);

    // Setup tool
    aubo_robot_namespace::ToolInEndDesc toolEnd;
    toolEnd.toolInEndPosition.x =    -0.177341;
    toolEnd.toolInEndPosition.y = 0.002327;
    toolEnd.toolInEndPosition.z = 0.146822;
    toolEnd.toolInEndOrientation.w = 1;
    toolEnd.toolInEndOrientation.x = 0;
    toolEnd.toolInEndOrientation.y = 0;
    toolEnd.toolInEndOrientation.z = 0;
    robotService.robotServiceSetToolKinematicsParam(toolEnd);

    // Set user coordinate system
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool userCoord;
    userCoord.coordType = aubo_robot_namespace::WorldCoordinate;
    userCoord.methods =
aubo_robot_namespace::Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrant
```

```
OfXOYPlane;

    userCoord.wayPointArray[0].jointPos[0] = -75.093279*M_PI/180;
    userCoord.wayPointArray[0].jointPos[1] = 28.544643*M_PI/180;
    userCoord.wayPointArray[0].jointPos[2] = -114.313905*M_PI/180;
    userCoord.wayPointArray[0].jointPos[3] = -62.769247*M_PI/180;
    userCoord.wayPointArray[0].jointPos[4] = -87.343517*M_PI/180;
    userCoord.wayPointArray[0].jointPos[5] = -27.888262*M_PI/180;

    userCoord.wayPointArray[1].jointPos[0] = -89.239837*M_PI/180;
    userCoord.wayPointArray[1].jointPos[1] = 23.936171*M_PI/180;
    userCoord.wayPointArray[1].jointPos[2] = -122.299277*M_PI/180;
    userCoord.wayPointArray[1].jointPos[3] = -65.208902*M_PI/180;
    userCoord.wayPointArray[1].jointPos[4] = -85.011123*M_PI/180;
    userCoord.wayPointArray[1].jointPos[5] = -41.87417*M_PI/180;

    userCoord.wayPointArray[2].jointPos[0] = -77.059212*M_PI/180;
    userCoord.wayPointArray[2].jointPos[1] = 35.509518*M_PI/180;
    userCoord.wayPointArray[2].jointPos[2] = -101.108547*M_PI/180;
    userCoord.wayPointArray[2].jointPos[3] = -56.433133*M_PI/180;
    userCoord.wayPointArray[2].jointPos[4] = -87.006734*M_PI/180;
    userCoord.wayPointArray[2].jointPos[5] = -29.827440*M_PI/180;

    userCoord.toolDesc = toolEnd;

    // Rotation axis and rotation angle
    double rotateAxis[3] = {1, 0, 0};
    double rotateAngle = 10.0*M_PI/180;

    // Rotation movement: the tool end rotates 10 degrees in the x+ direction in the
user coordinate system, and the current position remains unchanged
    robotService.robotServiceRotateMove(userCoord, rotateAxis, rotateAngle, true);

}
```

4.13 Move Track

4.13.1 robotServiceTrackMove Function

Example 1: Circular move

This example is a circular move in which a manipulator makes a trajectory motion.

The main process of this procedure is: (1) robot login (2) robot initialization (3) joint

movement to the initial position (4) emptying the global waypoint velocity (5) adding waypoints (6) setting the number of circles of circular move (7) doing circular motion by using the move track track.

Note: change the IP address (server_host) in the following code to the IP address of the corresponding server.

The demo1() function code in example_movetrack.cpp is as follows:

```
void Example_MoveTrack::demo1()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6              /*collision level*/,
                        true        /* Whether to allow reading pose, true by default */,
                                    true,       /* Leave the default as true */
                                    1000,       /* Leave the default as 1000 */
                                        result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
```

```
        std::cerr << " Robot initialization failed." << std::endl;
}

// Initialize move properties
robotService.robotServiceInitGlobalMoveProfile();

// Set the maximum acceleration for TCP move
double lineMaxAcc = 0.2;
robotService.robotServiceSetGlobalMoveEndMaxLineAcc(lineMaxAcc);

// Set the maximum speed for TCP move
double lineMaxVelc = 0.2;
robotService.robotServiceSetGlobalMoveEndMaxLineVelc(lineMaxVelc);

double jointAngle1[6] = {};
jointAngle1[0] = 60.443151*M_PI/180;
jointAngle1[1] = 42.275463*M_PI/180;
jointAngle1[2] = -97.679737*M_PI/180;
jointAngle1[3] = -49.990510*M_PI/180;
jointAngle1[4] = -90.007372*M_PI/180;
jointAngle1[5] = 62.567046*M_PI/180;

double jointAngle2[6] = {};
jointAngle2[0] = 83.411541*M_PI/180;
jointAngle2[1] = 39.625360*M_PI/180;
jointAngle2[2] = -103.796807*M_PI/180;
jointAngle2[3] = -53.491856*M_PI/180;
jointAngle2[4] = -90.021641*M_PI/180;
jointAngle2[5] = 85.530279*M_PI/180;

double jointAngle3[6] = {};
jointAngle3[0] = 81.206455*M_PI/180;
jointAngle3[1] = 28.381980*M_PI/180;
jointAngle3[2] = -129.233955*M_PI/180;
jointAngle3[3] = -67.700289*M_PI/180;
jointAngle3[4] = -90.019516*M_PI/180;
jointAngle3[5] = 83.325883*M_PI/180;

//Joint Move
robotService.robotServiceJointMove(jointAngle1, true);

// Empty the global waypoint vector
robotService.robotServiceClearGlobalWayPointVector();
```

```
    // Add waypoint
    robotService.robotServiceAddGlobalWayPoint(jointAngle1);
    // Add waypoint
    robotService.robotServiceAddGlobalWayPoint(jointAngle2);
    // Add waypoint
    robotService.robotServiceAddGlobalWayPoint(jointAngle3);

    // Set the number of loop motion circles
    robotService.robotServiceSetGlobalCircularLoopTimes(0);

    // arc movement
    robotService.robotServiceTrackMove(aubo_robot_namespace::ARC_CIR, true);
}
```

Example 2: Arc Move

This example is the arc move of the robotic arm doing the trajectory motion.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) Joint movement to the initial position (4) Clearing the global waypoint vector (5) Adding waypoints (6) Setting the number of loops of circular motion For 0 (7), do the arc move of the trajectory motion.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The demo2() function code in example_movetrack.cpp is as follows:

```
void Example_MoveTrack::demo2()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
```

```
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                                6             /*collision level*/,
                            true        /* Whether to allow reading pose, true by default */,
                                            true,       /* Leave the default as true */
                                            1000,       /* Leave the default as 1000 */
                                                result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    // Initialize move properties
    robotService.robotServiceInitGlobalMoveProfile();

    // Set the maximum acceleration for TCP move
    double lineMaxAcc = 0.2;
    robotService.robotServiceSetGlobalMoveEndMaxLineAcc(lineMaxAcc);

    // Set the maximum speed for TCP move
    double lineMaxVelc = 0.2;
    robotService.robotServiceSetGlobalMoveEndMaxLineVelc(lineMaxVelc);

    double jointAngle1[6] = {};
    jointAngle1[0] = 60.443151*M_PI/180;
    jointAngle1[1] = 42.275463*M_PI/180;
    jointAngle1[2] = -97.679737*M_PI/180;
    jointAngle1[3] = -49.990510*M_PI/180;
    jointAngle1[4] = -90.007372*M_PI/180;
    jointAngle1[5] = 62.567046*M_PI/180;

    double jointAngle2[6] = {};
    jointAngle2[0] = 83.411541*M_PI/180;
    jointAngle2[1] = 39.625360*M_PI/180;
```

```
    jointAngle2[2] = -103.796807*M_PI/180;
    jointAngle2[3] = -53.491856*M_PI/180;
    jointAngle2[4] = -90.021641*M_PI/180;
    jointAngle2[5] = 85.530279*M_PI/180;

    double jointAngle3[6] = {};
    jointAngle3[0] = 81.206455*M_PI/180;
    jointAngle3[1] = 28.381980*M_PI/180;
    jointAngle3[2] = -129.233955*M_PI/180;
    jointAngle3[3] = -67.700289*M_PI/180;
    jointAngle3[4] = -90.019516*M_PI/180;
    jointAngle3[5] = 83.325883*M_PI/180;

    //Joint Move
    robotService.robotServiceJointMove(jointAngle1, true);

    // Empty global waypoint container
    robotService.robotServiceClearGlobalWayPointVector();

    //Add waypoint
    robotService.robotServiceAddGlobalWayPoint(jointAngle1);
    // Add waypoint
    robotService.robotServiceAddGlobalWayPoint(jointAngle2);
    // Add waypoint
    robotService.robotServiceAddGlobalWayPoint(jointAngle3);

    // Set the number of circle move lops
    robotService.robotServiceSetGlobalCircularLoopTimes(3);

    // circular move
    robotService.robotServiceTrackMove(aubo_robot_namespace::ARC_CIR, true);
}
```

Example 3：MOVEP

This example is the MOVEP move of the robotic arm doing trajectory motion.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) Joint movement to the initial position (4) Clearing the global waypoint container (5) Adding waypoints (6) Setting the blend radius (7) Do the MOVEP motion of trajectory motion.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The demo3() function code in example_movetrack.cpp is as follows:

```
void Example_MoveTrack::demo3()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6              /*collision level*/,
                        true        /* Whether to allow reading pose, true by default */,
                                            true,       /* Leave the default as true */
                                            1000,       /* Leave the default as 1000 */
                                                result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    // Initialize move properties
    robotService.robotServiceInitGlobalMoveProfile();

    // Set the maximum acceleration for TCP move
```

```
double lineMaxAcc = 0.2;
robotService.robotServiceSetGlobalMoveEndMaxLineAcc(lineMaxAcc);

// Set the maximum speed for TCP move
double lineMaxVelc = 0.2;
robotService.robotServiceSetGlobalMoveEndMaxLineVelc(lineMaxVelc);

double jointAngle1[6] = {};
jointAngle1[0] = 60.443151*M_PI/180;
jointAngle1[1] = 42.275463*M_PI/180;
jointAngle1[2] = -97.679737*M_PI/180;
jointAngle1[3] = -49.990510*M_PI/180;
jointAngle1[4] = -90.007372*M_PI/180;
jointAngle1[5] = 62.567046*M_PI/180;

double jointAngle2[6] = {};
jointAngle2[0] = 83.411541*M_PI/180;
jointAngle2[1] = 39.625360*M_PI/180;
jointAngle2[2] = -103.796807*M_PI/180;
jointAngle2[3] = -53.491856*M_PI/180;
jointAngle2[4] = -90.021641*M_PI/180;
jointAngle2[5] = 85.530279*M_PI/180;

double jointAngle3[6] = {};
jointAngle3[0] = 81.206455*M_PI/180;
jointAngle3[1] = 28.381980*M_PI/180;
jointAngle3[2] = -129.233955*M_PI/180;
jointAngle3[3] = -67.700289*M_PI/180;
jointAngle3[4] = -90.019516*M_PI/180;
jointAngle3[5] = 83.325883*M_PI/180;

//Joint Move
robotService.robotServiceJointMove(jointAngle1, true);

// Empty the global waypoint container
robotService.robotServiceClearGlobalWayPointVector();

//Add waypoint
robotService.robotServiceAddGlobalWayPoint(jointAngle1);
// Add waypoint
robotService.robotServiceAddGlobalWayPoint(jointAngle2);
// Add waypoint
robotService.robotServiceAddGlobalWayPoint(jointAngle3);
```

```
    // Set blend radius
    float blendradius = 0.05;
    std::cout << "set blendradius ret is " <<
robotService.robotServiceSetGlobalBlendRadius(blendradius) <<
    std::endl;

    //MoveP Move

robotService.robotServiceTrackMove(aubo_robot_namespace::CARTESIAN_MOVEP,
true);
}
```

4.14 Teaching movement

In this example, the flange center is used for joint teaching, position teaching and orientation teaching in the base coordinate system.

The main flow of this program is: (1) Robot login (2) Robot initialization (3) Initialization of move properties (4) Setting the maximum speed and maximum acceleration (5) Setting the base coordinate system (6) JOINT1 joint teaching (7)) MOV_X position teaching (8) ROT_Z posture teaching.

Note: Modify the IP address (SERVER_HOST) in the code below to the IP address of the corresponding server.

The demo () function code in example_teachmove.cpp is as follows:

```
#include "example_teachmove.h"
#include "AuboRobotMetaType.h"
#include "serviceinterface.h"

#include <unistd.h>
#include <math.h>
#include <string.h>
#include <stdio.h>
#include <sstream>
#include <fstream>

#define SERVER_HOST "192.168.221.13"
#define SERVER_PORT 8899

Example_TeachMove::Example_TeachMove()
{

}
```

```
void Example_TeachMove::demo()
{
    ServiceInterface robotService;
    int ret = aubo_robot_namespace::InterfaceCallSuccCode;

    /** Call interface: login ***/
    ret = robotService.robotServiceLogin(SERVER_HOST, SERVER_PORT, "aubo",
"123456");
    if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "login successful" << std::endl;
    }
    else
    {
        std::cerr << "login failed" << std::endl;
    }

    /** If real manipulator is connected, the manipulator needs to be initialized    **/
    aubo_robot_namespace::ROBOT_SERVICE_STATE result;

    //Tool dynamics parameters
    aubo_robot_namespace::ToolDynamicsParam toolDynamicsParam;
    memset(&toolDynamicsParam, 0, sizeof(toolDynamicsParam));
    ret = robotService.rootServiceRobotStartup(toolDynamicsParam/** Tool dynamics
parameters **/,
                                               6              /*collision level*/,
                        true        /* Whether to allow reading pose, true by default */,
                                            true,       /* Leave the default as true */
                                            1000,       /* Leave the default as 1000 */
                                                result); /* Robot initialization */

if(ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
std::cerr << " Robot initialized successfully." << std::endl;
    }
    else
    {
        std::cerr << " Robot initialization failed." << std::endl;
    }

    // Initialize move properties
    robotService.robotServiceInitGlobalMoveProfile();
```

```
// Set the maximum acceleration of joint move
aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
jointMaxAcc.jointPara[0] = 30*M_PI/180;
jointMaxAcc.jointPara[1] = 30*M_PI/180;
jointMaxAcc.jointPara[2] = 30*M_PI/180;
jointMaxAcc.jointPara[3] = 30*M_PI/180;
jointMaxAcc.jointPara[4] = 30*M_PI/180;
jointMaxAcc.jointPara[5] = 30*M_PI/180;
robotService.robotServiceSetGlobalMoveJointMaxAcc(jointMaxAcc);

// Set the maximum move of joint move
aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
jointMaxVelc.jointPara[0] = 15*M_PI/180;
jointMaxVelc.jointPara[1] = 15*M_PI/180;
jointMaxVelc.jointPara[2] = 15*M_PI/180;
jointMaxVelc.jointPara[3] = 15*M_PI/180;
jointMaxVelc.jointPara[4] = 15*M_PI/180;
jointMaxVelc.jointPara[5] = 15*M_PI/180;
robotService.robotServiceSetGlobalMoveJointMaxVelc(jointMaxVelc);

// Set the teaching coordinate system as the base coordinate system
aubo_robot_namespace::CoordCalibrateByJointAngleAndTool baseCoord;
robotService.robotServiceSetTeachCoordinateSystem(baseCoord);

// Set teaching mode - joint teaching
aubo_robot_namespace::teach_mode teachmode1;
teachmode1 = aubo_robot_namespace::JOINT1;

// start teaching
robotService.robotServiceTeachStart(teachmode1, true);
sleep(2);

//Stop teaching
robotService.robotServiceTeachStop();

// Set teaching mode - position teach
aubo_robot_namespace::teach_mode teachmode2;
teachmode2 = aubo_robot_namespace::MOV_X;

// start teaching
robotService.robotServiceTeachStart(teachmode2, true);
sleep(5);
// Stop teaching
robotService.robotServiceTeachStop();
```

```
    // Set the teaching mode - posture teaching
    aubo_robot_namespace::teach_mode teachmode3;
    teachmode3 = aubo_robot_namespace::ROT_Z;

    //Start teaching
    robotService.robotServiceTeachStart(teachmode3, true);
    sleep(5);
    //Stop teaching
    robotService.robotServiceTeachStop();
}
```

4.15 Print waypoint information, joint status information, event information, diagnostic information

This program includes: function printWaypoint() for printing waypoint information, printJointStatus() for printing joint status information, printEventInfo() for printing event information, printRobotDiagnosis() for printing diagnostic information, and initJointAngleArray() for initializing joint angle array.

Note: Some use cases in the manual may use util.h to print waypoint info or print joint state info or print event info etc.

The code of util.h is as follows:

```
#ifndef UTIL_H
#define UTIL_H

#include "AuboRobotMetaType.h"
#include "serviceinterface.h"

class Util
{
public:
    Util();

public:
    /** Print the waypoints information **/
    static void printWaypoint(aubo_robot_namespace::wayPoint_S &wayPoint);

    /** Print the joint status information **/
    static void printJointStatus(const aubo_robot_namespace::JointStatus *jointStatus,
int len);
```

```
    /** Print the event info **/
    static void printEventInfo(const aubo_robot_namespace::RobotEventInfo
&eventInfo);

    /** Print diagnostic information **/
    static void printRobotDiagnosis(const aubo_robot_namespace::RobotDiagnosis
&robotDiagnosis);

    static void initJointAngleArray(double *array, double joint0,double joint1,double
joint2,double joint3,double joint4,double joint5);

};

#endif // UTIL_H
```

util.cpp 的代码如下:

```
#include "util.h"

#include "AuboRobotMetaType.h"
#include "serviceinterface.h"

#include <unistd.h>
#include <math.h>
#include <string.h>
#include <stdio.h>
#include <sstream>
#include <fstream>

Util::Util()
{

}

// Print the waypoints information
void Util::printWaypoint(aubo_robot_namespace::wayPoint_S &wayPoint)
{
    std::cout<<std::endl<<"start----------waypoint info----------"<<std::endl;
    //position info
    std::cout<<"position info: ";
    std::cout<<"x:"<<wayPoint.cartPos.position.x<<"    ";
    std::cout<<"y:"<<wayPoint.cartPos.position.y<<"    ";
    std::cout<<"z:"<<wayPoint.cartPos.position.z<<std::endl;

    //姿态信息
```

```
    std::cout<<"pose info: ";
    std::cout<<"w:"<<wayPoint.orientation.w<<"    ";
    std::cout<<"x:"<<wayPoint.orientation.x<<"    ";
    std::cout<<"y:"<<wayPoint.orientation.y<<"    ";
    std::cout<<"z:"<<wayPoint.orientation.z<<std::endl;

    ServiceInterface robotService;
    aubo_robot_namespace::Rpy tempRpy;
    robotService.quaternionToRPY(wayPoint.orientation,tempRpy);
    std::cout<<"RX:"<<tempRpy.rx*180/M_PI<<"
RY:"<<tempRpy.ry*180/M_PI<<"      RZ:"<<tempRpy.rz*180/M_PI<<std::endl;

    //Joint info
    std::cout<<"Joint info: "<<std::endl;
    for(int i=0;i<aubo_robot_namespace::ARM_DOF;i++)
    {
        std::cout<<"joint"<<i+1<<": "<<wayPoint.jointpos[i]<<" ~
"<<wayPoint.jointpos[i]*180.0/M_PI<<std::endl;
    }
}


// Print joint status information
void Util::printJointStatus(const aubo_robot_namespace::JointStatus *jointStatus, int
len)
{
    std::cout<<std::endl<<"start---------- Print joint status -------" << std::endl;

    for(int i=0; i<len; i++)
    {
        std::cout<<"Joint ID:"      <<i<<"    " ;
        std::cout<<"Current:"         <<jointStatus[i].jointCurrentI<<" ";
        std::cout<<"speed:"         <<jointStatus[i].jointSpeedMoto<<" ";
        std::cout<<"angle:"      <<jointStatus[i].jointPosJ<<" "<<" ~
"<<jointStatus[i].jointPosJ*180.0/M_PI;
        std::cout<<"voltage      :"    <<jointStatus[i].jointCurVol<<" ";
        std::cout<<"temperature      :"    <<jointStatus[i].jointCurTemp<<" ";
        std::cout<<"target current:"    <<jointStatus[i].jointTagCurrentI<<" ";
        std::cout<<"target motor speed:" <<jointStatus[i].jointTagSpeedMoto<<" ";
        std::cout<<"target joint angle :"    <<jointStatus[i].jointTagPosJ<<" ";
        std::cout<<"joint error      :"    <<jointStatus[i].jointErrorNum <<std::endl;
    }
    std::cout<<std::endl;
}
```

```
// print event information
void Util::printEventInfo(const aubo_robot_namespace::RobotEventInfo &eventInfo)
{
    std::cout<<"Event type:"<<eventInfo.eventType <<"
code:"<<eventInfo.eventCode<<"    content:"<<eventInfo.eventContent<<std::endl;
}

// print diagnostic information
void Util::printRobotDiagnosis(const aubo_robot_namespace::RobotDiagnosis
&robotDiagnosis)
{
    std::cout<<std::endl<<"start---------- Arm Diagnostic Information -------" <<
std::endl;

    std::cout<<std::endl<<"     "<<"CAN communication
status:"<<(int)robotDiagnosis.armCanbusStatus;
    std::cout<<std::endl<<"     "<<"power supply
current:"<<robotDiagnosis.armPowerCurrent;
    std::cout<<std::endl<<"     "<<"power supply
voltage:"<<robotDiagnosis.armPowerVoltage;

    (robotDiagnosis.armPowerStatus)? std::cout<<std::endl<<"     "<<"48V Power
supply status: On ":std::cout<<std::endl<<"     "<<"48V Power supply State: Off ";

    std::cout<<std::endl<<"     "<<"control box
temperature:"<<(int)robotDiagnosis.contorllerTemp;
    std::cout<<std::endl<<"     "<<"control box
humidity:"<<(int)robotDiagnosis.contorllerHumidity;
    std::cout<<std::endl<<"     "<<" Remote shutdown
signal:"<<robotDiagnosis.remoteHalt;
    std::cout<<std::endl<<"     "<<" Soft emergency
stop :"<<robotDiagnosis.softEmergency;
    std::cout<<std::endl<<"     "<<" Remote emergency stop
signal:"<<robotDiagnosis.remoteEmergency;
    std::cout<<std::endl<<"     "<<" collision detection
bit:"<<robotDiagnosis.robotCollision;
    std::cout<<std::endl<<"     "<<" Enter the force control mode
flag:"<<robotDiagnosis.forceControlMode;
    std::cout<<std::endl<<"     "<<" brake status:"<<robotDiagnosis.brakeStuats;
    std::cout<<std::endl<<"     "<<" end speed:"<<robotDiagnosis.robotEndSpeed;
    std::cout<<std::endl<<"     "<<" Maximum
acceleration:"<<robotDiagnosis.robotMaxAcc;
```

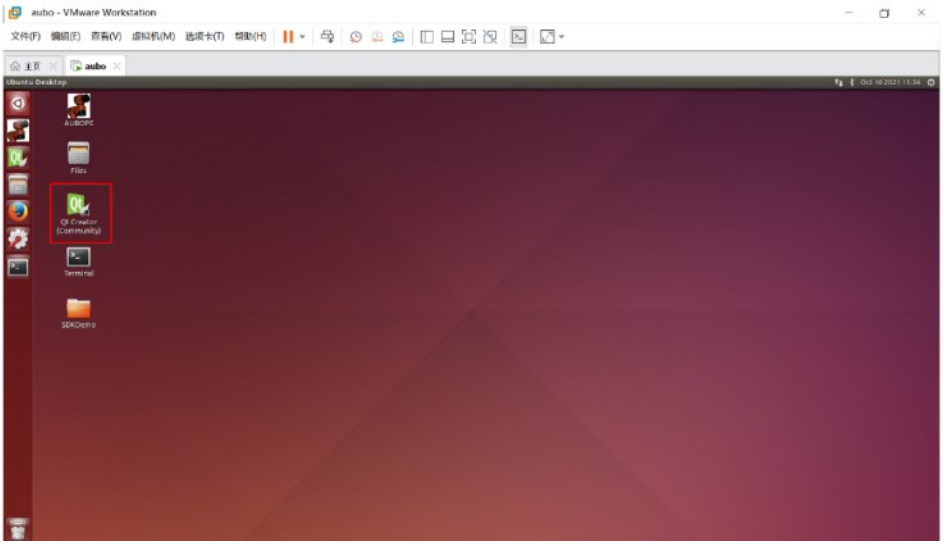
```
    std::cout<<std::endl<<"    "<<"upper computer software
status:"<<robotDiagnosis.orpeStatus;
    std::cout<<std::endl<<"    "<<" Pose read enable
bit:"<<robotDiagnosis.enableReadPose;
    std::cout<<std::endl<<"    "<<" Installation location
status:"<<robotDiagnosis.robotMountingPoseChanged;
    std::cout<<std::endl<<"    "<<" Magnetic encoder error
status:"<<robotDiagnosis.encoderErrorStatus;
    std::cout<<std::endl<<"    "<<" Static Collision Detection
Switch:"<<robotDiagnosis.staticCollisionDetect;
    std::cout<<std::endl<<"    "<<" Joint Collision
Detection:"<<robotDiagnosis.jointCollisionDetect;
    std::cout<<std::endl<<"    "<<" Photoelectric encoder inconsistent
error:"<<robotDiagnosis.encoderLinesError;
    std::cout<<std::endl<<"    "<<" joint error
status:"<<robotDiagnosis.jointErrorStatus;
    std::cout<<std::endl<<"    "<<" Singularity Overspeed
Warning:"<<robotDiagnosis.singularityOverSpeedAlarm;
    std::cout<<std::endl<<"    "<<" Current error
warning:"<<robotDiagnosis.robotCurrentAlarm;
    std::cout<<std::endl<<"    "<<"tool error:"<<(int)robotDiagnosis.toolIoError;
    std::cout<<std::endl<<"    "<<" The installation position is
wrong:"<<robotDiagnosis.robotMountingPoseWarning;
    std::cout<<std::endl<<"    "<<"mac Buffer
length:"<<robotDiagnosis.macTargetPosBufferSize;
    std::cout<<std::endl<<"    "<<"macBuffer valid data
length:"<<robotDiagnosis.macTargetPosDataSize;
    std::cout<<std::endl<<"    "<<"macdata
interruption:"<<robotDiagnosis.macDataInterruptWarning;

    std::cout<<std::endl<<"----------------------------------end."<<std::endl;
}

void Util::initJointAngleArray(double *array, double joint0, double joint1, double
joint2, double joint3, double joint4, double joint5)
{
    array[0] = joint0;
    array[1] = joint1;
    array[2] = joint2;
    array[3] = joint3;
    array[4] = joint4;
    array[5] = joint5;
}
```


5 Environment configuration instructions

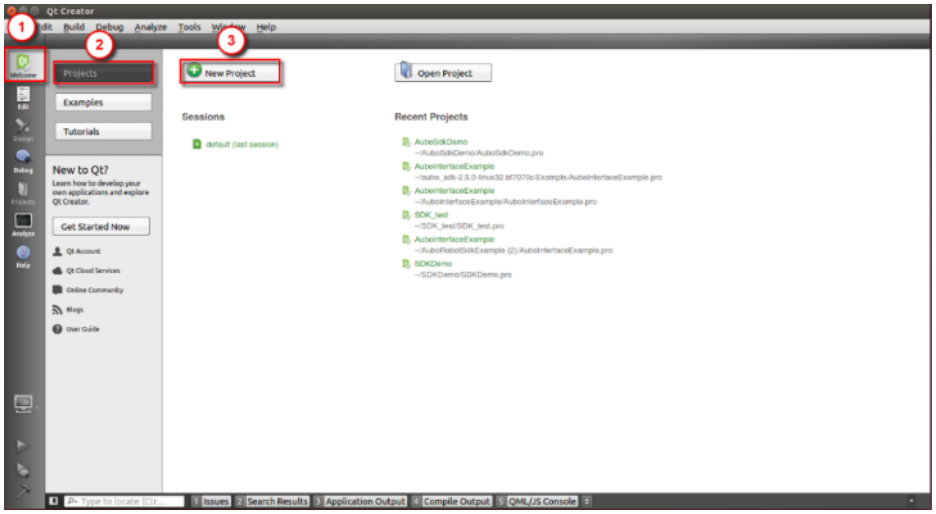
Note: The environment configuration instructions described in this manual are performed in a 32-bit AUBO virtual machine. Users can also refer to this instruction for secondary development of the SDK under 32-bit or 64-bit Linux systems.

5.1 Create your own program

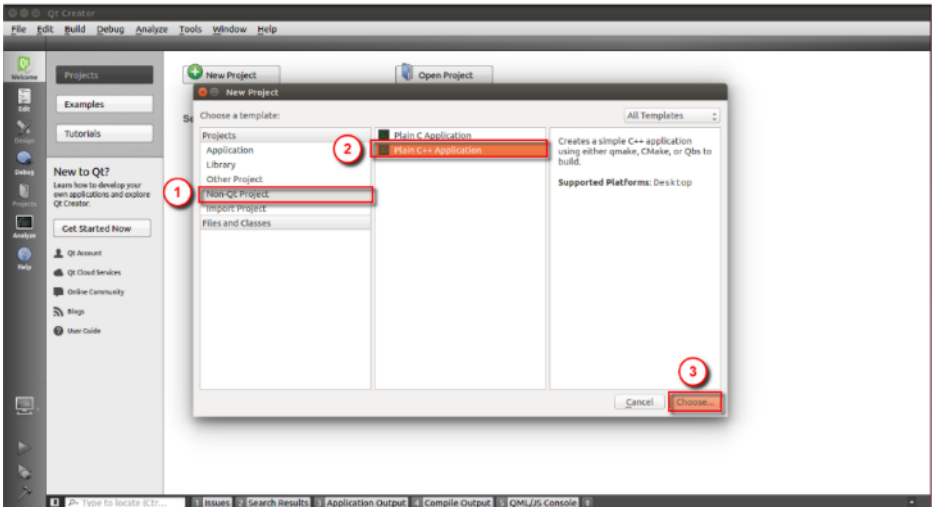
1. Open the AUBO virtual machine. On the desktop, open Qt Creator.



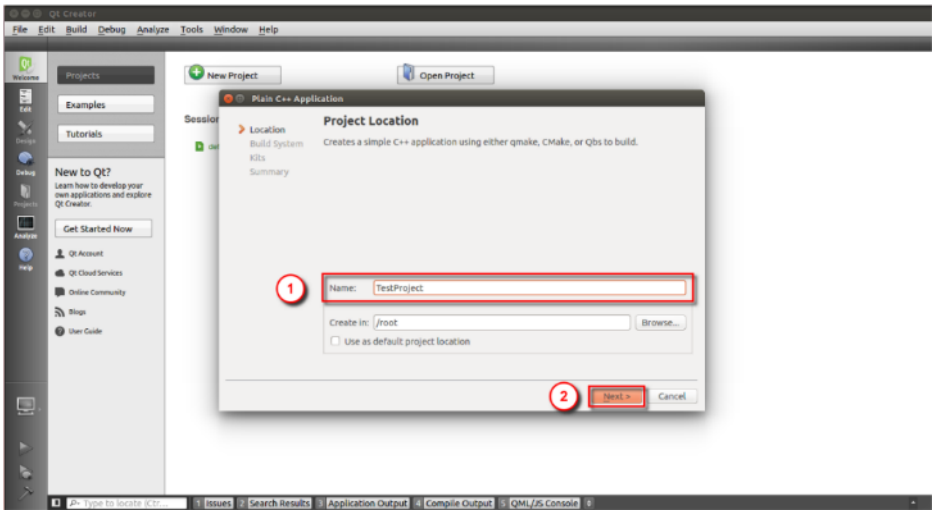
2. In the "Welcome" - "Projects" column, click "New Project". Create a new project.



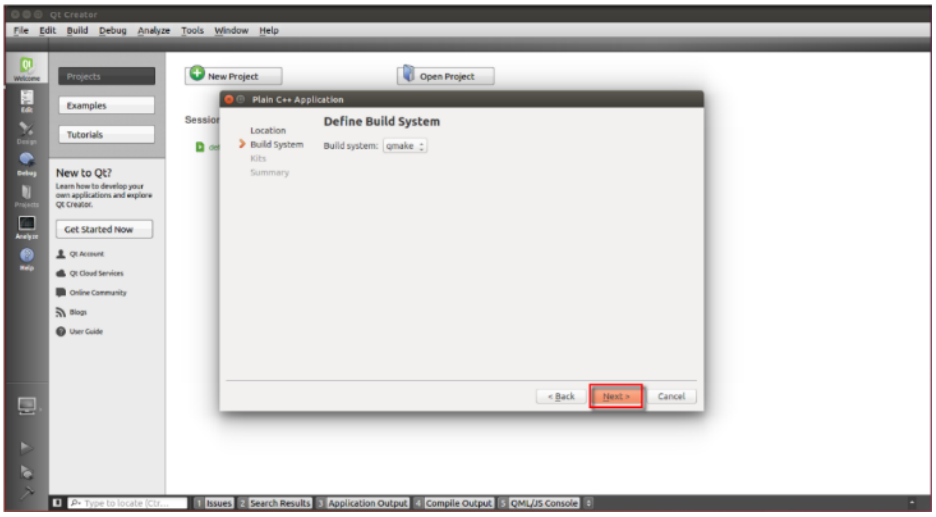
3. Select "Non-Qt Project" - "Plain C++ Application" and click "Choose...".



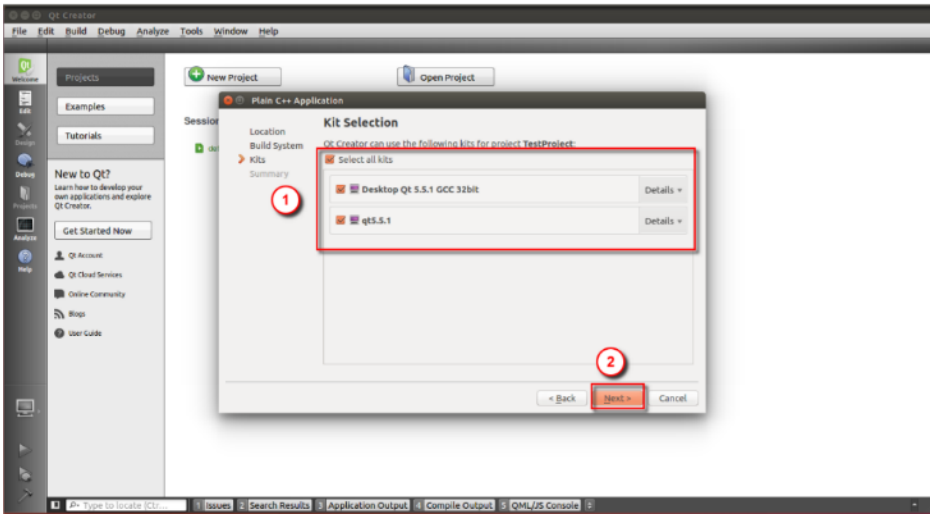
4. Name the project "TestProject" and click "Next".



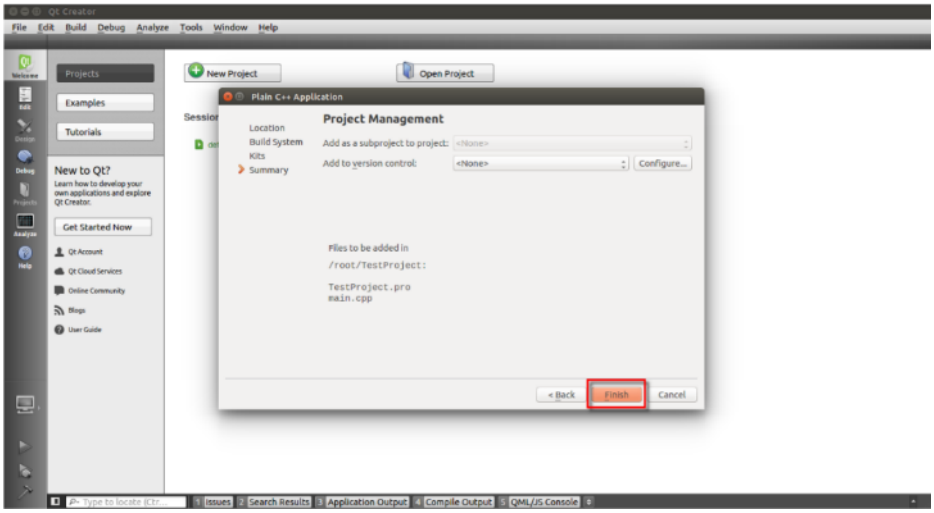
5. Click "Next".



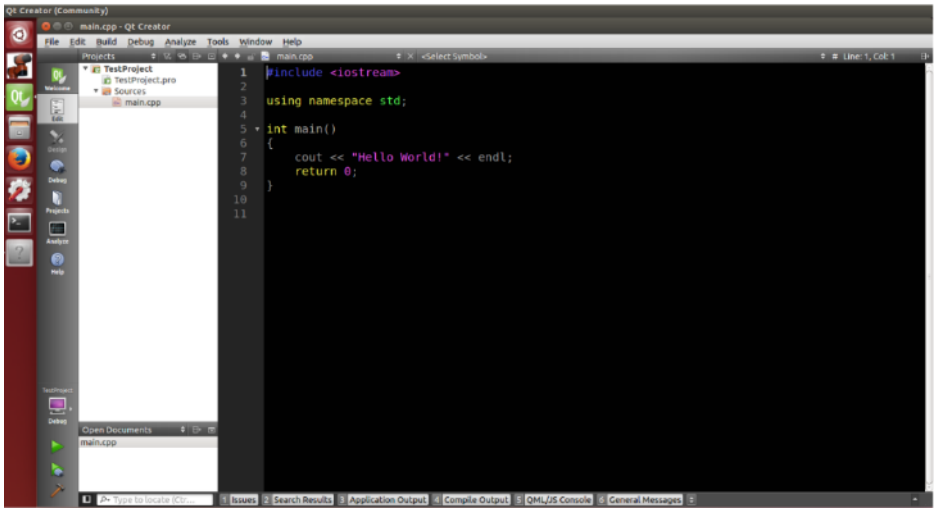
6. Click "Select all kits" and click "Next".



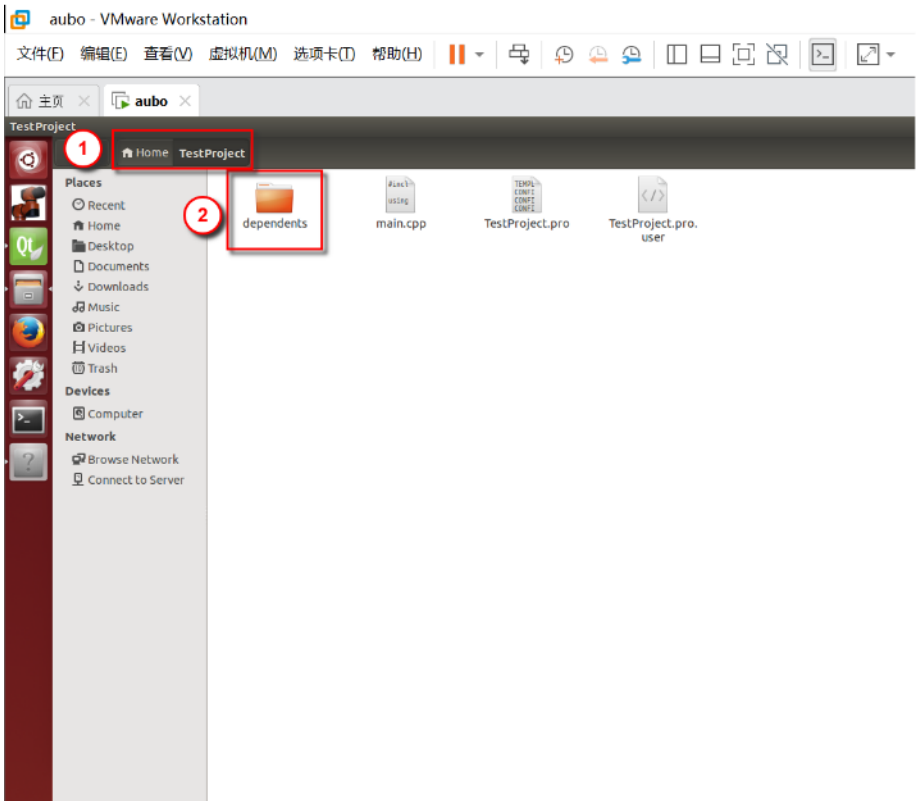
7. Click "Finish".



8. The interface is as follows.



9. Open the folder where the project is located (here is "TestProject"), and copy the dependents file in the SDK package to the folder.



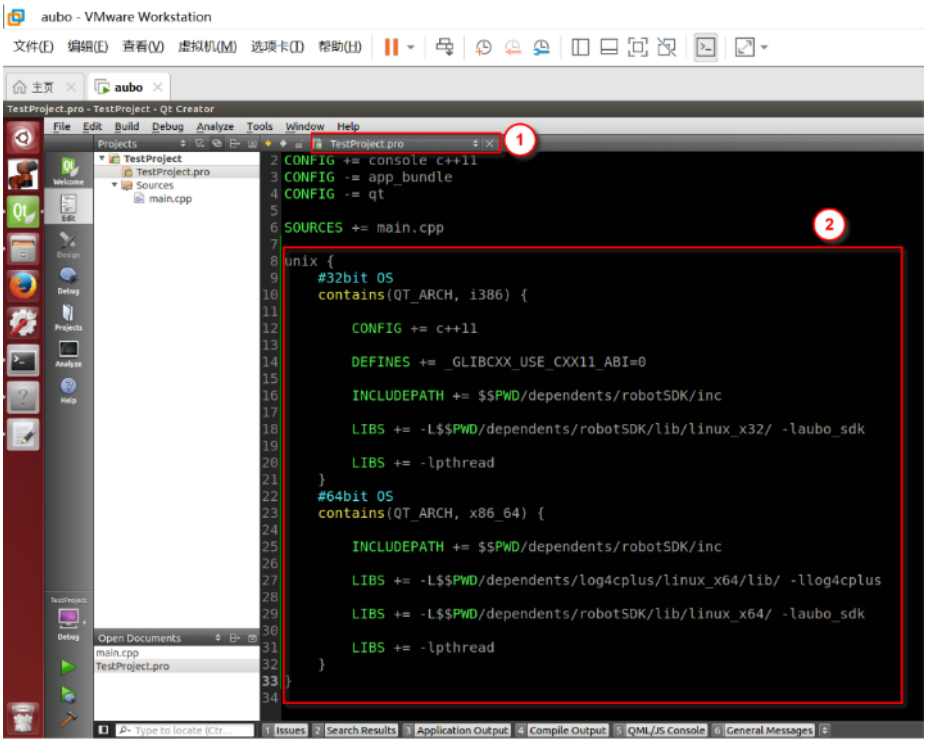
10. Open the TestProject.pro file, enter the code below and save.

Note: When importing header files and lib library files, users need to configure according to the actual received SDK package and the path where the files are actually stored.

```
1.  unix {
2.      #32bit OS
3.      contains(QT_ARCH, i386) {
4.
5.          CONFIG += c++11
6.
7.          DEFINES += _GLIBCXX_USE_CXX11_ABI=0
8.
9.          INCLUDEPATH += $$PWD/dependents/robotSDK/inc
10.
11.         LIBS += -L$$PWD/dependents/robotSDK/lib/linux_x32/ -laubo_sdk
12.
13.         LIBS += -lpthread
14.     }
15.     #64bit OS
16.     contains(QT_ARCH, x86_64) {
```

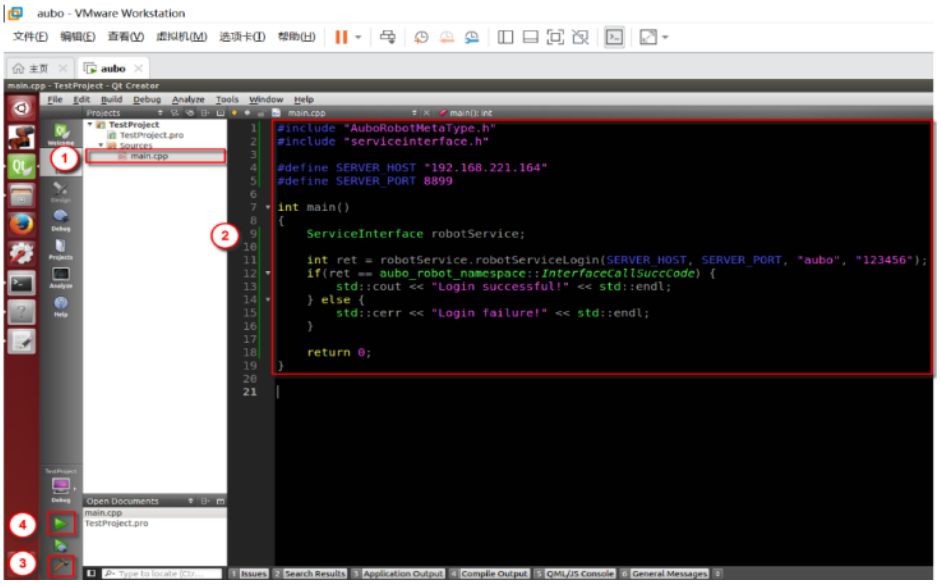
```
17.
18.         INCLUDEPATH += $$PWD/dependents/robotSDK/inc
19.
20.         LIBS += -L$$PWD/dependents/log4cplus/linux_x64/lib/ -llog4cplus
21.
22.         LIBS += -L$$PWD/dependents/robotSDK/lib/linux_x64/ -laubo_sdk
23.
24.         LIBS += -lpthread
25.     }
26. }
```

After entering the code, as shown in the image below.

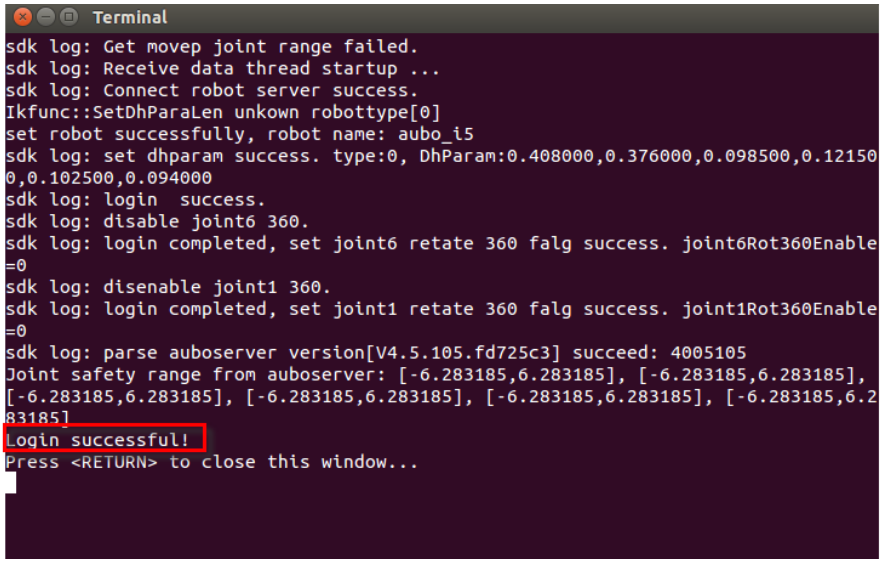


11. When entering the code in main.cpp, note that SERVER_HOST is the corresponding IP address. Run the program.

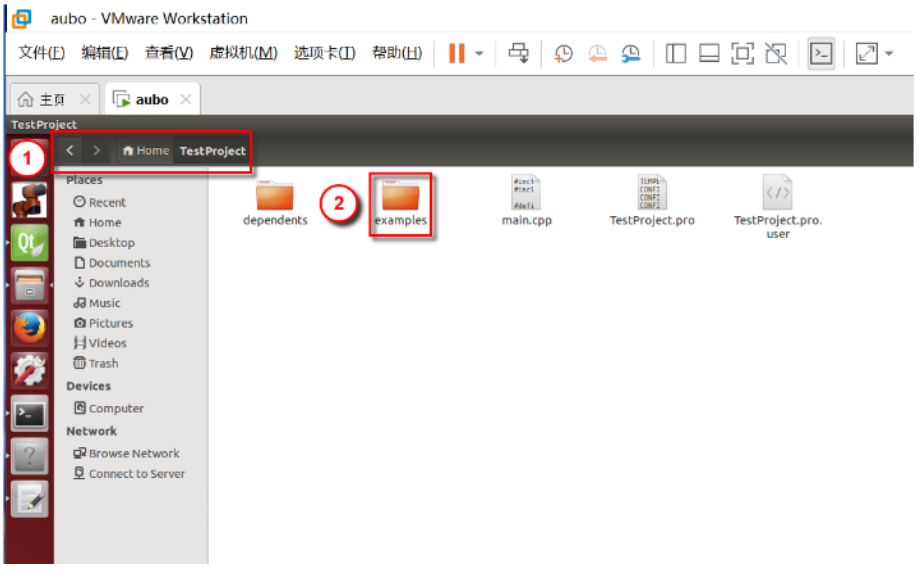
    Note: Before running the program, you need to open AUBOPE in the AUBO virtual machine or connect to the real robotic arm. And the IP address and port number in the program should be set correctly to ensure the successful login of the robotic arm.



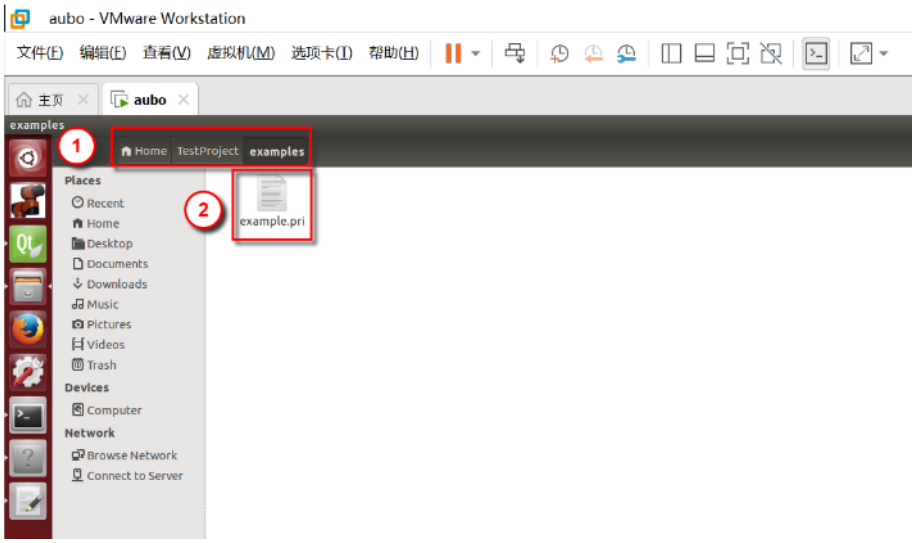
12. The running result shows that the login is successful. The environment configuration is successful.



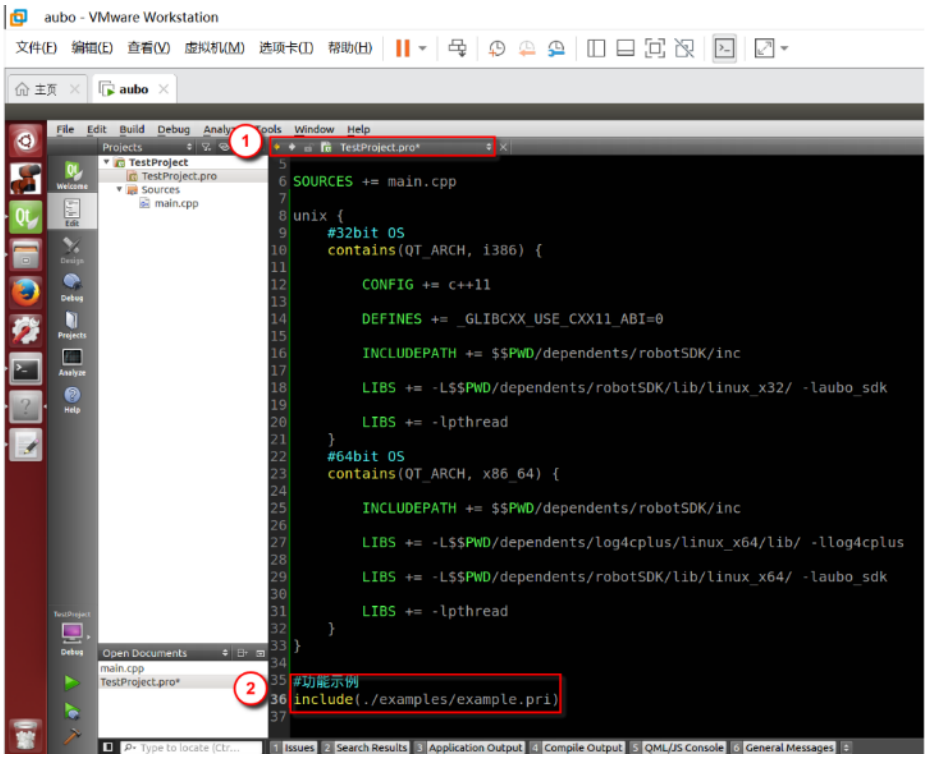
13. Create a new folder "examples" in the folder where the project is located ("TestProject").



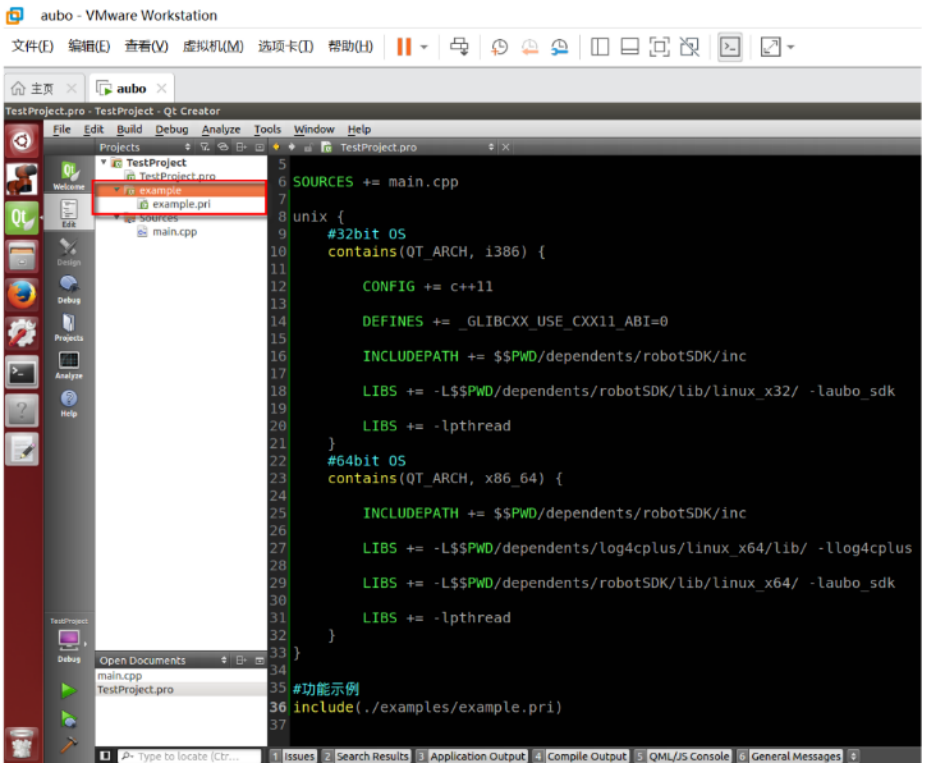
14. Create a new text in the "examples" folder and name it "example.pri".



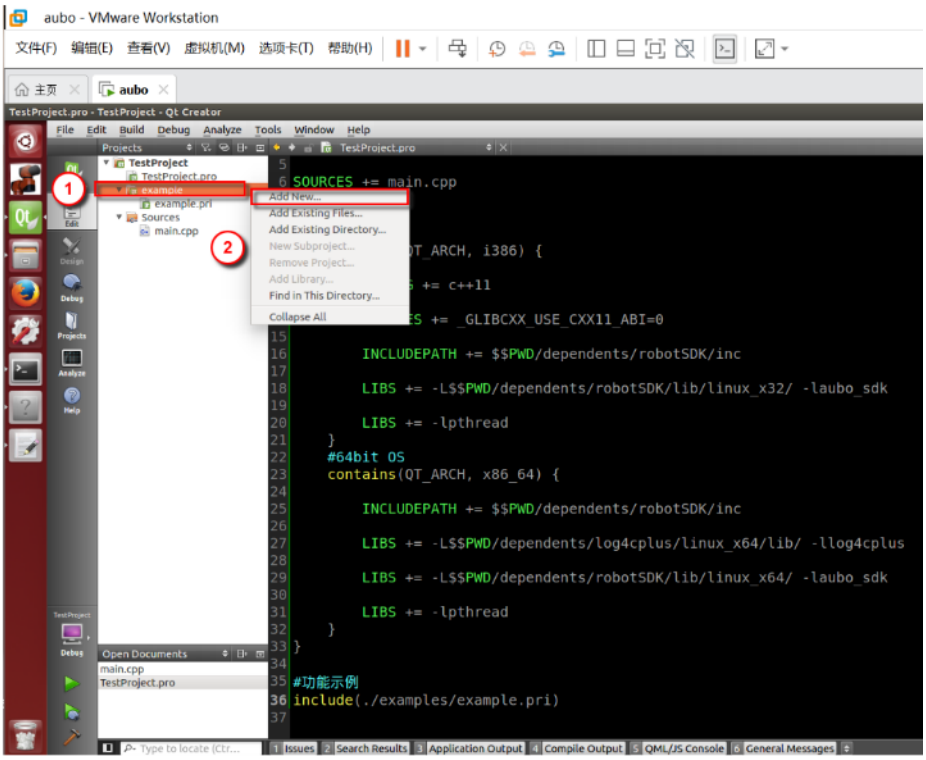
15. Open the TestProejct.pro file and add code to the "TestProject.pro" file, as shown in the following figure. save.



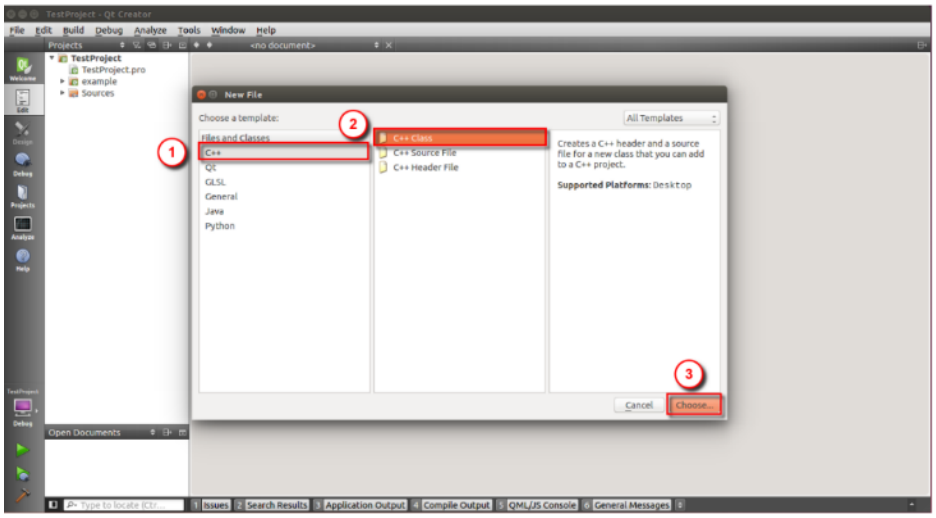
16. Then the following interface appears.



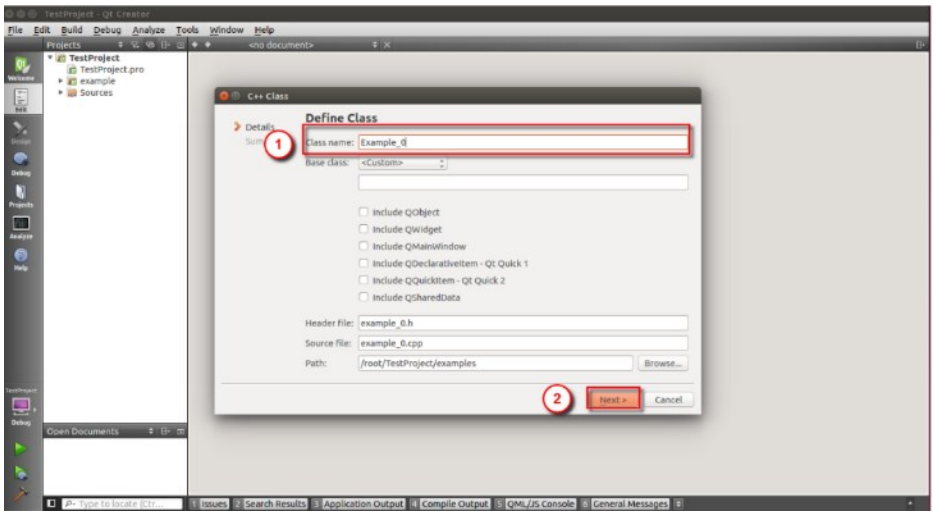
17. Select "example", right-click, and click "Add New...".



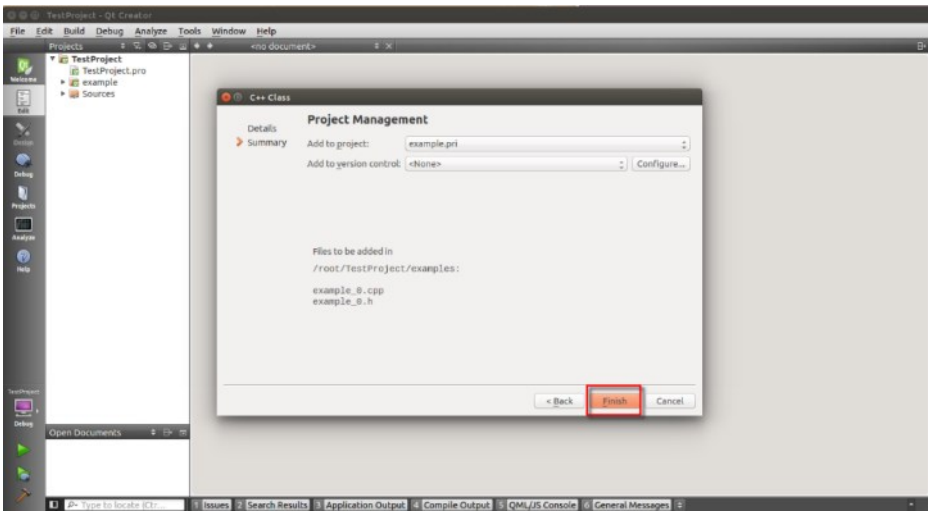
Select "C++" - "C++Class" and click "Choose...".



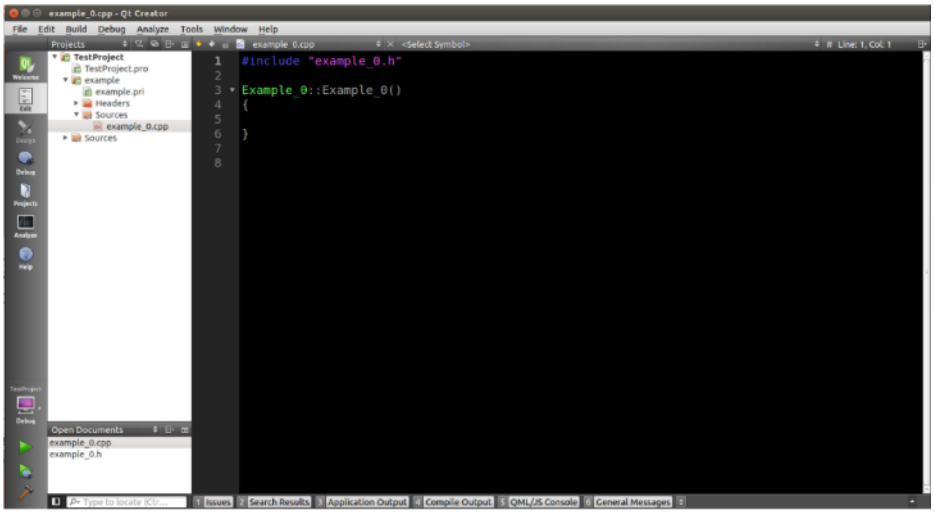
Enter the class name as "Example_0" and click "Next".



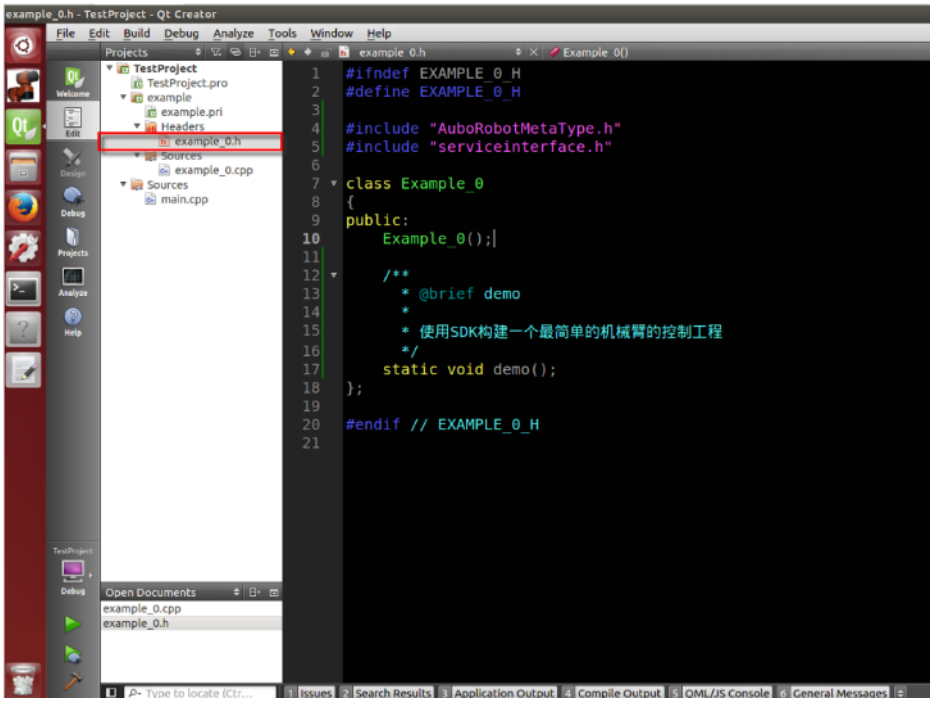
Click "Finish".



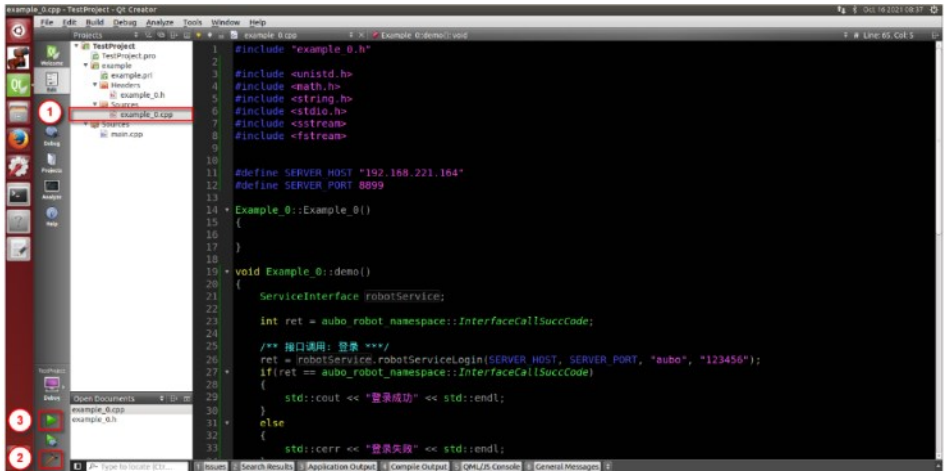
Appears as shown below.



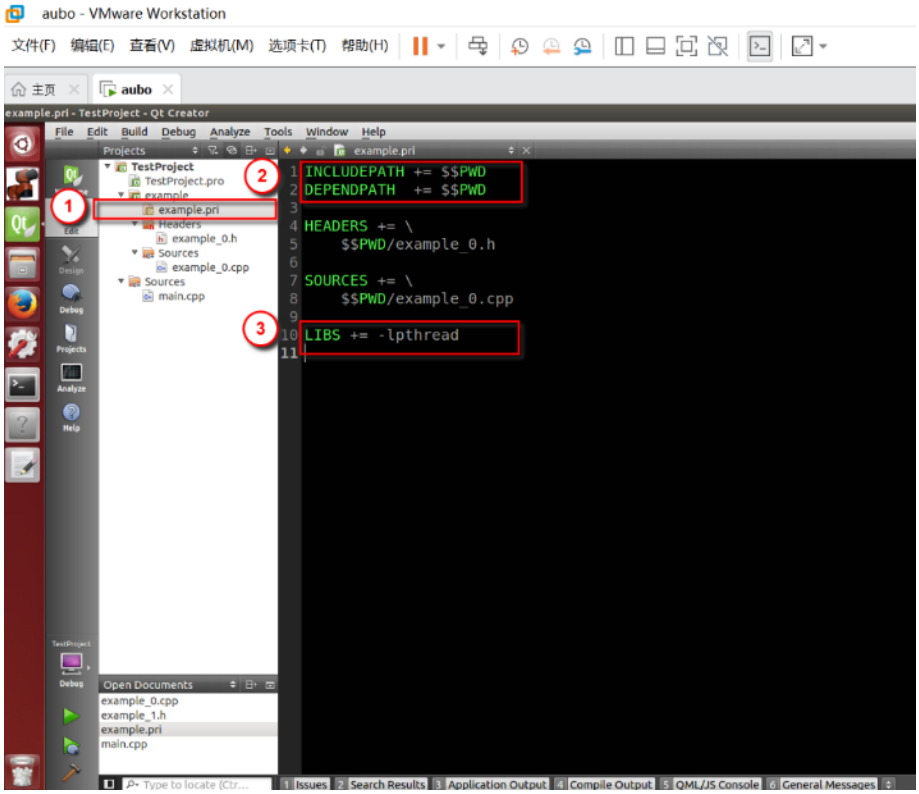
18. Select "example_0.h" and enter the code as shown below.



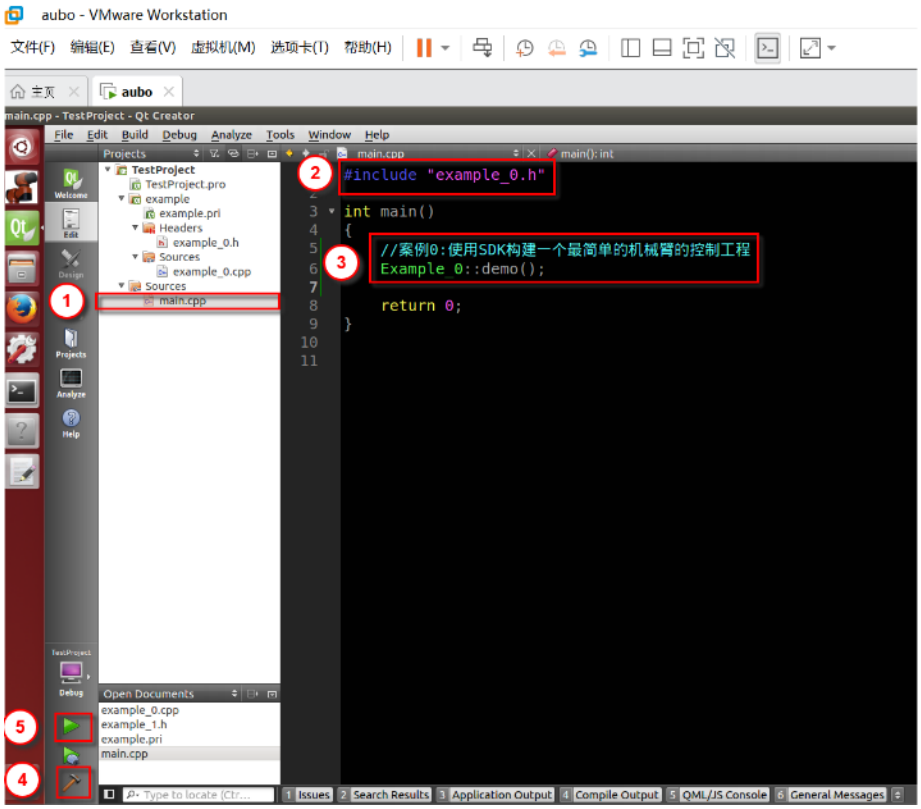
19. Select "example_0.cpp" and enter the code. Compile and run the program.



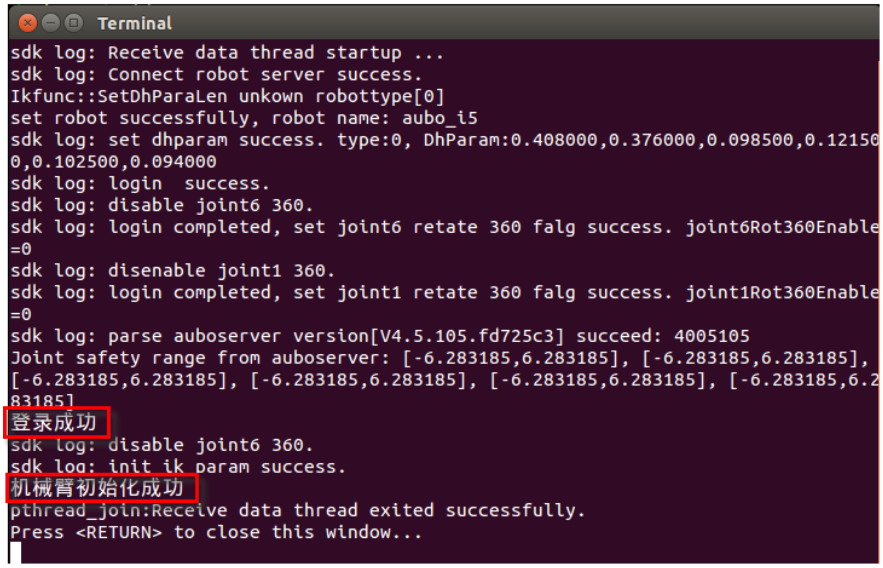
20. Add code in example.pri.



21. Add code in main.cpp. Compile and run the program.

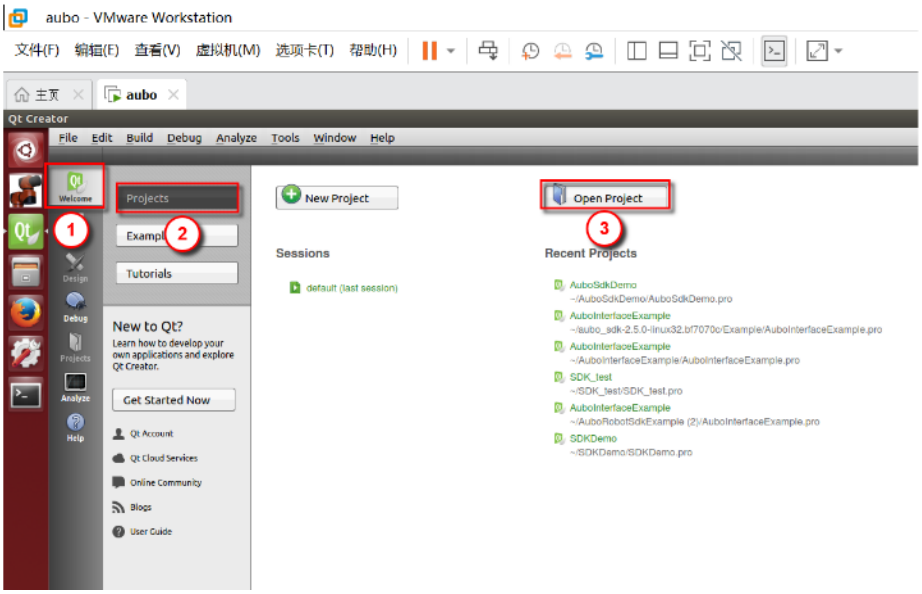


22. The running result is as follows. Indicates that the robot is successfully logged in and initialized.

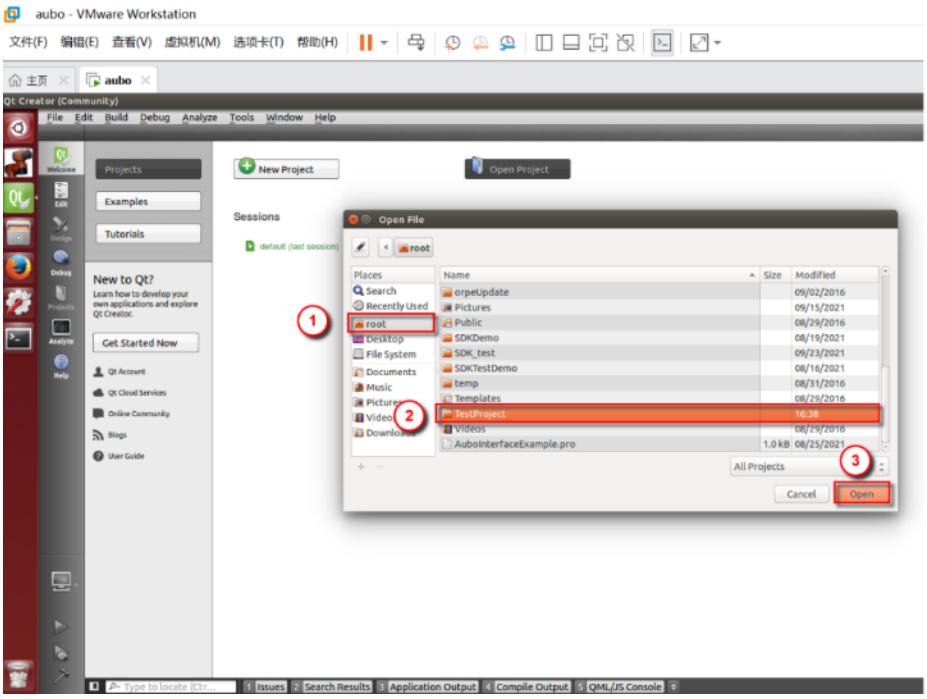


5.2 Open project

1. Open Qt Creator, in the "Welcome" - "Projects" column, click "Open Project".



2. Select the folder where the project is located, and then click "OPEN" (the project folder "TestProject" in the document is saved in the root path).



3. Select "TestProject.pro", click "Open", and the project is opened.