Coordinate control

Coordinate control is to make the robot arm move to a specified point with a specified posture, which is divided into x, y, z, rx, ry, rz. X, Y and Z represents the position of the robot arm head in space (The coordinate system is Cartesian coordinate system); [rx,ry,rz] represents the posture of such head at this point (The coordinate system is Euler coordinates).

1 Single parameter control

1.1 Sending single parameter coordinates

WriteCoord(Axis axis, double value, int speed = DefaultSpeed)
Return value: no
parameter description: Parameter 1: coordinate number (Axis enumeration type, int: 1-6 (X-RZ)); Parameter 2: coordinate (X, Y and Z range from -300 to 300.00 in mm; RX, RY, RZ ranges from -180 to 180); Parameter 3: speed (0-100), the example with 30 as a default
example:

```
mycobot::MyCobot::I().WriteCoord(mycobot::Axis::X, 10, 30);
```

2 Multi-parameter control

2.1 Get all coordinates

GetCoords() Return value: Coords type parameter description: no example: mycobot::Coords coords = mycobot::MyCobot::I().GetCoords();

2.2 Send all coordinates

WriteCoords(const Coords& coords, int speed = DefaultSpeed)
Return value: no
parameter description: Parameter 1: coordinates (X, Y and Z range from -300 to 300.00 in mm; RX, RY and RZ range from -180 to 180); Parameter 2: speed (0-100), the example with 30 as a default
example:

```
mycobot::Coords goal_coords = {5, 5, 5, 5, 5, 5 };
mycobot::MyCobot::I().WriteCoords(goal_coords, 30);
mycobot::Coords goal_coords = {5, 5, 5, 5, 5, 5 };
mycobot::MyCobot::I().WriteCoords(goal_coords, 30);
```

3 Complete use cases

```
int main(int argc, char* argv[])
{
    try {
        QCoreApplication a(argc, argv);
        using namespace std::chrono_literals;
```

```
        if (!mycobot::MyCobot::I().IsControllerConnected()) {
            std::cerr << "Robot is not connected\n";
            exit(EXIT_FAILURE);
        }
        std::cout << "Robot is connected\n";
        mycobot::MyCobot::I().PowerOn();
        mycobot::MyCobot::I().StopRobot();
        std::cout << "Robot is moving: " << mycobot::MyCobot::I().IsMoving() <<
"\n";
        mycobot::Angles angles = mycobot::MyCobot::I().GetAngles();
        std::this_thread::sleep_for(200ms);
        mycobot::Coords coords = mycobot::MyCobot::I().GetCoords();
        angles = mycobot::MyCobot::I().GetAngles();
        std::cout << "[" << angles[mycobot::J1] << ", " << angles[mycobot::J2] <<
", " << angles[mycobot::J3] << ", "
            << angles[mycobot::J4] << ", " << angles[mycobot::J5] << ", " <<
angles[mycobot::J6] << "]";
        mycobot::Angles goal_angles = { 5, 5, 5, 5, 5, 5 };
        mycobot::MyCobot::I().WriteAngles(goal_angles);
        while (!mycobot::MyCobot::I().IsInPosition(goal_angles, false)) {
            angles = mycobot::MyCobot::I().GetAngles();
            std::cout << "[" << angles[mycobot::J1] << ", " <<
angles[mycobot::J2] << ", "
                << angles[mycobot::J3] << ", " << angles[mycobot::J4] << ", "
                << angles[mycobot::J5] << ", " << angles[mycobot::J6] << "]" <<
std::flush;
            std::this_thread::sleep_for(200ms);
        }

        mycobot::MyCobot::I().JogAngle(mycobot::Joint::J1, 1, 5);
        std::this_thread::sleep_for(5000ms);
        mycobot::MyCobot::I().StopRobot();

        std::cout << "\n";
        exit(EXIT_SUCCESS);
    } catch (std::error_code&) {
        std::cerr << "System error. Exiting.\n";
        exit(EXIT_FAILURE);
    } catch (...) {
        std::cerr << "Unknown exception thrown. Exiting.\n";
        exit(EXIT_FAILURE);
    }
}
```