Joint control

For a serial multi-joint robot, the joint control is to control the variables of each joint of the robot arm so as to make each joint reaches a target position at a certain speed.

1 Single joint control

1.1 Sending the angle of single joint

WriteAngle(Joint joint, double value, int speed = DefaultSpeed)
Return value: no parameter
parameter description: Parameter 1: joint number (1-6) Parameter 2: angle (-170°- 170°)
Parameter 3: speed (0-100), the example with 30 as a default
example:
`mycobot::MyCobot::I().WriteAngle(mycobot::Joint::J1, 10, 30);`

2 Multi-joint control

2.1 Get the angles of all joints

GetAngles()
Return value: Angles type
parameter description: no example:
`mycobot::Angles angles= mycobot::MyCobot::I().GetAngles();`

2.2 Send the angles of all joints

WriteAngles(const Angles& angles, int speed = DefaultSpeed)
Return value: no
parameter description: Parameter 1: all angles (std::array, ranging from -170° to 170°) Parameter 2: speed (0-100), the example with 30 as a default

```
mycobot::Angles goal_angles = { 5, 5, 5, 5, 5, 5 };
mycobot::MyCobot::I().WriteAngles(goal_angles, 30);
mycobot::Angles goal_angles = { 5, 5, 5, 5, 5, 5 };
mycobot::MyCobot::I().WriteAngles(goal_angles, 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);
    }
}
```