Inverse Solution

Before starting this function, you need to close the large program and APP processes. If you need to restart the large program and APP later, start them from the terminal:

```
bash ~/dofbot_pro/APP_DOFBOT_PRO/start_app.sh
```

1. Function Description

After the node program starts, it will provide a /dofbot_kinemarics service. This service provides both FK and IK services. You can choose to call FK or IK by specifying kin.name. The **fk** service content is to **input the current angle values of 6 servos**, that is, joint values, and will **output the current pose of the robotic arm end**, where the pose includes xyz coordinates and Euler angles (roll, pitch, yaw); the **ik** service content is to **input the pose of the robotic arm end**, that is, xyz coordinates and Euler angle values, and will **output the angle values of 6 servos**, that is, joints. In subsequent operations when we grasp objects in 3D space, we need to first read the current pose of the robotic arm end, then identify the object's center to get a new pose coordinate through coordinate system conversion, and then call IK to calculate the value of each servo.

2. Startup and Operation

2.1. Startup

Enter in the terminal:

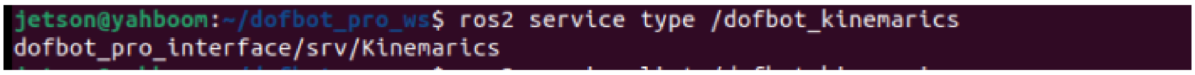
```
#Start inverse kinematics program
ros2 run dofbot_pro_info kinematics_dofbot
#Start ROS control robotic arm program
ros2 run dofbot_pro_driver arm_driver
```

After the inverse kinematics node program starts, it will provide a service. Use the following command to check. Enter in the terminal:

```
ros2 node info /kinemarics_dofbot
```

In the listed service list, there is a **/dofbot_kinemarics**, which is the inverse kinematics service. Use the following command to view the relevant information of this service:

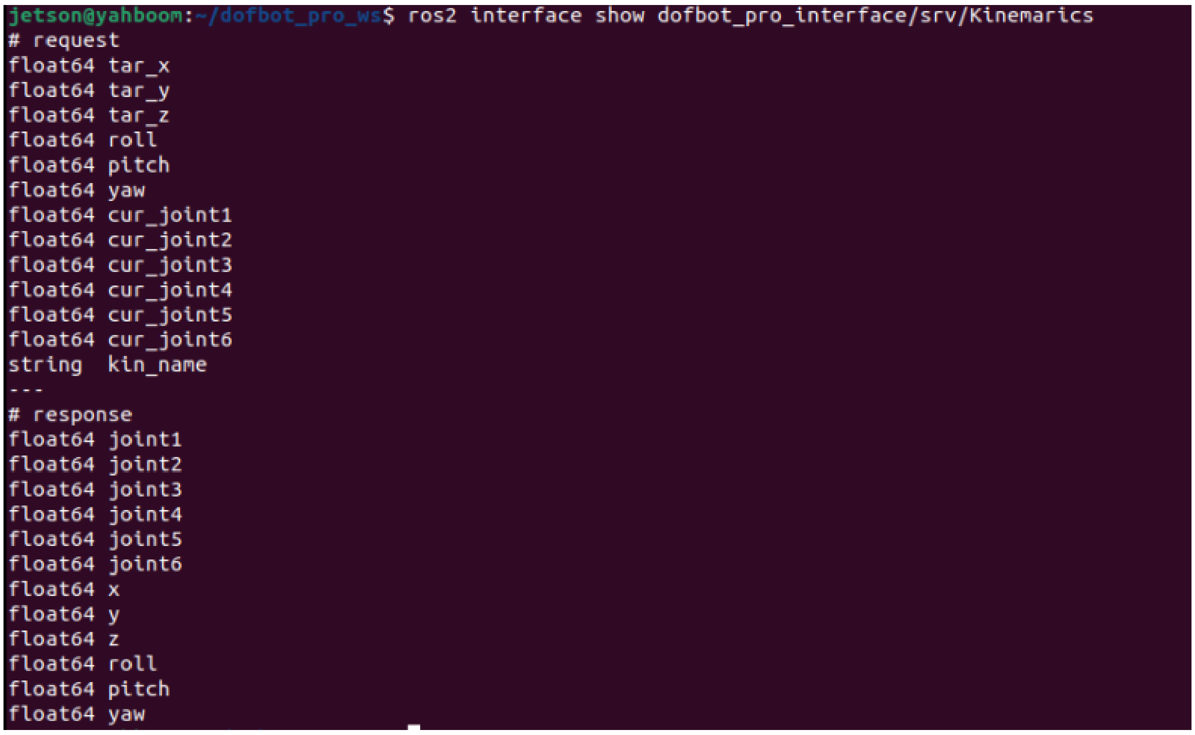
```
ros2 service type /dofbot_kinemarics
```



```
jetson@yahboom:~/dofbot_pro_ws$ ros2 service type /dofbot_kinemarics
dofbot_pro_interface/srv/Kinemarics
```

As shown in the figure above, it describes that the service type data of /dofbot_kinemarics is the custom **dofbot_pro_interface/srv/Kinemarics**. The specific parameters are the Args content below. We can use the following command to see what content is included in the **dofbot_pro_interface/srv/Kinemarics** data type:

```
ros2 interface show dofbot_pro_interface/srv/Kinemarics
```



```
jetson@yahboom:~/dofbot_pro_ws$ ros2 interface show dofbot_pro_interface/srv/Kinemarics
# request
float64 tar_x
float64 tar_y
float64 tar_z
float64 roll
float64 pitch
float64 yaw
float64 cur_joint1
float64 cur_joint2
float64 cur_joint3
float64 cur_joint4
float64 cur_joint5
float64 cur_joint6
string  kin_name
---
# response
float64 joint1
float64 joint2
float64 joint3
float64 joint4
float64 joint5
float64 joint6
float64 x
float64 y
float64 z
float64 roll
float64 pitch
float64 yaw
```

Here the --- divides it into two parts. The upper part is the request (can be understood as input parameters), and the lower part is the response (can be understood as output return values). The request part has an additional kin_name, which is used to distinguish whether we are calling IK or FK. If we want to call the IK service, assign IK to kin_name; conversely, assign the value of FK to kin_name to call the FK service. **Request part parameters**:

- tar_x: x value of target position, unit is m
- tar_y: y value of target position, unit is m
- tar_z: z value of target position, unit is m
- Roll: Roll value of target pose, unit is radians
- Pitch: Pitch value of target pose, unit is radians
- Yaw: Yaw value of target pose, unit is radians

The above 6 are parameters that must be assigned when calling IK.

- cur_joint1: Current angle value of servo 1, unit is degrees
- cur_joint2: Current angle value of servo 2, unit is degrees
- cur_joint3: Current angle value of servo 3, unit is degrees
- cur_joint4: Current angle value of servo 4, unit is degrees
- cur_joint5: Current angle value of servo 5, unit is degrees
- cur_joint6: Current angle value of servo 6, unit is degrees

The above 6 are parameters that must be assigned when calling FK.

- kin_name: Name of the service to call, ik or fk

Response part parameters:

- joint1: Angle value of servo 1, unit is degrees
- joint2: Angle value of servo 2, unit is degrees
- joint3: Angle value of servo 3, unit is degrees
- joint4: Angle value of servo 4, unit is degrees
- joint5: Angle value of servo 5, unit is degrees
- joint6: Angle value of servo 6, unit is degrees

The above 6 values are the angle values of 6 servos that will be returned after calling the IK service.

- x: Calculated x value of end position, unit is m
- y: Calculated y value of end position, unit is m
- z: Calculated z value of end position, unit is m
- Roll: Calculated Roll value of end position, unit is radians
- Pitch: Calculated Pitch value of end position, unit is radians
- Yaw: Calculated Yaw value of end position, unit is radians

The above six parameters are the end position and pose that will be returned after calling the FK service.

2.2. Operation

2.2.1. FK

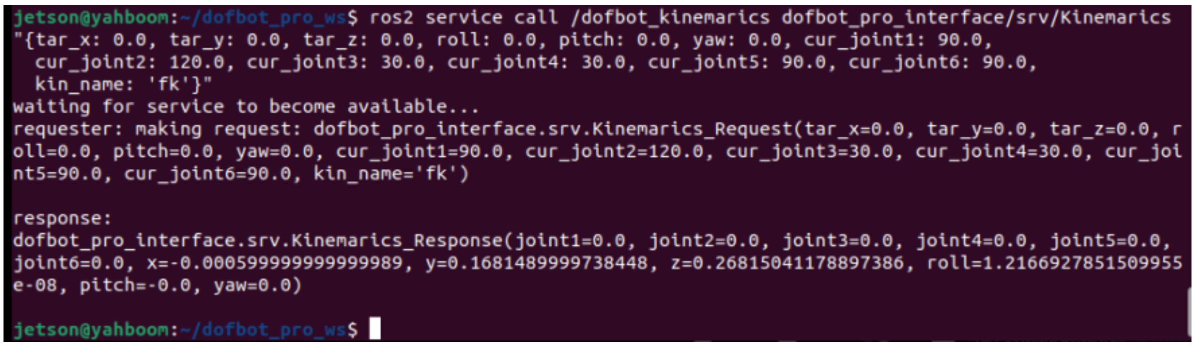
First, let's make the robotic arm move to a certain state, and then read out the end position and pose based on this state, and compare it with the actual measured values to see if it's accurate. From the previous lesson "ROS Control Robotic Arm" content, we can use topics to first publish a robotic arm pose, such as publishing a pose like [90,120,30,30,90,90]. Enter in the terminal:

```
ros2 topic pub /TargetAngle dofbot_pro_interface/ArmJoint "id: 0
run_time: 1000
angle: 0.0
joints: [90,120,30,30,90,90]"
```

Press Enter to publish the message, and the robotic arm moves to the pose we specified. Then, we call the FK service, write the previously set [90,120,30,30,90,90] into the current values of the six servos mentioned above, that is, cur_joint1 to cur_joint6, and assign fk to kin_name. Enter in the terminal:

```
ros2 service call /dofbot_kinemarics dofbot_pro_interface/srv/Kinemarics "
{tar_x: 0.0, tar_y: 0.0, tar_z: 0.0, roll: 0.0, pitch: 0.0, yaw: 0.0, cur_joint1:
90.0,
  cur_joint2: 120.0, cur_joint3: 30.0, cur_joint4: 30.0, cur_joint5: 90.0,
cur_joint6: 90.0,
  kin_name: 'fk'}"
```

Press Enter, and the response will be printed in the terminal as shown below:



```
jetson@yahboom:~/dofbot_pro_ws$ ros2 service call /dofbot_kinemarics dofbot_pro_interface/srv/Kinemarics
"{tar_x: 0.0, tar_y: 0.0, tar_z: 0.0, roll: 0.0, pitch: 0.0, yaw: 0.0, cur_joint1: 90.0,
  cur_joint2: 120.0, cur_joint3: 30.0, cur_joint4: 30.0, cur_joint5: 90.0, cur_joint6: 90.0,
  kin_name: 'fk'}"
waiting for service to become available...
requester: making request: dofbot_pro_interface.srv.Kinemarics_Request(tar_x=0.0, tar_y=0.0, tar_z=0.0, r
oll=0.0, pitch=0.0, yaw=0.0, cur_joint1=90.0, cur_joint2=120.0, cur_joint3=30.0, cur_joint4=30.0, cur_joi
nt5=90.0, cur_joint6=90.0, kin_name='fk')

response:
dofbot_pro_interface.srv.Kinemarics_Response(joint1=0.0, joint2=0.0, joint3=0.0, joint4=0.0, joint5=0.0,
joint6=0.0, x=-0.000599999999999989, y=0.1681489999738448, z=0.26815041178897386, roll=1.216692785150995
e-08, pitch=-0.0, yaw=0.0)

jetson@yahboom:~/dofbot_pro_ws$ 
```

From the printed response, we can see that the xyz representing the robotic arm end position and the Euler angles Roll, Pitch, Yaw representing the robotic arm end pose are respectively: x=-0.0006, y=0.1684148999974, z=0.268150411789, Roll=1.21669278515e-08, Pitch=-0.0, Yaw=0.0. From these data, we can know that the robotic arm end is 0.0006 meters to the right of the base coordinate system, 0.168414899974 meters forward, and 0.26810411789 meters high. The pose can be considered unchanged because the Euler angles are all equal to zero or infinitely close to

0. We can use a ruler to roughly measure the distance from the robotic arm end to the base coordinate system to see if it matches the calculated xyz values.

2.2.2. IK

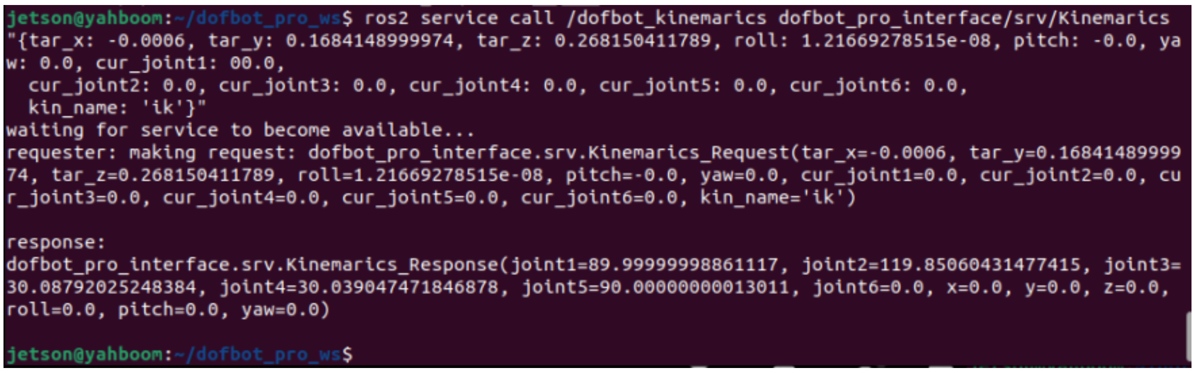
First, we send the following message to make the robotic arm extend upward. Enter in the terminal:

```
ros2 topic pub /TargetAngle dofbot_pro_interface/ArmJoint "id: 0
run_time: 1000
angle: 0.0
joints: [90,90,90,90,90,90]"
```

Then, we call the IK service, given the position and pose parameters of the robotic arm end, to calculate what the value of each servo of the robotic arm should be to reach this position and pose. We take the xyz and Euler angles calculated by the previous FK as the position and pose to reach. Enter in the terminal:

```
ros2 service call /dofbot_kinemarics dofbot_pro_interface/srv/Kinemarics "
{tar_x: -0.0006, tar_y: 0.1684148999974, tar_z: 0.268150411789, roll:
1.21669278515e-08, pitch: -0.0, yaw: 0.0, cur_joint1: 00.0,
  cur_joint2: 0.0, cur_joint3: 0.0, cur_joint4: 0.0, cur_joint5: 0.0,
cur_joint6: 0.0,
  kin_name: 'ik'}"
```

Press Enter, and the response will be printed in the terminal as shown below:



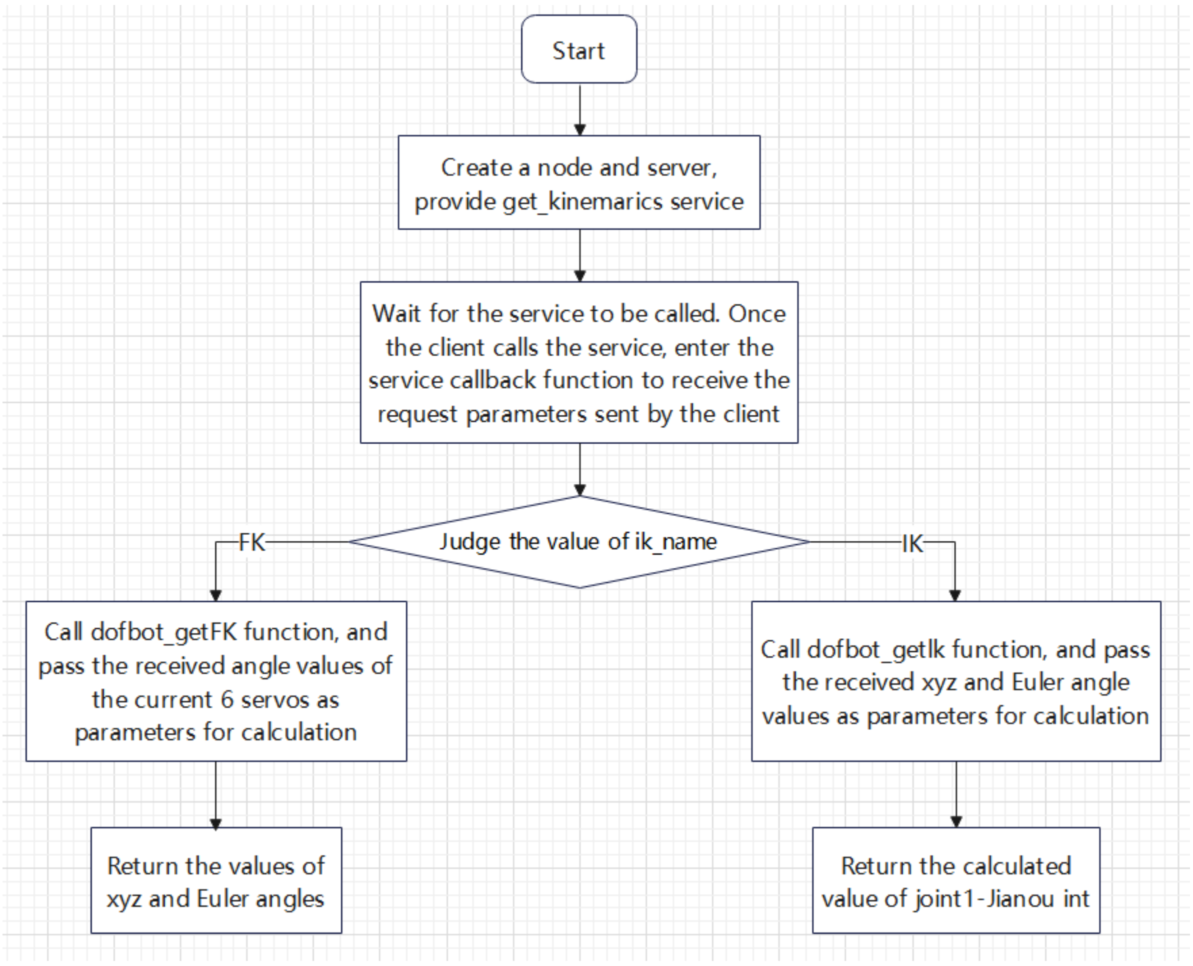
```
jetson@yahboom:~/dofbot_pro_ws$ ros2 service call /dofbot_kinemarics dofbot_pro_interface/srv/Kinemarics
"{tar_x: -0.0006, tar_y: 0.1684148999974, tar_z: 0.268150411789, roll: 1.21669278515e-08, pitch: -0.0, ya
w: 0.0, cur_joint1: 00.0,
  cur_joint2: 0.0, cur_joint3: 0.0, cur_joint4: 0.0, cur_joint5: 0.0, cur_joint6: 0.0,
  kin_name: 'ik'}"
waiting for service to become available...
requester: making request: dofbot_pro_interface.srv.Kinemarics_Request(tar_x=-0.0006, tar_y=0.16841489999
74, tar_z=0.268150411789, roll=1.21669278515e-08, pitch=-0.0, yaw=0.0, cur_joint1=0.0, cur_joint2=0.0, cu
r_joint3=0.0, cur_joint4=0.0, cur_joint5=0.0, cur_joint6=0.0, kin_name='ik')

response:
dofbot_pro_interface.srv.Kinemarics_Response(joint1=89.99999998861117, joint2=119.85060431477415, joint3=
30.08792025248384, joint4=30.039047471846878, joint5=90.00000000013011, joint6=0.0, x=0.0, y=0.0, z=0.0,
roll=0.0, pitch=0.0, yaw=0.0)

jetson@yahboom:~/dofbot_pro_ws$
```

From the printed response, we can see that the calculated returned joint1-joint6 correspond to the values of six servos. After rounding each joint, it is [90,120,30,30,90,0]. We try to publish the calculated joint1-joint6 through topics to the robotic arm control node to make it move to the pose we specified. Enter in the terminal:

```
ros2 topic pub /TargetAngle dofbot_pro_interface/ArmJoint "id: 0
run_time: 1000
angle: 0.0
joints: [90,120,30,30,90,0]"
```

Press Enter to make the robotic arm move to the pose we specified. Then, we use a ruler to measure whether the distance from the robotic arm end to the base coordinate system corresponds to the xyz we input. An error within 1cm is acceptable. This is because each servo of the robotic arm cannot guarantee complete consistency when leaving the factory, and there are small differences.

3. Program Flowchart



Start
Create a node and server, provide get_kinemarics service
Wait for the service to be called. Once the client calls the service, enter the service callback function to receive the request parameters sent by the client
Judge the value of ik_name
FK
IK
Call dofbot_getFK function, and pass the received angle values of the current 6 servos as parameters for calculation
Call dofbot_getIk function, and pass the received xyz and Euler angle values as parameters for calculation
Return the values of xyz and Euler angles
Return the calculated value of joint1-Jianou int

4. Core Code Analysis

Code path:

```
/home/jetson/dofbot_pro_ws/src/dofbot_pro_info/src/kinemarics_dofbotpro.cpp
```

Import relevant header files

```
#include <iostream>
#include <vector>
#include "rclcpp/rclcpp.hpp"
#include <kdl/chain.hpp>
#include <kdl/tree.hpp>
#include <kdl/segment.hpp>
#include <kdl/frames_io.hpp>
#include <kdl/chainiksolverpos_lma.hpp>
#include <kdl/chainfksolverpos_recursive.hpp>
#include <kdl_parser/kdl_parser.hpp>
#include "dofbot_pro_interface/srv/kinematics.hpp"
#include "dofbot_pro_info/kinemarics_dofbotpro.h"
```

Define some constants

```
// Radians to degrees
const float RA2DE = 180.0f / M_PI;
// Degrees to radians
const float DE2RA = M_PI / 180.0f;
//URDF model, used as parameter input for inverse kinematics calculation
const char *urdf_file =
"/home/jetson/dofbot_pro_ws/src/dofbot_pro_info/urdf/DOFBOT_Pro-V24.urdf";
```

Main function

```
auto server_node = rclcpp::Node::make_shared("kinemarics_dofbot"); // Create
service
auto service = server_node-
>create_service<dofbot_pro_interface::srv::Kinemarics>(
        "dofbot_kinemarics", srvicecallback);
```

Service callback function srvicecallback

FK service part

```
if (request->kin_name == "fk") {
        double joints[]{request->cur_joint1, request->cur_joint2, request-
>cur_joint3, request->cur_joint4,
                        request->cur_joint5};
        // Define target joint angle container
        vector<double> initjoints;
        // Define pose container
        vector<double> initpos;
        // Target joint angle unit conversion, from degrees to radians
        for (int i = 0; i < 5; ++i) initjoints.push_back((joints[i] - 90) *
DE2RA);
        // Call forward kinematics function to get current pose
        dofbot_pro.dofbot_getFK(urdf_file, initjoints, initpos);
        RCLCPP_INFO(rclcpp::get_logger("rclcpp"), "fk sent***");
        response->x = initpos.at(0);
        response->y = initpos.at(1);
        response->z = initpos.at(2);
        response->roll = initpos.at(3);
        response->pitch = initpos.at(4);
        response->yaw = initpos.at(5);
    }
```

IK service part

```
//Call ik service according to kin_name
if (request->kin_name == "ik") {

       // Gripper length
        //double tool_param = 0.12;
        // Grasping pose
        double Roll = request->roll ;
        double Pitch = request->pitch;
        double Yaw = request->yaw ;
        double x=request->tar_x;
        double y=request->tar_y;
        double z=request->tar_z;
        // End position (unit: m)
        double xyz[]{x, y, z};
        cout << x << y << z << endl;
        // End pose (unit: radians)
        //double rpy[]{Roll * DE2RA, Pitch * DE2RA, Yaw * DE2RA};
        double rpy[]{Roll , Pitch, Yaw };
        // Create output angle container
        vector<double> outjoints;
        // Create end position container
        vector<double> targetXYZ;
        // Create end pose container
        vector<double> targetRPY;
        for (int k = 0; k < 3; ++k) targetXYZ.push_back(xyz[k]);
        for (int l = 0; l < 3; ++l) targetRPY.push_back(rpy[l]);
        // Inverse kinematics to calculate joint angles to reach target point
        dofbot_pro.dofbot_getIK(urdf_file, targetXYZ, targetRPY, outjoints);
        // Print inverse kinematics results
        for (int i = 0; i < 5; i++) cout << (outjoints.at(i) * RA2DE) + 90 <<
",";
        cout << endl;
        a++;
        response->joint1 = (outjoints.at(0) * RA2DE) + 90;
        response->joint2 = (outjoints.at(1) * RA2DE) + 90;
        response->joint3 = (outjoints.at(2) * RA2DE) + 90;
        response->joint4 = (outjoints.at(3) * RA2DE) + 90;
        response->joint5 = (outjoints.at(4) * RA2DE) + 90;

        // redis.set("joint1", std::to_string(response->joint1));
        cout<<"-----------------"<<endl;



    }
```