

Inverse solution

Before starting this function, you need to close the process of the big program and APP. Enter the following program in the terminal to close the process of the big program and APP.

```
sh ~/app_Arm/kill_YahboomArm.sh
sh ~/app_Arm/stop_app.sh
```

If you need to start the big program and APP again later, start the terminal.

```
sudo systemctl start yahboom_arm.service
sudo systemctl start yahboom_app.service
```

1. Function description

After the node program is started, a /get_kinematics service will be provided. The service will provide two services, fk and ik. The specified kin.name is used to select whether to call fk or ik. The content of the **fk** service is **inputting the current angle values** of the 6 servos, that is, the joint values, and **outputting the current posture** of the end of the robot arm, where the posture includes xyz coordinates and Euler angles (roll, pitch, yaw); the content of the **ik** service is **inputting the posture** of the end of the robot arm, that is, the value of xyz coordinates and Euler angles, and **outputting the angle values** of the 6 servos, that is, joints. When we clamp objects in three-dimensional space later, we need to read the current posture of the end of the robot arm first, then identify the center of the object and obtain a new posture coordinate through coordinate system conversion, and then call ik to calculate the value of each servo.

2. Start and operate

2.1. Start

Terminal input,

```
#Start roscore. If the subsequent program starts launch, you do not need to start
roscore
roscore
#Start the inverse solution program
roslaunch dofbot_pro_info kinematics_dofbot_pro
#Start the ROS control robot arm program
roslaunch dofbot_pro_info arm_driver.py
```

After the inverse solution node program is started, a service will be provided. Check it out through the following command. Terminal input,

```
roslaunch dofbot_pro_info kinematics_dofbot
```

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

```
rosservice info /get_kinemarics
```

```
jetson@yahboom:~$ rosservice info /get_kinemarics
Node: /kinemarics_dofbot
URI: rosrpc://192.168.2.83:35853
Type: dofbot_pro_info/dofbot_pro_kinemarics
Args: tar_x tar_y tar_z Roll Pitch Yaw cur_joint1 cur_joint2 cur_joint3 cur_joint4 cur_joint5 cur_joint6 kin_name
jetson@yahboom:~$
```

As shown in the figure above, the service type data describing /get_kinemarics is the custom **dofbot_pro_info/dofbot_pro_kinemarics**, and the specific parameters are the content of Args below. We can use the following command to see what is included in the **dofbot_pro_info/dofbot_pro_kinemarics** data type,

```
rossrv show dofbot_pro_info/dofbot_pro_kinemarics
```

```
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
---
float64 joint1
float64 joint2
float64 joint3
float64 joint4
float64 joint5
float64 joint6
float64 x
float64 y
float64 z
float64 Roll
float64 Pitch
float64 Yaw
```

The --- here is divided into two parts, the upper part is the request (which can be understood as the input parameter), and the lower part is the response (which can be understood as the output return value). The request part has an additional kin_name, which distinguishes whether we are calling IK or FK. If we want to call the IK service, we assign IK to kin_name, otherwise, we assign the FK value to kin_name to call the FK service. **Parameters in the request part,**

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

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

- cur_joint1: The current angle value of servo No. 1, in degrees
- cur_joint2: The current angle value of servo No. 2, in degrees
- cur_joint3: The current angle value of servo No. 3, in degrees
- cur_joint4: The current angle value of servo No. 4, in degrees
- cur_joint5: The current angle value of servo No. 5, in degrees
- cur_joint6: The current angle value of servo No. 6, in degrees

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

- kin_name: the name of the service being called, ik or fk

Parameters in the response part,

- joint1: the angle value of servo No. 1, in degrees
- joint1: the angle value of servo No. 2, in degrees
- joint1: the angle value of servo No. 3, in degrees
- joint1: the angle value of servo No. 4, in degrees
- joint1: the angle value of servo No. 5, in degrees
- joint1: the angle value of servo No. 6, in degrees

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

- x: calculate the x value of the end position, in m
- y: calculate the y value of the end position, in m
- z: calculate the z value of the end position, in m
- Roll: calculate the Roll value of the end position, in radians
- Pitch: calculate the Pitch value of the end position, in radians
- Yaw: calculate the Yaw value of the end position, in radians

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

2.2, Operation

2.2.1, FK

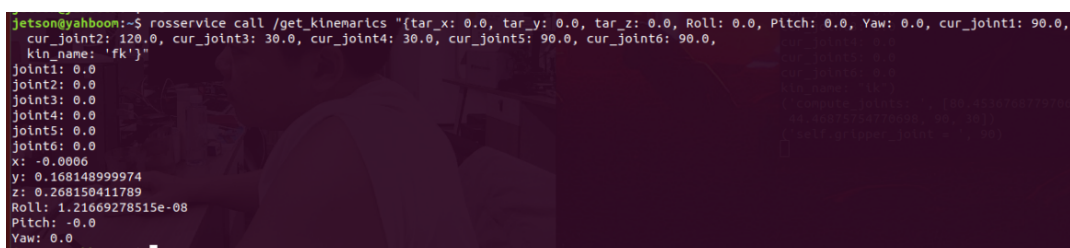
Let's first let the robot arm move to a certain state, and then read the position and posture of the end according to this state, and compare them with the actual measurement to see if they are accurate. Based on the content of the last lesson "ROS Control Robotic Arm", we can use the topic to publish the posture of a robot arm, such as publishing a posture like [90,120,30,30,90,90], and input in the terminal,

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

Press Enter to publish the message, and the robot arm moves to the posture we specified. Then, we call the FK service and write the [90,120,30,30,90,90] just set to the current values of the six servos just mentioned, that is, cur_joint1 to cur_joint6, and assign kin_name to fk, and input in the terminal,

```
rosservice call /get_kinematics "{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 out in the terminal, as shown in the figure below.



```
Jetson@yahboom:~$ rosservice call /get_kinematics "{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'}"
joint1: 0.0
joint2: 0.0
joint3: 0.0
joint4: 0.0
joint5: 0.0
joint6: 0.0
x: -0.0006
y: 0.168148999974
z: 0.268150411789
Roll: 1.21669278515e-08
Pitch: -0.0
Yaw: 0.0
Jetson@yahboom:~$
```

From the printed response, we can see that the xyz representing the end position of the robot arm and the Euler angles Roll, Pitch, and Yaw representing the end posture of the robot arm are $x=-0.0006$, $y=0.1684148999974$, $z=0.268150411789$, $\text{Roll}=1.21669278515\text{e-}08$, $\text{Pitch}=-0.0$, $\text{Yaw}=0.0$. From these data, we can know that the end of the robot arm is 0.0006 meters to the right of the base coordinate system, 0.1684148999974 meters in front, and 0.26810411789 meters high. The posture can be considered to be unchanged because the Euler angles are all equal to zero or infinitely close to 0. We can use a ruler to roughly measure the position of the end of the robot arm from the base coordinate system to see if it matches the calculated xyz value.

2.2.2. IK

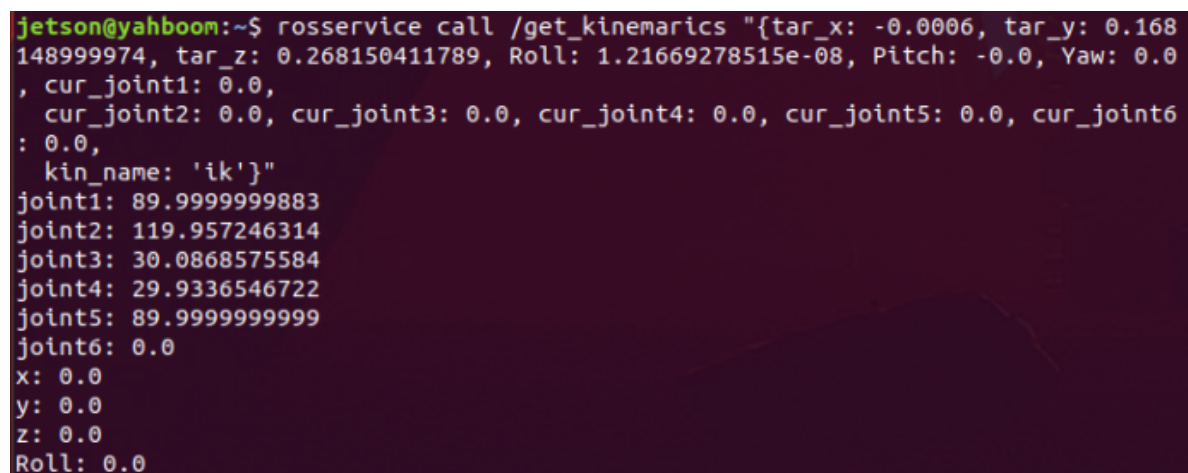
We first send the following message to let the robot arm stretch upwards, input in the terminal,

```
rostopic pub /TargetAngle dofbot_pro_info/ArmJoint "id: 0
run_time: 1000
angle: 0.0
joints: [90,90,90,90,90,90]"
```

Then, we call the IK service again, given the position and posture parameters of the end of the robot arm, and calculate what the value of each servo of the robot arm should be to achieve this position and posture. We take the xyz and Euler angles calculated by FK as the position and posture to be reached as an example, and input the terminal,

```
rosservice call /get_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 in the following figure.



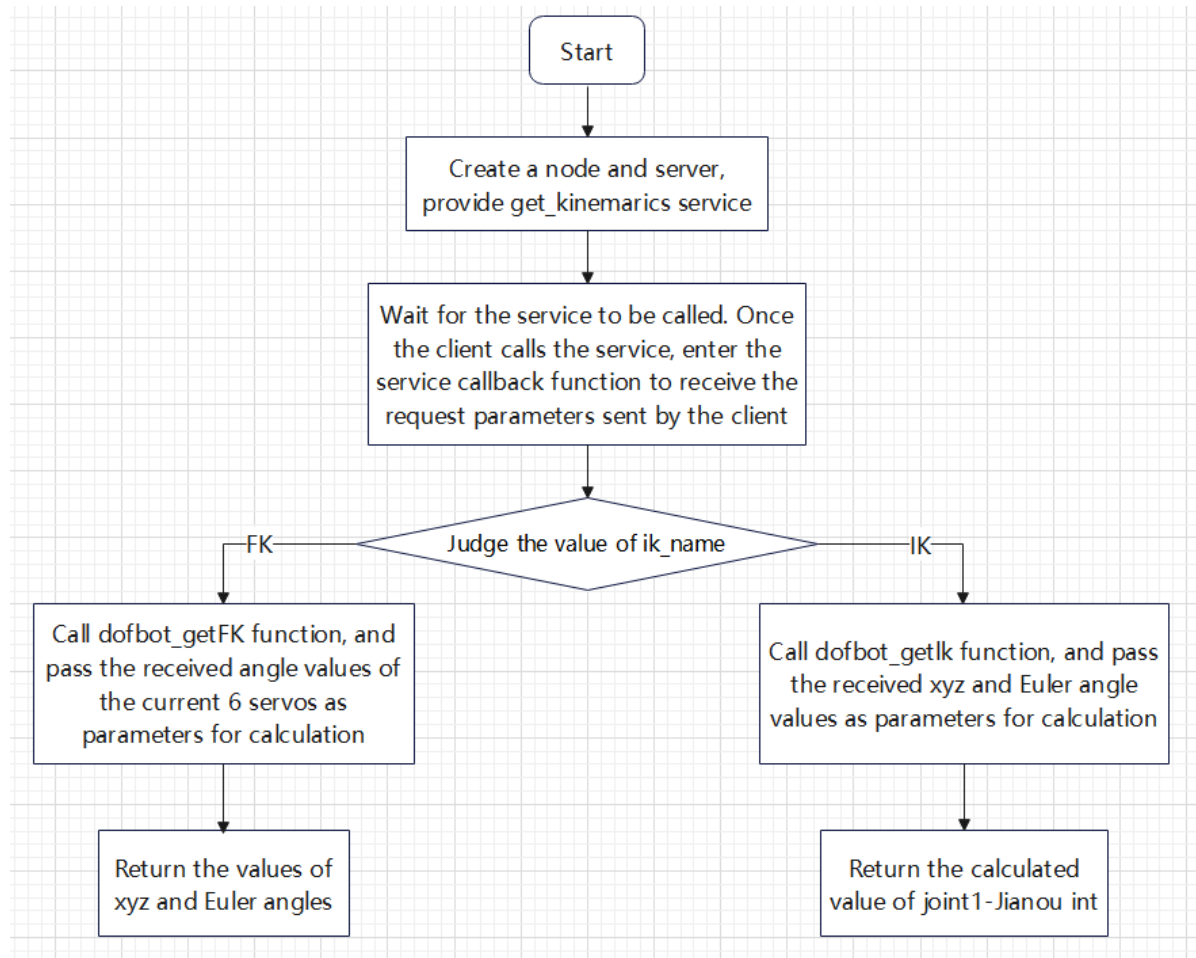
```
jetson@yahboom:~$ rosservice call /get_kinemarics "{tar_x: -0.0006, tar_y: 0.168
1489999974, tar_z: 0.268150411789, Roll: 1.21669278515e-08, Pitch: -0.0, Yaw: 0.0
, cur_joint1: 0.0,
  cur_joint2: 0.0, cur_joint3: 0.0, cur_joint4: 0.0, cur_joint5: 0.0, cur_joint6
: 0.0,
  kin_name: 'ik'}"
joint1: 89.9999999883
joint2: 119.957246314
joint3: 30.0868575584
joint4: 29.9336546722
joint5: 89.9999999999
joint6: 0.0
x: 0.0
y: 0.0
z: 0.0
Roll: 0.0
```

From the printed response, we can see that the calculated joint1-joint6 corresponds to the values of the six servos. After rounding each joint, it is [90,120,30,30,90,0]. We try to publish the calculated joint1-joint6 to the robot control node through the topic, so that it can move to the specified posture. Terminal input,

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

Press Enter to let the robot arm move to the specified posture. Then, we use a ruler to measure the distance from the end of the robot arm to the base coordinate system to see if it corresponds to the xyz we input. The error within 1cm is acceptable. This is because each servo of the robot arm cannot be guaranteed to be completely consistent when leaving the factory, and there are slight differences.

3. Program flow chart



4. Core code analysis

Code path:

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

Import related header files

```
#include <iostream>
#include "ros/ros.h"
#include <kd1/chain.hpp>
#include <kd1/tree.hpp>
#include <kd1/segment.hpp>
#include <kd1/frames_io.hpp>
#include <kd1/chainiksolverpos_lma.hpp>
#include <kd1/chainfksolverpos_recursive.hpp>
#include <kd1_parser/kd1_parser.hpp>
#include "dofbot_pro_info/dofbot_pro_kinematics.h"
#include "dofbot_pro_info/kinematics_dofbotpro.h"
```

Define some constants

```
// Convert radians to degrees
const float RA2DE = 180.0f / M_PI;
// Convert degrees to radians
const float DE2RA = M_PI / 180.0f;
// URDF model, when inverse solving, the urdf model will be used as a parameter
input calculation
const char *urdf_file =
"/home/jetson/dofbot_pro_ws/src/dofbot_pro_info/urdf/DOFBOT_Pro-V24.urdf";
```

main function

```
// Create and initialize nodes
ros::init(argc, argv, "kinemarics_dofbot");
// Create a server, provide a service named get_kinemarics, the callback function
is srvicecallback
ros::ServiceServer server = n.advertiseService("get_kinemarics",
srvicecallback);
```

Service callback function srvicecallback

fk service part

```
//根据kin_name调用的是fk服务
// The fk service is called according to kin_name
if (request.kin_name == "fk") {
    double joints[]{request.cur_joint1, request.cur_joint2, request.cur_joint3,
request.cur_joint4,
    request.cur_joint5};
    // 定义目标关节角容器 Define the target joint angle container
    vector<double> initjoints;
    // 定义位姿容器 Define pose container
    vector<double> initpos;
    // 目标关节角度单位转换,由角度转换成弧度
    // Target joint angle unit conversion, from degrees to radians
    for (int i = 0; i < 5; ++i) initjoints.push_back((joints[i] - 90) * DE2RA);
    // 调取FK函数,获取当前位姿
    //Call FK function to get current pose
    dofbot_pro.dofbot_getFK(urdf_file, initjoints, initpos);;
    cout << "fk已发出***" << endl;
    // 返回值存储在initpos, 然后给response的成员变量赋值, 返回的是表示位置的xyz和表示位姿的
    欧拉角
    // The return value is stored in initpos, and then assigned to the member
    variable of response. The returned value is xyz representing the position and the
    Euler angle representing the posture
    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

```

//根据kin_name调用的是ik服务
// Based on kin_name, the ik service is called
if (request.kin_name == "ik") {
    // 抓取的位姿 Grasping posture
    double Roll = request.Roll; //-135 由于夹爪的坐标系转换了, 所以roll控制俯仰角 -135
    Since the gripper's coordinate system is transformed, roll controls the pitch
    angle
    double Pitch = 0;
    double Yaw = 0;
    double x=request.tar_x;
    double y=request.tar_y;
    double z=request.tar_z;
    // 末端位置(单位: m) End position (unit: m)
    double xyz[]={x, y, z};
    cout << x << y << z << endl;
    // 末端姿态(单位: 弧度) End attitude (unit: radians)
    //double rpy[]={Roll * DE2RA, Pitch * DE2RA, Yaw * DE2RA};
    double rpy[]={Roll, Pitch, Yaw };
    // 创建输出角度容器 Create output angle container
    vector<double> outjoints;
    // 创建末端位置容器 Create an end position container
    vector<double> targetXYZ;
    // 创建末端姿态容器 Create an 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]);
    // 反解求到达目标点的各关节角度
    // Inversely solve the angles of each joint to reach the target point
    dofbot_pro.dofbot_getIK(urdf_file, targetXYZ, targetRPY, outjoints);
    // 打印反解结果 Print the inverse solution
    cout << "-----" << a << "-----" << endl;
    for (int i = 0; i < 5; i++) cout << (outjoints.at(i) * RA2DE) + 90 << ",";
    cout << endl;
    a++;
    //返回值存储在initpos, 然后给response的成员变量赋值, 返回的是joint1-joint6也就是每个舵
    机的目标角度值, 这里要把返回的弧度转换成角度
    //The return value is stored in initpos, and then assigned to the member
    variable of response. The returned value is joint1-joint6, which is the target
    angle value of each servo. Here, the returned radians are converted into
    degrees.
    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;
}

```