Forward kinematics code path： dofbot_ws/src/dofbot_moveit/src/dofbot_kinematics_fk.cpp

Forward kinematics code testing：

cd ~/dofbot_ws/

source devel/setup.bash

rosrun dofbot_moveit dofbot_kinematics_fk

Inverse kinematics code path： dofbot_ws/src/dofbot_moveit/src/dofbot_kinematics_ik.cpp

Inverse kinematics code testing：

cd ~/dofbot_ws/

source devel/setup.bash

rosrun dofbot_moveit dofbot_kinematics_ik

1. Overview

Kinematics of a manipulator arm is the study of the motion of the manipulator arm, regardless of the forces that produce the motion. This section only considers the position and pose relation of the manipulator in the static state.

A typical robotic arm consists of a number of serial connected joints and links. Each joint has one degree of freedom, translation or rotation. For a manipulator with n joints, the joints are numbered from 1 to n, and there are n+1 links, numbered from 0 to n. Link 0 is the base of the manipulator and is generally fixed. Link n has an end-effector on it. Joint i connects link i to link-1. A link can be viewed as a rigid body that determines the relative position between the two joint axes adjacent to it. A link can be described by two parameters, the length of the link and the torque of the link, which define the relative position in space of the two axes associated with it. The parameters of the first link and the last link are meaningless, which is generally chosen as 0. A joint is described by two parameters, one is the linkage offset, which refers to the distance from one link to the next link along the joint axis. The other is joint Angle, which refers to the rotation Angle of one joint relative to the next joint axis.

2. Pose description of three-dimensional space

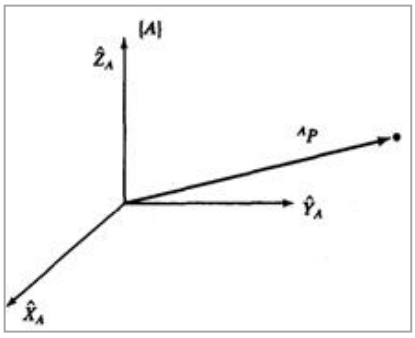
Firstly, a coordinate system is defined, relative to which the position of a point can be represented by a 3-dimensional column vector. The orientation of a rigid body can be represented by a 3 by 3 rotation matrix. The homogeneous transformation matrix of 4×4 can unify the description of rigid body position and attitude (pose), which has the following advantages:

(1) It can describe the position and pose of rigid body and the relative position and pose of coordinate system (description).

(2) It can represent the transformation of a point from the description of one coordinate system to the description of another coordinate system (mapping).

(3) It can represent the transformation (operator) of the orientation description before and after rigid body motion.

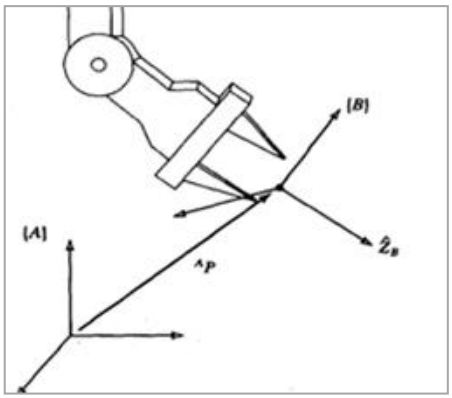
l Position description -- position vector



A coordinate system {A} is represented by three mutually orthogonal unit vectors with arrows. Then the spatial position of point p in the coordinate system {A} is expressed as:

$$^AP = \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix}$$

lDescription of orientation -- rotation Matrix



Commonly used rotation transformation matrices are image.png rotated by an Angle about the X, Y, or Z axes. They are respectively.

$$R(X,\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) \\ 0 & sin(\theta) & cos(\theta) \end{bmatrix} R(Y,\theta) = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} R(Z,\theta) = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

lInhomogeneous statement

After a rotation R and a translation t, we get a/ : a/=R*a+t

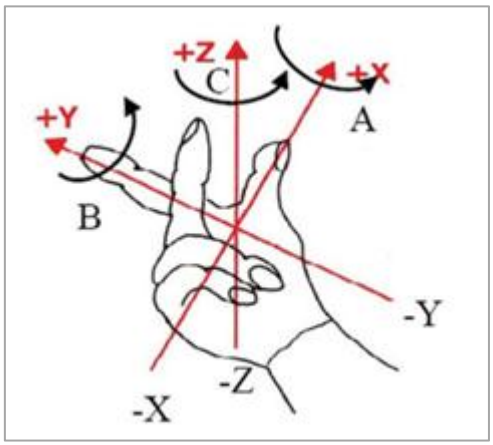
lHomogeneous statement

$$\begin{bmatrix} a\prime \\ 1 \end{bmatrix} = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} a \\ 1 \end{bmatrix} = T \begin{bmatrix} a \\ 1 \end{bmatrix}$$

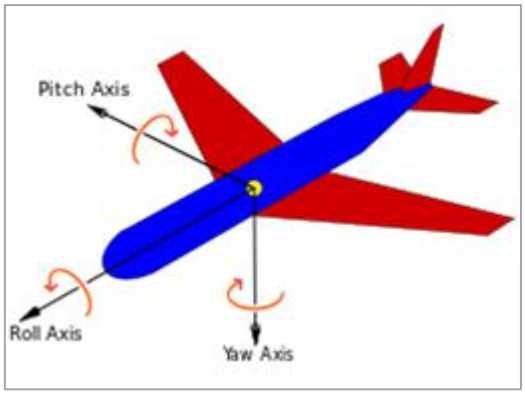
The universal rotation matrix R is:

$$R = R_z(\beta) R_y(\alpha) R_x(\theta)$$

The rotation matrix here rotates theta Angle about the X-axis, then α Angle about the Y-axis, and finally β Angle about the z-axis.
lright-hand rule



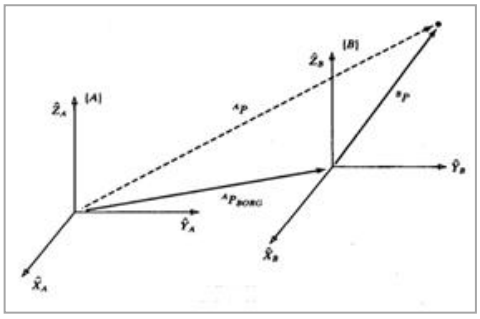
Your thumb points to x, your index finger to y, and your middle finger to z.
lRPY and Euler Angle



The rotation order in accordance with the external coordinate system (reference coordinate system) is x- >y- >z, called RPY (Roll Pitch Yaw);
When describing the same attitude, the above two representations are equivalent, that is, the Angle value of the Yaw Pitch Roll is the same.
Definition: Yaw drift Angle Ϋ; Pitch pitch Angle β; Roll roll Angle α
lquaternion algorithm
Roughly defined, vectors represented by scalars and three imaginary axes are called quaternions.

3. Mapping transformation

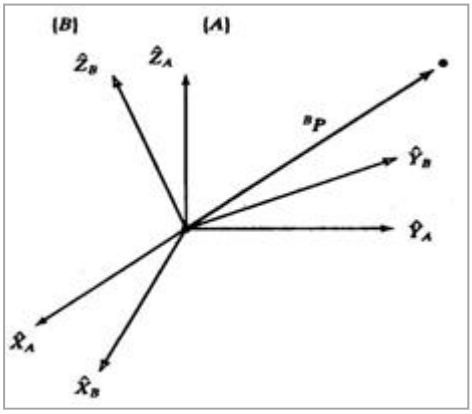
Mapping: Transformation from coordinate system to coordinate system, that is, representation of the same quantity in different coordinate systems. This is often used in robotics.
lmapping of translation coordinates



{A} and {B} have the same attitude, and {B} is different from {A} only translation, that is, {B} has no rotation with respect to {A}. The position of P with respect to {A} can be expressed by adding vectors as follows:

$$^{A}P = {}^{B}P + {}^{A}P_{BORG}$$

lMapping of rotating coordinate system (origin coincidence only rotation)



The components of any vector are the projection of the vector in the direction of the unit vector on the reference frame. The projection is calculated by the dot product of the vectors.

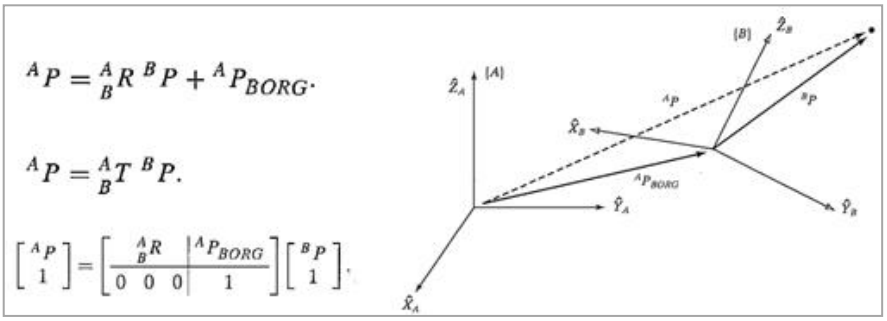
$$^{A}P_x = {}^{B}\widehat{X}_A^T * {}^{B}P \quad 、 \quad ^{A}P_y = {}^{B}\widehat{X}_A^T * {}^{B}P \quad 、 \quad ^{A}P_z = {}^{B}\widehat{X}_A^T * {}^{B}P$$

This can be simplified as:

$$^{A}P = {}^{A}_{B}R\,{}^{B}P$$

This map converts A description of a point P in space relative to {B} into a description of that point relative to {A}.

lMapping in general coordinates



Application scenario: Two coordinate systems {A} and {B}, known {B} relative to {A} translation and rotation are APBORG and image.png and BP, respectively, AP.
In combination with the previous transformation cases with only translation and rotation, set an intermediate coordinate system {C}, whose origin coincides with {B} and attitude is consistent with {A}, the following formula can be obtained:

$$^{A}P = {}^{A}_{B}R\,{}^{B}P + {}^{A}P_{BORG} = {}^{A}_{B}T\,{}^{B}P$$

This leads to the concept of an operator in matrix form, representing a mapping from one coordinate system to another. It's easier to write it as a block matrix

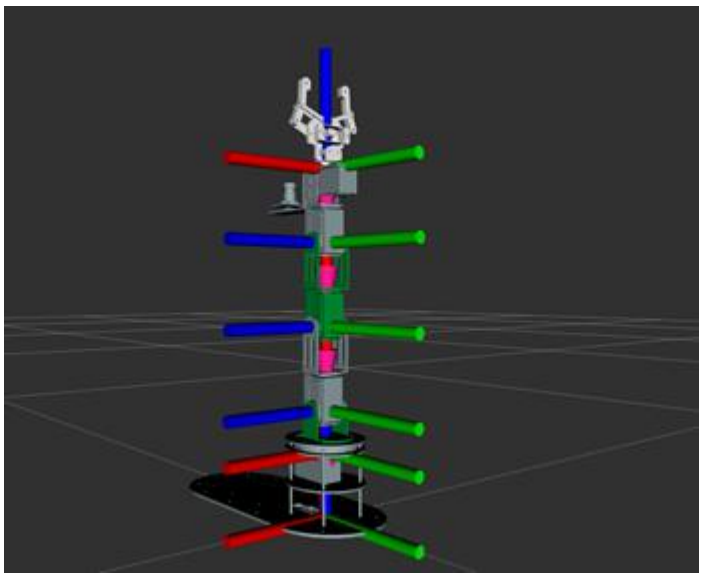
$$\begin{bmatrix} {}^{A}P \\ 1 \end{bmatrix} = \begin{bmatrix} {}^{A}_{B}R & {}^{A}P_{BORG} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} {}^{B}P \\ 1 \end{bmatrix}$$

The intermediate matrix above is called the homogeneous transformation matrix, and although the homogeneous transformation in the concise form is useful, it is not usually used in computer programs for vector transformations, because it consumes the time required to multiply 0 and 1.

4. Forward kinematics solution&&inverse kinematics solution

Forward kinematics solution: In a robotic arm dofbot, given the Angle of each joint, the end pose is solved.
inverse kinematics solution: In a robotic arm dofbot, given the position of the end (except the claw), find the Angle of each joint.
When the joint Angle of the manipulator is known, the end position and attitude of the manipulator can be solved, and the transformation equation of the end coordinate system relative to the base coordinate system can be obtained.
The forward kinematics of the manipulator can be obtained in the following steps:
l is to establish the coordinate system for the joints of the manipulator.
l Write out the DH table of mechanical arm according to the relation of coordinate system and parameters of connecting rod.
l The transformation matrix of each connecting rod is solved by the following formula and DH table.
l Solve the transformation matrix of the end relative to the base based on the continuous link transformation matrix.
The joint structure of a robot can be described by a string of characters. For example, Puma560 is "RRRRRR" and the Stanford arm is "RRPRRR", where each character represents the type of the

corresponding joint, R is the rotation pair and P is the moving pair. In 1955, Denavit and Hartenberg proposed a systematic method for describing the geometric relationships between links and joints on such serial links, which is now known as the D-H parameter method.
Dofbot's coordinate system is established as follows:



The red axis is the X-axis; The green axis is the Y-axis; The blue axis is the z-axis; The bottom coordinate system, we call it the base coordinate system.

α_{i-1}：以$\widehat{X}_{i-1}$方向看，$\widehat{Z}_{i-1}$和$\widehat{X}_i$间的夹角

a_{i-1}：以$\widehat{X}_{i-1}$方向看，$\widehat{Z}_{i-1}$和$\widehat{X}_i$间的距离$(a_i > 0)$

θ_i：以$\widehat{Z}_i$方向看，$\widehat{X}_{i-1}$和$\widehat{X}_i$间的夹角

d_i：以$\widehat{Z}_i$方向看，$\widehat{X}_{i-1}$和$\widehat{X}_i$间的距离

Dofbot 的 D-H 参数 ：

i	d_i(沿 Z 轴)	a_i(沿 X 轴)	β_i(绕 Y 轴)	θ_i(绕 Z 轴)
1	0.066	0	0	θ_1
2	0.04145	0	π/2	θ_2
3	0	-0.08285	0	θ_3
4	0	-0.08285	0	θ_4
5	0	-0.07385	-π/2	θ_5

三角函数公式补充:

cos(a+b)=cos(a)cos(b)-sin(a)sin(b) 简写成 ： c(a+b)=c(a)c(b)-s(a)s(b)

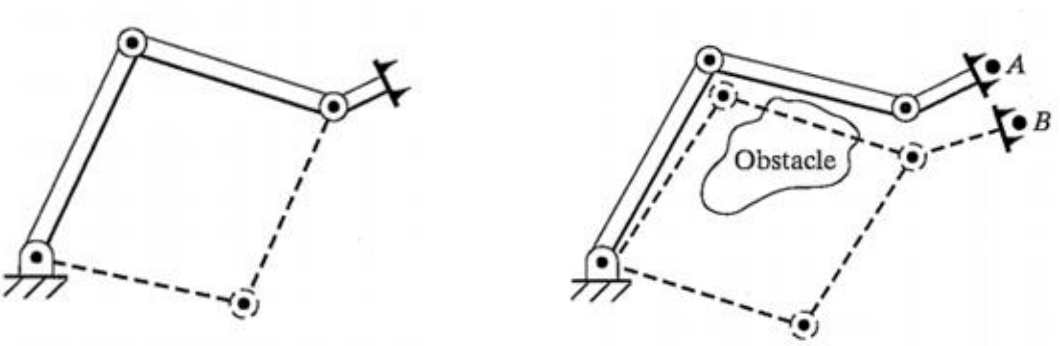
cos(a-b)=cos(a)cos(b)+sin(a)sin(b) 简写成 ： c(a-b)=c(a)c(b)+s(a)s(b)

sin(a±b)=sin(a)cos(b)±cos(a)sin(b)简写成: s(a±b)=s(a)c(b)±c(a)s(b)

sin(0°)=0 ； sin(90°)=1 ； cos(0°)=1 ； cos(90°)=0

l Inverse kinematics of manipulator

The inverse kinematics of the manipulator refers to solving the Angle of each revolute joint by knowing the position and pose of the end of the manipulator, namely, knowing the homogeneous transformation matrix. Therefore, the inverse kinematics of the manipulator can be understood as solving θ1, θ2, θ3, θ4 and θ5 of the joints through the forward kinematics equation. solvability should first be considered in calculating the inverse kinematics solution of robot, that is, the situation of no solution or multiple solutions should be considered. The target point outside the robot workspace is obviously unsolvable. In the case of multiple solutions, it can be seen from the following example that a planar two-bar manipulator (two joints can rotate 360°) has two solutions in the workspace:



If the inverse kinematics has multiple solutions, the control program must select one solution at runtime and send it to the driver to drive the robot joint rotation or translation. There are many different guidelines for how to choose the right solution. One of the most reasonable methods is to choose the closest solution. As you can see, if the robot is at point A and wants to go to point B, A

good choice would be the solution that minimizes the amount that each joint is required to move. A good choice would be the solution that minimizes the amount that each joint is required to move. So in the absence of obstacles, the dotted configuration above will be chosen as the inverse solution. When calculating the inverse solution we can consider the current position as an input parameter, so that we can choose the solution closest to the current position in joint space.

This "recent" can be defined in a variety of ways. When there is an obstacle, the path of the "nearest" solution collides with it, and another "farther" solution is chosen. So we need to figure out all the possible solutions when we're thinking about collisions, path planning, etc. the number of inverse solutions depends on the number of joints, link parameters and the allowable ranges of motion of the joints. The more non-zero values in the D-H parameter table that determine the robot configuration, the more solutions exist. For a universal 6 axis revolute joint manipulator, up to 16 different solutions may exist.

Besides, there are many ways to solve the inverse kinematics of robot, which are generally divided into two types: closed-form solutions and numerical solutions. Different scholars also put forward different solutions to inverse kinematics of the same robot. We should choose a better solution based on the efficiency and accuracy of the calculation method. Generally speaking, the numerical iterative method is slower and more time consuming than the analytical expression of the closed solution, so the existence of the closed solution should be considered when designing the robot configuration.

Difference between analytical solution and numerical solution: numerical solution is obtained by using some calculation methods, such as finite element method, numerical approximation, interpolation method. Others can only use the results of numerical calculations, rather than arbitrarily give the argument and calculate the calculated value. For example, the analytic solution of a quadratic equation with one variable is given above. When solving a quadratic equation with known coefficients, the numerical solution can be obtained by substituting the specific values of the coefficients. One way to understand the difference is that the analytical solution is a formula that applies to the solution of all such equations, while the numerical solution is a specific solution to a particular equation.

```
例如 ：方程 2y=x 的解：
   y=0.5x    —— 解析解
   x=1时，y=0.5    —— 数值解
```

5. Code solving

lEnvironment configuration

Put header file **dofbot_kinemarics.h** and dynamic library **libdofbot_kinemarics.so** into **dofbot_ws/src/dofbot_moveit/include/dofbot_moveit**. If the dynamic library is not found, the The dynamic library libdofbot_kinemarics.so is placed in **/usr/lib**. The system is stored in **/usr/lib** by default.

Open the **CMakeLists.txt** file in the feature pack and specify the location of the header file **dofbot_kinemarics.h**.

```
include_directories(
              include/dofbot_kinemarics
              ${catkin_INCLUDE_DIRS}
)
```

lSpecify a library to link to a library or executable target

```
add_executable(dofbot_kinematics_fk src/dofbot_kinematics_fk.cpp)
add_executable(dofbot_kinematics_ik src/dofbot_kinematics_ik.cpp)
target_link_libraries(dofbot_kinematics_fk ${catkin_LIBRARIES}
```

Import header file

```
#include "dofbot_kinemarics.h"
#include
```

Create a manipulator instance

```
Dofbot dofbot = Dofbot();
```

forward kinemarics

```
/**
* 正向运动学计算 根据旋转关节角获取当前位姿
* @param urdf_file 模型文件路径
* @param joints       当前关节角度
* @param cartpos     当前末端位姿
*/
bool dofbot_getFK(const char *urdf_file, vector
```

Sample code

```
// 当前各关节角速度
double joints[]{90, 135, 0, 0, 90};
// 定义目标关节角容器
vector
// 定义位姿容器
vector
// 目标关节角度单位转换,由角度[0,180]转换成弧度[-1.57,1.57]
for (int i = 0; i < 5; ++i) initjoints.push_back((joints[i] - 90) * DE2RA);
// 调取正解函数,获取当前位姿
dofbot.dofbot_getFK(urdf_file, initjoints, initpos);
cout <
cout << "X坐标 (cm): " << initpos.at(0) * 100    << "t"
       << "Y坐标 (cm): "    << initpos.at(1) * 100 << "t"
       << "Z坐标 (cm): "    << initpos.at(2) * 100 << endl;
cout << "Roll   (°):     " << initpos.at(3) * RA2DE << "t"
       << "Pitch (°):     " << initpos.at(4) * RA2DE << "t"
       << "Yaw     (°):  " << initpos.at(5) * RA2DE    << endl;
```

inverse kinemarics

```
/**
* 逆运动学计算 获取到到目标点各关节需要转动的角度
* @param urdf_file   模型文件路径
* @param targetXYZ   目标位置
* @param targetRPY   目标姿态
* @param outjoints   目标点关节角度
*/
bool dofbot_getIK(const char *urdf_file, vector
```

Sample code (note that this is the end-pose, not the end-effector)

```
// 末端的位姿
double Roll = -135;
double Pitch = 0;
double Yaw = 0;
// 求末端位置
double x = 0;
double y = 5.5;
double z = 17.375;
// 末端位置(单位: m)
double xyz[]{x, y, z};
// 末端姿态(单位: 弧度)
double rpy[]{Roll , Pitch, Yaw };
// 创建输出角度容器
vector
// 创建末端位置容器
vector
// 创建末端姿态容器
vector
// 末端位置单位转换,由cm转换成m
for (int k = 0; k < 3; ++k) targetXYZ.push_back(xyz[k] / 100);
// 末端姿态单位转换,由角度转换成弧度
for (int g = 0; g < 3; ++g) targetRPY.push_back(rpy[g] * DE2RA);
// 反解求到达目标点的各关节角度
dofbot.dofbot_getIK(urdf_file, targetXYZ, targetRPY, outjoints);
// 打印反解结果
cout << fixed << "IK kinematics result : " <<    endl;
// 将弧度[-1.57,1.57]转换成角度[0,180]
for (int i = 0; i < 5; i++) cout << outjoints.at(i) * RA2DE + 90    << "t";
```

For details, see **dofbot_kinematics_fk.cpp** and **dofbot_kinematics_ik.cpp** in **dofbot_ws/src/dofbot_moveit/src/**