HERIOT WATT UNIVERSITY

INTELLIGENT ROBOTICS

COURSEWORK 1A

Three Joint Arm Robot Manipulation

Mohit VAISHNAV, *VIBOT*

Supervisor
Dr. Nick TAYLOR

Submission Last Date 16^{th}, Oct.

Contents

List of Figures

1 Introduction

Robots are playing an important roles in today's fast moving world. To work along side humans, robots should have the property to move like them. So robotic arm manipulation plays an important role in research domain. An arm of a robot is made of many joints aligned together in such a way that they can have a smooth movements in the direction desired. With increment in each joints complexity and the difficulty both increases drastically. Each joint and arm could be given a particular orientation and degree of freedom which is to be designed.

Some building blocks associated with any robot are actuators, sensors, kinematics and dynamics. Sensors acts as a interface between physical world and the robot itself. Kinematics helps in understanding of robotic movements and the functions it could perform whereas dynamics gives the idea of joint angles for any position specified by the user. It provides the solution for placing a robotic arm.

2 Inverse Kinematics

It refers to the process where given the tip location, arm has to reach that point and all the calculations are done for the joint to align properly. Doing this, brings more than one solution into picture and it becomes difficult to decide. As an outcomes of inverse kinematics, some of the problems could be mentioned as possibility of non existent solution and singularities other than multiple solution. This problem is more persistent with increasing number of degree of freedom and there is redundancy in movement. Even non existent solution could be of different types like outside the reach of arm, physical constraint to reach to that point, or infinite acceleration because of singularity.

2.1 Singularity

Inverse kinematics brings singularities [2] into picture. While moving an arm. sometimes it becomes difficult remap Cartesian space to joint space. This positions are known as degeneracies or singularities. In a situation of singularity there are infinite solutions to reach to a particular position. Following which if the optimal solutions is not found out then joints reach to a position where there is no way to move ahead. Mobility of the arm gets reduced at any such point. There are different kind of singularity each causing a different issue. Boundary or work-space singularity are the most common form of it. Other form of singularity is internal or joint space. They are caused by the alignment of axes in space. For example, a rotation and counter rotation in two axes can lead to indeterminate location. At this point there could be infinite solutions for inverse kinematics equation. There is also a possibility of infinite joint velocity for small Cartesian motion which in itself is problematic. Such points could be located by analyzing the Jacobian matrix of manipulator. Usually a 6 axis robot has three different singularities wrt Fig. 1:

Wrist It makes the robot wrist (Joint 4 & 5) to spin 180° when its axis line up.

Shoulder This occurs when joint 1 is aligned with wrist which makes joint 4 & 1 to spin 180° continuously. *Alignment* singularity is another subset of this kind in which joint 1 & last are aligned together.

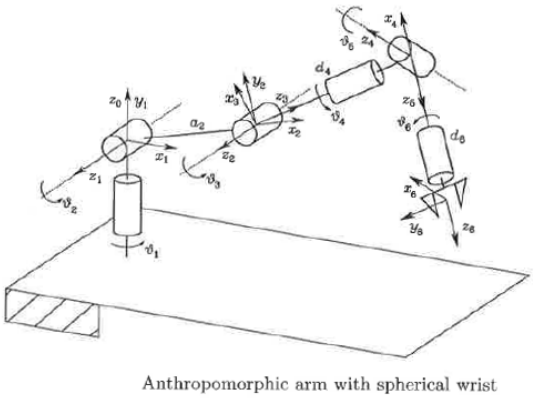

Figure 1: Robotic Arm (source: [2])

Elbow When joint 2 & 3 are in the same plane as that of wrist, it causes *elbow singularity*. It locks the elbow to a particular position.

2.2 Avoiding Singularity

During earlier days robot arm could be seen breaking down when encountering a singularity which makes on joint moves too fast. Now a days maximum speed is defined for each joint to avoid infinity type of situation. As soon as the singularity is passed, the robotic arm continues with the rest of the work with earlier velocity. There is a lot of mathematics associated with solving singularity and can be seen in tutorial [3]. Usually singularities appear in case of nearly zero degree joints so small angles are added to reduce this occurrence. Another approach to avoid this is the divide the work space in such a way that singularities are neglected.

2.3 Velocity Analysis

Along with the aim of reaching to the target, it is essential for many purposes to have a uniform speed. It helps in maintaining consistency in work (eg. welding/painting). So, the end effector velocity takes into account the joint velocities. It could be mathematically written as:

$$\dot{p} = J\dot{q}$$

where, $\dot{p}$ is the end effector velocity and $\dot{q}$ is the joint velocity (joint angles). Thus, inverse of Jacobian matrix (if non singular) provides the desired joint velocity.

2.4 Ambiguity

Calculating inverse causes many problems because there are more than one solution even in the range of one cycle (360°).

$$sin^{-1}y = x, \pi - x, \quad cos^{-1}y = x, -x, \quad tan^{-1}y = x, \pi + x$$

Similarly even for cosine and tangent there is ambiguity while taking inverse. Hence it is visible that there are more than one ways to reach any valid location asked.

3 Accuracy

In a robotic system, accuracy is controlled using resolution of the control system, solving mechanical inaccuracies and minimizing error. This error is estimated in Cartesian Space. Repeatability of the robot where it can again re-position itself is also a static measurement to check accuracy. Resolution of the control system is the minimum incremental change which a robot can identify.

4 Implementation

To make the robot move, first the aim was to put in place the right equations as shown below.

```
float theta234 = alpha;
theta1 = (float) Math.atan2(py,px);
float p1 = (float) (px* Math.cos(theta1) + py * Math.sin(theta1) - d4 * Math
    .cos(theta234));
float p2 = (float) (pz - d4 * Math.sin(theta234));
float c3 = (float) (p1*p1 + p2*p2 - d2*d2 - d3*d3)/(2*d2*d3);
theta3 = (float) Math.atan2(Math.sqrt(1-(c3*c3)),c3);
theta2 = (float) (Math.atan2(((d3* c3 + d2)* p2 - (d3 * Math.sin(theta3) *
    p1)), ((d3* c3 + d2)* p1 + (d3 * Math.sin(theta3) * p2))));
theta4 = (float) (alpha - theta2 - theta3);
```

Now to solve the degeneracy, boundary condition limit is taken into consideration. As the animation shows that the robot cannot have the workspace beyond 10m in all three coordinate system. So before taking anything into consideration, *px, py & pz* are checked using:

```
if (Math.abs(px) <=10 && Math.abs(py) <=10 && Math.abs(pz) <=10 && (Math.
    sqrt(sqr(px) + sqr(py) + sqr(pz)) <= 10))
```

Individually they cannot be greater than 10m in length for any direction and also their effective length has to be less than 10m because of the workspace sphere radius.

Now each of the boundary limit can be explicitly defined (like for pz) i.e when the position of the joint are all parallel to one another and forms a straight line.

$$if(px\ is\ 0,\ py\ is\ 0,\ (d2+d3+d4)\ is\ pz) \quad \{\theta_2 = (90^\circ);\ \theta_3 = (0^\circ);\ \theta_4 = (0^\circ);\}$$

Taking the condition even further, it can be stated that based on the quadrant θ_1 varies. As seen in above solutions that while taking inverse there are multiple solutions and one has to be selected. So in case of θ_1 where there is inverse of *tan* involved, either solution is the obtained or there is a factor of π. Hence in case where in the XY plane, quadrant is third or forth, factor of π has to be added.

Another constraint that is added to solve next degeneracy is taking the absolute value of Z coordinate. If it happens to be less than arm length of $d_2 + d_3$. This is done to check the movement of the arm and decide if it has to approach with elbow upwards or vice verse. So, θ_2 and θ_3 are taken in reverse order like.

$$\theta_3 = -\theta_3 + 2\pi$$

$$\theta_2 = -\theta_2 + 2\pi$$

5 Difficulties

Static friction is always said to be the highest from all three kinds of friction. This is true anywhere and applies here too. To start the program, basic libraries have to be installed in the personal computer which took quiet a bit of time. Once everything is in place, coding part started. Based on the lecture slides provided, it became easy to understand the dynamics and kinematics of the arm. As the variables are dependent on each other, it is very much essential to follow the right order else the maneuver is totally opposite from what is desired.

Even while solving degeneracies, it looked so difficult that if one is solved, the other becomes prominent and arm takes another weird turns. While solving for second kind of degeneracy, if the condition is taken for negative values of Z coordinate, as soon the arm crosses $Z = 0$, all the calculations changes and the arm end effector does not reach to the right position. Also, if the manipulator is checked using the interface provided it looks so visible to solve it but still was not possible.

6 Conclusion

This coursework is one of the most fundamental step in understanding the robotics kinematics. Theoretically and conceptually it made many things clear but when it comes to code such a program, real work is known. Though it looks so easy to understand those complexities but all could not be put to code and make the robot move as desired.

References

[1] http://home.iitk.ac.in/∼avinash/ME%20600.pdf

[2] https://robohub.org/3-types-of-robot-singularities-and-how-to-avoid-them/

[3] https://www.researchgate.net/profile/Bruno_Siciliano/publication/220061834_Kinematic_Control_of_Redundant_Robot_Manipulators_A_Tutorial/links/09e415134a0d011178000000.pdf

[4] http://www.cs.columbia.edu/∼ allen/F15/NOTES/jacobians.pdf

[5] http://www.ccs.neu.edu/home/rplatt/cs5335_fall2017/slides/inverse_kinematics.pdf

[6] https://www.seas.upenn.edu/∼ meam520/notes02/IntroRobotKinematics5.pdf

[7] http://home.deib.polimi.it/gini/robot/docs/Rob05.pdf

```
1       // To check if the coordinate is out of bound
2       if (Math.abs(px) <=10 && Math.abs(py) <=10 && Math.abs(pz) <=10 && (
    Math.sqrt(sqr(px) + sqr(py) + sqr(pz)) <= 10))
3       {
4         if (px == 0 && py ==0 && (d2+d3+d4)==pz)
5         {
6           theta2 = (float)  Math.toRadians(90);
7           theta3 = (float)  Math.toRadians(0);
8           theta4 = (float)  Math.toRadians(0);
9         }
10        else
11        {
12
13        float theta234 = alpha;
14        if (py < 0 || (py==0 && px < 0))
15        {
16          theta1 = (float) Math.atan2(py,px) + (float)  Math.toRadians(180);
     // k = 1;
17        }
18        else
19        {
20          theta1 = (float) Math.atan2(py,px); // k = 0;
21        }
22
23        float p1 = (float) (px* Math.cos(theta1) + py * Math.sin(theta1) -
    d4 * Math.cos(theta234));
24        float p2 = (float) (pz - d4 * Math.sin(theta234));
25        float c3 = (float) (p1*p1 + p2*p2 - d2*d2 - d3*d3)/(2*d2*d3);
26
27        if (Math.abs(pz) <= (d2+d3))
28        {
29          //theta3 =-(float)Math.atan2(Math.sqrt(1-(c3*c3)),c3)+ (float)
    Math.toRadians(360) ;
30
31          theta3 = (float) Math.atan2(Math.sqrt(1-(c3*c3)),c3);
32          theta2 = (float) (Math.atan2(((d3* c3 + d2)* p2 - (d3 * Math.sin(
    theta3) * p1)), ((d3* c3 + d2)* p1 + (d3 * Math.sin(theta3) * p2))));
33
34        }
35        else
36        {
37          //theta3 = (float) Math.atan2(Math.sqrt(1-(c3*c3)),c3);
38
39          theta3 =-(float)Math.atan2(Math.sqrt(1-(c3*c3)),c3)+ (float)  Math
    .toRadians(360) ;
40          theta2 =-(float) (Math.atan2(((d3* c3 + d2)* p2 - (d3 * Math.sin(
    theta3) * p1)), ((d3* c3 + d2)* p1 + (d3 * Math.sin(theta3) * p2)))) + (
    float)  Math.toRadians(360) ;
41
42        }
43        //theta3 = (float) Math.atan2(Math.sqrt(1-(c3*c3)),c3); // Math.
    atan2(-Math.sqrt(1-(c3*c3)),c3)
44        // theta2 = (float) (Math.atan2(((d3* c3 + d2)* p2 - (d3 * Math.sin(
    theta3) * p1)), ((d3* c3 + d2)* p1 + (d3 * Math.sin(theta3) * p2))));
45        theta4 = (float) (alpha - theta2 - theta3);
46
47
48        }
49      }
50
```

```
51        //if ((d2+d3)>=pz)
52        //{
53        //  theta3 = -theta3;
54        //}
55        //theta4 = (float) (alpha - theta2 - theta3);
56      }
57
58
59      /**
```