4 is harder than 6:
Inverse kinematics for underactuated robots

Peter Corke

February 2014

Many low-cost hobby class robots, such as shown in Figure 1 have only 4 joints (degrees of freedom). This document describes how to determine inverse kinematics for such a robot using the Robotics Toolbox[1] for MATLAB.

Underactuation complicates the process of finding an inverse kinematic solution, and it frustrates those who are new to robotics — those who just want to run the code and get an answer. For a robot with 6 joints it's quite straightforward, but underactuation requires some careful thought about the problem that you are trying to solve — you can't just blindly use the tools.

We will consider the problem in two parts. First the problem of moving the robot tool to a particular position. Second, moving the tool to a particular position and tool orientation.

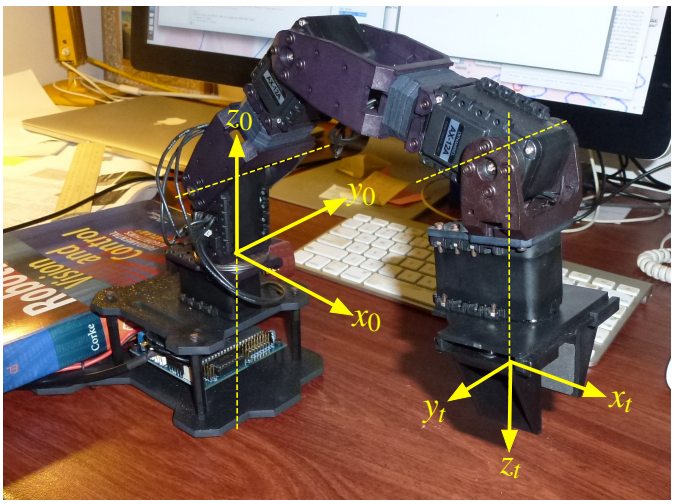


Figure 1: The PhantomX robot sitting on my desk. The world frame {0} and the tool frame {t} are shown. Note I'm using a book over the back feet to keep it from toppling over.

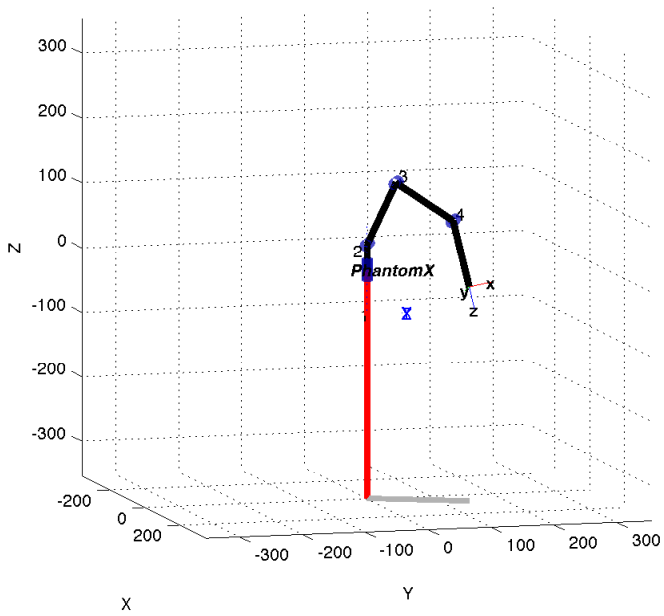


Figure 2: The PhantomX robot in a pose that might be useful for grabbing an object off the table top. Here the joint angles are $(1.12, 0.441, -1.69, -0.770)$ rad.

1 Moving to a position

We start by loading a model of a 4DOF robot, in this case the PhantomX as shown in Figure 1.

```
>> mdl_phantomx
```

and then plot it for default, all zero, joint angles

```
>> px.plot(qz)
```

and we see that the robot is pointing straight upwards. We can use the sliders in the teach pendant function

```
>> px.teach
```

to move the robot to the sort of pose that might be used for picking up an object as shown in Figure 2.

Now we will define the position we want the tool tip to move to

```
>> Td = transl([100 80 -70])
```

and then compute the numerical inverse kinematics


```
                                    only care about x, y, z position
>> q = px.ikine(Td, qz, [1 1 1 0 0 0])
Warning: Initial joint angles results in near-singular configuration, this may slow converge
> In SerialLink.ikine at 140
        new RBT version： 0.6747 1.7646 -2.2161 2.2219
q =
    0.6747    2.6132    1.9831   -2.3849
```

Note that we have specified a mask value of $(1, 1, 1, 0, 0, 0)$ which indicates that we only care about errors in the x-, y- and z-directions, rotational errors are to be ignored. The error message simply indicates that the Jacobian, required at each iteration, is singular for the initial set of joint angles.

```
>> px.plot(q)
```

which is shown in Figure 3(a). While this looks rather awkward the tool position is indeed what we requested

```
>> px.fkine(q)
ans =
    0.7746    0.6247    0.0986  100.0000
    0.6197   -0.7809    0.0789   80.0000
    0.1263   -0.0000   -0.9920  -70.0000
         0         0         0    1.0000
```

as indicated by the top three values in the rightmost column.

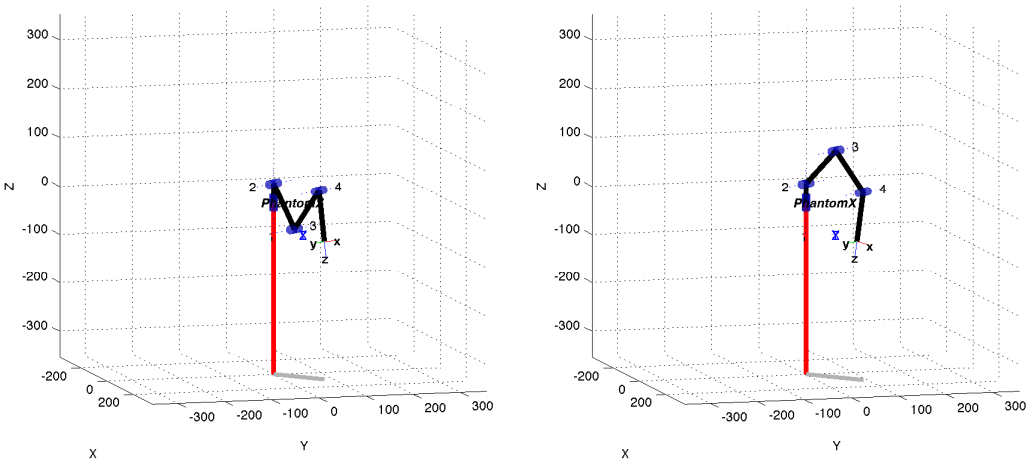
Inverse kinematics is, in general, not uniquely defined. There are several arm configurations that will give the same tool position. The numerical method has just chosen the awkward one. In the less awkward configuration we chose earlier using the teach pendant we see that q_3 was negative. With a bit of trial and error, guided by the configuration we achieved using the teach pendant, we find that

```
>> q = px.ikine(Td, [0.5 1 -1 -0.5], [1 1 1 0 0 0])
q =
    0.6747    0.8000   -1.6280   -0.8665
>> px.plot(q)
```

gives a reasonable looking solution, as shown in Figure 3(b).

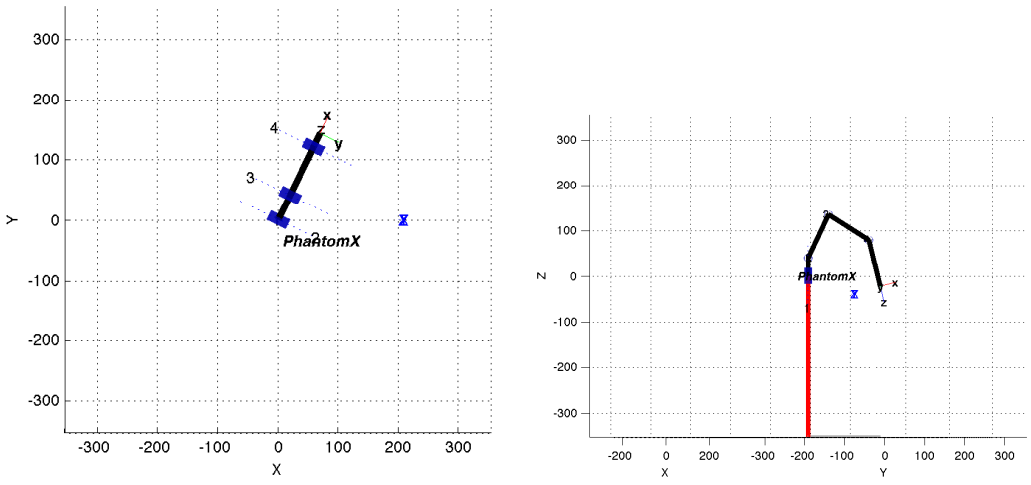
2 Moving to a pose

Looking from above as shown in Figure 4(a) we can see that the arm is a straight line and its angle is a function only of q_1, the waist joint. The effective length of the line is a function of q_2, q_3 and q_4. The height of the tool tip, as shown in Figure 4, is also a function of q_2, q_3 and q_4. So while there is only one possible value of q_1, there are infinite number of values of q_2, q_3 and q_4 — it is a 3-bar linkage pinned at each end. Each particular choice of q_2, q_3 and q_4 results in a different orientation of the tool's z-direction — this is an unconstrained degree of freedom.



(a) (b)

Figure 3: Two solutions for the tool tip position $(100, 80, -70)$.



(a) (b)

Figure 4: Two views of the robot at tool tip position $(100, 80, -70)$.

3 Moving to a pose

Now we want to control the orientation of the tool as well. The construction of the robot only allows us to control the tool z-direction. The only way we could change the tool x- or y-direction would be to rotate q_1 but this would violate the position requirement. For an underactuated robot arm it is critically important to understand what the mechanism can and cannot do, particularly when we are talking about controlling pose.

The first step is to specify the desired pose. If we look at the value we set above

```
>> Td
Td =
     1     0     0   100
     0     1     0    80
     0     0     1   -70
     0     0     0     1
```

we see that the rotation part is an identity matrix, which has desired tool z-direction pointing upward in the world z-direction, as defined by the third column. This is impossible for the robot to achieve, to be reaching down towards the table top with its finger tips pointing upwards, as well as being non-useful for picking up an object on the table. We need to set the orientation part of the desired tool pose and we do that by

```
>> Td = transl([100 80 -70])*oa2tr([0 -1 0], [0 0 -1])
Td =
     1     0     0   100
     0    -1     0    80
     0     0    -1   -70
     0     0     0     1
```

```
In RBT 10 version:
0  -1 0  100
0  0  -1 80
-1 0  0  -70
0  0  0  1
```

which postmultiplies by a rotational transform that has its z-axis in the negative world z-direction (downward) and its y-axis in the negative world y-direction. The third column now indicates negative world z-direction.

The only remaining degree of freedom is evident when we look at the robot side on, the tool z-axis is not pointing quite downward. In order to change things so that it is, we must rotate the end-effector frame

```
>> q = px.ikine(Td, [0.5 1 -1 -0.5], [1 1 1 1 0 0])
Warning: Initial joint angles results in near-singular configuration, this may slow converge
> In SerialLink.ikine at 140

q =

    0.6747    0.6954   -1.8288   -0.6174

>> px.fkine(q)

ans =

    0.7809    0.6247    0.0000  100.0000
```



```
In RBT 10 version:
q = px.ikine(Td, 'q0', [0.5 0.5 -1 -0.5], 'mask', [1 1 1 1 0 0], 'tol', 0.8)

q =

    0.6737    0.7993   -1.6284   -0.8671
```

```
    0.6247   -0.7809    0.0000   80.0000
    0.0000   -0.0000   -1.0000  -70.0000
         0         0         0    1.0000
>> px.plot(q)
```

```
ans =
    0.1403   -0.7514    0.6447       100
    0.1120   -0.6349   -0.7644     79.86
    0.9837    0.1795   -0.0049    -69.92
         0         0         0         1
```

4 4 DOF arm

The fundamental problem is that we normally specify the pose of the robot's end-effector in terms of the full orientation of the frame $\{E\}$. However with only one orientation degree of freedom we cannot achieve an arbitrary pose. At best we can specify the direction of one of the unit-vector columns of the orthonormal rotation matrix — the n, o or a vector. If we want the gripper jaws pointing straight down then $\boldsymbol{a} = [00-1]$. If we want the gripper jaws in the horizontal plane then $\boldsymbol{n} = [001]$. However if we wanted the gripper jaws to pointing downwards at 45 degrees the orientation would be a function of the gripper position — we lose the benefit of decoupling tool position and orientation.

Consider instead that the tool orientation was expressed with respect a new frame $\{V\}$ which has its origin coincident with $\{E\}$ but its y- and z-axes lie in the world horizontal plane.

References

[1] P. I. Corke, *Robotics, Vision & Control: Fundamental Algorithms in MATLAB*. Springer, 2011. ISBN 978-3-642-20143-1.

