Denavit-Hartenberg Notation

Forward Kinematics

Inverse Kinematics

Steven Dirven, Massey University, 2015

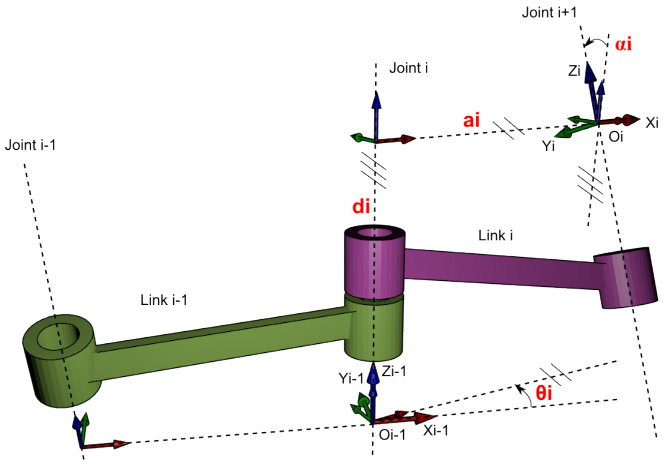
Revisiting Denavit-Hartenberg (DH) Notation

- Serial link manipulator comprises of bodies, called links, and joints.
- Each joint has 1 Degree of Freedom (Translation or Rotation)
- Denavit-Hartenberg Notation is a method of assigning coordinate frames onto a serial link robot to describe the relationships between its links and joints

http://www.asiamattersforamerica.org/sites/default/files/field/image/Automation_of_foundry_with_robot-290x290.jpg

Denavit-Hartenberg

Coordinate frame i is fixed at the distal end of link i (furtherest from base, closest to end effector) with z in the axis of the distal joint i+1.



http://upload.wikimedia.org/wikipedia/commons/d/d3/Classic-DHparameters.png

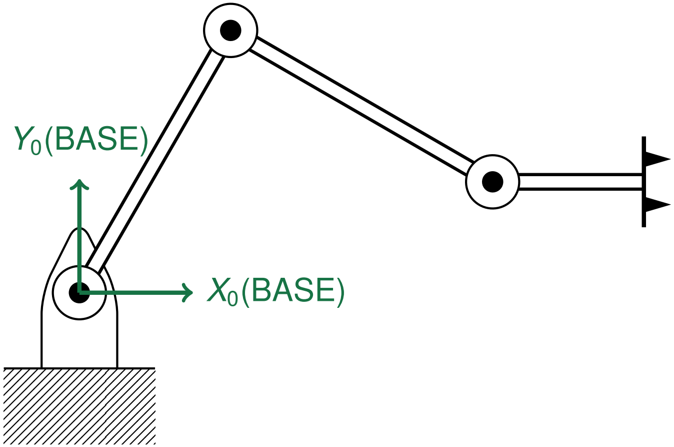
Denavit-Hartenberg : Rules for Assigning Frames

Coordinate frame i is fixed at the distal end of link i (furtherest from base, closest to end effector) with z in the axis of the distal joint i+1.

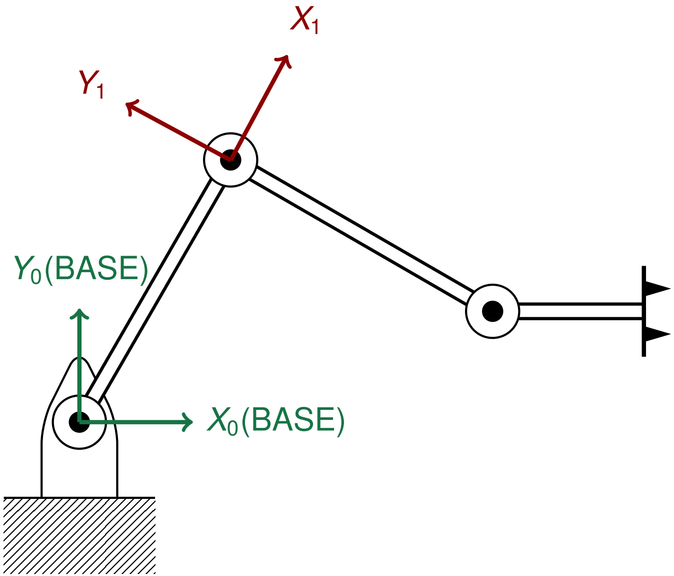
DH Coordinate System Assignment

1. The z axis is in the direction of the joint axis
2. The x axis is parallel to the common normal ($x_j = z_{j-1} \times z_j$)
3. The y axis is assigned to complete a right-handed coordinate system

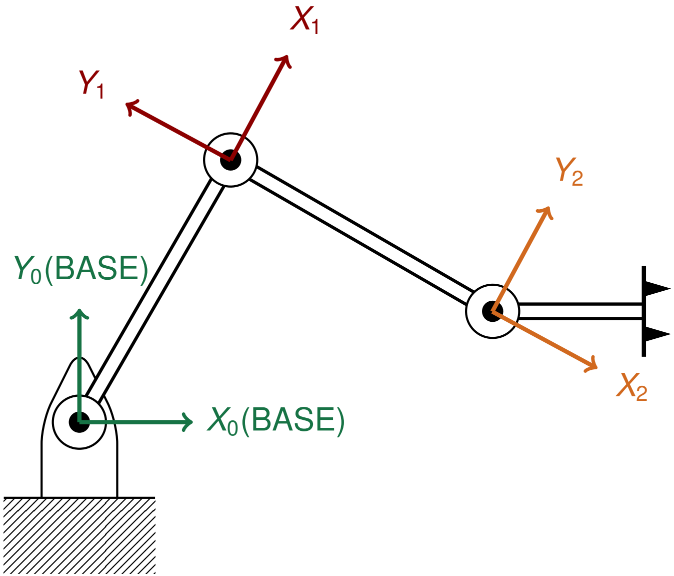
Denavit-Hartenberg : 2D 3 Link Assigning Frames



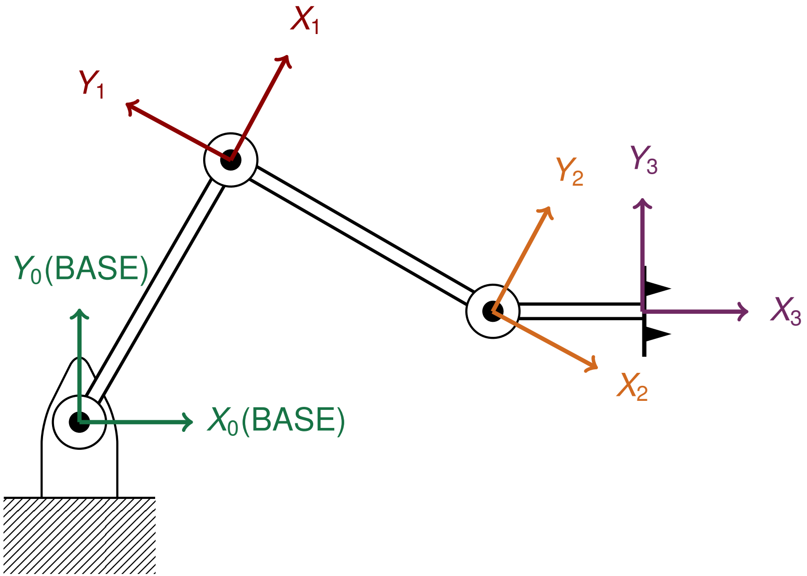
Denavit-Hartenberg : 2D 3 Link Assigning Frames



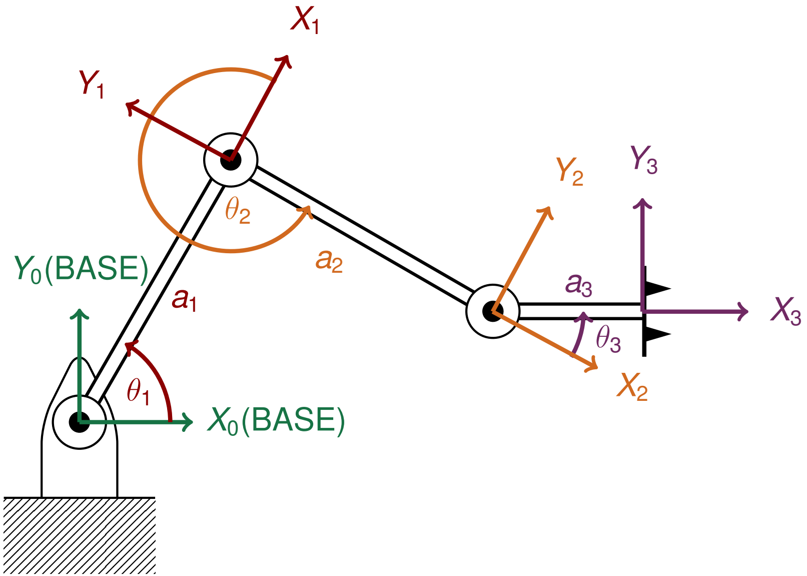
Denavit-Hartenberg : 2D 3 Link Assigning Frames



Denavit-Hartenberg : 2D 3 Link Assigning Frames



Denavit-Hartenberg : 2D 3 Link Assigning Frames



Denavit-Hartenberg : DH Parameters

DH parameters describe how the coordinate system j at the distal end of link j can be related back to the coordinate system located at joint j at its proximal end.

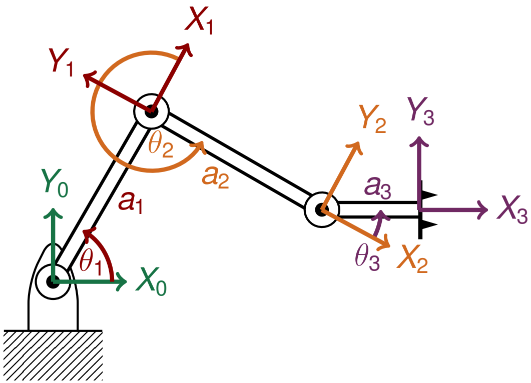
The coordinate system located at a links' proximal end is the coordinate system of the previous link (coordinate system j-1).

DH Parameters

θ_j	Angle	Angle about z_{j-1}, from x_{j-1} to x_j
d_j	Offset	Offset along z_{j-1} to the common normal
a_j	Length	Length of the common normal
α_j	Twist	Angle about common normal, from z_{j-1} to z_j

Denavit-Hartenberg : 2D 3 Link DH Parameters

A trivial solution given that there are no offsets (d_j) or twists (α_j)



DH Parameters

Link	θ_j	d_j	a_j	α_j
1	θ_1	0	a_1	0
2	θ_2	0	a_2	0
3	θ_3	0	a_3	0

Denavit-Hartenberg : Peter Corke Toolbox (PCT)

In practice, after finding the Denavit-Hartenberg parameters, a mathematical tool would be used.

Robotics, Vision and Control - Peter Corke

The MATLAB Peter Corke Toolbox can be downloaded for free from:

`http://www.petercorke.com/Robotics_Toolbox.html`

Installation instructions are available on the website.

Denavit-Hartenberg : Peter Corke Toolbox (PCT) (2D 3 Link)

DH Parameters Example code

Link	θ_j	d_j	a_j	α_j	Syntax: Link([θ_j,d_j,a_j,α_j,T_j)]
1	θ_1	0	$a_1 = 2$	0	L(1) = Link([0,0,2,0,0])
2	θ_2	0	$a_2 = 2$	0	L(2) = Link([0,0,2,0,0])
3	θ_3	0	$a_3 = 1$	0	L(3) = Link([0,0,1,0,0])

Where T_j = 0 for a revolute joint, or 1 for a prismatic joint.
The joint variable is replaced with a zero as a placeholder.
This will not be constrained to zero

Denavit-Hartenberg : Peter Corke Toolbox (PCT) (2D 3 Link)

DH Parameters Example code

```
L(1) = Link([0,0,2,0,0])
L(2) = Link([0,0,2,0,0])
L(3) = Link([0,0,1,0,0])
ThreeL = SerialLink(L,’name’,’3 Link’)
ThreeL
ThreeL.plot([0 0 0])
ThreeL.plot([pi/4 pi/4 pi/4])
```

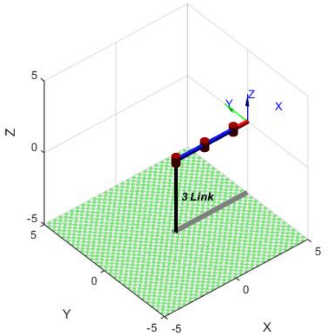
This turns the DH Parameters into the description of the 2D 3 Link RRR planar robot from before into a SerialLink object ThreeL.

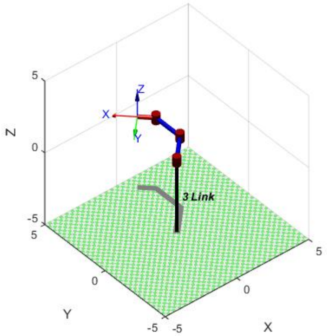
The variable names are arbitrary, the functions are provided by the PCT.

Denavit-Hartenberg : Peter Corke Toolbox (PCT) (2D 3 Link)

DH Parameters Example code

```
Syntax:  ThreeL.plot([θ1 θ2 θ3])
ThreeL.plot([0 0 0])
ThreeL.plot([pi/4 pi/4 pi/4])
```





Kinematics : Background

The process of describing a serial robot can be broken down into the following steps:

1. Assign co-ordinate systems (The Denavit-Hartenberg Technique)
2. Determine the transformation matrices by combining the translation and rotation aspects

From this, there are two types of problem to solve:

- Given the joint configurations, what is the pose of the robot?
- Given the pose of the robot, what are the joint configurations?

These are known as **forward** and **inverse** kinematics repsectively

Kinematics : Forward Kinematics

Forward Kinematics: Given the joint configurations (variables) what is the pose of the robot?

Forward Kinematics Technique

1. Assign DH Parameters to describe the coordinate systems' relationships
2. Substitute these parameter values into the DH Transform
3. Multiply these transforms together to find the overall transform relative to the base.

Forward Kinematics : DH Transform from Basic Transforms

The DH Transform can be found by multiplying the elementary transformations of:
Rotate by θ_j, translate by d_j, translate by a_j, and finally, rotate by α_j.

DH Transform

$$^{j-1}A_j(\theta_j, d_j, a_j, \alpha_j) = T_{Rz}(\theta_j) T_z(d_j) T_x(a_j) T_{Rx}(\alpha_j)$$

$$= \begin{bmatrix} cos(\theta_j) & -sin(\theta_j) & 0 & 0 \\ sin(\theta_j) & cos(\theta_j) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_j \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & \alpha_j \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\alpha_j) & -sin(\alpha_j) & 0 \\ 0 & sin(\alpha_j) & cos(\alpha_j) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^{j-1}A_j = \begin{bmatrix} cos(\theta_j) & -sin(\theta_j)cos(\alpha_j) & sin(\theta_j)sin(\alpha_j) & a_j cos(\theta_j) \\ sin(\theta_j) & cos(\theta_j)cos(\alpha_j) & -cos(\theta_j)sin(\alpha_j) & a_j sin(\theta_j) \\ 0 & sin(\alpha_j) & cos(\alpha_j) & d_j \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Forward Kinematics : Combining DH Transforms

Each joint is expressed in the DH Transform with the following nomenclature : ${}^{j-1}A_j$

These can be combined in the following manner:

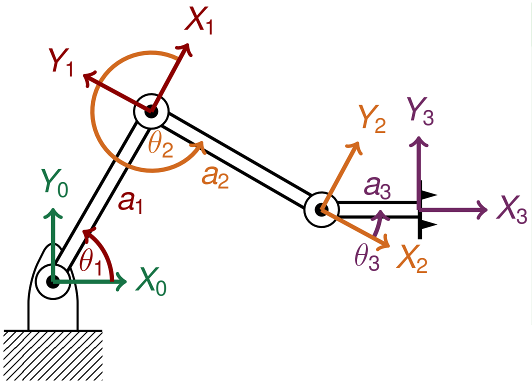
Combining DH Transforms by multiplication

$${}^0T_n = {}^0A_1 \times {}^1A_2 \ldots \times {}^{n-1}A_n$$

Remember that matrix multiplication is not commutative. That is $X \times Y \neq Y \times X$

Forward Kinematics : Step 1 : DH Parameters (2D 3 Link)

A trivial solution given that there are no offsets (d_j) or twists (α_j)



DH Parameters

Link	θ_j	d_j	a_j	α_j
1	θ_1	0	a_1	0
2	θ_2	0	a_2	0
3	θ_3	0	a_3	0

Forward Kinematics : Step 2 : DH Transforms (2D 3 Link)

Substitute values into the DH Transform for each link. For example ($a_1 = 2$, $a_2 = 2$, $a_3 = 1$)

Link	θ_j	d_j	a_j	α_j
1	θ_1	0	2	0
2	θ_2	0	2	0
3	θ_3	0	1	0

DH Substitution

$$^{0}A_1 = \begin{bmatrix} cos(\theta_1) & -sin(\theta_1) & 0 & 2cos(\theta_1) \\ sin(\theta_1) & cos(\theta_1) & 0 & 2sin(\theta_1) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^{1}A_2 = \begin{bmatrix} cos(\theta_2) & -sin(\theta_2) & 0 & 2cos(\theta_2) \\ sin(\theta_2) & cos(\theta_2) & 0 & 2sin(\theta_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^{2}A_3 = \begin{bmatrix} cos(\theta_3) & -sin(\theta_3) & 0 & cos(\theta_3) \\ sin(\theta_3) & cos(\theta_3) & 0 & sin(\theta_3) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Forward Kinematics : Step 3 : Overall Transform (2D 3 Link)

Overall transform

$$^0T_n = ^0A_1 \times ^1A_2 \ldots \times ^{n-1}A_n$$

Replaced the following for compactness s = sin, c = cos

$$^0T_3 = \begin{bmatrix} c\theta_1 & -s\theta_1 & 0 & 2c\theta_1 \\ s\theta_1 & c\theta_1 & 0 & 2s\theta_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c\theta_2 & -s\theta_2 & 0 & 2c\theta_2 \\ s\theta_2 & c\theta_2 & 0 & 2s\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c\theta_3 & -s\theta_3 & 0 & c\theta_3 \\ s\theta_3 & c\theta_3 & 0 & s\theta_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Begin multiplying together:

$$\begin{bmatrix} c\theta_1 c\theta_2 - s\theta_1 s\theta_2 & -c\theta_1 s\theta_2 - s\theta_1 c\theta_2 & 0 & 2c\theta_1 c\theta_2 - 2s\theta_1 s\theta_2 + 2c\theta_1 \\ s\theta_1 c\theta_2 + c\theta_1 s\theta_2 & -s\theta_1 s\theta_2 + c\theta_1 c\theta_2 & 0 & 2s\theta_1 c\theta_2 + 2c\theta_1 s\theta_2 + 2s\theta_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c\theta_3 & -s\theta_3 & 0 & c\theta_3 \\ s\theta_3 & c\theta_3 & 0 & s\theta_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

= Too much work

Forward Kinematics : MATLAB PCT (2D 3 Link)

MATLAB Example : DH Parameters

```
L(1) = Link([0,0,2,0,0])
L(2) = Link([0,0,2,0,0])
L(3) = Link([0,0,1,0,0])
ThreeL = SerialLink(L,'name','3 Link')
```

```
 ThreeL = 3 Link (3 axis, RRR, stdDH, fastRNE)
+---+-----------+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |    offset |
+---+-----------+-----------+-----------+-----------+-----------+
| 1 |         q1|          0|          2|          0|          0|
| 2 |         q2|          0|          2|          0|          0|
| 3 |         q3|          0|          1|          0|          0|
+---+-----------+-----------+-----------+-----------+-----------+

grav =     0  base = 1   0   0   0   tool =  1   0   0   0
           0         0   1   0   0           0   1   0   0
        9.81         0   0   1   0           0   0   1   0
                     0   0   0   1           0   0   0   1
```

Forward Kinematics : MATLAB PCT (2D 3 Link)

MATLAB Example : Forward Kinematics

```
Syntax:   name.fkine([q1 q2 ... qn])
Case1 = ThreeL.fkine([0 0 0])
Case2 = ThreeL.fkine([pi/4 pi/4 pi/4])
```

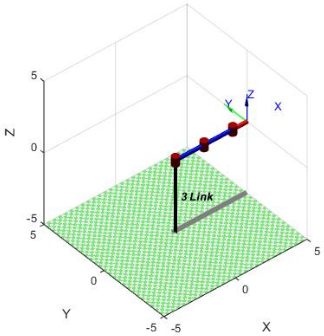
```
Case1 =

     1     0     0     5
     0     1     0     0
     0     0     1     0
     0     0     0     1

Case2 =

   -0.7071   -0.7071         0    0.7071
    0.7071   -0.7071         0    4.1213
         0         0    1.0000         0
         0         0         0    1.0000
```

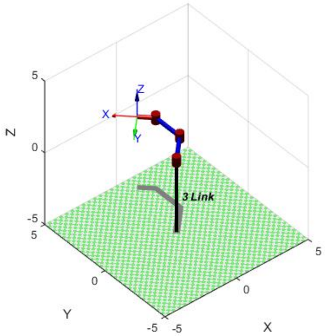
Forward Kinematics : MATLAB PCT (2D 3 Link)

Forward kinematics result

DH Parameters Example code

```
ThreeL.plot([0 0 0])
ThreeL.plot([pi/4 pi/4 pi/4])
```





MATLAB PCT Additional Features

The Peter Corke Toolbox has many useful methods for analysis of serial robots:

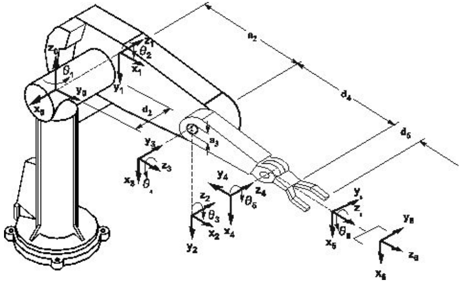
PCT Helpful Methods

```
name.n % Returns number of joints
name.links % The vector of link objects
name.RP % Returns if revolute or prismatic
name.a % Returns a, length of common
normal
name.offset % Returns the offset parameter
name.plot % Graphs the Forward Kinematic
Pose
```

There are many more...

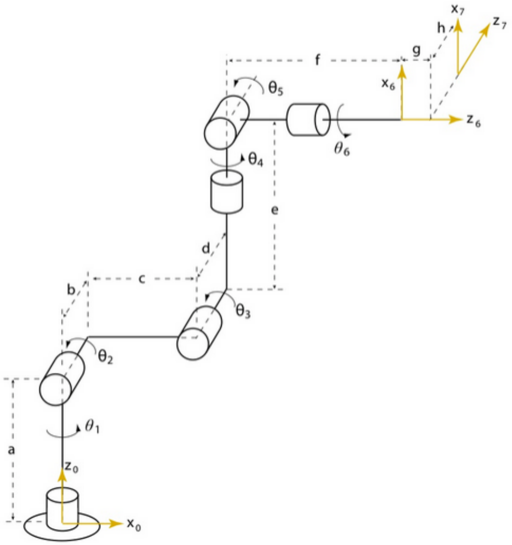
Forward Kinematics : PUMA Robot Background

Programmable Universal Manipulator for Assembly (PUMA)



http://www.ece.gatech.edu/academic/courses/ece4007/09spring/ece4007l01/ws5/images/head1.gif

Forward Kinematics : PUMA Robot Background

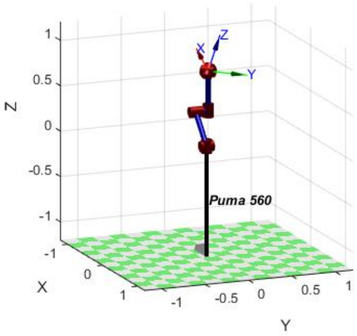


http://medesign.seas.upenn.edu/index.php/Courses/MEAM520-11A-P02

Forward Kinematics : MATLAB PCT (PUMA)

MATLAB PUMA Example

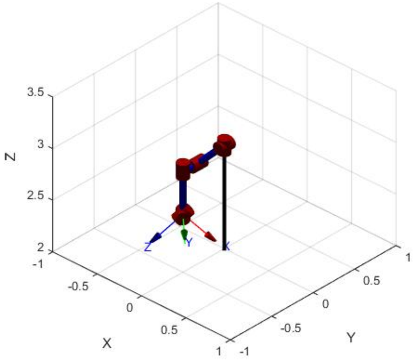
```
mdl_puma560
qpose = [pi pi/4 -pi/4 -pi/4 -pi/4 pi]
p560.plot(qpose)
```



Forward Kinematics : MATLAB PCT (PUMA)

MATLAB PUMA Example

```
mdl_puma560
p560.base = transl(0,0,3)*trotx(pi)
p560.kine(qpose)
```



Forward Kinematics : MATLAB PCT (PUMA Trajectory)

It is possible to specify many joint configurations, for which the forward kinematics technique can be applied to find the poses. This represents a trajectory, or path, of the robot with respect to time

Example Trajectory (PUMA)

$$\text{trajectory} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & .04 & -.04 & 0 & 0 & 0 \\ 0 & .2 & -.2 & 0 & 0 & 0 \\ 0 & .6 & -.6 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Example Code for Trajectory (PUMA)

```
trajectory = [0 0 0 0 0 0;0 .04 -.04 0 0
0;0 .2 -.2 0 0 0;0 .6 -.6 0 0 0;0 1 1 0 0
0]
```

Forward Kinematics : MATLAB PCT (PUMA Trajectory)

http://www.asiamattersforamerica.org/sites/default/files/field/image/Automation_of_foundry_with_robot-290x290.jpg

Inverse Kinematics : Background

Inverse kinematics solves the opposite problem of forward kinematics.

It solves for the joint angles, given the location and pose of the end effector.

Why would we like to do this, and how can it be done?

http://www.asiamattersforamerica.org/sites/default/files/field/image/Automation_of_foundry_with_robot-290x290.jpg

Inverse Kinematics : Calculation Methods

The methods for inverse kinematics can be divided into:

1. Closed-form (CF) analytical solution
   - A necessary condition for a closed-form solution of a 6-axis robot is that the three wrist axes intersect at a certain point (Spherical wrist).
2. Numerical (N) solution methods (Iterative)
   - This method can be applied to all types of robots, though it is computationally expensive.

Inverse Kinematics : Necessary Conditions for Closed Form (CF)

How do the necessary conditions for a closed form solution help us solve the inverse kinematics problem?

- Because the robot has a spherical wrist, the last 3 links can only achieve changes in orientation
- Therefore, the first 3 links dictate the translation in three dimensions. The joint configurations to solve this problem only has a limited number of solutions (maximum of 8)
- Complex transformation is broken up into separate translation and rotation problems that can be solved

Inverse Kinematics : Advantages and Disadvantages of CF and N methods

	ADVANTAGES	DISADVANTAGES
Closed Form	Very fast to calculate, perfectly accurate solution	Cannot always be used because it requires the necessary conditions
Numerical	Can be calculated for any number of links (even for redundant actuators)	Computationally expensive calculations, does not always converge to most useful configuration

Inverse Kinematics : MATLAB PCT PUMA CF and N

Inverse Kinematics (PUMA Closed form)

```
mdl_puma560
pose = [pi pi/4 -pi/4 -pi/4 -pi/4 pi]
Transform = p560.fkine(pose)
pose_inverse = p560.ikine(Transform)
pose_inverse2 = p560.ikine6s(Transform)
```

```
pose          = 3.1416   0.7854  -0.7854  -0.7854  -0.7854   3.1416
pose_inverse  = 3.1416   0.7854  -0.7854   0.8662   1.4934   1.9232
pose_inverse2 = 5.4196   1.6173  -0.7854   2.2861   0.0732  -2.2320
Transform =
     0.5000    0.7071   -0.5000   -0.3256
    -0.5000    0.7071    0.5000    0.1501
     0.7071    0.0000    0.7071    0.7371
          0         0         0    1.0000
```

Inverse Kinematics : MATLAB PCT PUMA CF and N

So far there are 3 different solutions, so which one is right?

Inverse Kinematics : MATLAB PCT PUMA CF and N

So far there are 3 different solutions, so which one is right?

They are all mathematically "right", but in practice typically only a few are useful or achievable

Limitations can include:

Inverse Kinematics : MATLAB PCT PUMA CF and N

So far there are 3 different solutions, so which one is right?

They are all mathematically "right", but in practice typically only a few are useful or achievable

Limitations can include:

- Appendages of the robot may intersect as links have finite geometry
- Joints may have limits as to how far they can extend or rotate
- In order to avoid other components in the workspace

Inverse Kinematics : MATLAB PCT PUMA CF Configurations

For a 6 link manipulator such as the PUMA 560, there are a maximum of 8 configurations. These can be specified as arguments to the closed form solution function.

Types are:

Left or right handed	'l','r'
Elbow up or down	'u','d'
Wrist flipped or not flipped	'f','n'

Syntax for inverse kinematics pose

```
pose = p560.ikine6s(Transform,'ru')
% Right handed, elbow up configuration
```

Inverse Kinematics : MATLAB PCT PUMA N Configurations

Alternatively, the numerical technique could (or may have to) be used to find the joint angles.

The numerical technique can be seeded, and will **hopefully** converge to the desired outcome.

Syntax for inverse kinematics pose

```
pose = p560.ikine(Transform,Seed)
pose = p560.ikine(Transform,[0 0 3 0 0 0])
% Converges to elbow up configuration
```

If not the required outcome, need to reseed.

Kinematics : 6 DOF Manipulators

A 6 DOF manipulator should theoretically be able to achieve any pose, consisting of a location and orientation, in its workspace.

In the physical domain, however, joint limits and singularities prevent this from being the case.

Adding additional links and joints can overcome this problem.

Kinematics : Robots with Fewer, or Extra, Degrees of Freedom

The PUMA 560 robot that has been studied has 6 Degrees of Freedom (DOF), and a spherical wrist, which makes inverse kinematics easy to calculate.

Having 6 DOF means that the X, Y, Z location and Roll, Pitch, Yaw orientation are all independant except at the boundary or certain working poses - eg. Singularities (Will discuss futher later)

Robots that do not have 6 DOF can be classified as being:

- an **under-actuated** manipulator, which have fewer than 6 DOF
- or a **redundant** manipulator, having greater than 6 DOF

Kinematics : Robots with Fewer, or Extra, Degrees of Freedom

How does having more, or fewer, than 6 DOF affect inverse kinematics solution methods?

What is known, and what can be ignored?

Kinematics : Robots with Fewer, or Extra, Degrees of Freedom

How does having more, or fewer, than 6 DOF affect inverse kinematics solution methods?

What is known, and what can be ignored?

This can result in limitations and advantages respectively:

- An **under-actuated** manipulator will exhibit some DOF that will not be able to be controlled
- whereas a **redundant** manipulator has superfluous DOF, resulting in an infinte number of solutions. How to choose?

Kinematics : Under-Actuated Manipulators MATLAB PCT

For a two link manipulator, can only specify 2 DOF (eg. X and Y).

Under-actuated Manipulators MATLAB PCT

```
Syntax:
model.ikine(tool_pose,[0 0],[XYZRPY mask])

mdl_twolink
twolink.base = eye(4) % Joint 1 in Z
T = transl(0.4,0.5,0.6)
q = twolink.ikine(T,[0 0],[1 1 0 0 0 0])
Tf = twolink.fkine(q)
```

```
T =
    1.0000         0         0    0.4000
         0    1.0000         0    0.5000
         0         0    1.0000    0.6000
         0         0         0    1.0000
```

```
Tf =
   -0.5398   -0.8418         0    0.4000
    0.8418   -0.5398         0    0.5000
         0         0    1.0000         0
         0         0         0    1.0000
```

Kinematics : Under-Actuated Manipulators MATLAB PCT

From the previous slide:

```
T =                                                      Tf =
    1.0000         0         0    0.4000                    -0.5398   -0.8418         0    0.4000
         0    1.0000         0    0.5000                     0.8418   -0.5398         0    0.5000
         0         0    1.0000    0.6000                          0         0    1.0000         0
         0         0         0    1.0000                          0         0         0    1.0000
```

It is observed that only the X and Y components can be solved for (of course this robot cannot rotate out of this plane).

Under-actuated manipulators can only have as many degrees of freedom controlled as their physical configuration permits. In this case, 2 DOF.

Kinematics : Redundant Manipulators MATLAB PCT

Redundant manipulators have more than 6 DOF. This example concatenates a PP robot (mobile platform) with a PUMA560 robot (RRRRRR) to become a PPRRRRRR robot. Notice that they are concatenated where p8 = ...

Redundant Manipulators MATLAB PCT (P8 example)

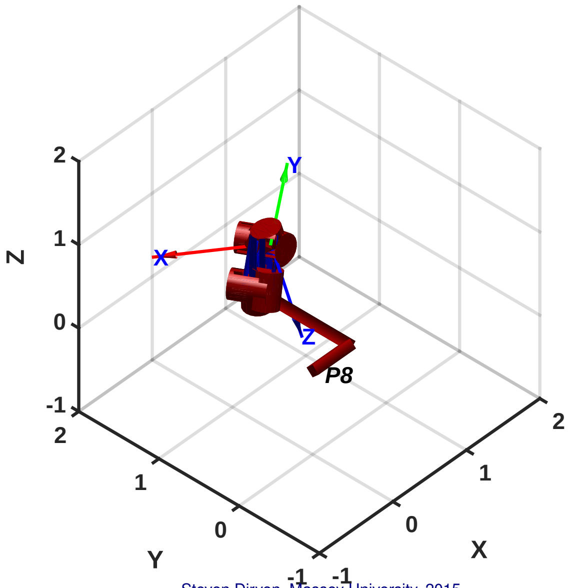
```
mdl_puma560
platform = SerialLink([0 0 0 -pi/2 1; -pi/2 0 0
pi/2 1],'base',troty(pi/2),'name','platform')
p560.links(1).d = .6
p8 = SerialLink([platform,p560],'name','P8')
T = transl(0.5,1.0,0.7) * rpy2tr(0,3*pi/4,0)
qf = p8.ikine(T)
p8.plot(qf,'workspace',[-1 2 -1 2 -1 2],'nobase',
'noshadow')
```

Kinematics : Redundant Manipulators MATLAB PCT

The Denavit-Hartenberg Table in MATLAB PCT for the p8 manipulator:

```
p8 =

P8 (8 axis, PPRRRRRR, stdDH, fastRNE)

+---+-----------+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |    offset |
+---+-----------+-----------+-----------+-----------+-----------+
| 1 |          0|         q1|          0|     -1.571|          0|
| 2 |     -1.571|         q2|          0|      1.571|          0|
| 3 |         q3|        0.6|          0|      1.571|          0|
| 4 |         q4|          0|     0.4318|          0|          0|
| 5 |         q5|       0.15|     0.0203|     -1.571|          0|
| 6 |         q6|     0.4318|          0|      1.571|          0|
| 7 |         q7|          0|          0|     -1.571|          0|
| 8 |         q8|          0|          0|          0|          0|
+---+-----------+-----------+-----------+-----------+-----------+

grav =     0  base =  0  0  1  0   tool =  1  0  0  0
           0          0  1  0  0           0  1  0  0
        9.81         -1  0  0  0           0  0  1  0
                      0  0  0  1           0  0  0  1
```

Kinematics : Redundant Manipulators MATLAB PCT



Kinematics : Robots with Fewer, or Extra, Degrees of Freedom Summary

- < 6 DOF: Can only control as many DOF as are available in the robot. eg. # of joints

- 6 DOF: Theoretically can achieve any position and orientation (pose) within its workspace. However, due to physical limitations and singularities, this is not always the case. Special case of spherical wrist facilitates analytical inverse kinematics

- > 6 DOF: Has improved dexterity, compared to a 6DOF robot. Inverse kinematics by numerical method. More complex to solve, but depending on the solution algorithm may find a more suitable configuration.