```
function m = TotalMass(j)
global uLINK

if j == 0
   m = 0;
else
   m = uLINK(j).m + TotalMass(uLINK(j).sister) + TotalMass(uLINK(j).child);
end
```

Fig. 2.18 TotalMass.m: Sum of Each Link's Mass

2.5 Kinematics of a Humanoid Robot

2.5.1 Creating the Model

To explain the kinematics of a humanoid robot, we will use a 12 DoF model shown in Fig. 2.19, which consists of two legs. The link names and their ID numbers are indicated in Fig. 2.19(a). Splitting this model by the manner of Fig. 2.14(a), each link will have just one joint to drive it except the BODY link. Thus we can identify a link by joint name or its ID number. For example,

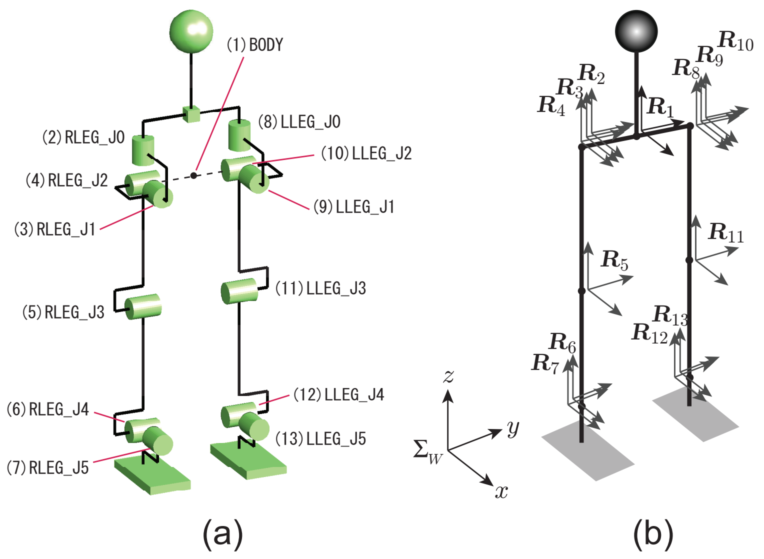


Fig. 2.19 (a) Structure of a 12 degree of freedom biped robot. Numbers in brackets refer to the ID number (b) Rotation matrix which describes attitude of each link.

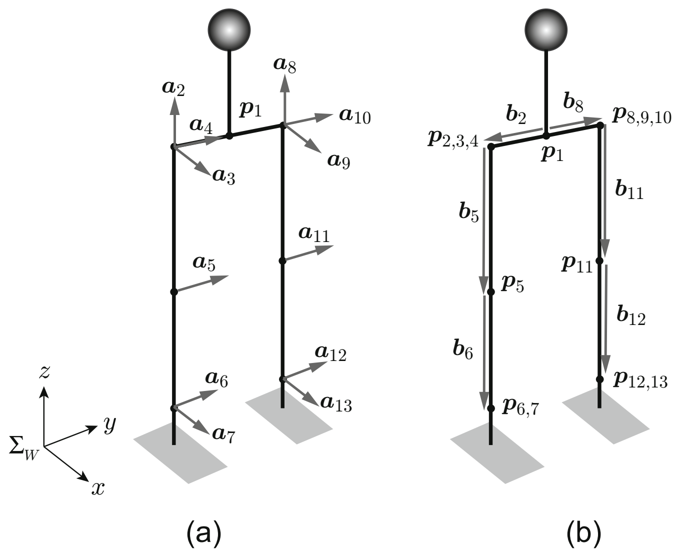


Fig. 2.20 (a) Joint Axis Vector $\boldsymbol{a}_j$ (b) Relative Position Vector $\boldsymbol{b}_j$ and the Origin of Local Coordinates $\boldsymbol{p}_j$

by the ID number 5, we refer the joint RLEG_J3 as well as the link of the right lower leg.

First, we must define the local coordinates for each link. An origin of each local coordinates can be set anywhere on its own joint axis. For each hip, however, it is reasonable to assign the origins of the three frames at the same point where the three joint axes intersect. In the same way, for each ankle, we assign the origins of two ankle frames on the same point where the two joint axes intersect.

Then, all rotation matrices which describe attitude of links are set to match the world coordinates when the robot is in its initial state, standing upright with fully stretched knees. Therefore we set thirteen matrices as,

$$\boldsymbol{R}_1 = \boldsymbol{R}_2 = \ldots = \boldsymbol{R}_{13} = \boldsymbol{E}.$$

We show the local coordinates defined at this stage in Fig. 2.19(b).

Next we will define the joint axis vectors $\boldsymbol{a}_j$ and the relative position vector $\boldsymbol{b}_j$ as indicated in Fig. 2.20. The joint axis vector is a unit vector which defines the axis of the joint rotation in the parent link's local coordinates. The positive (+) joint rotation is defined as the way to tighten a right-hand screw placed in the same direction with the joint axis vector. Using the knee joints as an example, the joint axis vectors would be, $\boldsymbol{a}_5, \boldsymbol{a}_{11} = [0\ 1\ 0]^T$. When we rotate this link in the + direction, a straight knee will flex in the same direction as a human's. The relative position vector $\boldsymbol{b}_j$ is the vector that indicates where the origin of a local coordinate lies in the parent link's local coordinates. When they lie in the same place as it is in the case of the ankle roll joint, $\boldsymbol{b}_7, \boldsymbol{b}_{13} = \mathbf{0}$[14].

In the following section we will use the description above to calculate Forward Kinematics, the Jacobian Matrix and Inverse Kinematics. To do this we will need a lot more information such as the shape, joint angle, joint velocity, etc. The full list of link parameters is shown in Table 2.1.

Table 2.1 Link Parameters

Link Parameter	Symbol for Equation	uLINK data field
Self ID	j	-
Sister ID	None	sister
Child ID	None	child
Parent ID	i	mother
Position in World Coordinates	$\boldsymbol{p}_j$	p
Attitude in World Coordinates	$\boldsymbol{R}_j$	R
Linear Velocity in World Coordinates	$\boldsymbol{v}_j$	v
Angular Velocity in World Coordinates	$\boldsymbol{\omega}_j$	w
Joint Angle	q_j	q
Joint Velocity	$\dot{q}_j$	dq
Joint Acceleration	$\ddot{q}_j$	ddq
Joint Axis Vector(Relative to Parent)	$\boldsymbol{a}_j$	a
Joint Relative Position(Relative to Parent)	$\boldsymbol{b}_j$	b
Shape(Vertex Information, Link Local)	$\bar{\boldsymbol{p}}_j$	vertex
Shape(Vertice Information (Point Connections)	None	face
Mass	m_j	m
Center of Mass(Link Local)	$\bar{\boldsymbol{c}}_j$	c
Moment of Inertia(Link Local)	$\bar{\boldsymbol{I}}_j$	I

2.5.2 *Forward Kinematics: Calculating the Position of the Links from Joint Angles*

Forward Kinematics is a calculation to obtain the position and attitude of a certain link from a given robot structure and its joint angles. This is required when you want to calculate the Center of Mass of the whole robot, when you just want to display the current state of the robot, or when you want to detect collisions of the robot with the environment. Thus forward kinematics forms the basis of robotics simulation.

Forward kinematics can be calculated by using the chain rule of homogeneous transforms. First we will start off by calculating the homogeneous

[14] There is a well known method you can use to describe the link structure of a robot called the Denavit-Hartenberg (DH) method [18]. We also used this method at first. However this method has a restriction which requires you to change the orientation of the link coordinates with each link. We found the implementation with this restriction to be rather error-prone, so we instead adopted the method outlined above.

transform of a single link as shown in Fig. 2.21. We need to set a local coordinate system Σ_j which has it's origin on the joint axis. The joint axis vector seen from the parent coordinates is $\boldsymbol{a}_j$ and the origin of Σ_j is $\boldsymbol{b}_j$. The joint angle is q_j and the attitude of the link when the joint angle is 0 is, $\boldsymbol{E}$.

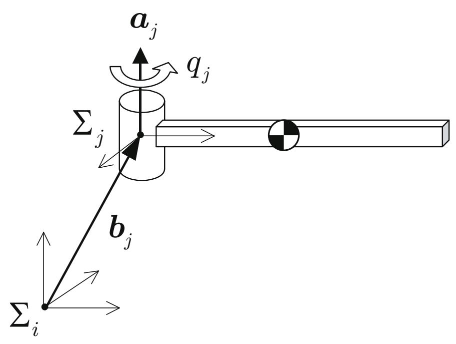


Fig. 2.21 The Position, Attitude and Rotation of a single link $\boldsymbol{a}_j, \boldsymbol{b}_j$ each specify the joint axis vector and location of the origin viewed from the parent coordinate system

The homogeneous transform relative to the parent link is:

$$^i\boldsymbol{T}_j = \begin{bmatrix} e^{\widehat{a}_j q_j} & \boldsymbol{b}_j \\ 0\,0\,0 & 1 \end{bmatrix}. \tag{2.56}$$

Next let us assume there are two links as shown in Fig. 2.22. We will assume that the absolute position and attitude of the parent link $\boldsymbol{p}_i, \boldsymbol{R}_i$ is known. Therefore, the homogeneous transform to Σ_i becomes:

$$\boldsymbol{T}_i = \begin{bmatrix} \boldsymbol{R}_i & \boldsymbol{p}_i \\ 0\,0\,0 & 1 \end{bmatrix}. \tag{2.57}$$

From the chain rule of homogeneous transforms Σ_j is:

$$\boldsymbol{T}_j = \boldsymbol{T}_i {}^i\boldsymbol{T}_j. \tag{2.58}$$

From (2.56), (2.57) and (2.58) the absolute position ($\boldsymbol{p}_j$) and attitude ($\boldsymbol{R}_j$) of Σ_j can be calculated as being,

$$\boldsymbol{p}_j = \boldsymbol{p}_i + \boldsymbol{R}_i \boldsymbol{b}_j \tag{2.59}$$

$$\boldsymbol{R}_j = \boldsymbol{R}_i e^{\widehat{a}_j q_j} \tag{2.60}$$

Using this relationship and the recursive algorithm, the Forward Kinematics can be performed by an extremely simple script shown in Fig. 2.23. To use this program, we first set the absolute position and attitude of the base

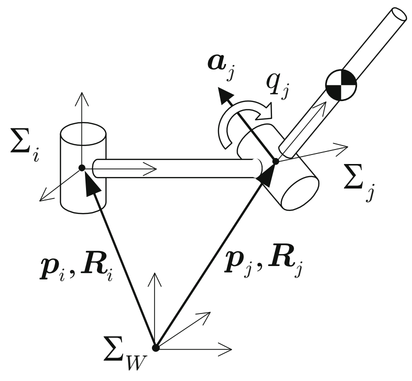

$\boldsymbol{a}_j$
q_j
Σ_i
Σ_j
$\boldsymbol{p}_i, \boldsymbol{R}_i$
$\boldsymbol{p}_j, \boldsymbol{R}_j$
Σ_W

Fig. 2.22 Relative Position of Two Links

```
function ForwardKinematics(j)
global uLINK

if j == 0 return; end
if j ~= 1
    i = uLINK(j).mother;
    uLINK(j).p = uLINK(i).R * uLINK(j).b + uLINK(i).p;
    uLINK(j).R = uLINK(i).R * Rodrigues(uLINK(j).a, uLINK(j).q);
end
ForwardKinematics(uLINK(j).sister);
ForwardKinematics(uLINK(j).child);
```

Fig. 2.23 ForwardKinematics.m calculate forward kinematics for all joints

link (BODY) and all joint angles. Then by executing ForwardKinematics(1), we can update the positions and attitudes of all links in the robot.

Figure 2.24 shows what the 12 degree of freedom biped robot looks like when you give it random values for joint angles to all 12 joints. This should help you to imagine how a simple mechanism embodies a large amount of complexity.

2.5.3 Inverse Kinematics: Calculating the Joint Angles from a Link's Position and Attitude

Next we will discuss how to calculate joint angles when we have the position and attitude of the body and the foot to realize. What we need to do in this cases is **Inverse Kinematics**. For instance, suppose our robot is in the front of stairs and we want to place one of the foot on the first step whose height and depth are already known. We certainly need to determine the amount of

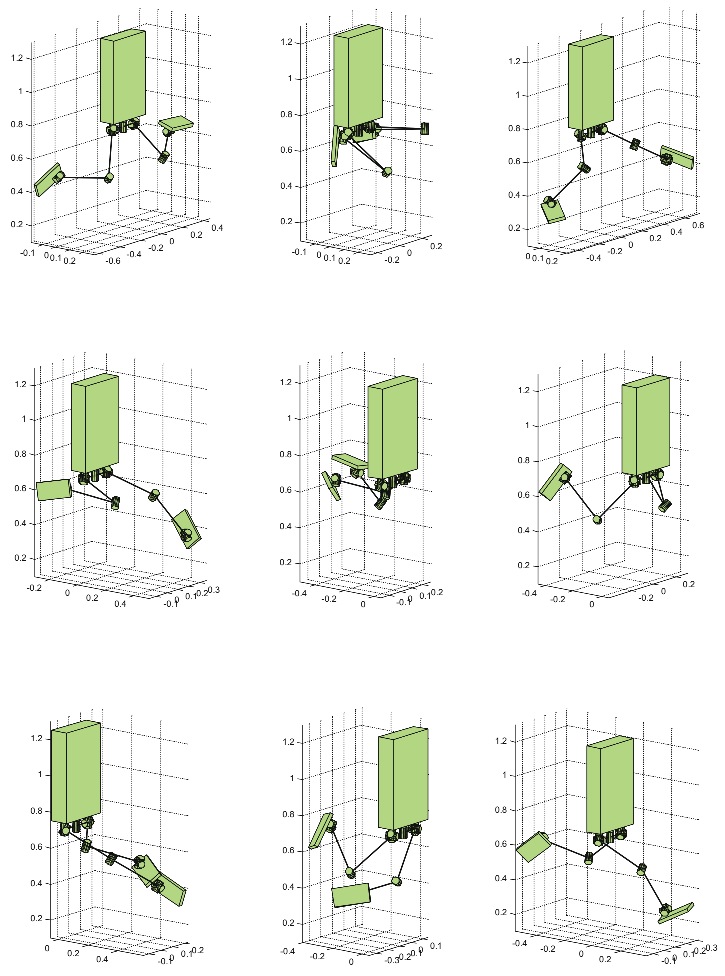
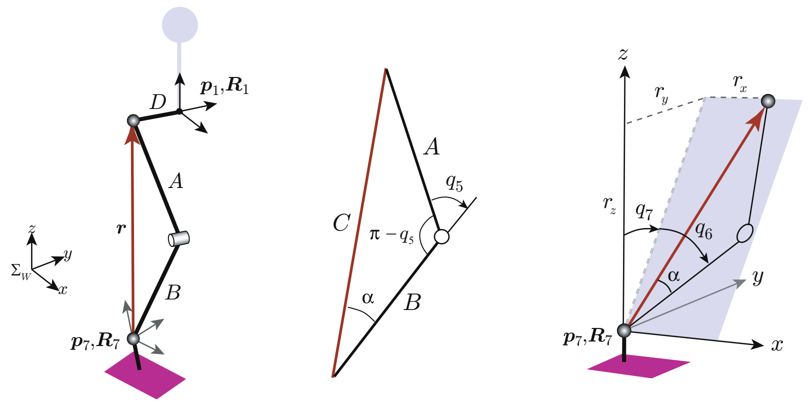


Fig. 2.24 Random poses calculated using ForwardKimematics. The knees are limited to $[0,\ \pi]$[rad], other joints are restricted to $[-\frac{\pi}{3},\ \frac{\pi}{3}]$[rad].

joint rotation for the hip, knee and the ankle. Inverse Kinematics is necessary for such a case.

There exist both an analytical method and a numerical method of solving Inverse Kinematics. First we will explain how to solve it analytically. Let's focus on the right leg of the model shown in Fig. 2.19. The position and attitude of the body and right leg will be $(\boldsymbol{p}_1, \boldsymbol{R}_1)$ and $(\boldsymbol{p}_7, \boldsymbol{R}_7)$ respectively. To simplify the equation we will define D, which is the distance between the Body origin and the hip joint. The upper leg length is A, and the lower leg length is B, as shown in Fig. 2.25(a). So therefore, the position of the hip would be



(a) Definition (b) Get the knee angle (c) Get the ankle angles

Fig. 2.25 Calculating Inverse Kinematics of the Legs

$$\boldsymbol{p}_2 = \boldsymbol{p}_1 + \boldsymbol{R}_1 \begin{bmatrix} 0 \\ D \\ 0 \end{bmatrix}.$$

Next, we calculate the position of the crotch viewed from the ankle coordinate space

$$\boldsymbol{r} = \boldsymbol{R}_7^T(\boldsymbol{p}_2 - \boldsymbol{p}_7) \equiv [r_x \; r_y \; r_z]^T. \tag{2.61}$$

From this we can calculate the distance between the ankle and the hip, which we will define as

$$C = \sqrt{r_x^2 + r_y^2 + r_z^2}.$$

As shown in Fig. 2.25(b), if we consider the triangle ABC we get the angle of the knee q_5. From the **cosine rule** we get,

$$C^2 = A^2 + B^2 - 2AB\cos(\pi - q_5).$$

So the angle of the knees will be,

$$q_5 = -\cos^{-1}\left(\frac{A^2 + B^2 - C^2}{2AB}\right) + \pi.$$

If we define the angle at the lower end of the triangle as α, from the **sine rule** we get,

$$\frac{C}{\sin(\pi - q_5)} = \frac{A}{\sin\alpha}.$$

So therefore,

$$\alpha = \sin^{-1}\left(\frac{A\sin(\pi - q_5)}{C}\right).$$

Next we will focus on the ankle local coordinates. As shown in Fig. 2.25(c), from vector $\boldsymbol{r}$ you can calculate the ankle roll and pitch angles. So,

$$\begin{aligned} q_7 &= \operatorname{atan2}(r_y, r_z) \\ q_6 &= -\operatorname{atan2}\left(r_x, \operatorname{sign}(r_z)\sqrt{r_y^2 + r_z^2}\right) - \alpha. \end{aligned}$$

The function $\operatorname{atan2}(y, x)$ calculates the angle between vector (x, y) and the x axis as it has already appeared in Section 2.2.7. It is built into Matlab and is also available as a built-in function in most programming languages. Also $sign(x)$ is a function that returns $+1$ if x is a positive value and -1 if it is negative.

What remains is the yaw, roll and pitch angles at the base of the leg. From the equations that define each joint

$$\boldsymbol{R}_7 = \boldsymbol{R}_1\boldsymbol{R}_z(q_2)\boldsymbol{R}_x(q_3)\boldsymbol{R}_y(q_4)\boldsymbol{R}_y(q_5 + q_6)\boldsymbol{R}_x(q_7),$$

we obtain

$$\boldsymbol{R}_z(q_2)\boldsymbol{R}_x(q_3)\boldsymbol{R}_y(q_4) = \boldsymbol{R}_1^T\boldsymbol{R}_7\boldsymbol{R}_x(-q_7)\boldsymbol{R}_y(-q_5 - q_6).$$

By expanding the left side of this equation and calculating the right hand side we get the following

$$\begin{bmatrix} c_2c_4 - s_2s_3s_4 & -s_2c_3 & c_2s_4 + s_2s_3c_4 \\ s_2c_4 + c_2s_3s_4 & c_2c_3 & s_2s_4 - c_2s_3c_4 \\ -c_3s_4 & s_3 & c_3c_4 \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix}$$

where $c_2 \equiv \cos q_2$, and $s_2 \equiv \sin q_2$.

By looking carefully at the left hand side of this equation we get,

$$q_2 = \operatorname{atan2}(-R_{12}, R_{22}) \tag{2.62}$$
$$q_3 = \operatorname{atan2}(R_{32}, -R_{12}s_2 + R_{22}c_2) \tag{2.63}$$
$$q_4 = \operatorname{atan2}(-R_{31}, R_{33}). \tag{2.64}$$

An implementation of the above is shown in Fig. 2.26[15]. For the left leg, we could invert the sign on D and apply the same program.

This implementation can only be applied to a robot which has the same layout as the one in Fig. 2.19. If the robot does not have three joint axes that intersect each other at one point, we need an entirely different algorithm. There are many different algorithms outlined in the robot textbooks

[15] As a practical implementation, our program covers the target position exceeding the leg length and joint angle limits.

```
function q = IK_leg(Body,D,A,B,Foot)

r = Foot.R' * (Body.p + Body.R * [0 D 0]'- Foot.p);  % crotch from ankle
C = norm(r);
c5 = (C^2-A^2-B^2)/(2.0*A*B);
if c5 >= 1
    q5 = 0.0;
elseif c5 <= -1
    q5 = pi;
else
    q5 = acos(c5);  % knee pitch
end
q6a = asin((A/C)*sin(pi-q5));   % ankle pitch sub

q7 = atan2(r(2),r(3));  % ankle roll -pi/2 < q(6) < pi/2
if q7 > pi/2, q7=q7-pi; elseif q7 < -pi/2, q7=q7+pi; end
q6 = -atan2(r(1),sign(r(3))*sqrt(r(2)^2+r(3)^2)) -q6a; % ankle pitch

R = Body.R' * Foot.R * Rroll(-q7) * Rpitch(-q6-q5); %% hipZ*hipX*hipY
q2  = atan2(-R(1,2),R(2,2));   % hip yaw
cz = cos(q2); sz = sin(q2);
q3 = atan2(R(3,2),-R(1,2)*sz + R(2,2)*cz);  % hip roll
q4 = atan2( -R(3,1), R(3,3));               % hip pitch

q = [q2 q3 q4 q5 q6 q7]';
```

Fig. 2.26 IK_leg.m Example implementation of an analytical solution to Inverse Kinematics. CAUTION! When using this program on a real robot, you need to continuously check whether the joint angles exceed their limits. In the worst case, you could destroy your robot or otherwise cause major injury or death.

(for instance [96, 127]), but in general it requires a large amount of heavy calculation, so it is more common to use the numerical solution which we will go over in the next section.

2.5.4 Numerical Solution to Inverse Kinematics

Compared to solving the inverse kinematics analytically, forward kinematics calculation is simple (Section 2.5.2). So it isn't all that far fetched to think of a trial and error method of solving the inverse kinematics by using forward kinematics, as shown in Fig. 2.27. A sample algorithm could be,

Step 1. Prepare the position and attitude $(\boldsymbol{p}^{ref}, \boldsymbol{R}^{ref})$ of the base link
Step 2. Prepare the position and attitude $(\boldsymbol{p}^{ref}, \boldsymbol{R}^{ref})$ of the target link

Step 3. Define vector $\boldsymbol{q}$ which holds the joint angles from the base link to the target link
Step 4. Use forward kinematics to calculate the position and attitude $(\boldsymbol{p}, \boldsymbol{R})$ of the target link
Step 5. Calculate the difference in position and attitude $(\Delta\boldsymbol{p}, \Delta\boldsymbol{R}) = (\boldsymbol{p}^{ref} - \boldsymbol{p}, \boldsymbol{R}^T\boldsymbol{R}^{ref})$
Step 6. If $(\Delta\boldsymbol{p}, \Delta\boldsymbol{R})$ are small enough stop the calculation
Step 7. If $(\Delta\boldsymbol{p}, \Delta\boldsymbol{R})$ are not small enough calculate $\Delta\boldsymbol{q}$ which would reduce the error
Step 8. Update joint angles by $\boldsymbol{q} := \boldsymbol{q} + \Delta\boldsymbol{q}$ and return to **Step 4**

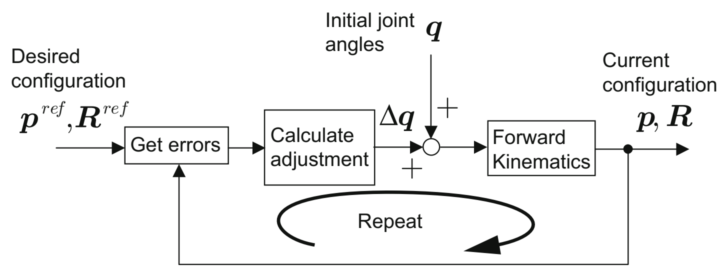


Fig. 2.27 Basic Concept Behind Numerical Approach to Inverse Kinematics: Use forward kinematics to and adjust the joint angles to narrow the difference.

To actually implement this you first need to surmount the next two hurdles.

1. What do we really mean by the position and attitude errors $(\Delta\boldsymbol{p}, \Delta\boldsymbol{R})$ being small enough? (**Step 6**)
2. How do we actually go about calculating $\Delta\boldsymbol{q}$, to narrow the gap? (**Step 7**)

The first problem can be solved relatively easily. Zero position error and zero attitude error can be described with the following equations

$$\Delta\boldsymbol{p} = 0$$
$$\Delta\boldsymbol{R} = \boldsymbol{E}.$$

An example function which returns positive scalar depending on the error magnitude is the following[16]

[16] A more general function would be $err(\Delta\boldsymbol{p}, \Delta\boldsymbol{R}) = \alpha\|\Delta\boldsymbol{p}\|^2 + \beta\|\Delta\boldsymbol{\theta}\|^2$. Here α and β are some positive number with the direction requiring more precision being larger.

$$err(\Delta\boldsymbol{p}, \Delta\boldsymbol{R}) = \|\Delta\boldsymbol{p}\|^2 + \|\Delta\boldsymbol{\theta}\|^2, \tag{2.65}$$

$$\Delta\boldsymbol{\theta} \equiv (\ln \Delta\boldsymbol{R})^{\vee}. \tag{2.66}$$

This function becomes zero only at both position and attitude error is zero. You can say the position and attitude errors are small enough when $err(\Delta\boldsymbol{p}, \Delta\boldsymbol{R})$ becomes smaller than predefined value, for example 1×10^{-6}.

How about the second problem? All we really need to do is come up with a set of joint angles $\Delta\boldsymbol{q}$ which lowers $err(\Delta\boldsymbol{p}, \Delta\boldsymbol{R})$. One idea would be to use random numbers each time. If we are able to lower $err(\Delta\boldsymbol{p}, \Delta\boldsymbol{R})$ by even a small amount, we will use it for the joint angles and start over. The robot uses trial and error to search for the joint angles itself so we have the **illusion of intelligence**[17].

Although it is an enticing idea, none of the robots today would use this method. The reason is that there is a method which is much faster and far more precise. In this method which is called the Newton-Raphson method we first start off by considering what happens to the position and attitude $(\delta\boldsymbol{p}, \delta\boldsymbol{\theta})$ when you change the joint angles using a minute value of $\delta\boldsymbol{q}$

$$\delta\boldsymbol{p} = \boldsymbol{X}_p(\boldsymbol{q}, \delta\boldsymbol{q}) \tag{2.67}$$

$$\delta\boldsymbol{\theta} = \boldsymbol{X}_\theta(\boldsymbol{q}, \delta\boldsymbol{q}). \tag{2.68}$$

Here, $\boldsymbol{X}_p$ and $\boldsymbol{X}_\theta$ are unknown, but when $\delta\boldsymbol{q}$ is small let us say that we can describe it simply with addition and multiplication. If we use a matrix we get,

$$\begin{bmatrix} \delta\boldsymbol{p} \\ \delta\boldsymbol{\theta} \end{bmatrix} = \begin{bmatrix} J_{11} & J_{12} & J_{13} & J_{14} & J_{15} & J_{16} \\ J_{21} & J_{22} & J_{23} & J_{24} & J_{25} & J_{26} \\ J_{31} & J_{32} & J_{33} & J_{34} & J_{35} & J_{36} \\ J_{41} & J_{42} & J_{43} & J_{44} & J_{45} & J_{46} \\ J_{51} & J_{52} & J_{53} & J_{54} & J_{55} & J_{56} \\ J_{61} & J_{62} & J_{63} & J_{64} & J_{65} & J_{66} \end{bmatrix} \delta\boldsymbol{q}. \tag{2.69}$$

Here $J_{ij}, (i, j = 1 \ldots 6)$ are constants which are defined by the current position and attitude of the robots links. There are 6 because of the number of links in the leg. It is too much to write all the components each time so we will simplify it by

$$\begin{bmatrix} \delta\boldsymbol{p} \\ \delta\boldsymbol{\theta} \end{bmatrix} = \boldsymbol{J}\ \delta\boldsymbol{q}. \tag{2.70}$$

[17] This idea is something anyone would think of, but for some reason a lot of people tend to think that they are the only ones to think of it. Actually the author happened to be one of them :-).

The matrix $\boldsymbol{J}$ is called the **Jacobian**[18]. Once we have (2.70) we can calculate the required adjustment by simply taking the inverse of this matrix

$$\delta \boldsymbol{q} = \lambda\, \boldsymbol{J}^{-1} \begin{bmatrix} \delta \boldsymbol{p} \\ \delta \boldsymbol{\theta} \end{bmatrix}. \tag{2.71}$$

This is the equation to calculate the adjustments of the joint angles based on the errors in position and attitude. The value $\lambda \in (0\ 1]$ is a coefficient used to stabilize the numeric calculation. Figure 2.28 shows a sample implementation of the inverse kinematics algorithm written in Matlab. On the 7th line you will see the function CalcJacobian which is used to calculate the Jacobian. We will go over this in more detail in the next section. The 10th line is the actual implementation of (2.71). The operator \ "backslash" efficiently solves the linear equations without doing explicit matrix inversion.

```
function InverseKinematics(to, Target)
global uLINK

lambda = 0.5;
ForwardKinematics(1);
idx = FindRoute(to);
for n = 1:10
  J   = CalcJacobian(idx);
  err = CalcVWerr(Target, uLINK(to));
  if norm(err) < 1E-6 return, end;
  dq = lambda * (J \ err);
  for nn=1:length(idx)
    j = idx(nn);
    uLINK(j).q = uLINK(j).q + dq(nn);
  end
  ForwardKinematics(1);
end
```

Fig. 2.28 InvserseKinematics.m Numerical Solution to Inverse Kinematics

The function FindRoute returns the links that you need to go through to get to the target link from the base link. CalcVWerr is a function which calculates the difference in position and attitude. You can find implemented versions of these functions in the appendix at the end of this chapter.

[18] From the German mathematician Carl Gustav Jacobi (1804-1851). When mathematicians refer to the Jacobian it means the determinant of this matrix, but when roboticists talk about the Jacobian they usually mean the matrix itself. Some people think that this is a mistake but this is not something that is local to Japan, it is done the world over.

We show an example use of Inverse Kinematics as Matlab command input in Fig. 2.29. Here we use `SetupBipedRobot` to set robot data, `GoHalfSitting` to get non-singular posture, and `DrawAllJoints()` to display the biped robot. The function `rpy2rot()` is a implementation of (2.13). They can be obtained from the download material.

```
>> SetupBipedRobot;  % Set robot parameters
>> GoHalfSitting;    % Set knee bending posture

>> Rfoot.p = [-0.3, -0.1, 0]';
>> Rfoot.R = rpy2rot(0, ToRad*20.0,0);
>> InverseKinematics(RLEG_J5, Rfoot);

>> Lfoot.p = [ 0.3, 0.1, 0]';
>> Lfoot.R = rpy2rot(0, -ToRad*30.0,0);
>> InverseKinematics(LLEG_J5, Lfoot);

>> DrawAllJoints(1);  % Show the robot
```

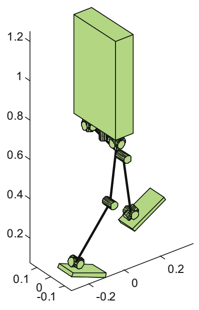


Fig. 2.29 Sample using InverseKinematics and Results of Calculation

2.5.5 Jacobian

In the previous section we introduced the Jacobian which gives you the relationship between small joint movements and spatial motion. Through the Jacobian we can also calculate the torque requirements of the joints in order to generate external forces through the hands and feet. As this is used extensively in robot control, papers on robotics that do not have a Jacobian somewhere in them are a rare thing indeed[19].

[19] We can find out how the Jacobian is used in robotics by reading Dr Yoshikawa's textbook [144].