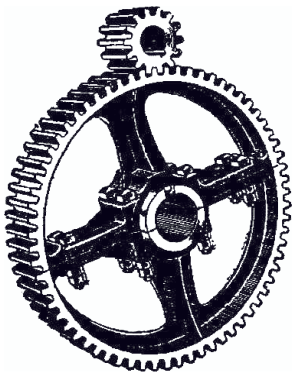
Chapter

8 Velocity Relationships



In this chapter we consider the relationship between the rate of change of joint coordinates, the joint velocity, and the velocity of the end-effector. The 3-dimensional end-effector pose $\xi \in SE(3)$ has a velocity which is represented by a 6-vector known as a spatial velocity. The joint velocity and the end-effector velocity are related by the manipulator Jacobian matrix which is a function of manipulator pose.

Section 8.1 uses a numerical approach to introduce the manipulator Jacobian. Next we introduce additional Jacobians to transform velocity between coordinate frames and angular velocity between different angular representations. The numerical properties of the Jacobian matrix are shown to provide insight into the dexterity of the manipulator – the directions in which it can move easily and those in which it cannot – and understanding about singular configurations. In Sect. 8.2 the inverse Jacobian is used to generate Cartesian paths without requiring inverse kinematics, and this can be applied to over- and under-actuated robots. Section 8.3 demonstrates how the Jacobian transpose is used to transform forces from the end-effector to the joints and between coordinate frames. Finally, in Sect. 8.4 the numeric inverse kinematic solution, used in the previous chapter, is fully described.

8.1 Manipulator Jacobian

We start by investigating how small changes in joint coordinates affect the pose of the end-effector. Using the homogeneous transformation representation of pose we can approximate its derivative with respect to joint coordinates by a first-order difference

$$\frac{\mathrm{d}\boldsymbol{T}}{\mathrm{d}\boldsymbol{q}} \approx \frac{\boldsymbol{T}(\boldsymbol{q}+\delta_q) - \boldsymbol{T}(\boldsymbol{q})}{\delta_q}$$

and recalling the definition of $\boldsymbol{T}$ from Eq. 2.19 we can write

$$\frac{\mathrm{d}\boldsymbol{T}}{\mathrm{d}\boldsymbol{q}} \approx \frac{1}{\delta_q}\left(\begin{array}{c|c} \boldsymbol{R}(\boldsymbol{q}+\delta_q) - \boldsymbol{R}(\boldsymbol{q}) & \begin{matrix}\delta_x\\ \delta_y\\ \delta_z\end{matrix} \\ \hline 000 & 0 \end{array}\right) \tag{8.1}$$

where $(\delta_x, \delta_y, \delta_z)$ is the translational displacement of the end-effector.

For the purpose of example we again use the Puma 560 robot

```
>> mdl_puma560
```

and at the nominal configuration `qn` shown in Fig. 8.1 the end-effector pose is

```
>> T0 = p560.fkine(qn);
```

We perturb joint one by a small but finite amount `dq`

```
>> dq = 1e-6;
```

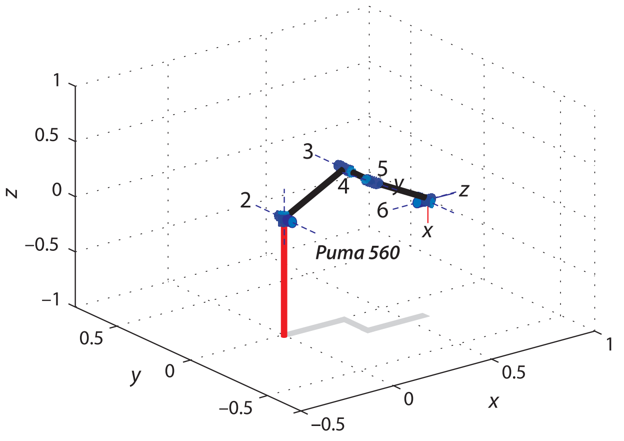


Fig. 8.1.
Puma robot in its nominal pose `qn`. The end-effector *z*-axis points in the world *x*-direction, and the *x*-axis points downward

and compute the new end-effector pose

```
>> Tp = p560.fkine(qn + [dq 0 0 0 0 0]);
```

and using Eq. 8.1 the derivative is

```
>> dTdq1 = (Tp - T0) / dq
dTdq1 =
         0   -1.0000   -0.0000    0.1500
   -0.0000   -0.0000    1.0000    0.5963
         0         0         0         0
         0         0         0         0
```

This is clearly no longer a homogeneous transformation – the uppper left 3 × 3 matrix is not an orthonormal matrix and the lower-right element is not a 1. What does this result mean?

Equating the elements of column four of `dTdq1` and the matrix in Eq. 8.1 we can write

$$\begin{pmatrix}\delta_x\\ \delta_y\\ \delta_z\end{pmatrix} = \begin{pmatrix}0.1500\\ 0.5963\\ 0\end{pmatrix}\delta_{q_1}$$

which is the displacement of the end-effector position as a function of a displacement in q_1. From Fig. 8.1 this makes sense – a small rotation of the waist joint (q_1) will move the end-effector in the horizontal *xy*-plane but not vertically.

We can repeat the process for the next joint

```
>> Tp = p560.fkine(qn + [0 dq 0 0 0 0]);
>> dTdq2 = (Tp - T0) / dq
dTdq =
    1.0000   -0.0000   -0.0000    0.0144
    0.0000         0    0.0000         0
    0.0000    0.0000    1.0000    0.5963
         0         0         0         0
```

and write

$$\begin{pmatrix}\delta_x\\ \delta_y\\ \delta_z\end{pmatrix} = \begin{pmatrix}0.0144\\ 0\\ 0.5963\end{pmatrix}\delta_{q_2}$$

A small motion of the shoulder joint causes end-effector motion in the vertical *xz*-plane, as expected, but not the *y*-direction. Dividing both sides by an infinitesimal time step δ_t we find a relationship

$$\begin{pmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\end{pmatrix}=\begin{pmatrix}0.0144\\ 0\\ 0.5963\end{pmatrix}\dot{q}_1$$

between joint angle velocity and end-effector velocity.

Now we consider the top-left 3 × 3 submatrix of the matrix in Eq. 8.1 and multiply it by δ_q / δ_t to achieve a first-order approximation to the derivative of $\boldsymbol{R}$

$$\dot{\boldsymbol{R}} \approx \left(\frac{\boldsymbol{R}(\boldsymbol{q}+\delta_q)-\boldsymbol{R}(\boldsymbol{q})}{\delta_q}\right)\frac{\delta_q}{\delta_t}$$

Recalling an earlier definition of the derivative of an orthonormal rotation matrix Eq. 3.4 we write

$$\boldsymbol{S}(\omega)\boldsymbol{R} \approx \frac{\boldsymbol{R}(\boldsymbol{q}+\delta_q)-\boldsymbol{R}(\boldsymbol{q})}{\delta_q}\dot{q}_1$$

$$\boldsymbol{S}(\omega) \approx \left(\frac{\boldsymbol{R}(\boldsymbol{q}+\delta_q)-\boldsymbol{R}(\boldsymbol{q})}{\delta_q}\boldsymbol{R}^T\right)\dot{q}_1$$

from which we find a relationship between end-effector angular velocity and joint velocity

$$\omega \approx \operatorname{vex}\left(\frac{\boldsymbol{R}(\boldsymbol{q}+\delta_q)-\boldsymbol{R}(\boldsymbol{q})}{\delta_q}\boldsymbol{R}^T\right)\dot{q}_1$$

Continuing with the joint 1 derivative computed above

```
>> dRdq1 = dTdq1(1:3,1:3);
>> R = T0(1:3, 1:3);
>> S = dRdq1 * R'
S =
   -0.0000   -1.0000    0.0000
    1.0000   -0.0000    0.0000
         0         0         0
```

which is a skew symmetric matrix from which we extract the angular velocity vector

```
>> vex(S)
ans =
     0     0     1
```

and finally we write

$$\begin{pmatrix}\omega_x\\ \omega_y\\ \omega_z\end{pmatrix}=\begin{pmatrix}0\\ 0\\ 1\end{pmatrix}\dot{q}_1$$

From Fig. 8.1 this makes sense. The end-effector's angular velocity is the same as the angular velocity of joint one which rotates about the world z-axis.

Repeating the process for small motion of the second joint we obtain

```
>> dRdq2 = dTdq2(1:3,1:3);
>> S = dRdq2 * inv( R )
S =
   -0.0000   -0.0000   -1.0000
    0.0000   -0.0000   -0.0000
    1.0000   -0.0000   -0.0000
>> vex(S)
    0.0000   -1.0000    0.0000
```

A Jacobian is the matrix equivalent of the derivative – the derivative of a vector-valued function of a vector with respect to a vector. If $\boldsymbol{y} = F(\boldsymbol{x})$ and $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{y} \in \mathbb{R}^m$ then the Jacobian is the $m \times n$ matrix

$$J = \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}} \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{pmatrix}$$

The Jacobian is named after Carl Jacobi, and more more details are given in Appendix G.

from which we write

$$\begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} = \begin{pmatrix} 0 \\ -1 \\ 0 \end{pmatrix} \dot{q}_2$$

Once again, with reference to Fig. 8.1, this makes sense. Rotation of joint two, whose rotational axis is in the negative y-direction, results in an angular velocity in the negative y-direction.

We have established, numerically, the relationship between the velocity of individual joints and the translational and angular velocity of the robot's end-effector. Earlier Eq. 7.3 we wrote the forward kinematics in functional form as

$$\xi = \mathcal{K}(\boldsymbol{q})$$

and taking the derivative we write

$$\boldsymbol{\nu} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{8.2}$$

which is the instantaneous forward kinematics where $\boldsymbol{\nu} = (v_x, v_y, v_z, \omega_x, \omega_y, \omega_z) \in \mathbb{R}^6$ is a spatial velocity and comprises translational and rotational velocity components. The matrix $\boldsymbol{J}(\boldsymbol{q}) \in \mathbb{R}^{6\times N}$ is the manipulator Jacobian or the geometric Jacobian.

The Jacobian matrix can be computed directly by the `jacob0` method of the `SerialLink` object

The function `jacob0` does not use finite differences. It has a direct form based on the Denavit-Hartenberg parameters of the robot arm (Paul and Shimano 1978).

```
>> J = p560.jacob0(qn)
J =
    0.1501    0.0144    0.3197         0         0         0
    0.5963    0.0000    0.0000         0         0         0
         0    0.5963    0.2910         0         0         0
         0   -0.0000   -0.0000    0.7071   -0.0000   -0.0000
         0   -1.0000   -1.0000   -0.0000   -1.0000   -0.0000
    1.0000    0.0000    0.0000   -0.7071    0.0000   -1.0000
```

The rows correspond to Cartesian degrees of freedom and the columns correspond to joints – they are the end-effector spatial velocities corresponding to unit velocity of the corresponding joints. The results we computed earlier, using derivative approximation, can be seen in the first two columns. The 3×3 block of zeros in the top right indicates that motion of the wrist joints has no effect on the end-effector translational motion – this is an artifact of the spherical wrist and a zero length tool.

8.1.1 Transforming Velocities between Coordinate Frames

Consider two frames $\{A\}$ and $\{B\}$ related by

Carl Gustav Jacob Jacobi (1804–1851) was a Prussian mathematician. He obtained a Doctor of Philosophy degree from Berlin University in 1825. In 1827 he was appointed professor of mathematics at Königsberg University and held this position until 1842 when he suffered a breakdown from overwork.

Jacobi wrote a classic treatise on elliptic functions in 1829 and also described the derivative of *m* functions of *n* variables which bears his name. He was elected a foreign member of the Royal Swedish Academy of Sciences in 1836. He is buried in the Friedhof I der Dreifaltigkeits-Kirchengemeinde in Berlin.

$$^{A}\boldsymbol{T}_{B} = \begin{pmatrix} \boldsymbol{R} & \boldsymbol{t} \\ 0 & 1 \end{pmatrix}$$

then a spatial velocity with respect to frame $\{A\}$ can be expressed relative to frame $\{B\}$ by

$$^{B}\nu = {}^{B}\boldsymbol{J}_{A}\,{}^{A}\nu \tag{8.3}$$

where the Jacobian

$$^{B}\boldsymbol{J}_{A} = \boldsymbol{J}_{\nu}\left({}^{A}\mathbf{T}_{B}\right) = \begin{pmatrix} \boldsymbol{R}^{T} & \left(\boldsymbol{S}(\boldsymbol{t})\boldsymbol{R}\right)^{T} \\ \boldsymbol{0}_{3\times3} & \boldsymbol{R}^{T} \end{pmatrix} \tag{8.4}$$

As discussed on page 16 the middle indices in Eq. 8.3, the "A"s, can be considered to *cancel out*.

is a 6 × 6 matrix and a function of the relative pose between the frames. The matrix has an interesting structure. The lower-left block of zeros means that rotational velocity in frame $\{B\}$ depends only on the rotational velocity in frame $\{A\}$. If the frames are displaced by a translation the top-right 3 × 3 block is non-zero, and this means that the translational velocity in frame $\{B\}$ has a component due to both translational and rotational velocity in frame $\{A\}$. The rotational velocity in $\{A\}$ is simply rotated to frame $\{B\}$.

For example if the second frame is related to the first by the transform

```
>> T = transl(1, 0, 0)*troty(pi/2);
```

then the Jacobian is given by

```
>> J = tr2jac(T)
J =
    0.0000         0   -1.0000         0    1.0000         0
         0    1.0000         0         0         0    1.0000
    1.0000         0    0.0000         0   -0.0000         0
         0         0         0    0.0000         0   -1.0000
         0         0         0         0    1.0000         0
         0         0         0    1.0000         0    0.0000
```

A unit velocity in the x-direction of frame $\{A\}$ is transformed to

```
>> vB = J*[1 0 0 0 0 0]';
>> vB'
ans =
    0.0000         0    1.0000         0         0         0
```

in frame $\{B\}$.

8.1.2 Jacobian in the End-Effector Coordinate Frame

The Jacobian computed by the method `jacob0` maps joint velocity to the end-effector spatial velocity expressed in the *world coordinate* frame – hence the zero suffix for the method `jacob0`. To obtain the spatial velocity in the end-effector coordinate frame we use the method `jacobn` instead

```
>> p560.jacobn(qn)
ans =
   -0.0000   -0.5963   -0.2910         0         0         0
    0.5963    0.0000    0.0000         0         0         0
    0.1500    0.0144    0.3197         0         0         0
   -1.0000         0         0    0.7071         0         0
   -0.0000   -1.0000   -1.0000   -0.0000   -1.0000         0
   -0.0000    0.0000    0.0000    0.7071    0.0000    1.0000
```

The code for the two Jacobian methods reveals that `jacob0` discussed earlier is actually based on `jacobn` with a velocity transformation from the end-effector frame to the world frame based on the inverse of the $\mathbf{T}_6$ matrix. Starting with Eq. 8.3 we write

$$\begin{aligned} {}^0\boldsymbol{\nu} &= {}^0\boldsymbol{J}_N\, {}^N\boldsymbol{\nu} \\ &= \boldsymbol{J}_v({}^N\boldsymbol{T}_0)\, {}^N\boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} \\ &= {}^0\boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} \end{aligned}$$

8.1.3 Analytical Jacobian

In Eq. 8.2 the spatial velocity was expressed in terms of translational and angular velocity vectors. It can be more intuitive to consider the rotational velocity in terms of rates of change of roll-pitch-yaw angles or Euler angles. Consider the case of roll-pitch-yaw angles $\boldsymbol{\Gamma} = (\theta_r, \theta_p, \theta_y)$ for which the rotation matrix is

$$\begin{aligned} \boldsymbol{R} &= \boldsymbol{R}_x(\theta_r)\boldsymbol{R}_y(\theta_p)\boldsymbol{R}_z(\theta_y) \\ &= \begin{pmatrix} c\theta_p c\theta_y & -c\theta_p s\theta_y & s\theta_p \\ c\theta_r s\theta_y + c\theta_y s\theta_p s\theta_r & -s\theta_p s\theta_r s\theta_y + c\theta_r c\theta_y & -c\theta_p s\theta_r \\ s\theta_r s\theta_y - c\theta_r c\theta_y s\theta_p & c\theta_r s\theta_p s\theta_y + c\theta_y s\theta_r & c\theta_p c\theta_r \end{pmatrix} \end{aligned}$$

where we use the shorthand $c\theta$ and $s\theta$ to mean $\cos\theta$ and $\sin\theta$ respectively. With some tedium we can write the derivative $\dot{\boldsymbol{R}}$ and recalling Eq. 3.4

$$\dot{\boldsymbol{R}} = \boldsymbol{S}(\omega)\boldsymbol{R}$$

we can solve for ω in terms of roll-pitch-yaw angles and rates to obtain

$$\begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} = \begin{pmatrix} s\theta_p \dot{\theta}_y + \dot{\theta}_r \\ -c\theta_p s\theta_r \dot{\theta}_y + c\theta_r \dot{\theta}_p \\ c\theta_p c\theta_r \dot{\theta}_y + s\theta_r \dot{\theta}_p \end{pmatrix}$$

which can be factored as

$$\omega = \begin{pmatrix} 1 & 0 & s\theta_p \\ 0 & c\theta_r & -c\theta_p s\theta_r \\ 0 & s\theta_r & c\theta_p c\theta_r \end{pmatrix} \begin{pmatrix} \dot{\theta}_r \\ \dot{\theta}_p \\ \dot{\theta}_y \end{pmatrix}$$

and written concisely as

$$\omega = \boldsymbol{B}(\boldsymbol{\Gamma})\dot{\boldsymbol{\Gamma}}$$

This matrix $\boldsymbol{B}$ is itself a Jacobian that maps roll-pitch-yaw angle rates to angular velocity. It can be computed by the Toolbox function

```
>> rpy2jac(0.1, 0.2, 0.3)
ans =
    1.0000         0    0.1987
         0    0.9950   -0.0978
         0    0.0998    0.9752
```

The analytical Jacobian is

$$\boldsymbol{J}_a(\boldsymbol{q}) = \begin{pmatrix} \boldsymbol{I}_{3\times 3} & \boldsymbol{0}_{3\times 3} \\ \boldsymbol{0}_{3\times 3} & \boldsymbol{B}^{-1}(\boldsymbol{\Gamma}) \end{pmatrix} \boldsymbol{J}(\boldsymbol{q})$$

provided that $\boldsymbol{B}$ is not singular. $\boldsymbol{B}$ is singular when $\cos\phi = 0$ or pitch angle $\phi = \pm\frac{\pi}{2}$ and is referred to as a representational singularity. A similar approach can be taken for Euler angles using the corresponding function `eul2jac`.

The analytical Jacobian can be computed by passing an extra argument to the Jacobian function `jacob0`, for example

```
>> p560.jacob0(qn, 'eul');
```

to specify the Euler angle analytical form.

8.1.4 Jacobian Condition and Manipulability

We have discussed how the Jacobian matrix maps joint rates to end-effector Cartesian velocity but the inverse problem has strong practical use – what joint velocities are needed to achieve a required end-effector Cartesian velocity? We can invert Eq. 8.2 and write

$$\dot{\boldsymbol{q}} = \boldsymbol{J}(\boldsymbol{q})^{-1}\boldsymbol{\nu} \tag{8.5}$$

provided that $\boldsymbol{J}$ is square and non singular. This is the basis of a motion control algorithm known as resolved-rate motion control which will be discussed in the next section.

For an N-link robot the Jacobian is a $6 \times N$ matrix so a square Jacobian requires a robot with 6 joints. A robot configuration $\boldsymbol{q}$ at which $\det(\boldsymbol{J}(\boldsymbol{q})) = 0$ is described as singular or degenerate. Singularities occur when one or more axes become aligned resulting in the loss of degrees of freedom – the gimbal lock problem again.

For example at the Puma's *ready* pose two of the wrist joints (joints 4 and 6) are aligned resulting in the loss of one degree of freedom. The Jacobian is this case is

```
>> J = p560.jacob0(qr)
J =
    0.1500   -0.8636   -0.4318         0         0         0
    0.0203    0.0000    0.0000         0         0         0
         0    0.0203    0.0203         0         0         0
         0         0         0         0         0         0
         0   -1.0000   -1.0000         0   -1.0000         0
    1.0000    0.0000    0.0000    1.0000    0.0000    1.0000
```

which is rank deficient. The rank is only

```
>> rank(J)
ans =
     5
```

compared to a maximum of six for the 6×6 Jacobian. Looking at the Jacobian it is clear that columns 4 and 6 are identical meaning that motion of these joints will result in the same Cartesian velocity. The function `jsingu` performs this analysis automatically, for example

```
>> jsingu(J)
1 linearly dependent joints:
  q6 depends on: q4
```

For the Puma 560 robot arm joints 4 and 6 are the only ones that can become aligned and lead to singularity. The offset distances, d_j and a_j, between links prevents other axes becoming aligned.

indicating velocity of q_6 can be expressed completely in terms of the velocity of q_4.

However if the robot is close to, but not actually at, a singularity we encounter problems where some Cartesian end-effector velocities require very high joint rates – at the singularity those rates will go to infinity. We can illustrate this by choosing a pose slightly away from `qr` which we just showed was singular. We set q_5 to a small but non-zero value of 5 deg

```
>> q = qr
>> q(5) = 5 * pi/180
q =
         0    1.5708   -1.5708         0    0.0873         0
```

and the Jacobian is now

```
>> J=p560.jacob0(q);
```

To achieve relatively slow end-effector motion of 0.1 m s^{-1} in the z-direction requires

```
>> qd = inv(J)*[0 0 0.1 0 0 0]' ;
>> qd'
ans =   -0.0000   -4.9261    9.8522    0.0000   -4.9261         0
```

very high-speed motion of the shoulder and elbow – the elbow would have to move at 9.85 rad s^{-1} or nearly 600 deg s^{-1}. The reason is that although the robot is no longer at a singularity, the determinant of the Jacobian is still very small

```
>> det(J)
ans =
  -1.5509e-05
```

Alternatively we can say that its condition number is very high

```
>> cond(J)
ans =
  235.2498
```

and the Jacobian is *poorly conditioned*.

However for some motions, such as rotation in this case, the poor condition of the Jacobian is not problematic. If we wished to rotate the tool about the y-axis then

```
>> qd = inv(J)*[0 0 0 0 0.2 0]';
>> qd'
ans =    0.0000   -0.0000         0    0.0000   -0.2000         0
```

the required joint rates are very modest.

This leads to the concept of manipulability. Consider the set of joint velocities with a unit norm

$$\dot{\boldsymbol{q}}^T \dot{\boldsymbol{q}} = 1$$

which lie on the surface of a hypersphere in the N-dimensional joint velocity space. Substituting Eq. 8.5 we can write

$$\boldsymbol{\nu}^T \left(\boldsymbol{J}(\boldsymbol{q}) \boldsymbol{J}(\boldsymbol{q})^T \right)^{-1} \boldsymbol{\nu} = 1$$

which is the equation of points on the surface of a 6-dimensional ellipsoid in the end-effector velocity space. If this ellipsoid is close to spherical, that is, its radii are of the same order of magnitude then all is well – the end-effector can achieve arbitrary Cartesian velocity. However if one or more radii are very small this indicates that the end-effector cannot achieve velocity in the directions corresponding to those small radii.

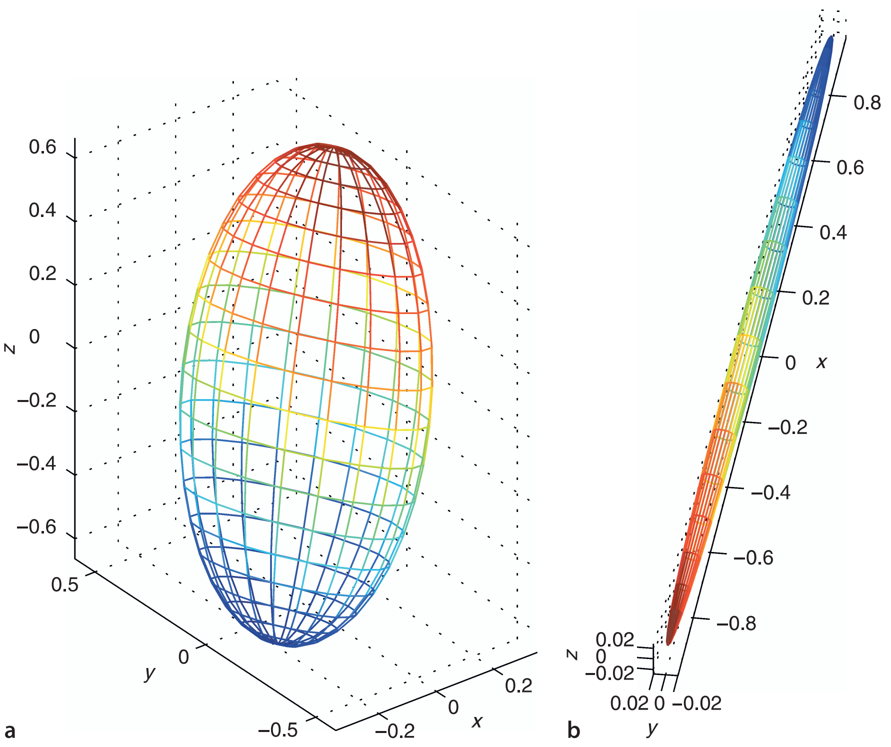


Fig. 8.2. End-effector velocity ellipsoids. **a** Translational velocity ellipsoid for the nominal pose; **b** rotational velocity ellipsoid for a near singular pose, the ellipsoid is an elliptical plate

To illustrate this we return the robot to the *nominal* configuration and compute the ellipsoid corresponding to *translational* velocity◂ in the world frame

Since we can only plot three dimensions.

```
>> J = p560.jacob0(qn);
>> J = J(1:3, :);
```

and plot the corresponding velocity ellipsoid

```
>> plot_ellipse(J*J')
```

which is shown in Fig. 8.2a. We see that the end-effector can achieve higher velocity in the y- and z-directions than in the x-direction. Ellipses and ellipsoids are discussed in more detail in Appendix E.

The *rotational* velocity ellipsoid for the near singular case

```
>> J = p560.jacob0(qr);
>> J = J(4:6, :);
>> plot_ellipse(J*J')
```

is shown in Fig. 8.2b and is an elliptical plate with almost zero thickness.◂ This indicates an inability to rotate about the direction corresponding to the small radius, which in this case is rotation about the x-axis. This is the degree of freedom that was lost – both joints 4 and 6 provide rotation about the world z-axis, joint 5 provides provides rotation about the world y-axis, but none allow rotation about the world x-axis. The shape of the ellipsoid describes how *well-conditioned* the manipulator is for making certain motions. Manipulability is a succinct scalar measure that describes how spherical the ellipsoid is, for instance the ratio of the smallest to the largest radius.◂ The Toolbox method `maniplty` computes Yoshikawa's manipulability measure

This is much easier to see if you change the viewpoint interactively.

The radii are the square roots of the eigenvalues of the $\boldsymbol{J}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q})^T$ as discussed in Appendix E.

$$m = \sqrt{\det(\boldsymbol{J}\boldsymbol{J}^T)}$$

which is proportional to the volume of the ellipsoid. For example

```
>> p560.maniplty(qr, 'yoshikawa')
ans =
     0
```

indicates a total lack of manipulability in this pose – the robot is at a singularity. At the nominal pose the manipulability is higher

The manipulability measure combines translational and rotational velocity information which have different units. The options 'T' and 'R' can be used to compute manipulability on just the translational or rotational velocity respectively.

```
>> p560.maniplty(qn, 'yoshikawa')
ans =
    0.0786
```

but still not particularly high. In practice we find that the seemingly large workspace of a robot is greatly reduced by joint limits, self collision, singularities and regions of reduced manipulability. This measure is based only on the kinematics of the mechanism and does not take into account mass and inertia – it is easier to move a small wrist joint than the larger waist joint. Other manipulability measures, based on acceleration and inertia, take this into account and are discussed in Sect. 9.1.6.

8.2 Resolved-Rate Motion Control

Resolved-rate motion control exploits Eq. 8.5

$$\dot{q} = J(q)^{-1}\nu$$

to map or *resolve* desired Cartesian velocity to joint velocity without requiring inverse kinematics as we used earlier. For now we will assume that the Jacobian is square (6 × 6) and non-singular but we will relax these constraints later.

The motion control scheme is typically implemented in discrete-time form as

$$\begin{aligned}\dot{q}^*\langle k\rangle &= J\big(q\langle k\rangle\big)^{-1}\nu^* \\ q^*\langle k+1\rangle &= q\langle k\rangle + \delta t\dot{q}^*\langle k\rangle\end{aligned} \tag{8.6}$$

which is essentially an integrator that gives the desired joint angles for the next time step, $q^*\langle k+1\rangle$ in terms of the current joint angles and the desired end-effector velocity ν^*.

The algorithm is implemented by the Simulink® model

```
>> sl_rrmc
```

shown in Fig. 8.3. The Cartesian velocity is a constant 0.1 m s^{-1} in the y-direction. The `Jacobian` block has as its input the current manipulator joint angles and outputs a 6 × 6 Jacobian matrix. This is inverted and multiplied by the desired velocity to form the desired joint rates. The robot is modelled by an integrator, that is, it acts as a velocity servo.

In this model we assume that the robot is perfect, that is, the actual joint angles are equal to the desired joint angles q^*. The issue of tracking error is discussed in Sect. 9.4.

To run the simulation

```
>> r = sim('sl_rrmc');
```

and we see an animation of the manipulator end-effector moving at constant velocity in Cartesian space. Simulation results are returned in the simulation object `r` from which we extract time and joint coordinates

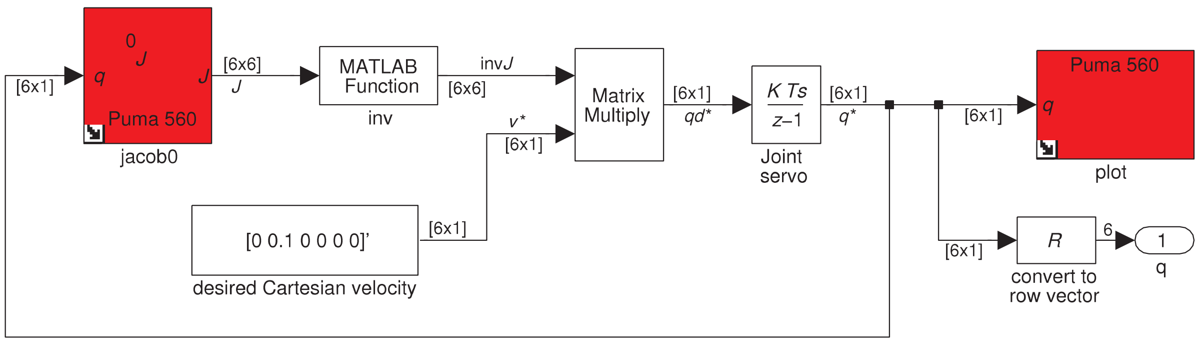


Fig. 8.3. The Simulink® model `sl_rrmc` for resolved-rate motion control for constant end-effector velocity

```
>> t = r.find('tout');
>> q = r.find('yout');
```

We apply forward kinematics to determine the end-effector position

```
>> T = p560.fkine(q);
>> xyz = transl(T);
```

The function `mplot` is a Toolbox utility that plots columns of a matrix in separate subgraphs.

which we then plot◄ as a function of time

```
>> mplot(t, xyz(:,1:3))
```

which is shown in Fig. 8.4a. The Cartesian motion is 0.1 m s^{-1} in the y-direction as demanded but we observe some small and unwanted motion in the x- and z-directions. The motion of the first three joints

```
>> mplot(t, q(:,1:3))
```

is shown in Fig. 8.4b and is not linear with time – reflecting the changing kinematic configuration of the arm.

The approach just described, based purely on integration, suffers from an accumulation of error which we observed as the unwanted x- and z-direction motion in Fig. 8.4a. We can eliminate this by changing the algorithm to a *closed-loop* form based on the difference between the desired and actual pose

$$\dot{\boldsymbol{q}}^*\langle k\rangle = \boldsymbol{J}\left(\boldsymbol{q}\langle k\rangle\right)^{-1}\left(\xi^*\langle k\rangle \ominus \mathcal{K}\left(\boldsymbol{q}\langle k\rangle\right)\right) \tag{8.7}$$

$$\boldsymbol{q}^*\langle k{+}1\rangle = \boldsymbol{q}\langle k\rangle + K_p \delta t \dot{\boldsymbol{q}}^*\langle k\rangle$$

where K_p is a proportional gain. The input is now the desired pose $\xi^*\langle k\rangle$ as a function of time rather than ν^*. The current pose is determined by forward kinematics based on the measured joint coordinates. The difference between two poses is a 6-vector computed using the function $\Delta(\cdot)$ given by Eq. 3.10 and implemented by the Toolbox function `tr2delta`.

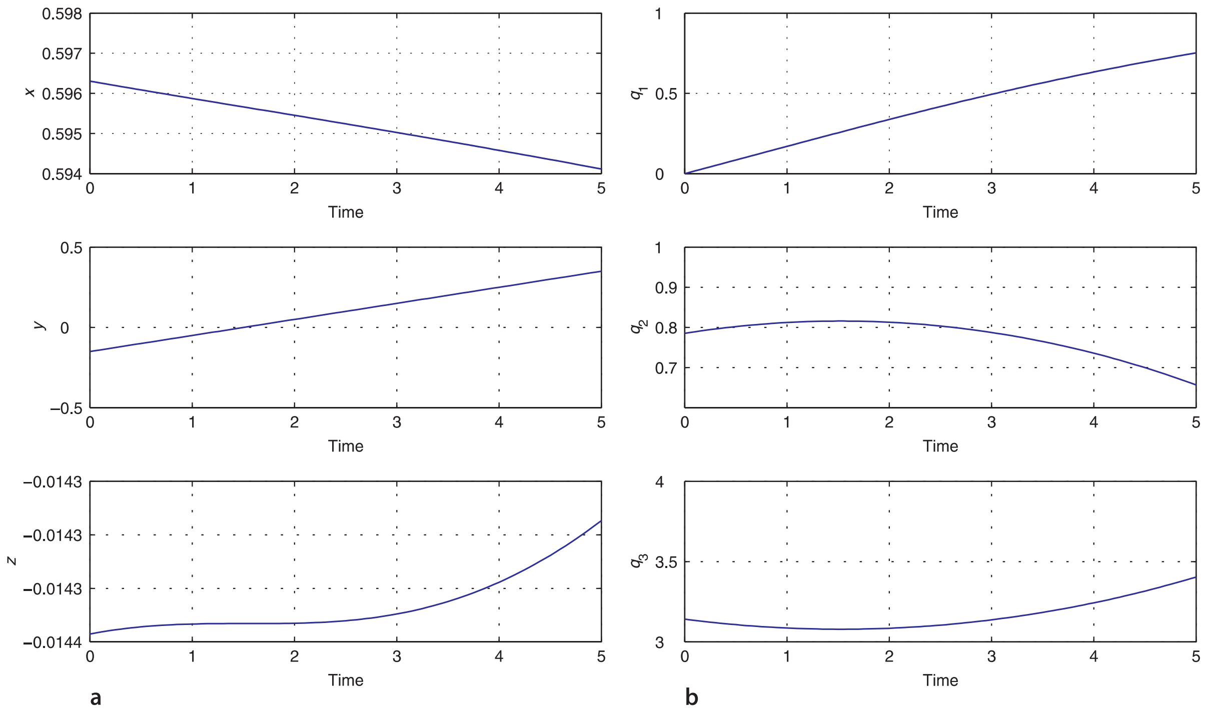


Fig. 8.4. Resolved-rate motion control, Cartesian and joint coordinates versus time. **a** Cartesian end-effector position; **b** joint coordinates

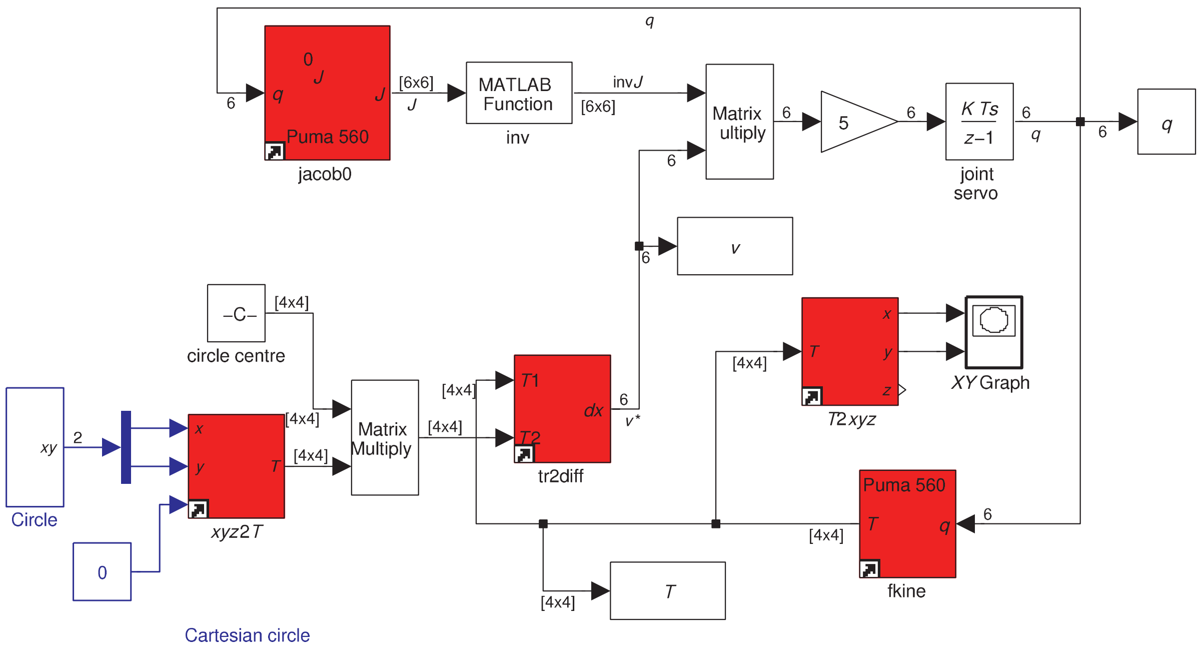


Fig. 8.5. The Simulink® model `sl_rrmc2` for closed-loop resolved-rate motion control with circular end-effector motion

A Simulink® example to demonstrate this for a circular path is

```
>> sl_rrmc2
```

shown in Fig. 8.5. The tool of a Puma 560 robot traces out a circle of radius 50 mm. The *x*-, *y*- and *z*-coordinates as a function of time are computed by the blue colored blocks and converted to a homogenous transformation. The difference between the Cartesian demand and the current Cartesian pose is computed by the `tr2delta` block which produces a Cartesian differential, or scaled spatial velocity, described by a 6-vector. The Jacobian block has as its input the current manipulator joint angles and outputs the Jacobian matrix. The result, after application of a proportional gain, is the joint-space motion required to correct the Cartesian error.

So far we have assumed that the Jacobian is square. For the non-square cases it is helpful to consider the velocity relationship

$$\nu = J(q)\dot{q}$$

in the diagrammatic form shown in Fig. 8.6. The Jacobian is a $6 \times N$ matrix, the joint velocity is an N-vector, and ν is a 6-vector.

The case of $N < 6$ is referred to as under-actuated robot, and $N > 6$ is over-actuated or redundant. The under-actuated case cannot be solved because the system of equations is under-constrained but the system can be *squared up* by deleting some rows of ν and J – accepting that some Cartesian degrees of freedom are not controllable given the low number of joints. For the over-actuated case the system of equations is over-constrained and we could find a least squares solution. Alternatively we can *square up* the Jacobian to make it invertible by deleting some columns – effectively *locking* the corresponding axes.

8.2.1 Jacobian Singularity

For the case of a square Jacobian where $\det(J(q)) = 0$ we cannot solve Eq. 8.5 directly. One simple strategy to deal with singularity is to replace the inverse with the damped inverse Jacobian

$$\dot{q} = (J(q) + pI)^{-1}\nu$$

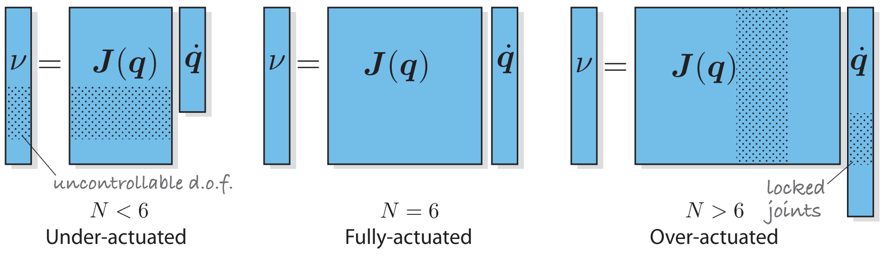


Fig. 8.6. Schematic of Jacobian, ν and $\dot{q}$ for different cases of N. The dotted areas represent matrix regions that could be deleted in order to create a square subsystem capable of solution

where p is a small constant added to the diagonal which places a *floor* under the determinant. However this will introduces some error in $\dot{q}$, which integrated over time could lead to a significant discrepancy in tool position. The closed-loop resolved-rate motion scheme of Eq. 8.7 would minimize any such error.

The pseudo-inverse of the Jacobian J^+ has the property that

$$J^+ J = I$$

just as the inverse does, and is defined as

$$J^+ = (J^T J)^{-1} J^T$$

This is the left generalized- or pseudo-inverse, see Appendix D for more details.

and readily computed using the MATLAB® builtin function `pinv`. The solution

$$\dot{q} = J(q)^+ \nu$$

provides a least squares solution for which $|J\dot{q} - \nu|$ is smallest.

Yet another approach is to delete from the Jacobian all those columns that are linearly dependent on other columns. This is effectively locking the joints corresponding to the deleted columns and we now have an underactuated system which we treat as per the next section.

8.2.2 Jacobian for under-Actuated Robot

An under-actuated robot has $N < 6$, and a Jacobian that is taller than it is wide. For example the two-link manipulator from Sect. 7.3.3 at a nominal pose

```
>> mdl_twolink
>> qn = [1 1];
```

has the Jacobian

```
>> J = jacob0(twolink, qn)
J =
   -1.7508   -0.9093
    0.1242   -0.4161
         0         0
         0         0
         0         0
    1.0000    1.0000
```

We cannot solve the inverse problem Eq. 8.5 using the pseudo-inverse since it will attempt to satisfy motion constraints that the manipulator cannot meet. For example the desired motion of 0.1 m s^{-1} in the x-direction gives the required joint velocity

```
>> qd = pinv(J) * [0.1 0 0 0 0 0]'
qd =
   -0.0698
    0.0431
```

which results in end-effector velocity

```
>> xd = J*qd;
ans =
    0.0829   -0.0266         0         0         0   -0.0266
```

This has the desired motion in the x-direction but undesired motion in y-axis translation and z-axis rotation. The end-effector rotation cannot be independently controlled (since it is a function of q_1 and q_2) yet this solution has taken it into account in the least squares solution.

We have to confront the reality that we have *only* two degrees of freedom which we will use to control just v_x and v_y. We rewrite Eq. 8.2 in partitioned form as

$$\left(\begin{array}{c} v_x \\ v_y \\ \hline v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{array}\right) = \begin{pmatrix} J_{xy} \\ J_0 \end{pmatrix} \begin{pmatrix} \dot{q}_1 \\ \dot{q}_2 \end{pmatrix}$$

and taking the top partition, the first two rows, we write

$$\begin{pmatrix} v_x \\ v_y \end{pmatrix} = J_{xy} \begin{pmatrix} \dot{q}_1 \\ \dot{q}_2 \end{pmatrix}$$

where J_{xy} is a 2×2 matrix. We invert this

$$\begin{pmatrix} \dot{q}_1 \\ \dot{q}_2 \end{pmatrix} = J_{xy}^{-1} \begin{pmatrix} v_x \\ v_y \end{pmatrix}$$

which we can solve if $\det(J_{xy}) \neq 0$.

```
>> Jxy = J(1:2,:);
>> qd = inv(Jxy)* [0.1 0]'
qd =
   -0.0495
   -0.0148
```

which results in end-effector velocity

```
>> xd = J*qd;
>> xd'
ans =
    0.1000    0.0000         0         0         0   -0.0642
```

We have achieved the desired x-direction motion with no unwanted motion apart from the z-axis rotation which is unavoidable – we have used the two degrees of freedom to control x- and y-translation, not z-rotation.

8.2.3 Jacobian for over-Actuated Robot

An over-actuated or redundant robot has $N > 6$, and a Jacobian that is wider than it is tall. In this case we rewrite Eq. 8.5 to use the left pseudo-inverse

$$\dot{\boldsymbol{q}} = \boldsymbol{J}(\boldsymbol{q})^{+} \boldsymbol{\nu} \tag{8.8}$$

which, of the infinite number of solutions possible, will yield the one for which $|\dot{\boldsymbol{q}}|$ is smallest – the minimum-norm solution.

For example, consider the 8-axis P8 robot from Sect. 7.3.4 at a nominal pose

```
>> qn8 = [0 0 qn];
```

and its Jacobian

```
>> J = jacob0(p8, qn8);
>> about(J)
J [double] : 6x8 (384 bytes)
```

is a 6×8 matrix. Now consider that we want the end-effector to move at 0.2 m s^{-1} in the x-, y- and z-directions. Using Eq. 8.8 we compute the required joint rates

```
>> xd = [0.2 0.2 0.2 0 0 0]';
>> q = pinv(J) * xd;
>> q'
ans =
    0.1801    0.1800    0.0336    0.3197    0.0322    0.0475   -0.3519   -0.0336
```

We see that all eight joints have non-zero velocity and contribute to the desired end-effector motion. If the robot follows a repetitive path the joint angles may *drift* over time, that is they may not follow a repetitive path, potentially moving toward joint limits. We can use null-space control to provide additional constraints.

The Jacobian has eight columns and a rank of six

```
>> rank(J)
ans =
     6
```

and a null-space whose basis has two columns

```
>> N = null(J)
N =
     0.2543   -0.0320
     0.1086    0.2635
    -0.1821   -0.4419
     0.3543   -0.1534
    -0.7260    0.3144
    -0.2576   -0.6250
     0.3718   -0.1610
     0.1821    0.4419
```

These columns are orthogonal vectors that span the null-space, that is, any joint velocity that is a linear combination of these two column vectors will result in *no* end-effector motion. We can demonstrate this by

```
>> norm( J * (N(:,1) + N(:,2)) )
ans =
   5.7168e-16
```

This is remarkably useful because it allows Eq. 8.8 to be written as

$$\dot{\boldsymbol{q}} = \underbrace{\boldsymbol{J}(\boldsymbol{q})^{+}\boldsymbol{\nu}}_{\text{end-effector motion}} + \underbrace{\boldsymbol{N}\boldsymbol{N}^{+}\dot{\boldsymbol{q}}_{ns}}_{\text{null-space motion}} \tag{8.9}$$

where the $N \times N$ matrix $\boldsymbol{N}\boldsymbol{N}^{+}$ *projects* the desired joint motion into the null-space so that it will not affect the end-effector Cartesian motion, allowing the two motions to be superimposed.

Null-space motion can be used for highly-redundant robots to avoid collisions between the links and obstacles (including other links), or to keep joint coordinates away from their mechanical limit stops. Consider that in addition to the desired Cartesian velocity `xd` we wish to simultaneously move joint 5 (the Puma's elbow joint) angle closer to zero (it is currently π), so we set a desired joint velocity

```
>> qd_ns = [0 0 0 0 -0.1 0 0 0]';
```

and project it into the null-space

```
>> qp = N * pinv(N) * qd_ns;
>> qp'
    0.0195   -0.0004    0.0007    0.0305   -0.0626    0.0009    0.0321   -0.0007
```

The projection has introduced a scaling, the joint 5 velocity is not the -0.1 we desired but we can apply a scale factor to correct this

```
>> qp = qp / qp(5) * qd_ns(5)
qp =
    0.0312   -0.0006    0.0011    0.0487   -0.1000    0.0014    0.0513   -0.0011
```

The other joint velocities provide the required compensating motion in order that the end-effector pose is not disturbed as shown by

```
>> norm( J * qp)
ans =
   3.1107e-17
```

8.3 Force Relationships

Forces and wrenches are properly the subject of the next chapter, about dynamics, but it is helpful now to introduce another very useful property of the Jacobian.

In the earlier discussion of motion we introduced the concept of a spatial velocity $\boldsymbol{\nu} = (v_x, v_y, v_z, \omega_x, \omega_y, \omega_z)$. For forces there is a spatial equivalent called a wrench $\boldsymbol{g} = (f_x, f_y, f_z, m_x, m_y, m_z) \in \mathbb{R}^6$ which is a vector of forces and moments.

8.3.1 Transforming Wrenches between Frames

Just as the Jacobian transforms a spatial velocity from one coordinate frame to another using Eq. 8.3 it can be used to transform wrenches between coordinate frames

$$^{B}\boldsymbol{g} = ({}^{A}\boldsymbol{J}_{B})^{T}\, {}^{A}\boldsymbol{g} \tag{8.10}$$

where ${}^{A}J_{B}$ is given by Eq. 8.4 and is a function of the relative pose ${}^{A}\boldsymbol{T}_{B}$ from frame $\{A\}$ to frame $\{B\}$. Note that the force transform differs from the velocity transform in using the transpose rather than the inverse of the Jacobian, and is therefore never singular.

For example consider two frames, displaced by 2 m in the x-direction. We create the Jacobian

```
>> J = tr2jac( transl(2, 0, 0) );
```

Then a force of 3 N in the y-direction of the first frame is transformed to

```
>> F = J'*[0 3 0 0 0 0]';
>> F'
ans =
     0     3     0     0     0     6
```

a force of 3 N force in the y-direction *plus* a moment of 6 N m about the z-axis in the second frame due to a *lever arm* effect.

8.3.2 Transforming Wrenches to Joint Space

The manipulator Jacobian transforms joint velocity to an end-effector spatial velocity according to Eq. 8.2 and the Jacobian transpose transforms a wrench applied at the end-effector to torques and forces experienced at the joints▸

Derived through the principle of virtual work, see for instance Spong et al. (2006, sect. 4.10).

$$\boldsymbol{Q} = {}^{0}\boldsymbol{J}(\boldsymbol{q})^{T}\, {}^{0}\boldsymbol{g} \tag{8.11}$$

where $\boldsymbol{g}$ is a wrench in the world coordinate frame and $\boldsymbol{Q}$ is the generalized joint force vector. The elements of $\boldsymbol{Q}$ are joint torque or force for revolute or prismatic joints respectively.

If the wrench is defined in the end-effector coordinate frame then we use instead

$$\boldsymbol{Q} = {}^{N}\boldsymbol{J}(\boldsymbol{q})^{T}\,{}^{N}\boldsymbol{g} \tag{8.12}$$

Interestingly this mapping from external quantities (the wrench) to joint quantities (the generalized forces) can never be singular as it can be for velocity. We exploit this property in the next section to solve the inverse kinematic problem numerically.

For the Puma 560 robot in its nominal pose, see Fig. 8.1, a force of 20 N in the world y-direction results in joint torques of

```
>> tau = p560.jacob0(qn)' * [0 20 0 0 0 0]';
>> tau'
ans =
   11.9261    0.0000    0.0000         0         0         0
```

The force pushes the arm *sideways* and only the waist joint will rotate in response – experiencing a torque of 11.93 Nm due to a lever arm effect. A force of 20 N applied in the world x-direction results in joint torques of

```
>> tau = p560.jacob0(qn)' * [20 0 0 0 0 0]';
>> tau'
ans =
    3.0010    0.2871    6.3937         0         0         0
```

which is pulling the end-effector away from the base which results in torques being applied to the first three joints.

8.4 Inverse Kinematics: a General Numerical Approach

In Sect. 7.3 we solved the inverse kinematic problem using an explicit solution that required the robot to have 6 joints and a spherical wrist. For the case of robots which do not meet this specification, for example those with more or less than 6 joints, we need to consider a numerical solution. Here we will develop an approach based on the forward kinematics and the Jacobian transpose which we can compute for any manipulator configuration – these functions have no singularities.

The principle is shown in Fig. 8.7. The virtual robot is drawn solidly in its current pose and faintly in the desired pose. From the overlaid pose graph we write

$$\xi_E^* = \xi_E \oplus \xi_\Delta$$

which we can rearrange as

$$\xi_\Delta = \ominus\xi_E \oplus \xi_E^*$$

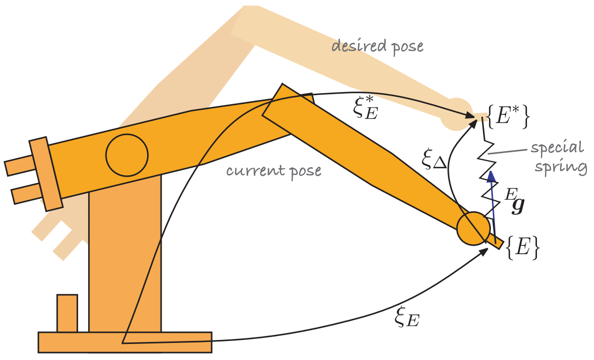


Fig. 8.7. Schematic of the numerical inverse kinematic approach, showing the current ξ_E and the desired ξ_E^* manipulator pose

We postulate a *special* spring between the end-effector of the two poses which is pulling (and twisting) the robot's end-effector toward the desired pose▸ with a wrench proportional to the *difference* in pose

It has to be a *special* spring in order to change the orientation as well as the position of the end-effector.

$$^{E}\boldsymbol{g} \propto \Delta(\xi_E, \xi_E^*)$$

where the function $\Delta(\cdot)$ given by Eq. 3.10 approximates the difference between two poses as a 6-vector comprising translational and rotational displacement.▸ The wrench is also a 6-vector and comprises forces and moments. We write

This assumes that the difference in orientation between the two poses is "small" so the approximation becomes better as the solution converges.

$$^{E}\boldsymbol{g} = \gamma\Delta(\xi_E, \xi_E^*) \tag{8.13}$$

where γ is a constant and the current pose is computed using forward kinematics

$$\xi_E\langle k\rangle = \mathcal{K}\big(\boldsymbol{q}\langle k\rangle\big) \tag{8.14}$$

where $\boldsymbol{q}\langle k\rangle$ is the current estimate of the inverse kinematic solution.

The end-effector wrench Eq. 8.13 is *resolved* to joint forces

$$\boldsymbol{Q}\langle k\rangle = {}^{N}\boldsymbol{J}\big(\boldsymbol{q}\langle k\rangle\big)^{T}\, {}^{E}\boldsymbol{g}\langle k\rangle \tag{8.15}$$

using the Jacobian transpose Eq. 8.12. We assume that the virtual robot has no joint motors only viscous dampers so the joint velocity due to the applied forces will be proportional

$$\dot{\boldsymbol{q}}\langle k\rangle = \boldsymbol{Q}\langle k\rangle / B$$

where B is the joint damping coefficients (we assume all dampers are the same). Now we can write a discrete-time update for the joint coordinates

$$\boldsymbol{q}\langle k{+}1\rangle = \alpha\dot{\boldsymbol{q}}\langle k\rangle + \boldsymbol{q}\langle k\rangle \tag{8.16}$$

where α is some well chosen gain. We iterate Eq. 8.13 to Eq. 8.16 until the magnitude of the wrench $^{E}\boldsymbol{g}$ is sufficiently small. Using the Jacobian transpose we do not face the problem of having to invert a Jacobian which is potentially non-square or singular.

In Section 7.3.3 we used a mask vector when computing the inverse kinematics of a robot with $N < 6$. The mask vector $\boldsymbol{m}$ can be included in Eq. 8.15 which becomes

$$\boldsymbol{Q}\langle k\rangle = {}^{N}\boldsymbol{J}\big(\boldsymbol{q}\langle k\rangle\big)^{T} \operatorname{diag}(\boldsymbol{m})\, {}^{E}\boldsymbol{g}\langle k\rangle \tag{8.17}$$

8.5 Wrapping Up

In this chapter we have learnt about Jacobians and their application to robotics. The manipulator Jacobian captures the relationship between the rate of change of joint coordinates and the spatial velocity of the end-effector. The numerical properties of the Jacobian tell us about manipulability, that is how well the manipulator is able to move in different directions. The extreme case, singularity, is indicated by linear dependence between columns of the Jacobian. We showed how the inverse Jacobian can be used to resolve desired Cartesian velocity into joint velocity as an alternative means of generating Cartesian paths for under- and over-actuated robots. For over-actuated robots we showed how null-space motions can be used to move the robot's joints without affecting the end-effector pose.

We created other Jacobians as well. A Jacobian can be used to to map spatial velocities between coordinate frames. The analytic Jacobian maps angular velocity to roll-pitch-yaw or Euler angle rates.

The Jacobian transpose is used to map wrenches applied at the end-effector to joint torques, and also to map wrenches between coordinate frames. We showed, finally, how to use the Jacobian transpose and forward kinematics to compute inverse kinematics numerically for arbitrary robots and singular poses.

Further Reading

The manipulator Jacobian is covered by almost all standard robotics texts such as Spong et al. (2006), Craig (2004), Siciliano et al. (2008), Paul (1981), and the handbook (Siciliano and Khatib 2008, § 1). An excellent discussion of manipulability and velocity ellipsoids is provided by Siciliano et al. (2008), and the most common manipulability measure is that propsed by Yoshikawa (1984). Computing the manipulator Jacobian based on Denavit-Hartenberg was first described by Paul and Shimano (1978).

The resolved-rate motion control scheme was proposed by Whitney (1969). Extensions such as pseudo-inverse Jacobian-based control are reviewed by Klein and Huang (1983) and damped least-square methods are reviewed by Deo and Walker (1995). The approach to numeric inverse kinematics used in the Toolbox is based on results for control through singularities (Chiaverini et al. 1991).

Exercises

1. For the Puma 560 robot can you devise a configuration in which three joint axes are parallel?
2. Derive the analytical Jacobian for Euler angles.
3. Manipulability (page 177)
   a) Plot the velocity ellipse (x- and y-velocity) for the two-link manipulator at a grid of end-effector positions in its workspace. Each ellipsoid should be centred on the end-effector position.
   b) For the Puma 560 manipulator find a configuration where manipulability is greater than at `qn`.
   c) Overlay the translational or rotational velocity ellipsoid on the manipulator as displayed by the `plot` method, and create an animation that shows how it changes shape as the robot moves along a trajectory.
4. Resolved-rate motion control (page 180)
   a) Experiment with different Cartesian translational and rotational velocity demands, and combinations.
   b) Extend the Simulink® system of Fig. 8.4 to also record the determinant of the Jacobian matrix to the workspace.
   c) In Fig. 8.4 the robot's motion is simulated for 5 s. Extend the simulation time to 10 s and explain what happens.
   d) Set the initial pose and direction of motion to mimic that of Sect. 7.4.3. What happens when the robot reaches the singularity?
   e) Replace the Jacobian inverse block in Fig. 8.3 with the MATLAB® function `pinv`.
   f) Replace the Jacobian inverse block in Fig. 8.3 with a damped least squares function, and investigate the effect of different values of the damping factor.
   g) Replace the Jacobian inverse block in Fig. 8.3 with a block based on the MATLAB® function `lscov`.
   h) Modify the simulation of Fig. 8.5 to use the 8-axis robot from Sect. 7.3.4. Observe the joint coordinate trajectory, is it repetitive like the end-effector trajectory?

   i) Modify the above to include null-space motion to keep the robot joints away from their limits.
5. For the over-actuated P8 robot (page 150)
   a) Develop a null-space controller that keeps the last six joints in the middle of their working range by using the first two joints to position the base of the Puma. Modify this so as to maximize the manipulability of the P8 robot. Consider now that the Puma robot is mounted on a non-holonomic robot, create a controller that generates appropriate steering and velocity inputs to the mobile robot (challenging).
   b) For an arbitrary pose and end-point spatial velocity we will move six joints and lock two joints. Write an algorithm to determine which two joints should be locked.