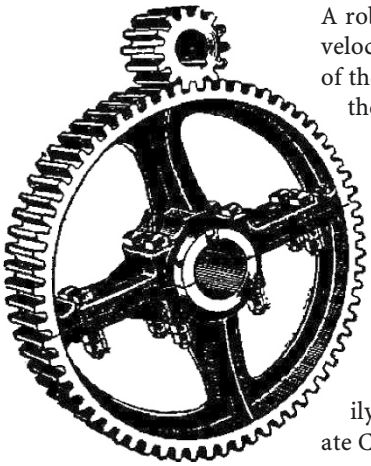
Chapter

8 Manipulator Velocity



A robot's end-effector moves in Cartesian space with a translational and rotational velocity – a spatial velocity. However that velocity is a consequence of the velocities of the individual robot joints. In this chapter we introduce the relationship between the velocity of the joints and the spatial velocity of the end-effector.

The 3-dimensional end-effector pose $\xi_E \in \mathbf{SE}(3)$ has a rate of change which is represented by a 6-vector known as a spatial velocity that was introduced in Sect. 3.1. The joint rate of change and the end-effector velocity are related by the manipulator Jacobian matrix which is a function of manipulator configuration.

Section 8.1 uses a simple planar robot to introduce the manipulator Jacobian and then extends this concept to more general robots. Section 8.2 discusses the numerical properties of the Jacobian matrix which 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. In Sect. 8.3 we use the inverse Jacobian to generate Cartesian paths without requiring inverse kinematics, and this can be applied to over- and under-actuated robots which are discussed in Sect. 8.4. Section 8.5 demonstrates how the Jacobian transpose is used to transform forces from the end-effector to the joints and between coordinate frames. Finally, in Sect. 8.6 the numeric inverse kinematic solution, used in the previous chapter, is introduced and its dependence on the Jacobian matrix is fully described.

8.1 Manipulator Jacobian

In the last chapter we discussed the relationship between joint coordinates and end-effector pose – the manipulator kinematics. Now we investigate the relationship between the rate of change of these quantities – between joint *velocity* and *velocity* of the end-effector. This is called the velocity or differential kinematics of the manipulator.

8.1.1 Jacobian in the World Coordinate Frame

We illustrate the basics with our now familiar 2-dimensional example, see Fig. 8.1, this time defined using Denavit-Hartenberg notation

```
>> mdl_planar2_sym
>> p2
two link:: 2 axis, RR, stdDH
+---+-----------+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |    offset |
+---+-----------+-----------+-----------+-----------+-----------+
|  1|         q1|          0|         a1|          0|          0|
|  2|         q2|          0|         a2|          0|          0|
+---+-----------+-----------+-----------+-----------+-----------+
```

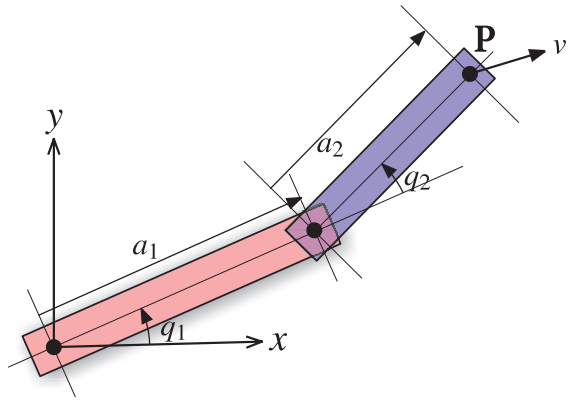


Fig. 8.1. Two-link robot showing the end-effector position $\boldsymbol{p} = (x, y)$ and the Cartesian velocity vector $\nu = \mathrm{d}\boldsymbol{p} / \mathrm{d}t$

and define two real-valued symbolic variables to represent the joint angles

```
>> syms q1 q2 real
```

and then compute the forward kinematics using these

```
>> TE = p2.fkine( [q1 q2] );
```

The position of the end-effector $\boldsymbol{p} = (x, y) \in \mathbb{R}^2$ is

The Toolbox considers robot pose in 3-dimensions using SE(3). This robot operates in a plane, a subset of SE(3), so we select $\boldsymbol{p} = (x, y)$.

```
>> p = TE.t;  p = p(1:2)
p =
 a2*cos(q1 + q2) + a1*cos(q1)
 a2*sin(q1 + q2) + a1*sin(q1)
```

and we compute the derivative of $\boldsymbol{p}$ with respect to the joints variables $\boldsymbol{q}$. Since $\boldsymbol{p}$ and $\boldsymbol{q}$ are both vectors the derivative

$$\frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}\boldsymbol{q}} = \boldsymbol{J}(\boldsymbol{q}) \tag{8.1}$$

will be a matrix – a Jacobian matrix

```
>> J = jacobian(p, [q1 q2])
J =
[ - a2*sin(q1 + q2) - a1*sin(q1), -a2*sin(q1 + q2)]
[   a2*cos(q1 + q2) + a1*cos(q1),  a2*cos(q1 + q2)]
```

which is typically denoted by the symbol $\boldsymbol{J}$ and in this case is 2×2.

To determine the relationship between joint *velocity* and end-effector *velocity* we rearrange Eq. 8.1 as

$$\mathrm{d}\boldsymbol{p} = \boldsymbol{J}(\boldsymbol{q})\mathrm{d}\boldsymbol{q}$$

and divide through by $\mathrm{d}t$ to obtain

$$\frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t} = \boldsymbol{J}(\boldsymbol{q})\frac{\mathrm{d}\boldsymbol{q}}{\mathrm{d}t}$$
$$\dot{\boldsymbol{p}} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}$$

The Jacobian matrix maps velocity from the joint coordinate or configuration space to the end-effector's Cartesian coordinate space and is itself a function of the joint coordinates.

More generally we write the forward kinematics in functional form, Eq. 7.4, as

$${}^0\xi = \mathcal{K}(\boldsymbol{q})$$

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 $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$ 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 details are given in Appendix E.

and taking the derivative we write

$$^{0}\boldsymbol{\nu} = {}^{0}\boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{8.2}$$

where $^{0}\boldsymbol{\nu} = (v_x, v_y, v_z, \omega_x, \omega_y, \omega_z) \in \mathbb{R}^6$ is the spatial velocity, as discussed in Sect. 3.1.1, of the end-effector in the world frame and comprises translational and rotational velocity components. The matrix $^{0}\boldsymbol{J}(\boldsymbol{q}) \in \mathbb{R}^{6\times N}$ is the manipulator Jacobian or the geometric Jacobian. This relationship is sometimes referred to as the instantaneous forward kinematics.

For a realistic 3-dimensional robot this Jacobian matrix can be numerically computed by the `jacob0` method of the `SerialLink` object, based on its Denavit-Hartenberg parameters. For the Puma robot in the pose shown in Fig. 8.2 the Jacobian is

```
>> 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
```

τ is the task space of the robot, typically $\tau \subset \mathbf{SE}(3)$, and $\mathcal{C} \subset \mathbb{R}^N$ is the configuration or joint space of the robot where N is the number of robot joints.

and is a matrix with dimensions dim $\mathcal{T} \times$ dim $\mathcal{C}$ – in this case 6×6. Each row corresponds to a Cartesian degree of freedom. Each column corresponds to a joint – it is the end-effector spatial velocity created by unit velocity of the corresponding joint. In this configuration, motion of joint 1, the first column, causes motion in the world x- and y-directions and rotation about the z-axis. Motion of joints 2 and 3 cause motion in the world x- and z-directions and negative rotation about the y-axis.

Physical insight comes from Fig. 8.2 which shows the joint axes in space. Alternatively you could use the `teach` method

```
>> p560.teach(qn)
```

and jog the various joint angles and observe the change in end-effector pose.

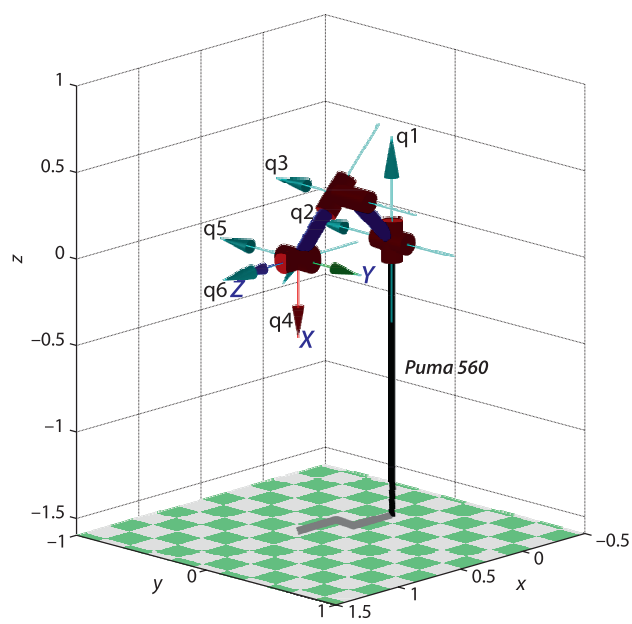


Fig. 8.2. Puma robot in its nominal pose `qn`. The end-effector z-axis points in the world x-direction, and the x-axis points downward

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 Cementary I of the Trinity Church (Dreifaltigkeitskirche) in Berlin.

The 3×3 block of zeros in the top right indicates that motion of the wrist joints have no effect on the end-effector translational motion – this is a consequence of the spherical wrist and the default zero-length tool.

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. To obtain the spatial velocity in the end-effector coordinate frame we introduce the velocity transformation Eq. 3.4 from the world frame to the end-effector frame which is a function of the end-effector pose

$$
{}^{E}\boldsymbol{\nu} = {}^{E}\boldsymbol{J}_0\left({}^{E}\boldsymbol{\xi}_0\right){}^{0}\boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} = \underbrace{\begin{pmatrix} {}^{E}\boldsymbol{R}_0 & \boldsymbol{0}_{3\times3} \\ \boldsymbol{0}_{3\times3} & {}^{E}\boldsymbol{R}_0 \end{pmatrix}}{}^{0}\boldsymbol{J}(\boldsymbol{q})\,\dot{\boldsymbol{q}} = {}^{E}\boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}
$$

which results in a new Jacobian for end-effector velocity.

For historical reasons the Toolbox implementation computes the end-effector Jacobian directly and applies a velocity transform for the world frame Jacobian.

In the Toolbox this Jacobian is computed by the method `jacobe` and for the Puma robot at the pose used above is

```
>> p560.jacobe(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
```

8.1.3 Analytical Jacobian

In Eq. 8.2 the spatial velocity was expressed in terms of translational and angular velocity vectors, however angular velocity is not a very intuitive concept. For some applications it can be more intuitive to consider the rotational velocity in terms of rates of change of roll-pitch-yaw angles or Euler angles. Analytical Jacobians are those where the rotational velocity is expressed in a representation other than angular velocity, commonly in terms of triple-angle rates.

Consider the case of XYZ roll-pitch-yaw angles $\boldsymbol{\Gamma} = (\theta_r, \theta_p, \theta_y)$ for which the rotation matrix is

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

where we use the shorthand $c\theta$ and $s\theta$ to mean $\cos\theta$ and $\sin\theta$ respectively. With some effort we can write the derivative $\dot{\boldsymbol{R}}$ and recalling Eq. 3.1

$$\dot{\boldsymbol{R}} = [\boldsymbol{\omega}]_\times \boldsymbol{R}$$

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

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

which can be factored as

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

and written concisely as

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

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

```
>> rpy2jac(0.1, 0.2, 0.3)
ans =
    0.1987         0    1.0000
   -0.2896    0.9553         0
    0.9363    0.2955         0
```

where the arguments are the roll, pitch and yaw angles. The analytical Jacobian is

$$\boldsymbol{J}_a(\boldsymbol{q}) = \begin{pmatrix} \boldsymbol{I}_{3\times3} & \boldsymbol{0}_{3\times3} \\ \boldsymbol{0}_{3\times3} & \boldsymbol{A}^{-1}(\boldsymbol{\Gamma}) \end{pmatrix} \boldsymbol{J}(\boldsymbol{q})$$

provided that $\boldsymbol{A}$ is not singular. $\boldsymbol{A}$ 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.

Another useful analytical Jacobian expresses angular rates as the rate of change of exponential coordinates $\boldsymbol{s} = \hat{\boldsymbol{v}}\theta \in \mathbf{so}(3)$

$$\omega = \boldsymbol{A}(\boldsymbol{s})\dot{\boldsymbol{s}}$$

where

$$\boldsymbol{A}(s) = \boldsymbol{I} - \frac{1-\cos\theta}{\theta}[\hat{\boldsymbol{v}}]_\times + \frac{\theta - \sin\theta}{\theta}[\hat{\boldsymbol{v}}]_\times^2$$

Implemented by the Toolbox functions `trlog` and `tr2rotvec` or the `SE3` method `torotvec`.

and $\hat{\boldsymbol{v}}$ and θ can be determined from the end-effector rotation matrix via the matrix logarithm.◀

8.2 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

$$\dot{\boldsymbol{q}} = \boldsymbol{J}(\boldsymbol{q})^{-1}\boldsymbol{\nu} \tag{8.3}$$

provided that $\boldsymbol{J}$ is square and nonsingular. The Jacobian is a dim $\mathcal{T} \times$ dim $\mathcal{C}$ matrix so in order to achieve a square Jacobian matrix a robot operating in the task space $\mathcal{T} \subset \mathbf{SE}(3)$, which has 6 spatial degrees-of-freedom, requires a robot with 6 joints.

8.2.1 Jacobian Singularities

A robot configuration $\boldsymbol{q}$ at which $\det(\boldsymbol{J}(\boldsymbol{q})) = 0$ is described as singular or degenerate. Singularities occur when the robot is at maximum reach or 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 the Jacobian

```
>> 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
```

is singular

```
>> det(J)
ans =
     0
```

Digging a little deeper we see that the Jacobian rank is only

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

compared to a maximum of six for a 6×6 matrix. The rank deficiency of one means that one column is equal to a linear combination of other columns. Looking at the Jacobian it is clear that columns 4 and 6 are identical meaning that two of the wrist joints (joints 4 and 6) are aligned. This leads to the loss of one degree of freedom since motion of these joints results in the same Cartesian velocity, leaving motion in one Cartesian direction unaccounted for. The function `jsingu` performs this analysis automatically, for example

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.

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

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 configuration slightly away from `qr` which we just showed was singular. We set q_5 to a small but nonzero value of 5 deg

We observed this effect in Fig. 7.12.

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

and the Jacobian is now

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

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 particular joint configuration is therefore good for certain motions but poor for others.

8.2.2 Manipulability

Consider the set of generalized 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. 8.3 we can write

$$\boldsymbol{\nu}^T \left(\boldsymbol{J}(\boldsymbol{q}) \boldsymbol{J}(\boldsymbol{q})^T \right)^{-1} \boldsymbol{\nu} = 1 \tag{8.4}$$

which is the equation of points on the surface of an ellipsoid within the dim $\mathcal{T}$-dimensional 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.

We will load the numerical, rather than symbolic model, for the planar robot arm of Fig. 8.1

```
>> mdl_planar2
```

which allows us to plot the velocity ellipse for an arbitrary pose

```
>> p2.vellipse([30 40], 'deg')
```

We can also interactively explore how its shape changes with configuration by

```
>> p2.teach('callback', @(r,q) r.vellipse(q), 'view', 'top')
```

which is shown in Fig. 8.3.

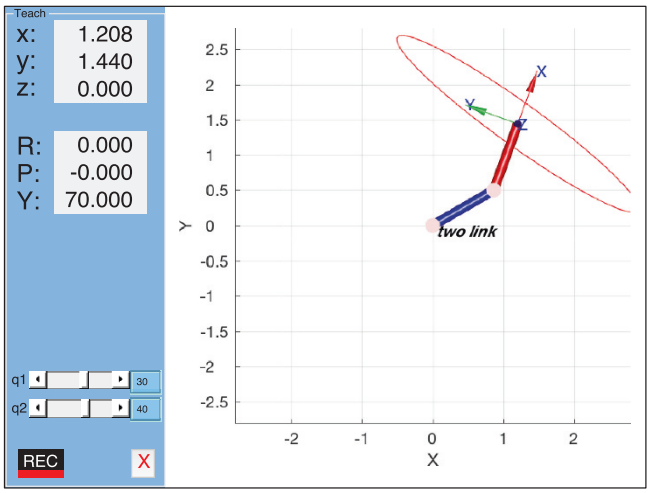


Fig. 8.3. Two-link robot with overlaid velocity ellipse

For a robot with a task space $\mathcal{T} \subset \mathbf{SE}(3)$ Eq. 8.4 describes a 6-dimensional ellipsoid which is impossible to visualize. However we can extract that part of the Jacobian relating to *translational* velocity▸ in the world frame

Since we can only plot three dimensions.

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

and plot the corresponding velocity ellipsoid

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

which is shown in Fig. 8.4a. The Toolbox provides a shorthand for this

```
>> p560.vellipse(qns, 'trans');
```

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 Sect. C.1.4.

The *rotational* velocity ellipsoid for the near singular case

```
>> p560.vellipse(qns, 'rot')
```

is shown in Fig. 8.4b 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.

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

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

The radii are the square roots of the eigenvalues of the $J(\boldsymbol{q})J(\boldsymbol{q})^T$ as discussed in Sect. C.1.4.

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

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

```
>> m = p560.maniplty(qr)
m =
     0
```

indicates a singularity. If the method is called with no output arguments it displays the volume of the translational and rotational velocity ellipsoids

```
>> p560.maniplty(qr)
Manipulability: translation 0.00017794, rotation 0
```

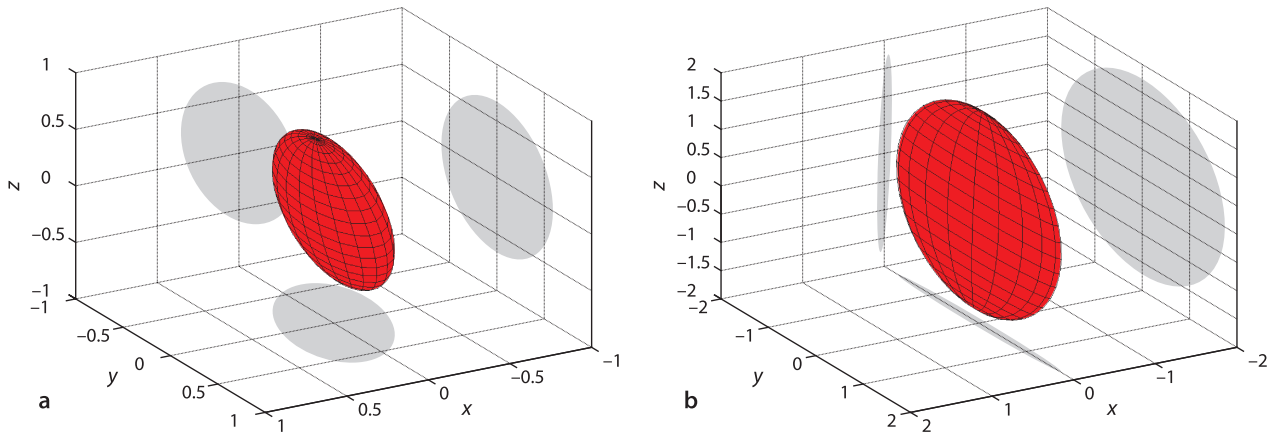


Fig. 8.4. End-effector velocity ellipsoids. **a** Translational velocity ellipsoid for the nominal pose (m s^{-1}); **b** rotational velocity ellipsoid for a near singular pose (rad s^{-1}), the ellipsoid is an elliptical plate

which indicates very poor manipulability for translation and zero for rotation. At the nominal pose the manipulability is higher

```
>> p560.maniplty(qn)
Manipulability: translation 0.111181, rotation 2.44949
```

The manipulability measure combines translational and rotational velocity information which have different units. The options `'trans'` and `'rot'` can be used to compute manipulability on just the translational or rotational velocity respectively.

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. The manipulability measure discussed here is based only on the kinematics of the mechanism. The fact that it is easier to move a small wrist joint than the larger waist joint suggests that mass and inertia should be taken into account and such manipulability measures are discussed in Sect. 9.2.7.

8.3 Resolved-Rate Motion Control

Resolved-rate motion control is a simple and elegant algorithm to generate straight line motion by exploiting Eq. 8.3

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

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

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

$$\begin{aligned} \dot{\boldsymbol{q}}^*\langle k\rangle &= \boldsymbol{J}(\boldsymbol{q}\langle k\rangle)^{-1}\boldsymbol{\nu}^* \\ \boldsymbol{q}^*\langle k{+}1\rangle &\leftarrow \boldsymbol{q}\langle k\rangle + \delta_t\dot{\boldsymbol{q}}^*\langle k\rangle \end{aligned} \tag{8.5}$$

where δ_t is the sample interval. The first equation computes the required joint velocity as a function of the current joint configuration and the desired end-effector velocity $\boldsymbol{\nu}^*$. The second performs forward rectangular integration to give the desired joint angles for the next time step, $\boldsymbol{q}^*\langle k{+}1\rangle$.

An example of the algorithm is implemented by the Simulink® model

```
>> sl_rrmc
```

In this model we assume that the robot is perfect, that is, the actual joint angles are equal to the desired joint angles $\boldsymbol{q}^*$. The issue of tracking error is discussed in Sect. 9.1.7.

shown in Fig. 8.5. The Cartesian velocity is a constant 0.05 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 modeled by a discrete-time integrator – an ideal velocity controller.

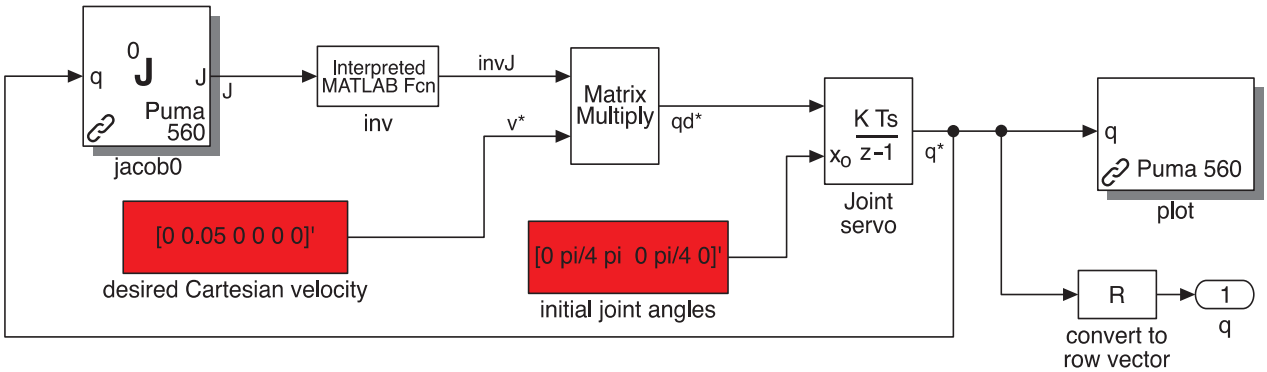


Fig. 8.5. The Simulink® model `sl_rrmc` for resolved-rate motion control for constant end-effector velocity

Running the simulation

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

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

```
>> 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);
```

which we then plot▸ as a function of time

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

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

which is shown in Fig. 8.6a. The Cartesian motion is 0.05 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. 8.6b and are 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. 8.6a. We can eliminate this by changing the algorithm to a *closed-loop* form based on the difference between the desired and actual pose

$$\dot{\boldsymbol{q}}^*\langle k\rangle \leftarrow K_p \boldsymbol{J}\big(\boldsymbol{q}\langle k\rangle\big)^{-1} \Delta\Big(\mathcal{K}\big(\boldsymbol{q}\langle k\rangle\big), \xi^*\langle k\rangle\Big) \tag{8.6}$$

where K_p is a proportional gain, $\Delta(\cdot) \in \mathbb{R}^6$ is a spatial displacement▸ and the desired pose $\xi^*\langle k\rangle$ is a function of time.

See Sect. 3.1.4 for definition.

A Simulink example to demonstrate this for a circular path is

```
>> sl_rrmc2
```

shown in Fig. 8.7 and 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 and converted to a homogeneous transformation by the blocks in the grey area. The *difference* between the desired pose and the current pose from forward kinematics using the $\Delta(\cdot)$ operator is computed by the `tr2delta` block. The result is a spatial displacement, a translation and a rotation described by a 6-vector which is used as

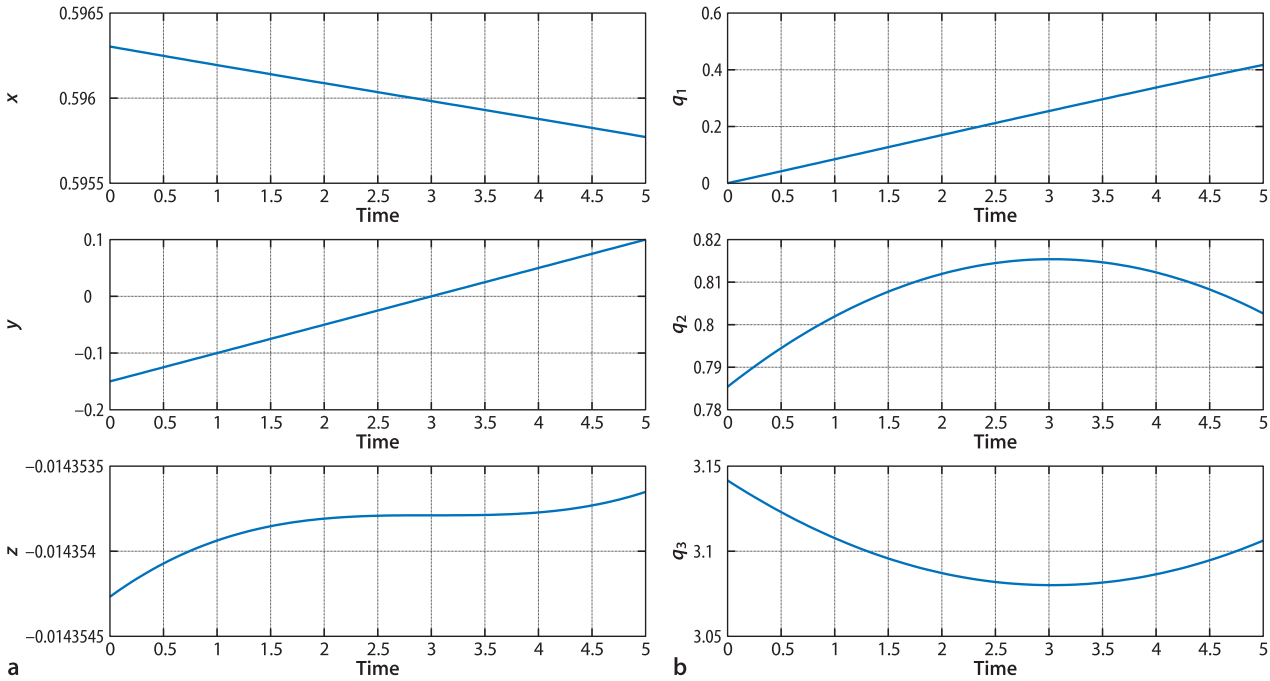


Fig. 8.6. Resolved-rate motion control, Cartesian and joint coordinates versus time. **a** Cartesian end-effector motion. Note the small, but unwanted motion in the *x*- and *z*-directions; **b** joint motion

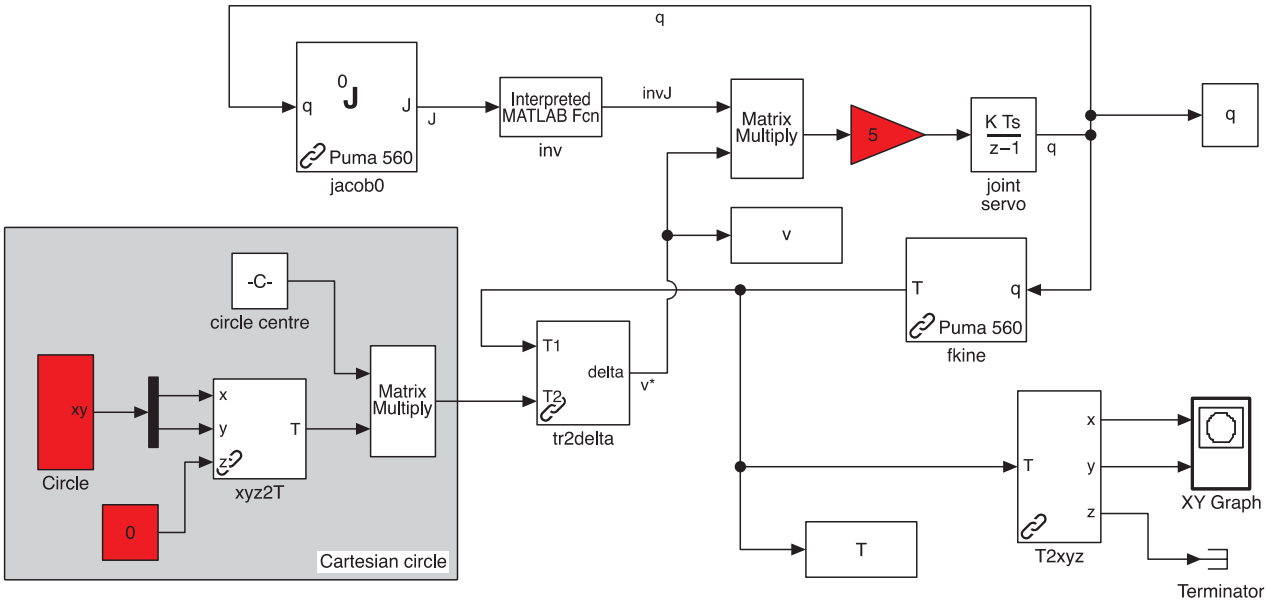


Fig. 8.7. The Simulink® model `sl_rrmc2` for closed-loop resolved-rate motion control with circular end-effector motion

the desired spatial velocity to drive the end-effector toward the desired pose. The Jacobian matrix is computed from the current manipulator joint angles and is inverted so as to transform the desired spatial velocity to joint angle rates. These are scaled by a proportional gain, to become the desired joint-space velocity that will correct any Cartesian error.

8.3.1 Jacobian Singularity

For the case of a square Jacobian where $\det(\boldsymbol{J}(\boldsymbol{q})) = 0$ we cannot solve Eq. 8.3 directly. One strategy to deal with singularity is to replace the inverse with the damped inverse

$$\dot{\boldsymbol{q}} = \left(\boldsymbol{J}(\boldsymbol{q}) + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{\nu}$$

where λ is a small constant added to the diagonal which places a *floor* under the determinant. This will introduces some error in $\dot{\boldsymbol{q}}$, which integrated over time could lead to a significant discrepancy in tool position but the closed-loop resolved-rate motion scheme of Eq. 8.6 would minimize this.

An alternative is to use the pseudo-inverse of the Jacobian $\boldsymbol{J}^{+}$ which has the property

$$\boldsymbol{J}^{+}\boldsymbol{J} = \boldsymbol{I}$$

just as the inverse does. It is defined as

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

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

This is the left generalized- or pseudoinverse, see Sect. F.1.1 for more details.

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

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

A matrix expression like $\boldsymbol{\nu} = \boldsymbol{J}\dot{\boldsymbol{q}}$ is a system of scalar equations which we can solve for $\dot{\boldsymbol{q}}$. At singularity some of the equations are the same, leading to more unknowns than equations, and therefore an infinite number of solutions. The pseudo-inverse computes a solution that satisfies the equation and has the minumum norm.

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 under-actuated system which we treat as per the next section.

8.4 Under- and Over-Actuated Manipulators

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

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

in the diagrammatic form shown in Fig. 8.8. The Jacobian is a $6 \times N$ matrix, the joint velocity is an N-vector, and $\boldsymbol{\nu}$ is a 6-vector.

The case of $N < 6$ is referred to as an 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 $\boldsymbol{\nu}$ and $\boldsymbol{J}$ – 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 under-constrained and the best we can do is find a least-squares solution as described in the previous section. Alternatively we can *square up* the Jacobian to make it invertible by deleting some columns – effectively *locking* the corresponding joints.

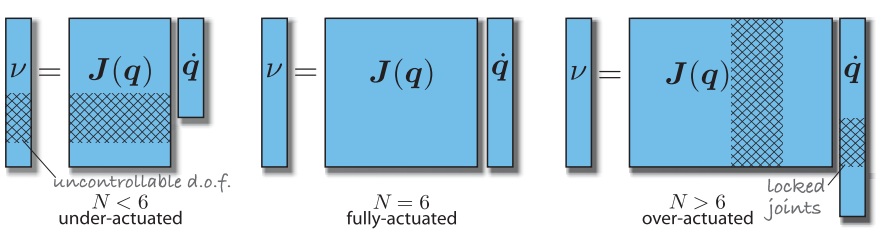


Fig. 8.8. Schematic of Jacobian, $\boldsymbol{\nu}$ and $\dot{\boldsymbol{q}}$ for different cases of N. The *hatched* areas represent matrix regions that could be deleted in order to create a square subsystem capable of solution

8.4.1 Jacobian for Under-Actuated Robot

An under-actuated robot has $N < 6$, and a Jacobian that is taller than it is wide. For example a 2-joint manipulator at a nominal pose

```
>> mdl_planar2
>> qn = [1 1];
```

has the Jacobian

```
>> J = p2.jacob0(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. 8.3 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;
>> xd'
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

$$\begin{pmatrix} v_x \\ v_y \\ \hline v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} = \begin{pmatrix} J_{xy} \\ \hline J_0 \end{pmatrix} \begin{pmatrix} \dot{q}_1 \\ \dot{q}_2 \end{pmatrix}$$

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.

8.4.2 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. 8.3 to use the left pseudo-inverse

$$\dot{\boldsymbol{q}} = \boldsymbol{J}(\boldsymbol{q})^{+}\boldsymbol{\nu} \tag{8.7}$$

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

We will demonstrate this for the left arm of the Baxter robot from Sect. 7.2.2.4 at a nominal pose

```
>> mdl_baxter
>> TE = SE3(0.8, 0.2, -0.2) * SE3.Ry(pi);
>> q = left.ikine(TE)
```

and its Jacobian

```
>> J = jacob0(left, q);
>> about J
J [double] : 6x7 (336 bytes)
```

is a 6×7 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. 8.7 we compute the required joint rates

```
>> xd = [0.2 0.2 0.2 0 0 0]';
>> qd = pinv(J) * xd;
>> qd'
ans =
        0.0895   -0.0464   -0.4259    0.6980   -0.4248    1.0179    0.2998
```

We see that all joints have nonzero velocity and contribute to the desired end-effector motion.▸

If the robot end-effector follows a repetitive path using RRMC the joint angles may *drift* over time and *not* follow a repetitive path, potentially moving toward joint limits. We can use null-space control to provide additional constraints to prevent this.

This Jacobian has seven columns and a rank of six

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

and therefore a null space▸ whose basis has just one vector

See Appendix B.

```
>> N = null(J)
N =
   -0.2244
   -0.1306
    0.6018
    0.0371
   -0.7243
    0.0653
    0.2005
```

In the case of a Jacobian matrix any joint velocity that is a linear combination of its null-space vectors will result in *no* end-effector motion. For this robot there is only one vector and we can show that this *null-space joint motion* causes no end-effector motion

```
>> norm( J * N(:,1))
ans =
   2.6004e-16
```

This is remarkably useful because it allows Eq. 8.7 to be written as

$$\dot{q} = \underbrace{J(q)^{+}\nu}_{\text{end-effector motion}} + \underbrace{NN^{+}\dot{q}_{\text{null}}}_{\text{null-space motion}} \tag{8.8}$$

where the matrix $NN^{+} \in \mathbb{R}^{N\times N}$ *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 increase joint 5 in order to move the arm away from some obstacle. We set a desired joint velocity

```
>> qd_null = [0 0 0 0 1 0 0]';
```

and project it into the null space

```
>> qp = N * pinv(N) * qd_null;
>> qp'
   0.1625    0.0946   -0.4359   -0.0269    0.5246   -0.0473   -0.1452
```

A scaling has been introduced but this joint velocity, or a scaled-up version of, this will increase the joint 5 angle without changing the end-effector pose. Other joints move as well – they provide the required compensating motion in order that the end-effector pose is not disturbed as shown by

```
>> norm( J * qp)
ans =
   1.9541e-16
```

A highly redundant snake robot like that shown in Fig. 8.9 would have a null space with 14 dimensions (20-6). This can be used to control the shape of the arm which is critical when moving within confined spaces.

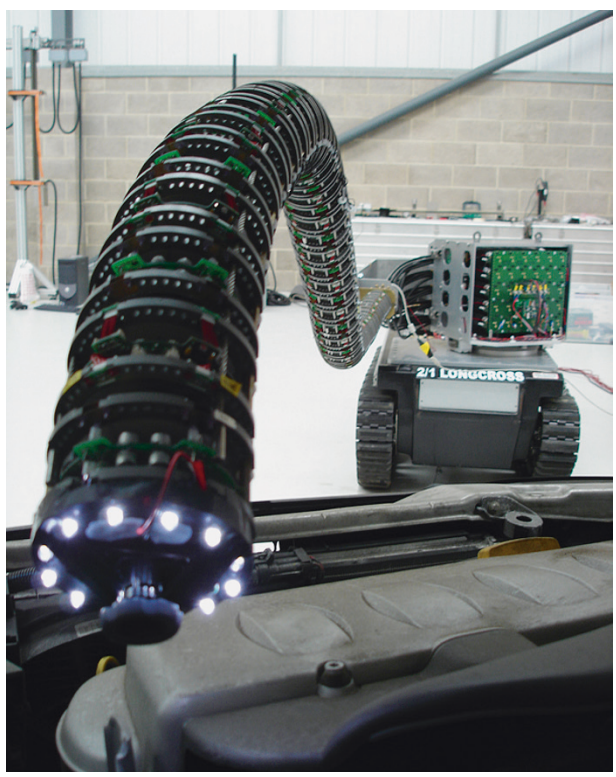


Fig. 8.9. 20-DOF snake-robot arm: 2.5 m reach, 90 mm diameter and payload capacity of 25 kg (image courtesy of OC Robotics)

8.5 Force Relationships

In Sect. 3.2.2 we introduced wrenches $\boldsymbol{W} = (f_x, f_y, f_z, m_x, m_y, m_z) \in \mathbb{R}^6$ which are a vector of forces and moments.

8.5.1 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).

$$\boldsymbol{Q} = {}^0\boldsymbol{J}(\boldsymbol{q})^T \, {}^0\boldsymbol{W} \tag{8.9}$$

where $\boldsymbol{W}$ 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.

The mapping for velocity, from end-effector to joints, involves the inverse Jacobian which can potentially be singular. The mapping of forces and torques, from end-effector to joints, is different – it involves the transpose of the Jacobian which can never be singular. We exploit this property in the next section to solve the inverse-kinematic problem numerically.

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

$$\boldsymbol{Q} = {}^E\boldsymbol{J}(\boldsymbol{q})^T \, {}^E\boldsymbol{W} \tag{8.10}$$

For the Puma 560 robot in its nominal pose, see Fig. 8.2, 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 N m 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.

8.5.2 Force Ellipsoids

In Sect. 8.2.2 we introduced the velocity ellipse and ellipsoid which describe the directions in which the end-effector is best able to move. We can perform a similar analysis for the forces and torques at the end-effector – the end-effector wrench. We start with a set of generalized joint forces with a unit norm

$$\boldsymbol{Q}^T\boldsymbol{Q} = 1$$

and substituting Eq. 8.9 we can write

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

which is the equation of points on the surface of a 6-dimensional ellipsoid in the end-effector wrench space. For the planar robot arm of Fig. 8.1 we can plot this ellipse

```
>> p2.fellipse([30 40], 'deg')
```

or we can interactively explore how its shape changes with configuration by

```
>> p2.teach(qn, 'callback', @(r,q) r.fellipse(q), 'view', 'top')
```

If this ellipsoid is close to spherical, that is, its radii are of the same order of magnitude then the end-effector can achieve an arbitrary wrench. However if one or more radii are very small this indicates that the end-effector cannot exert a force along, or a moment about, the axes corresponding to those small radii.

The force and velocity ellipsoids provide complementary information about how well suited the configuration of the arm is to a particular task. We know from personal experience that to throw an object quickly we have our arm outstretched and orthogonal to the throwing direction, whereas to lift something heavy we hold our arms close in to our body.

8.6 Inverse Kinematics: a General Numerical Approach

In Sect. 7.2.2.1 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 since these functions have no singularities.

8.6.1 Numerical Inverse Kinematics

The principle is shown in Fig. 8.10 where the robot in its current configuration is drawn solidly and the desired configuration is faint. From the overlaid pose graph the error between actual ξ_E and desired pose ξ_E^* is ξ_Δ which can be described by a spatial displacement as discussed in Sect. 3.1.4

$$^{E}\boldsymbol{\Delta} = \Delta\left(\xi_E, \xi_E^*\right) = \left(\boldsymbol{t}, \hat{\boldsymbol{v}}\theta\right) \in \mathbb{R}^6$$

where the current pose is computed using forward kinematics $\xi_E = \mathcal{K}(\boldsymbol{q})$.

Imagine 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 spatial displacement

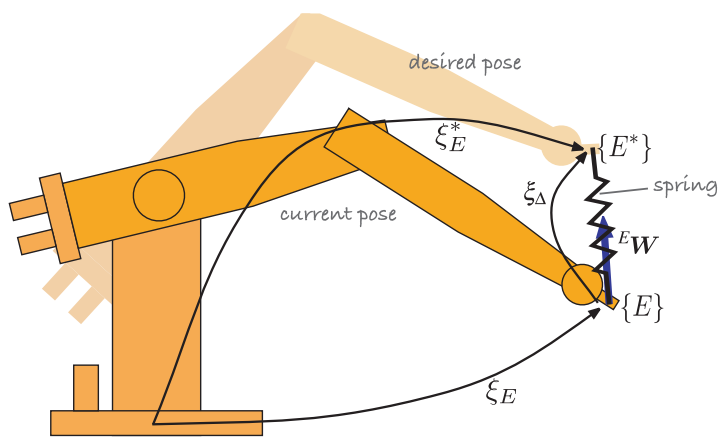


Fig. 8.10. Schematic of the numerical inverse kinematic approach, showing the current ξ_E and the desired ξ_E^* manipulator pose

$$ {}^{E}\boldsymbol{W} = \gamma\, {}^{E}\boldsymbol{\Delta} \tag{8.11} $$

which is *resolved* to generalized joint forces

$$ \boldsymbol{Q} = {}^{E}\boldsymbol{J}(\boldsymbol{q})^{T}\, {}^{E}\boldsymbol{W} $$

using the Jacobian transpose Eq. 8.10. We assume that this virtual robot has no joint motors only viscous dampers so the joint velocity will be proportional to the applied forces

$$ \dot{\boldsymbol{q}} = \boldsymbol{Q}/B $$

where B is the joint damping coefficient (assuming all dampers are the same). Putting all this together we can write

$$ \dot{\boldsymbol{q}} = \frac{1}{B}\boldsymbol{J}(\boldsymbol{q})^{T}\, \Delta\Big(\mathcal{K}(\boldsymbol{q}), \xi_E^*\Big) $$

which gives the joint velocities that will drive the forward kinematic solution toward the desired end-effector pose. This can be solved iteratively by

$$ \begin{aligned} \delta_q\langle k\rangle &= \alpha\boldsymbol{J}\big(\boldsymbol{q}\langle k\rangle\big)^{T}\, \Delta\Big(\mathcal{K}\big(\boldsymbol{q}\langle k\rangle\big), \xi_E^*\Big) \\ \boldsymbol{q}\langle k+1\rangle &\leftarrow \boldsymbol{q}\langle k\rangle + \delta_q\langle k\rangle \end{aligned} \tag{8.12} $$

until the norm of the update $\|\delta_q\langle k\rangle\|$ is sufficiently small and where $\alpha > 0$ is a well-chosen constant. Since the solution is based on the Jacobian transpose rather than inverse the algorithm works when the Jacobian is nonsquare or singular. In practice however this algorithm is slow to converge and very sensitive to the choice of α.

More practically we can formulate this as a least-squares problem in the world coordinate frame and minimize the scalar cost

$$ E = \boldsymbol{\Delta}^{T}\boldsymbol{M}\boldsymbol{\Delta} $$

where $\boldsymbol{M} = \operatorname{diag}(\boldsymbol{m}) \in \mathbb{R}^{6\times 6}$ and $\boldsymbol{m}$ is the mask vector introduced in Sect. 7.2.2.3. The update becomes

$$ \delta_q\langle k\rangle = \Big(\boldsymbol{J}\big(\boldsymbol{q}\langle k\rangle\big)^{T}\boldsymbol{M}\boldsymbol{J}\big(\boldsymbol{q}\langle k\rangle\big)\Big)^{-1}\boldsymbol{J}\big(\boldsymbol{q}\langle k\rangle\big)^{T}\boldsymbol{M}\,\Delta\Big(\mathcal{K}\big(\boldsymbol{q}\langle k\rangle\big), \xi_E^*\Big) $$

which is much faster to converge but can behave poorly near singularities. We remedy this by introducing a damping constant λ

$$ \boxed{\delta_q\langle k\rangle = \Big(\boldsymbol{J}\big(\boldsymbol{q}\langle k\rangle\big)^{T}\boldsymbol{M}\boldsymbol{J}\big(\boldsymbol{q}\langle k\rangle\big) + \lambda\boldsymbol{I}_{N\times N}\Big)^{-1}\boldsymbol{J}\big(\boldsymbol{q}\langle k\rangle\big)^{T}\boldsymbol{M}\,\Delta\Big(\mathcal{K}\big(\boldsymbol{q}\langle k\rangle\big), \xi_E^*\Big)} $$

which ensures that the term being inverted can never be singular.

An effective way to choose λ is to test whether or not an iteration reduces the error, that is if $\|\delta_q\langle k\rangle\| < \|\delta_q\langle k-1\rangle\|$. If the error is reduced we can decrease λ in order to speed convergence. If the error has increased we revert to our previous estimate of $\boldsymbol{q}\langle k\rangle$ and increase λ. This adaptive damping factor scheme is the basis of the well-known Levenberg-Marquardt optimization algorithm.

This algorithm is implemented by the `ikine` method and works well in practice. As with all optimization algorithms it requires a reasonable initial estimate of $\boldsymbol{q}$ and this can be explicitly given using the option `'q0'`. A brute-force search for an initial value can be requested by the option `'search'`. The simple Jacobian-transpose approach of Eq. 8.12 can be invoked using the option `'transpose'` along with the value of α.

8.7 Advanced Topics

8.7.1 Computing the Manipulator Jacobian Using Twists

In Sect. 7.1.2.2 we computed the forward kinematics as a product of exponentials based on the screws representing the joint axes in a zero-joint angle configuration. It is easy to differentiate the product of exponentials with respect to motion about each screw axis which leads to the Jacobian matrix

$$ {}^{0}\boldsymbol{J}^{\mathcal{V}} = \left(S_1 \;\; \mathrm{Ad}\left(e^{[S_1]q_1}\right) S_2 \;\; \cdots \;\; \mathrm{Ad}\left(e^{[S_1]q_1} \cdots e^{[S_{N-1}]q_{N-1}}\right) S_N \right) $$

for velocity in the world coordinate frame. The Jacobian is very elegantly expressed and can be easily built up column by column. Velocity in the end-effector coordinate frame is related to joint velocity by the Jacobian matrix

$$ {}^{E}\boldsymbol{J}^{\mathcal{V}} = \mathrm{Ad}\left({}^{E}\xi_0\right) {}^{0}\boldsymbol{J}^{\mathcal{V}} $$

where Ad $(\cdot)$ is the adjoint matrix introduced in Sect. 3.1.2.

> However, compared to the Jacobian of Sect. 8.1, these Jacobians give the velocity of the end-effector as a *velocity twist*, not a spatial velocity as defined on page 65.

To obtain the Jacobian that gives spatial velocity as described in Sect. 8.1 we must apply a velocity transformation

$$ {}^{0}\boldsymbol{J} = \begin{pmatrix} \boldsymbol{I}_{3\times3} & -\left[{}^{0}\boldsymbol{t}_E\right]_\times \\ \boldsymbol{0}_{3\times3} & \boldsymbol{I}_{3\times3} \end{pmatrix} {}^{0}\boldsymbol{J}^{\mathcal{V}} $$

8.8 Wrapping Up

Jacobians are an important concept in robotics, relating changes in one space to changes in another. We previously encountered Jacobians for estimation in Chap. 6 and will use them later for computer vision and control.

In this chapter we have learned about the manipulator Jacobian which describes the relationship between the rate of change of joint coordinates and the spatial velocity of the end-effector expressed in either the world frame or the end-effector frame. 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. The numerical properties of the Jacobian tell us about manipulability, that is how well the manipulator is able to move, or exert force, in different directions. At a singularity, indicated by linear dependence between columns of the Jacobian, the robot is unable to move in certain directions. We visualized this by means of the velocity and force ellipsoids.

We also created Jacobians to map angular velocity to roll-pitch-yaw or Euler angle rates, and these were used to form the analytic Jacobian matrix. The Jacobian transpose is used to map wrenches applied at the end-effector to joint torques, and also to map wrenches between coordinate frames. It is also the basis of numerical inverse kinematics for arbitrary robots and singular poses.

Further Reading

The manipulator Jacobian is covered by almost all standard robotics texts such as the robotics handbook (Siciliano and Khatib 2016), Lynch and Park (2017), Siciliano et al. (2008), Spong et al. (2006), Craig (2005), and Paul (1981). An excellent discussion of manipulability and velocity ellipsoids is provided by Siciliano et al. (2009), and the most common manipulability measure is that proposed by Yoshikawa (1984). Computing the manipulator Jacobian based on Denavit-Hartenberg parameters, as used in this Toolbox, 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-squares methods are reviewed by Deo and Walker (1995).

MATLAB and Toolbox Notes

The MATLAB Robotics System Toolbox™ describes a serial-link manipulator using an instance of the `RigidBodyTree` class. Jacobians can be computed using the class method `GeometricJacobian`.

Exercises

1. For the simple 2-link example (page 230) compute the determinant symbolically and determine when it is equal to zero. What does this mean physically?
2. For the Puma 560 robot can you devise a configuration in which three joint axes are parallel?
3. Derive the analytical Jacobian for Euler angles.
4. Velocity and force ellipsoids for the two link manipulator (page 236, 245). Perhaps using the interactive `teach` method with the `'callback'` option:
   a) What configuration gives the best manipulability?
   b) What configuration is best for throwing a ball in the positive x-direction?
   c) What configuration is best for carrying a heavy weight if gravity applies a force in the negative y-direction?
   d) 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 centered on the end-effector position.
5. Velocity and force ellipsoids for the Puma manipulator (page 237)
   a) For the Puma 560 manipulator find a configuration where manipulability is greater than at `qn`.
   b) Use the `teach` method with the `'callback'` option to interactively animate the ellipsoids. You may need to use the `'workspace'` option to `teach` to prevent the ellipsoid being truncated.
6. Resolved-rate motion control (page 237)
   a) Experiment with different Cartesian translational and rotational velocity demands, and combinations.
   b) Extend the Simulink system of Fig. 8.6 to also record the determinant of the Jacobian matrix to the workspace.
   c) In Fig. 8.6 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. 7.3.4. What happens when the robot reaches the singularity?
   e) Replace the Jacobian inverse block in Fig. 8.5 with the MATLAB function `pinv`.
   f) Replace the Jacobian inverse block in Fig. 8.5 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. 8.5 with a block based on the MATLAB function `lscov`.

7. The model `mdl_p8` describes an 8-joint robot (PPRRRRRR) comprising an *xy*-base (PP) carrying a Puma arm (RRRRRR).
   a) Compute a Cartesian end-effector path and use numerical inverse kinematics to solve for the joint coordinates. Analyze how the motion is split between the base and the robot arm.
   b) With the end-effector at a constant pose explore null-space control. Set a velocity for the mobile base and see how the arm configuration accomodates that.
   c) 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.
   d) Consider now that the Puma robot is mounted on a nonholonomic robot, create a controller that generates appropriate steering and velocity inputs to the mobile robot (challenging).
   e) 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.
8. The model `mdl_hyper3d(20)` is a 20-joint robot that moves in 3-dimensional space.
   a) Explore the capabilities of this robot.
   b) Compute a Cartesian end-effector trajectory that traces a circle on the ground, and use numerical inverse kinematics to solve for the joint coordinates.
   c) Add a null-space control strategy that keeps all joint angles close to zero while it is moving.
   d) Define an end-effector pose on the ground that the robot must reach after passing through two holes in vertical planes. Can you determine the joint configuration that allows this?
9. Write code to compute the Jacobian of a robot represented by a `SerialLink` object using twists as described in Sect. 8.7.1.
10. Consider the Puma 560 robot moving in the *xz*-plane. Divide the plane into 2-cm grid cells and for each cell determine if it is reachable, and if it is then determine the manipulability for the first three joints of the robot arm and place that value in the corresponding grid cell. Display a heat map of the robot's manipulability in the plane.