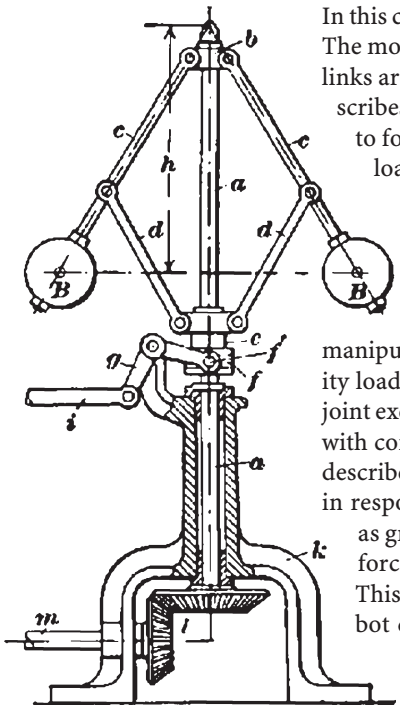
9 Dynamics and Control



In this chapter we consider the dynamics and control of a serial-link manipulator arm. The motion of the end-effector is the composition of the motion of each link, and the links are ultimately moved by *forces* and *torques* exerted by the joints. Section 9.1 describes the key elements of a robot joint control system that enables a single joint to follow a desired trajectory; and the challenges involved such as friction, gravity load and varying inertia.

Each link in the serial-link manipulator is supported by a reaction force and torque from the preceding link, and is subject to its own weight as well as the reaction forces and torques from the links that it supports. Section 9.2 introduces the *rigid-body* equations of motion, a set of coupled dynamic equations, that describe the joint torques necessary to achieve a particular manipulator state. These equations can be factored into terms describing inertia, gravity load and gyroscopic coupling which provide insight into how the motion of one joint exerts a disturbance force on other joints, and how inertia and gravity load varies with configuration and payload. Section 9.3 introduces the forward dynamics which describe how the manipulator moves, that is, how its configuration evolves with time in response to forces and torques applied by the joints and by external forces such as gravity. Section 9.4 introduces control systems that compute the required joint forces based on the desired trajectory as well as the rigid-body dynamic forces. This enables improved control of the end-effector trajectory, despite changing robot configuration, as well as compliant motion. Section 9.5 covers an important application of what we have learned about joint control – series-elastic actuators for human-safe robots.

9.1 Independent Joint Control

A robot drive train comprises an actuator or motor, and a transmission to connect it to the link. A common approach to robot joint control is to consider each joint or axis as an independent control system that attempts to accurately follow its joint angle trajectory. However as we shall see, this is complicated by various *disturbance* torques due to gravity, velocity and acceleration coupling, and friction that act on the joint. A very common control structure is the nested control loop. The outer loop is responsible for maintaining position and determines the velocity of the joint that will minimize position error. The inner loop is responsible for maintaining the velocity of the joint as demanded by the outer loop.

9.1.1 Actuators

The vast majority of robots today are driven by rotary electric motors (Fig. 9.1). Large industrial robots typically use brushless servo motors while small laboratory or hobby robots use brushed DC motors or stepper motors. Manipulators for very large payloads as used in mining, forestry or construction are typically hydraulically driven using electrically operated hydraulic valves – electro-hydraulic actuation.

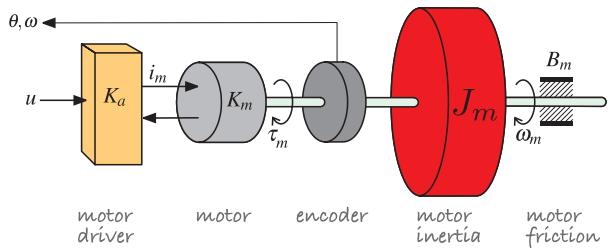


Fig. 9.1.
Key components of a robot-joint actuator. A demand voltage u controls the current i_m flowing into the motor which generates a torque τ_m that accelerates the rotational inertia J_m and is opposed by friction $B_m\,\omega_m$. The encoder measures rotational speed and angle

Electric motors can be either current or voltage controlled.▸ Here we assume current control where a motor driver or amplifier provides current

$$i_m = K_a u$$

that is linearly related to the applied control voltage u and where K_a is the transconductance of the amplifier with units of $\mathrm{A\,V^{-1}}$. The torque generated by the motor is proportional to current

$$\tau_m = K_m i_m$$

where K_m is the motor torque constant with units of $\mathrm{N\,m\,A^{-1}}$. The torque accelerates the rotational inertia J_m, due to the rotating part of the motor itself, which has a rotational velocity of ω. Frictional effects are modeled by B_m.

Current control is implemented by an electronic constant current source, or a variable voltage source with feedback of actual motor current. A variable voltage source is most commonly implemented by a pulse-width modulated (PWM) switching circuit. Voltage control requires that the electrical dynamics of the motor due to its resistance and inductance, as well as back EMF, must be taken into account when designing the control system.

9.1.2 Friction

Any rotating machinery, motor or gearbox, will be affected by friction – a force or torque that *opposes* motion. The net torque from the motor is

$$\tau' = \tau_m - \tau_f$$

where τ_f is the friction torque which is function of velocity

$$\tau_f = B\omega + \tau_C \tag{9.1}$$

where the slope $B > 0$ is the viscous friction coefficient and the offset is Coulomb friction. The latter is frequently modeled by the nonlinear function

$$\tau_C = \begin{cases} \tau_C^+ & \omega > 0 \\ 0 & \omega = 0 \\ \tau_C^- & \omega < 0 \end{cases} \tag{9.2}$$

In general the friction coefficients depend on the direction of rotation and this asymmetry is more pronounced for Coulomb than for viscous friction.

The total friction torque as a function of rotational velocity is shown in Fig. 9.2. At very low speeds, highlighted in grey, an effect known as stiction becomes evident. The applied torque must exceed the stiction torque before rotation can occur – a process known as *breaking stiction*. Once the machine is moving the stiction force rapidly decreases and viscous friction dominates.

There are several sources of friction *experienced* by the motor. The first component is due to the motor itself: its bearings and, for a brushed motor, the brushes rubbing on the commutator. The friction parameters are often provided in the motor manufacturer's data sheet. Other sources of friction are the gearbox and the bearings that support the link.

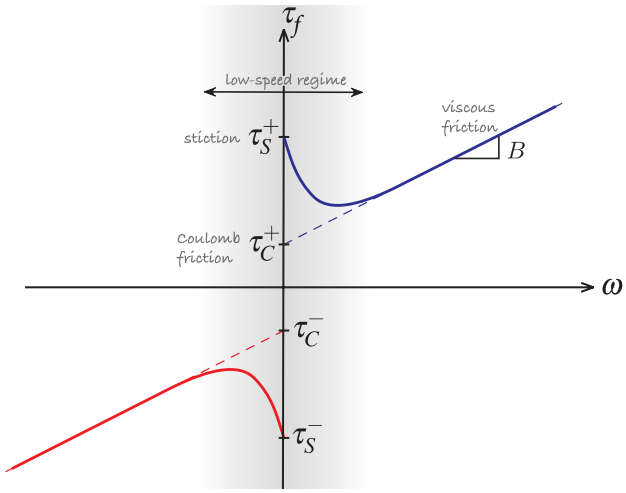


Fig. 9.2. Typical friction versus speed characteristic. The *dashed lines* depict a simple piecewise-linear friction model characterized by slope (viscous friction) and intercept (Coulomb friction). The low-speed regime is *shaded* and shown in exaggerated fashion

Charles-Augustin de Coulomb (1736–1806) was a French physicist. He was born in Angoulême to a wealthy family and studied mathematics at the Collége des Quatre-Nations under Pierre Charles Monnier, and later at the military school in Méziéres. He spent eight years in Martinique involved in the construction of Fort Bourbon and there he contracted tropical fever.

Later he worked at the shipyards in Rochefort which he used as laboratories for his experiments in static and dynamic friction of sliding surfaces. His paper *Théorie des machines simples* won the Grand Prix from the Académie des Sciences in 1781. His later research was on electromagnetism and electrostatics and he is best known for the formula on electrostatic forces, named in his honor, as is the SI unit of charge. After the revolution he was involved in determining the new system of weights and measures.

9.1.3 Effect of the Link Mass

A motor in a robot arm does not exist in isolation, it is connected to a link as shown schematically in Fig. 9.3. The link has two obvious significant effects on the motor – it adds extra inertia and it adds a torque due to the weight of the arm and both vary with the configuration of the joint.

With reference to the simple 2-joint robot shown in Fig. 9.4 consider the first joint which is directly attached to the first link which is colored red. If we assume the mass of the red link is concentrated at its center of mass (CoM) the extra inertia of the link will be $m_1 r_1^2$. The motor will also experience the inertia of the blue link and this will depend on the value of q_2 – the inertia of the arm when it is straight is greater than the inertia when it is folded.

We also see that gravity acting on the center of mass of the red link will create a torque on the joint 1 motor which will be proportional to $\cos q_1$. Gravity acting on the center of mass of the blue link also creates a torque on the joint 1 motor, and this is more pronounced since it is acting at a greater distance from the motor – the *lever arm effect* is greater.

These effects are clear from even a cursory examination of Fig. 9.4 but the reality is even more complex. Jumping ahead to material we will cover in the next section, we can use the Toolbox◂ to determine the torque acting on each of the joints as a function of the position, velocity and acceleration of the joints

◂ This requires the MATLAB Symbolic Math Toolbox™.

```
>> mdl_twolink_sym
>> syms q1 q2 q1d q2d q1dd q2dd real
>> tau = twolink.rne([q1 q2], [q1d q2d], [q1dd q2dd]);
```

and the result is a symbolic 2-vector, one per joint, with surprisingly many terms which we can summarize as:

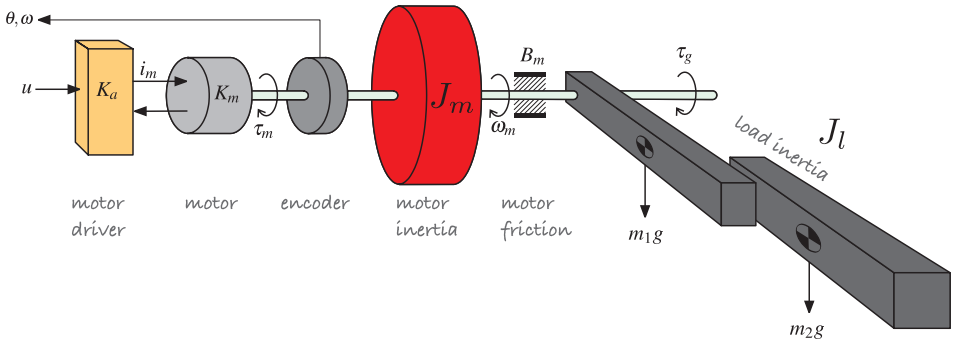


Fig. 9.3.
Robot joint actuator with attached links. The center of mass of each link is indicated by ◔

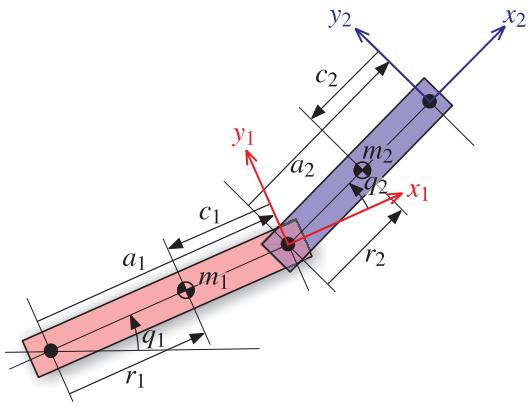


Fig. 9.4. Notation for rigid-body dynamics of two-link arm showing link frames and relevant dimensions. The center of mass (CoM) of each link is indicated by ◔. The CoM is a distance of r_i from the axis of joint i, and c_i from the origin of frame $\{i\}$ as defined in Fig. 7.5 – therefore $r_i = a_i + c_i$

$$
\begin{aligned}
\tau_1 &= \boldsymbol{M}_{11}(q_2)\ddot{q}_1 + \underbrace{\boldsymbol{M}_{12}(q_2)\ddot{q}_2 + \boldsymbol{C}_1(q_2)\dot{q}_1\dot{q}_2 + \boldsymbol{C}_2(q_2)\dot{q}_2^2 + g(q_1, q_2)}_{\text{disturbance}} \\
\boldsymbol{M}_{11} &= m_1\left(a_1^2 + 2a_1c_1 + c_1^2\right) + m_2\left(a_1^2 + (a_2 + c_2)^2 + (2a_1a_2 + 2a_1c_2)\cos q_2\right) \\
\boldsymbol{M}_{12} &= m_2(a_2 + c_2)(a_2 + c_2 + a_1\cos q_2) \\
\boldsymbol{C}_1 &= -2a_1m_2(a_2 + c_2)\sin q_2 \\
\boldsymbol{C}_2 &= -a_1m_2(a_2 + c_2)\sin q_2 \\
g &= (a_1m_1 + a_1m_2 + c_1m_1)\cos q_1 + (a_2m_2 + c_2m_2)\cos(q_1 + q_2)
\end{aligned}
\tag{9.3}
$$

We have already discussed the first and last terms in a qualitative way – the inertia is dependent on q_2 and the gravity torque g is dependent on q_1 and q_2. What is perhaps most surprising is that the torque applied to joint 1 depends on the velocity and the acceleration of q_2 and this will covered in more detail in Sect. 9.2.

In summary, the effect of joint motion in a series of mechanical links is nontrivial. The motion of any joint is affected by the motion of *all* the other joints and for a robot with many joints this becomes quite complex.

9.1.4 Gearbox

Electric motors are compact and efficient and can rotate at very high speed, but produce very low torque. Therefore it is common to use a reduction gearbox to tradeoff speed for increased torque. For a prismatic joint the gearbox might convert rotary motion to linear. The disadvantage of a gearbox is increased cost, weight, friction, backlash, mechanical noise and, for harmonic gears, torque ripple. Very high-performance robots, such as those used in high-speed electronic assembly, use expensive high-torque motors with a direct drive or a very low gear ratio achieved using cables or thin metal bands rather than gears.

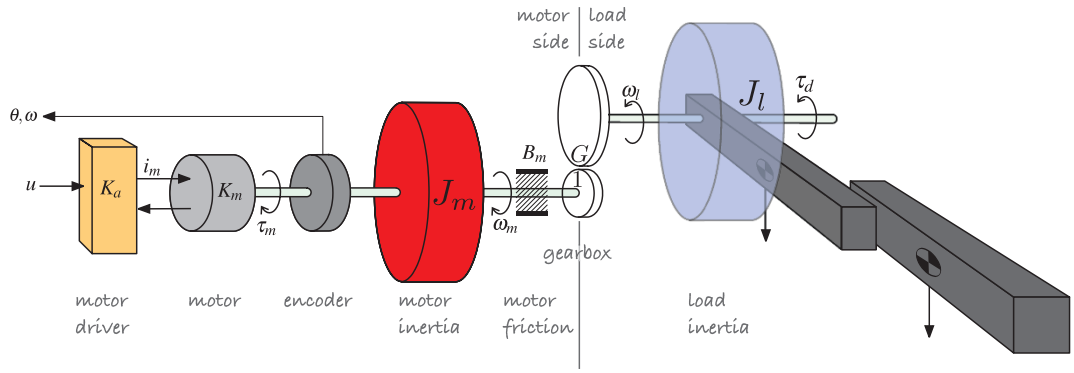


Fig. 9.5. Schematic of complete robot joint including gearbox. The effective inertia of the links is shown as J_l and the disturbance torque due to the link motion is τ_d

For example if you turned the motor shaft by hand you would *feel* the inertia of the load through the gearbox but it would be reduced by G^2.

Table 9.1. Relationship between load and motor referenced quantities for reduction gear ratio G

${}^{l}J = G^2\,{}^{m}J$
${}^{l}B = G^2\,{}^{m}B$
${}^{l}\tau_C = G\,{}^{m}\tau_C$
${}^{l}\tau = G\,{}^{m}\tau$
${}^{l}\omega = {}^{m}\omega/G$
${}^{l}\dot{\omega} = {}^{m}\dot{\omega}/G$

Figure 9.5 shows the complete drive train of a typical robot joint. For a $G:1$ reduction drive the torque at the link is G times the torque at the motor. For rotary joints the quantities measured at the link, reference frame l, are related to the motor referenced quantities, reference frame m, as shown in Table 9.1. The inertia of the load is reduced by a factor of G^2 and the disturbance torque by a factor of G.

There are two components of inertia *seen* by the motor. The first is due to the rotating part of the motor itself, its rotor. It is denoted J_m and is a constant intrinsic characteristic of the motor and the value is provided in the motor manufacturer's data sheet. The second component is the variable load inertia J_l which is the inertia of the driven link and all the other links that are attached to it. For joint j this is element M_{jj} of the configuration dependent inertia matrix of Eq. 9.3.

9.1.5 Modeling the Robot Joint

The complete motor drive comprises the motor to generate torque, the gearbox to amplify the torque and reduce the effects of the load, and an encoder to provide feedback of position and velocity. A schematic of such a device is shown in Fig. 9.6.

Collecting the various equations above we can write the torque balance on the motor shaft as

$$K_m K_a u - B'\omega - \tau_C'(\omega) - \frac{\tau_d(\boldsymbol{q})}{G} = J'\dot{\omega} \tag{9.4}$$

where B', τ_C' and J' are the effective total viscous friction, Coulomb friction and inertia due to the motor, gearbox, bearings and the load

$$B' = B_m + \frac{B_l}{G^2}, \quad \tau_C' = \tau_{C,m} + \frac{\tau_{C,l}'}{G}, \quad J' = J_m + \frac{J_l}{G^2} \tag{9.5}$$

In order to analyze the dynamics of Eq. 9.4 we must first linearize it, and this can be done simply by setting all additive constants to zero

$$J'\dot{\omega} + B'\omega = K_m K_a u$$

and then applying the Laplace transformation

$$sJ'\Omega(s) + B'\Omega(s) = K_m K_a U(s)$$

where $\Omega(s)$ and $U(s)$ are the Laplace transform of the time domain signals $\omega(t)$ and $u(t)$ respectively. This can be rearranged as a linear transfer function

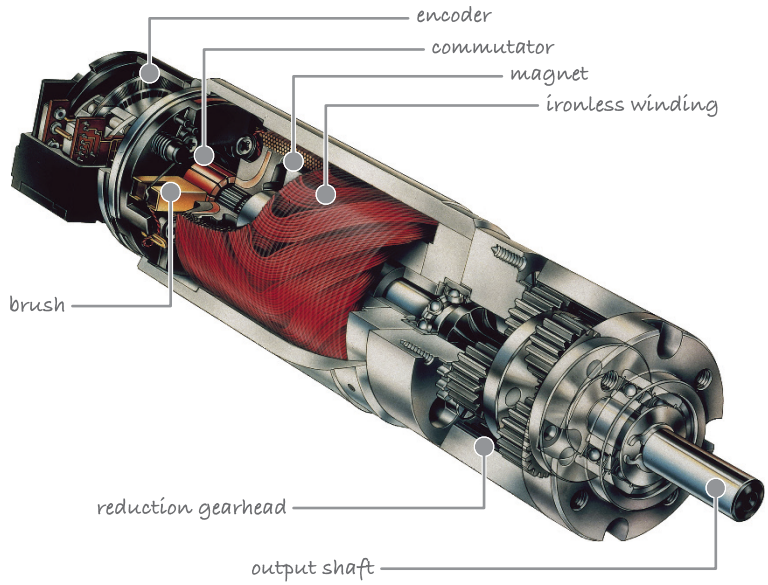


Fig. 9.6.
Schematic of an integrated motor-encoder-gearbox assembly (courtesy of maxon precision motors, inc.)

$$\frac{\Omega(s)}{U(s)} = \frac{K_m K_a}{J's + B'}$$

relating motor speed to control input, and has a single pole▸ at $s = -B'/J'$.

The mechanical pole.

We will use data for joint 2 – the shoulder – of the Puma 560 robot since its parameters are well known and are listed in Table 9.2. In the absence of other information we will take $B' = B_m$. The link inertia M_{22} experienced by the joint 2 motor as a function of configuration is shown in Fig. 9.16c and we see that it varies significantly – from 3.66 to 5.21 kg m^2. Using the mean value of the extreme inertia values, which is 4.43 kg m^2, the effective inertia is

$$\begin{aligned} J' &= J_m + \frac{1}{G^2} M_{22} \\ &= 200 \times 10^{-6} + \frac{4.43}{(107.815)^2} \\ &= 200 \times 10^{-6} + 380 \times 10^{-6} = 580 \times 10^{-6} \text{ kg m}^2 \end{aligned}$$

and we see that the inertia of the link referred to the motor side of the gearbox is comparable to the inertia of the motor itself.

The Toolbox can automatically generate▸ a dynamic model suitable for use with the MATLAB control design tools

This requires the Control Systems Toolbox™.

```
>> tf = p560.jointdynamics(qn);
```

is a vector of continuous-time linear-time-invariant (LTI) models, one per joint, computed for the particular pose `qn`. For the shoulder joint we are considering here that transfer function is

```
>> tf(2)
ans =
            1
  ----------------------
  0.0005797 s + 0.000817
Continuous-time transfer function.
```

which is similar to that above except that it does not account for K_m and K_a since these are not parameters of the `Link` object. Once we have a model of this form we can plot the step response and use a range of standard control system design tools.

Table 9.2. Motor and drive parameters for Puma 560 shoulder joint with respect to the motor side of the gearbox (Corke 1996b)

Parameter	Symbol	Value	Unit
Motor torque constant	K_m	0.228	N m A^{-1}
Motor inertia	J_m	200 $\times 10^{-6}$	kg m^2
Drive viscous friction	B_m	817 $\times 10^{-6}$	N m s rad^{-1}
Drive Coulomb friction	τ_C^+ τ_C^-	0.126 –0.709	N m N m
Gear ratio	G	107.815	
Maximum torque	τ_{max}	0.900	N m
Maximum speed	$\dot{q}_{max}$	165	rad s^{-1}

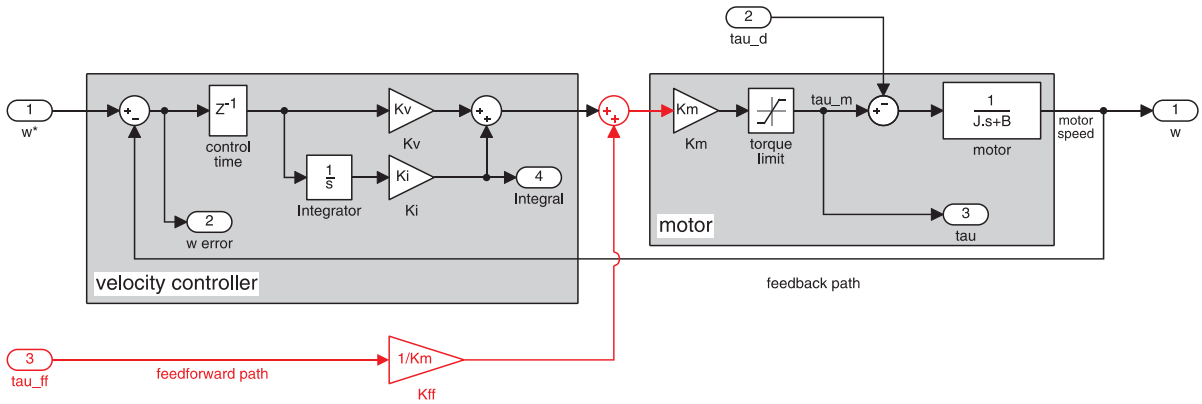


Fig. 9.7. Velocity control loop, Simulink model `vloop`

9.1.6 Velocity Control Loop

A very common approach to controlling the position output of a motor is the nested control loop. The outer loop is responsible for maintaining position and determines the velocity of the joint that will minimize position error. The inner loop – the velocity loop – is responsible for maintaining the velocity of the joint as demanded by the outer loop. Motor speed control is important for all types of robots, not just arms. For example it is used to control the speed of the wheels for car-like vehicles and the rotors of a quadrotor as discussed in Chap. 4.

The Simulink® model is shown in Fig. 9.7. The input to the motor driver is based on the error between the demanded and actual velocity.◂ A delay of 1 ms is included to model the computational time of the velocity loop control algorithm and a saturator models the finite maximum torque that the motor that can deliver.

The motor velocity is typically computed by taking the difference in motor position at each sample time, and the position is measured by a shaft encoder. This can be problematic at very low speeds where the encoder tick rate is lower than the sample rate. In this case a better strategy is to measure the time between encoder ticks.

We first consider the case of proportional control where $K_i = 0$ and

$$u^* = K_v\left(\dot{q}^* - \dot{q}\right) \tag{9.6}$$

To test this velocity controller we create a test harness

```
>> vloop_test
```

with a trapezoidal velocity demand which is shown in Fig. 9.8. Running the simulator

```
>> sim('vloop_test');
```

and with a little experimentation we find that a gain of $K_v = 0.6$ gives satisfactory performance as shown in Fig. 9.9. There is some minor overshoot at the discontinuity but less gain leads to increased velocity error and more gain leads to oscillation – as always control engineering is all about tradeoffs.

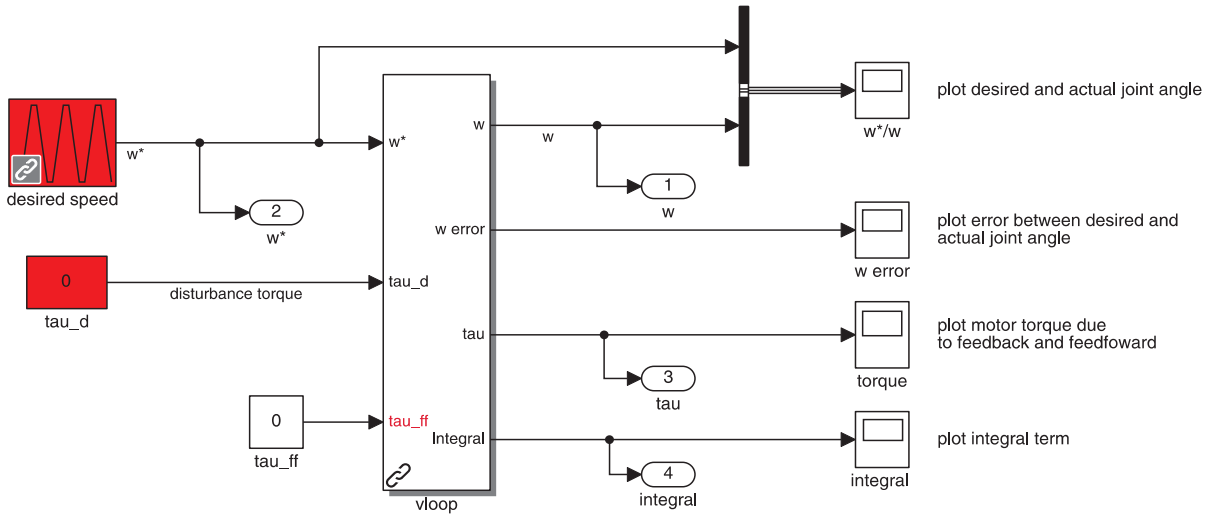


Fig. 9.8. Test harness for the velocity control loop, Simulink model `vloop_test`. The input `tau_d` is used to simulate a disturbance torque acting on the joint

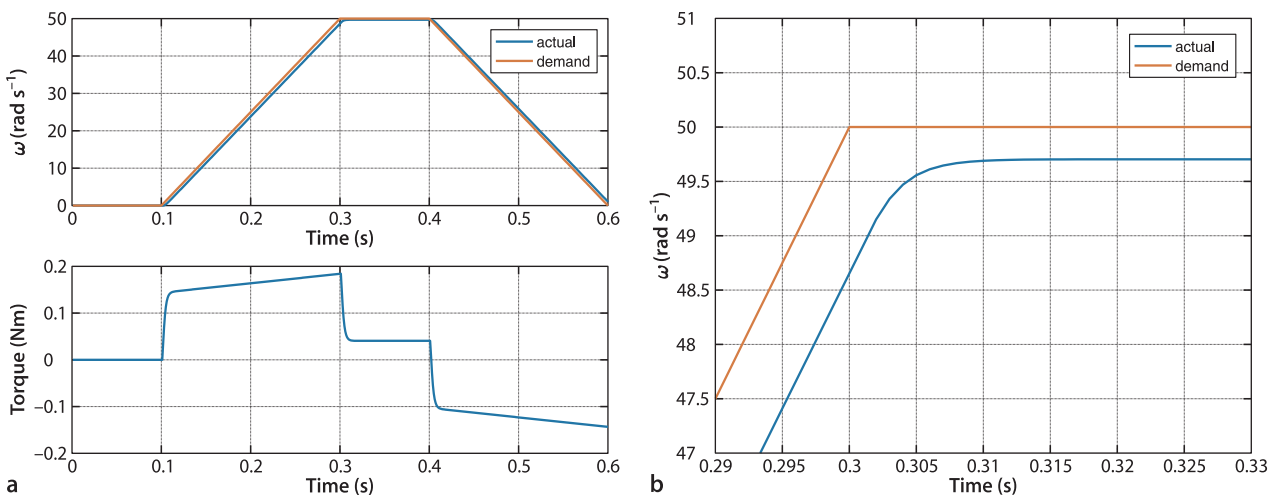


Fig. 9.9. Velocity loop with a trapezoidal demand. **a** Response; **b** closeup of response

We also observe a very slight steady-state error – the actual velocity is less than the demand at all times. From a classical control system perspective the velocity loop contains no integrator block and is classified as a Type 0 system – a characteristic of Type 0 systems is they exhibit a finite error for a constant input. More intuitively we can argue that in order to move at constant speed the motor must generate a finite torque to overcome friction, and since motor torque is proportional to velocity error there must be a finite velocity error.

Now we will investigate the effect of inertia variation on the closed-loop response. Using Eq. 9.5 and the data from Fig. 9.16c we find that the minimum and maximum joint inertia at the motor are 515×10^{-6} and 648×10^{-6} kg m^2 respectively. Figure 9.10 shows the velocity tracking error using the control gains chosen above for various values of link inertia. We can see that the tracking error decays more slowly for larger inertia, and is showing signs of instability for the case of zero link inertia. For a case where the inertia variation is more extreme the gain should be chosen to achieve satisfactory closed-loop performance at both extremes.

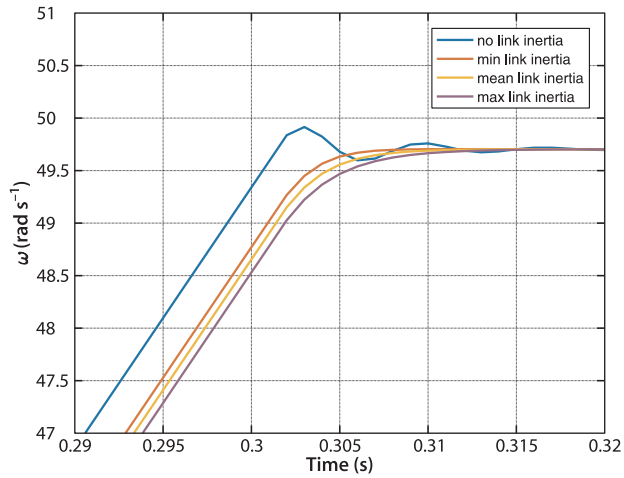


Fig. 9.10. Velocity loop response with a trapezoidal demand for varying inertia M_{22}

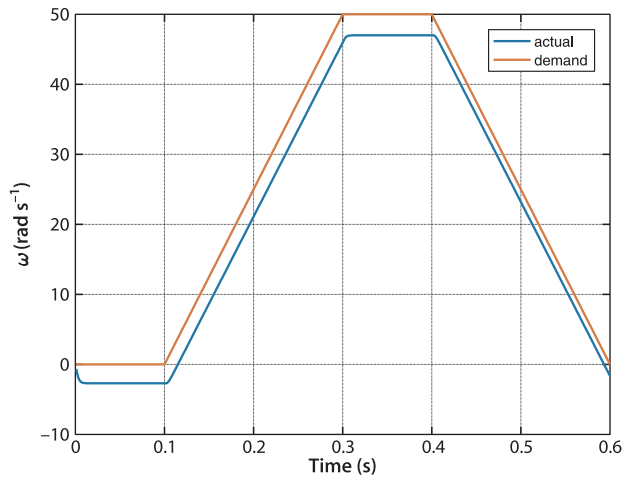


Fig. 9.11. Velocity loop response to a trapezoidal demand with a gravity disturbance of 20 N m

Motor limits. Electric motors are limited in both torque and speed. The maximum torque is defined by the maximum current the drive electronics can provide. A motor also has a maximum rated current beyond which the motor can be damaged by overheating or demagnetization of its permanent magnets which irreversibly reduces its torque constant. As speed increases so does friction and the maximum speed is $\omega_{\max} = \tau_{\max} / B$.

The product of motor torque and speed is the mechanical output power and also has an upper bound. Motors can tolerate some overloading, peak power and peak torque, for short periods of time but the sustained rating is significantly lower than the peak.

Figure 9.15a shows that the gravity torque on this joint varies from approximately −40 to 40 N m. We now add a disturbance torque equal to just half that maximum amount, 20 N m applied on the load side of the gearbox. We do this by setting a nonzero value in the `tau_d` block and rerunning the simulation. The results shown in Fig. 9.11 indicate that the control performance has been badly degraded – the tracking error has increased to more than 2 rad s^{-1}. This has the same root cause as the very small error we saw in Fig. 9.9 – a Type 0 system exhibits a finite error for a constant input or a constant disturbance.

There are three common approaches to counter this error. The first, and simplest, is to increase the gain. This will reduce the tracking error but push the system toward instability and increase the overshoot.

The second approach, commonly used in industrial motor drives, is to add integral action – adding an integrator changes the system to Type 1 which has zero error

for a constant input or constant disturbance. We change Eq. 9.6 to a proportional-integral controller

$$u^* = \left(K_v + \frac{K_i}{s}\right)\left(\dot{q}^* - \dot{q}\right), \quad K_i > 0$$

In the Simulink model of Fig. 9.7 this is achieved by setting `Ki` to a nonzero value. With some experimentation we find the gains $K_v = 1$ and $K_i = 10$ work well and the performance is shown in Fig. 9.12. The integrator state evolves over time to cancel out the disturbance term and we can see the error decaying to zero. In practice the disturbance varies over time and the integrator's ability to track it depends on the value of the integral gain K_i. In reality other disturbances affect the joint, for instance Coulomb friction and torques due to velocity and acceleration coupling. The controller needs to be well tuned so that these have minimal effect on the tracking performance.

As always in engineering there are some tradeoffs. The integral term can lead to increased overshoot so increasing K_i usually requires some compensating reduction of K_v. If the joint actuator is pushed to its performance limit, for instance the torque limit is reached, then the tracking error will grow with time since the motor acceleration will be lower than required. The integral of this increasing error will grow leading to a condition known as integral windup. When the joint finally reaches its destination the large accumulated integral keeps driving the motor forward until the integral decays – leading to large overshoot. Various strategies are employed to combat this, such as limiting the maximum value of the integrator, or only allowing integral action when the motor is close to its setpoint.

These two approaches are collectively referred to as disturbance rejection and are concerned with reducing the effect of an unknown disturbance. However if we think about the problem in its robotics context the gravity disturbance is not unknown. In Sect. 9.1.3 we showed how to compute the torque due to gravity that acts on each joint. If we know this torque, and the motor torque constant, we can *add* it to the output of the PI controller.

Even if the gravity load is known imprecisely this trick will reduce the magnitude of the disturbance.

The third approach is therefore to predict the disturbance and cancel it out – a strategy known as torque feedforward control. This is shown by the red wiring in Fig. 9.7 and can be demonstrated by setting the `tau_ff` block of Fig. 9.8 to the same, or approximately the same, value as the disturbance.

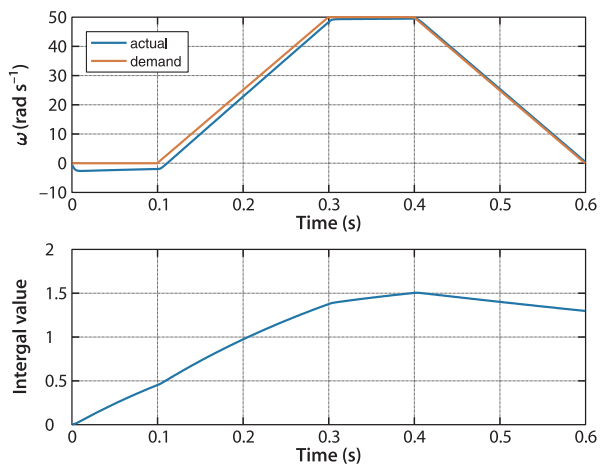


Fig. 9.12.
Velocity loop response to a trapezoidal demand with a gravity disturbance of 20 N m and proportional-integral control

Back EMF. A spinning motor acts like a generator and produces a voltage V_b called the back EMF which opposes the current flowing into the motor. Back EMF is proportional to motor speed $V_b = K_m\omega$ where K_m is the motor torque constant whose units can also be interpreted as V s rad^{-1}. When this voltage equals the maximum possible voltage from the drive electronics then no more current can flow into the motor and torque falls to zero. This provides a practical upper bound on motor speed, and torque at high speeds.

9.1.7 Position Control Loop

Another common approach is to use a proportional-integral-derivative (PID) controller for position but it can be shown that the D gain of this controller is related to the P gain of the inner velocity loop.

The outer loop is responsible for maintaining position and we use a proportional controller◂ based on the error between actual and demanded position to compute the desired speed of the motor

$$\dot{q}^* = K_p\left(q^*(t) - q\right) \tag{9.7}$$

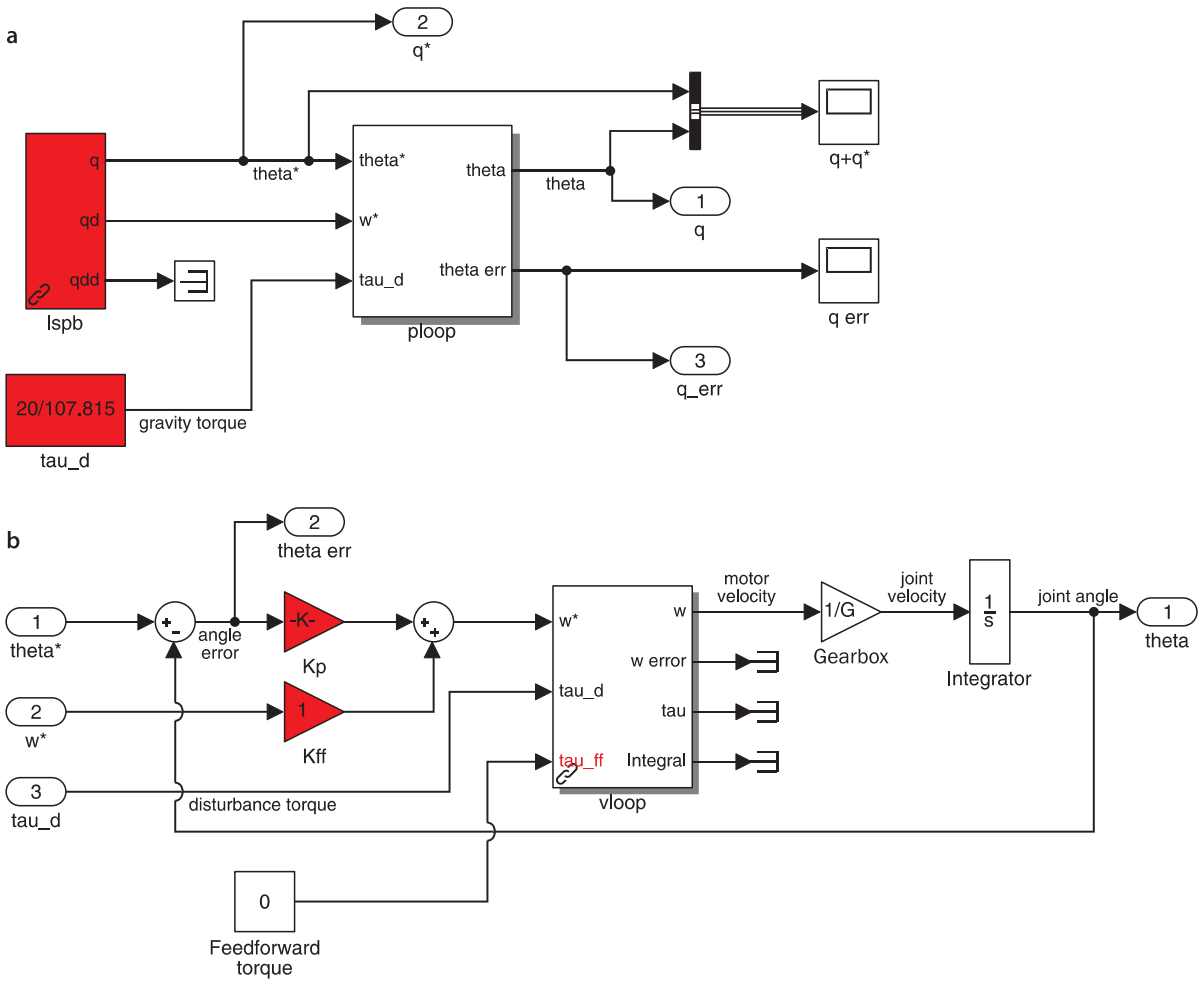
A Simulink model is shown in Fig. 9.13 and the position demand $q^*(t)$ comes from an LSPB trajectory generator that moves from 0 to 0.5 rad in 1 s with a sample rate of 1 000 Hz. Joint position is obtained by integrating joint velocity, obtained from the motor velocity loop via the gearbox. The error between the motor and desired position provides the velocity demand for the inner loop.

We load this control loop model

```
>> ploop_test
```

and its performance is tuned by adjusting the three gains: K_p, K_v, K_i in order to achieve good tracking performance along the trajectory. For $K_p = 40$ the tracking and error responses are shown in Fig. 9.14a. We see that the final error is zero but there is some tracking error along the path where the motor position lags behind the demand. The error between the demand and actual curves is due to the cumulative velocity error of the inner loop which has units of angle.

Fig. 9.13. Position control loop, Simulink model `ploop_test`. **a** Test harness for following an LSPB angle trajectory. **b** The position loop `ploop` which is a proportional controller around the inner velocity loop of Fig. 9.7



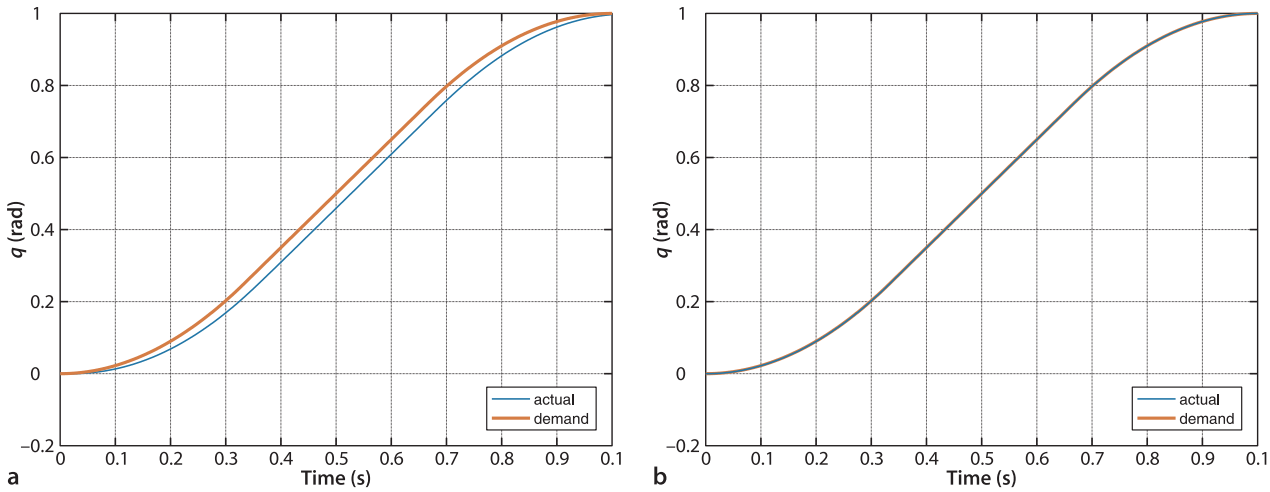


Fig. 9.14. Position loop following an LSPB trajectory. **a** Proportional control only **b** proportional control plus velocity demand feedforward

The position loop, like the velocity loop is based on classical negative feedback. Having zero position error while tracking a ramp would mean zero demanded velocity to the inner loop which is actually contradictory. More formally, we know that a Type 1 system exhibits a constant error to a ramp input. If we care about reducing this tracking error there are two common remedies. We can add an integrator to the position loop – making it a proportional-integral controller but this gives us yet another parameter to tune. A simple and effective alternative is velocity feedforward control – we add the desired velocity to the output of the proportional control loop, which is the input to the velocity loop. The LSPB trajectory function computes velocity as a function of time as well as position. The time response with velocity feedforward is shown in Fig. 9.14b and we see that tracking error is greatly reduced.

Since the model contains an integrator after the velocity loop.

9.1.8 Independent Joint Control Summary

A common structure for robot joint control is the nested control loop. The inner loop uses a proportional or proportional-integral control law to generate a torque so that the actual velocity closely follows the velocity demand. The outer loop uses a proportional control law to generate the velocity demand so that the actual position closely follows the position demand. Disturbance torques due to gravity and other dynamic coupling effects impact the performance of the velocity loop as do variation in the parameters of the plant being controlled, and this in turn leads to errors in position tracking. Gearing reduces the magnitude of disturbance torques by $1 / G$ and the variation in inertia and friction by $1 / G^2$ but at the expense of cost, weight, increased friction and mechanical noise.

The velocity loop performance can be improved by adding an integral control term, or by feedforward of the disturbance torque which is largely predictable. The position loop performance can also be improved by feedforward of the desired joint velocity. In practice control systems use both feedforward and feedback control. Feedforward is used to inject signals that we can compute, in this case the joint velocity, and in the earlier case the gravity torque. Feedback control compensates for all remaining sources of error including variation in inertia due to manipulator configuration and payload, changes in friction with time and temperature, and all the disturbance torques due to velocity and acceleration coupling. In general the use of feedforward allows the feedback gain to be reduced since a large part of the demand signal now comes from the feedforward.

9.2 Rigid-Body Equations of Motion

Consider the motor which actuates the j^{th} revolute joint of a serial-link manipulator. From Fig. 7.5 we recall that joint j connects link $j-1$ to link j. The motor exerts a torque that causes the outward link, j, to rotationally accelerate but it also exerts a reaction torque on the inward link $j-1$. Gravity acting on the outward links j to N exert a weight force, and rotating links also exert gyroscopic forces on each other. The inertia that the motor *experiences* is a function of the configuration of the outward links.

The situation at the individual link is quite complex but for the *series* of links the result can be written elegantly and concisely as a set of coupled differential equations in matrix form

$$\boldsymbol{Q} = \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{F}(\dot{\boldsymbol{q}}) + \boldsymbol{G}(\boldsymbol{q}) + \boldsymbol{J}(\boldsymbol{q})^T \boldsymbol{W} \tag{9.8}$$

where $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$ are respectively the vector of generalized joint coordinates, velocities and accelerations, $\boldsymbol{M}$ is the joint-space inertia matrix, $\boldsymbol{C}$ is the Coriolis and centripetal coupling matrix, $\boldsymbol{F}$ is the friction force, $\boldsymbol{G}$ is the gravity loading, and $\boldsymbol{Q}$ is the vector of generalized actuator forces associated with the generalized coordinates $\boldsymbol{q}$. The last term gives the joint forces due to a wrench $\boldsymbol{W}$ applied at the end-effector and $\boldsymbol{J}$ is the manipulator Jacobian. This equation describes the manipulator rigid-body dynamics and is known as the inverse dynamics – given the pose, velocity and acceleration it computes the required joint forces or torques.

The recursive form of the inverse dynamics does not explicitly calculate the matrices $\boldsymbol{M}$, $\boldsymbol{C}$ and $\boldsymbol{G}$ of Eq. 9.8. However we can use the recursive Newton-Euler algorithm to calculate these matrices and the Toolbox functions `inertia` and `coriolis` use Walker and Orin's (1982) *'Method 1'*. While the recursive forms are computationally efficient for the inverse dynamics, to compute the coefficients of the individual dynamic terms ($\boldsymbol{M}$, $\boldsymbol{C}$ and $\boldsymbol{G}$) in Eq. 9.8 is quite costly – $O(N^3)$ for an N-axis manipulator.

These equations can be derived using any classical dynamics method such as Newton's second law and Euler's equation of motion, as discussed in Sect. 3.2.1, or a Lagrangian energy-based approach. A very efficient way for computing Eq. 9.8 is the recursive Newton-Euler algorithm which starts at the base and working outward adds the velocity and acceleration of each joint in order to determine the velocity and acceleration of each link. Then working from the tool back to the base, it computes the forces and moments acting on each link and thus the joint torques. The recursive Newton-Euler algorithm has $O(N)$ complexity and can be written in functional form as

$$\boldsymbol{Q} = \mathcal{D}^{-1}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}) \tag{9.9}$$

Not all robot arm models in the Toolbox have dynamic parameters, see the "dynamics" tag in the output of the models() command, or use models('dyn') to list models with dynamic parameters. The Puma 560 robot is used for the examples in this chapter since its dynamic parameters are reliably known.

In the Toolbox it is implemented by the `rne` method of the `SerialLink` object. Consider the Puma 560 robot

```
>> mdl_puma560
```

at the nominal pose, and with zero joint velocity and acceleration. To achieve this state, the required generalized joint forces, or joint torques in this case, are

```
>> Q = p560.rne(qn, qz, qz)
Q =
   -0.0000   31.6399    6.0351    0.0000    0.0283         0
```

Since the robot is not moving (we specified $\dot{\boldsymbol{q}} = \ddot{\boldsymbol{q}} = 0$) these torques must be those required to *hold the robot up* against gravity. We can confirm this by computing the torques required in the absence of gravity

```
>> Q = p560.rne(qn, qz, qz, 'gravity', [0 0 0])
ans =
     0     0     0     0     0     0
```

by overriding the object's default gravity vector.

Like most Toolbox methods `rne` can operate on a trajectory

```
>> q = jtraj(qz, qr, 10)
>> Q = p560.rne(q, 0*q, 0*q)
```

which has returned

```
>> about(Q)
Q [double] : 10x6 (480 bytes)
```

a 10×6 matrix with each row representing the generalized force required for the corresponding row of `q`. The joint torques corresponding to the fifth time step are

```
>> Q(5,:)
ans =
    0.0000   29.8883    0.2489         0         0         0
```

Consider now a case where the robot is moving. It is *instantaneously* at the nominal pose but joint 1 is moving at 1 rad s^{-1} and the acceleration of all joints is zero. Then in the absence of gravity, the required joint torques

```
>> p560.rne(qn, [1 0 0 0 0 0], qz, 'gravity', [0 0 0])
   30.5332    0.6280   -0.3607   -0.0003   -0.0000         0
```

are nonzero. The torque on joint 1 is that needed to overcome friction which always opposes the motion. More interesting is that torques need to be exerted on joints 2, 3 and 4. This is to oppose the gyroscopic effects (centripetal and Coriolis forces) – referred to as velocity coupling torques since the rotational velocity of one joint has induced a torque on several other joints.

The elements of the matrices $\boldsymbol{M}$, $\boldsymbol{C}$, $\boldsymbol{F}$ and $\boldsymbol{G}$ are complex functions of the link's kinematic parameters (θ_j, d_j, a_j, α_j) and inertial parameters. Each link has ten independent inertial parameters: the link mass m_j; the center of mass (COM) $\boldsymbol{r}_j$ with respect to the link coordinate frame; and six second moments which represent the inertia of the link about the COM but with respect to axes aligned with the link frame $\{j\}$, see Fig. 7.5. We can view the dynamic parameters of a robot's link by

```
>> p560.links(1).dyn
Revolute(std): theta=q, d=0, a=0, alpha=1.5708, offset=0
  m     = 0
  r     = 0           0           0
  I     = | 0           0           0           |
          | 0           0.35        0           |
          | 0           0           0           |
  Jm    = 0.0002
  Bm    = 0.00148
  Tc    = 0.395      (+) -0.435     (-)
  G     = -62.61
  qlim = -2.792527 to 2.792527
```

which in order are: the kinematic parameters, link mass, COM position, link inertia matrix, motor inertia, motor friction, Coulomb friction, reduction gear ratio and joint angle limits.

The remainder of this section examines the various matrix components of Eq. 9.8.

9.2.1 Gravity Term

$$\boldsymbol{Q} = \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{F}(\dot{\boldsymbol{q}}) + \boldsymbol{G}(\boldsymbol{q}) + \boldsymbol{J}(\boldsymbol{q})^T \boldsymbol{W}$$

We start our detailed discussion with the gravity term because it is generally the dominant term in Eq. 9.8 and is present even when the robot is stationary or moving slowly. Some robots use counterbalance weights▸ or even springs to reduce the *gravity* torque that needs to be provided by the motors – this allows the motors to be smaller and thus lower in cost.

▸ Counterbalancing will however increase the inertia associated with a joint since it adds additional mass at the end of a lever arm, and increase the overall mass of the robot.

In the previous section we used the `rne` method to compute the gravity load by setting the joint velocity and acceleration to zero. A more convenient approach is to use the `gravload` method

```
>> gravload = p560.gravload(qn)
gravload =
   -0.0000   31.6399    6.0351    0.0000    0.0283         0
```

The `SerialLink` object contains a default gravitational acceleration vector which is initialized to the nominal value for Earth◂

The `'gravity'` option for the `SerialLink` constructor can change this.

```
>> p560.gravity'
ans =
         0         0    9.8100
```

We could change gravity to the lunar value

```
>> p560.gravity = p560.gravity/6;
```

resulting in reduced joint torques

```
>> p560.gravload(qn)
ans =
    0.0000    5.2733    1.0059    0.0000    0.0047         0
```

or we could turn our lunar robot upside down

```
>> p560.base = SE3.Rx(pi);
>> p560.gravload(qn)
ans =
    0.0000   -5.2733   -1.0059   -0.0000   -0.0047         0
```

and see that the torques have changed sign. Before proceeding we bring our robot back to Earth and right-side up

```
>> mdl_puma560
```

The torque exerted on a joint due to gravity acting on the robot depends very strongly on the robot's pose. Intuitively the torque on the shoulder joint is much greater when the arm is stretched out horizontally

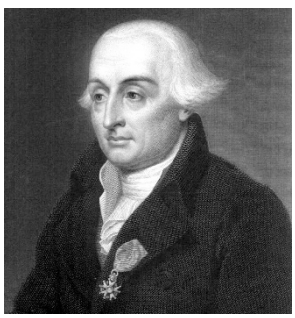
```
>> Q = p560.gravload(qs)
Q =
   -0.0000   46.0069    8.7722    0.0000    0.0283         0
```

than when the arm is pointing straight up

```
>> Q = p560.gravload(qr)
Q =
         0   -0.7752    0.2489         0         0         0
```

The gravity torque on the elbow is also very high in the first pose since it has to support the lower arm and the wrist. We can investigate how the gravity load on joints 2 and 3 varies with joint configuration by

```
[Q2,Q3] = meshgrid(-pi:0.1:pi, -pi:0.1:pi);
for i=1:numcols(Q2),
    for j=1:numcols(Q3);
    g = p560.gravload([0 Q2(i,j) Q3(i,j) 0 0 0]);
    g2(i,j) = g(2);
    g3(i,j) = g(3);
    end
end
surfl(Q2, Q3, g2); surfl(Q2, Q3, g3);
```



Joseph-Louis Lagrange (1736–1813) was an Italian-born (Giuseppe Lodovico Lagrangia) French mathematician and astronomer. He made significant contributions to the fields of analysis, number theory, classical and celestial mechanics. In 1766 he succeeded Euler as the director of mathematics at the Prussian Academy of Sciences in Berlin, where he stayed for over twenty years, producing a large body of work and winning several prizes of the French Academy of Sciences. His treatise on analytical mechanics "Mécanique Analytique" first published in 1788, offered the most comprehensive treatment of classical mechanics since Newton and formed a basis for the development of mathematical physics in the nineteenth century. In 1787 he became a member of the French Academy, was the first professor of analysis at the École Polytechnique, helped drive the decimalization of France, was a member of the Legion of Honour and a Count of the Empire in 1808. He is buried in the Panthéon in Paris.

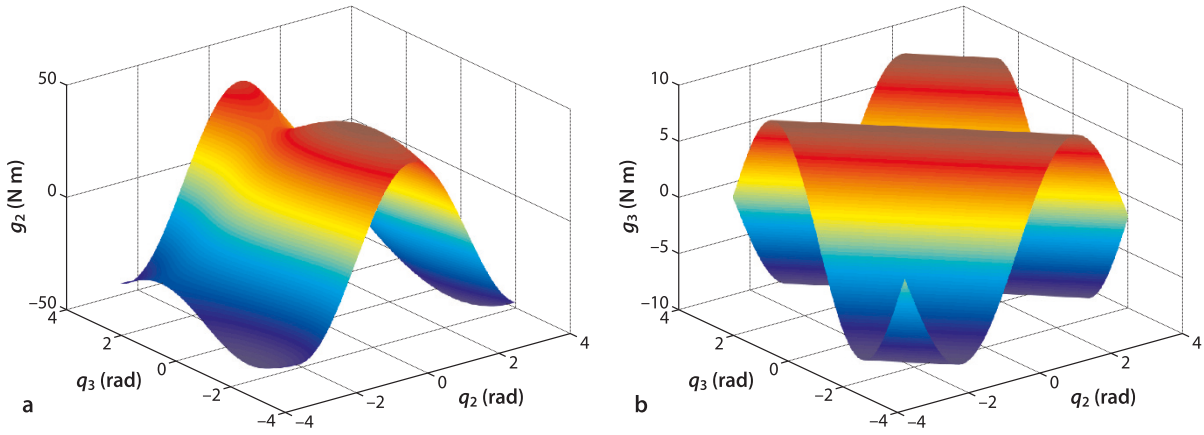


Fig. 9.15. Gravity load variation with manipulator pose. **a** Shoulder gravity load, $g_2(q_2, q_3)$; **b** elbow gravity load $g_3(q_2, q_3)$

and the results are shown in Fig. 9.15. The gravity torque on joint 2 varies between ± 40 N m and for joint 3 varies between ± 10 N m. This type of analysis is very important in robot design to determine the required torque capacity for the motors.

9.2.2 Inertia Matrix

$$Q = M(q)\ddot{q} + C(q, \dot{q})\dot{q} + F(\dot{q}) + G(q) + J(q)^T W$$

The joint-space inertia is a positive definite, and therefore symmetric, matrix▸

The diagonal elements of this inertia matrix includes the motor armature inertias, multiplied by G^2.

```
>> M = p560.inertia(qn)
M =
    3.6594   -0.4044    0.1006   -0.0025    0.0000   -0.0000
   -0.4044    4.4137    0.3509    0.0000    0.0024    0.0000
    0.1006    0.3509    0.9378    0.0000    0.0015    0.0000
   -0.0025    0.0000    0.0000    0.1925    0.0000    0.0000
    0.0000    0.0024    0.0015    0.0000    0.1713    0.0000
   -0.0000    0.0000    0.0000    0.0000    0.0000    0.1941
```

which is a function of the manipulator configuration. The diagonal elements M_{jj} describe the inertia *experienced* by joint j, that is, $Q_j = M_{jj}\ddot{q}_j$. Note that the first two diagonal elements, corresponding to the robot's waist and shoulder joints, are large since motion of these joints involves rotation of the heavy upper- and lower-arm links. The off-diagonal terms $M_{ij} = M_{ji}$, $i \neq j$ are the products of inertia and represent coupling of acceleration from joint j to the generalized force on joint i.

We can investigate some of the elements of the inertia matrix and how they vary with robot configuration using the simple (but slow▸) commands

Displaying the value of the robot object `>> p560` displays a tag `slowRNE` or `fastRNE`. The former indicates all calculations are done in MATLAB code. Build the MEX version, provided in the mex folder, to enable the `fastRNE` mode which is around 100 times faster.

```
[Q2,Q3] = meshgrid(-pi:0.1:pi, -pi:0.1:pi);
for i=1:numcols(Q2)
    for j=1:numcols(Q3)
        M = p560.inertia([0 Q2(i,j) Q3(i,j) 0 0 0]);
        M11(i,j) = M(1,1);
        M12(i,j) = M(1,2);
    end
end
surfl(Q2, Q3, M11); surfl(Q2, Q3, M12);
```

The results are shown in Fig. 9.16 and we see significant variation in the value of M_{11} which changes by a factor of

```
>> max(M11(:)) / min(M11(:))
ans =
    2.1558
```

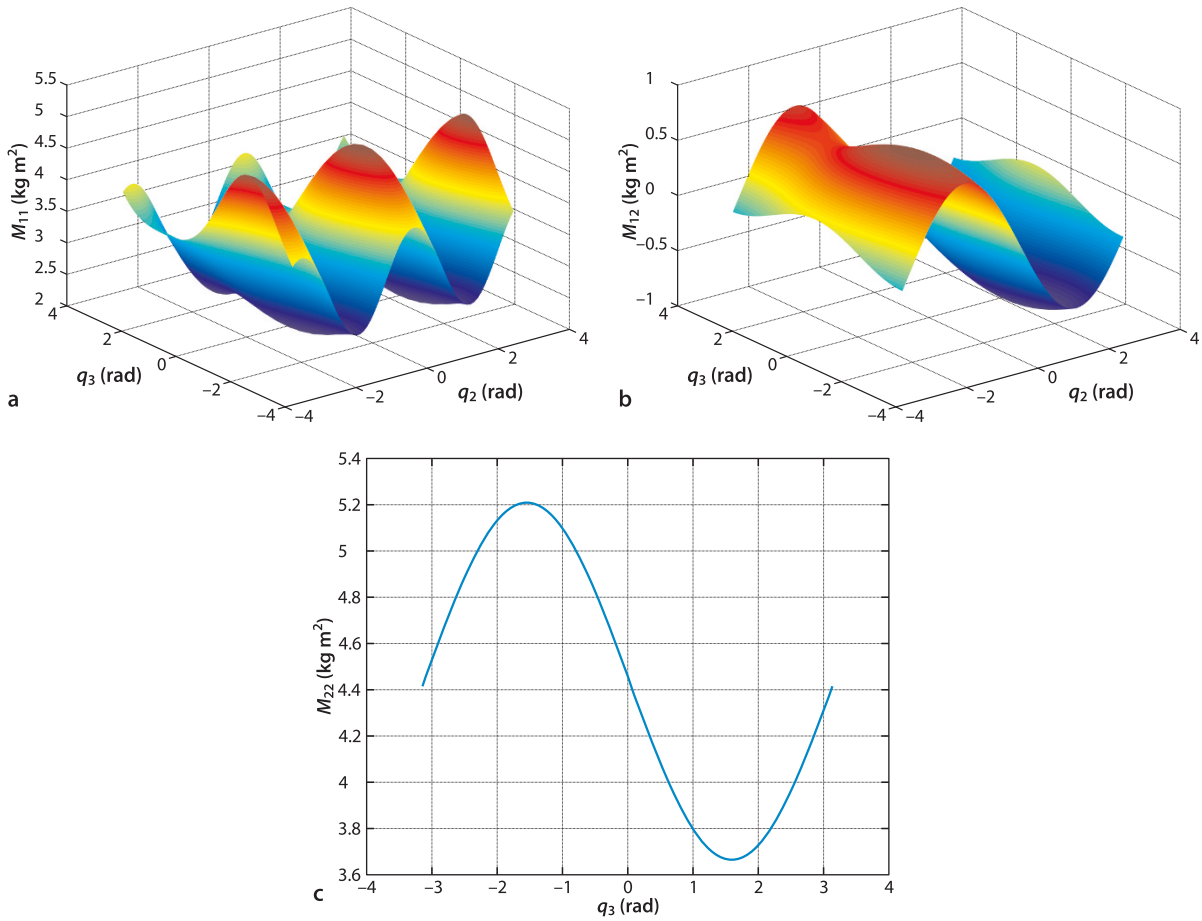


Fig. 9.16. Variation of inertia matrix elements as a function of manipulator pose. **a** Joint 1 inertia as a function of joint 2 and 3 angles $M_{11}(q_2, q_3)$; **b** product of inertia $M_{12}(q_2, q_3)$; **c** joint 2 inertia as a function of joint 3 angle $M_{22}(q_3)$

This is important for robot design since, for a fixed maximum motor torque, inertia sets the upper bound on acceleration which in turn effects path following accuracy.

The off-diagonal term M_{12} represents coupling between the angular acceleration of joint 2 and the torque on joint 1. That is, if joint 2 accelerates then a torque will be exerted on joint 1 and vice versa.

9.2.3 Coriolis Matrix

$$\boldsymbol{Q} = \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{F}(\dot{\boldsymbol{q}}) + \boldsymbol{G}(\boldsymbol{q}) + \boldsymbol{J}(\boldsymbol{q})^T \boldsymbol{W}$$

The Coriolis matrix $\boldsymbol{C}$ is a function of joint coordinates and joint velocity. The centripetal torques are proportional to $\dot{q}_j^2$, while the Coriolis torques are proportional to $\dot{q}_i \dot{q}_j$. For example, at the nominal pose with the elbow joint moving at 1 rad s^{-1}

```
>> qd = [0 0 1 0 0 0];
```

the Coriolis matrix is

```
>> C = p560.coriolis(qn, qd)
C =
    0.8992   -0.2380   -0.2380    0.0005   -0.0375    0.0000
   -0.0000    0.9106    0.9106         0   -0.0036         0
    0.0000    0.0000   -0.0000         0   -0.0799         0
   -0.0559    0.0000    0.0000   -0.0000    0.0000   -0.0000
   -0.0000    0.0799    0.0799   -0.0000         0         0
    0.0000         0         0    0.0000         0         0
```

The off-diagonal terms $C_{i,j}$ represent coupling of joint j velocity to the generalized force acting on joint i. $C_{2,3} = 0.9106$ represents significant coupling from joint 3 velocity to torque on joint 2 – rotation of the elbow exerting a torque on the shoulder. Since the elements of this matrix represents a coupling from velocity to joint force they have the same dimensions as viscous friction or damping, however the sign can be positive or negative. The joint torques due to the motion of just this one joint are

```
>> C*qd'
ans =
   -0.2380
    0.9106
   -0.0000
    0.0000
    0.0799
         0
```

9.2.4 Friction

$$Q = M(q)\ddot{q} + C(q,\dot{q})\dot{q} + F(\dot{q}) + G(q) + J(q)^T W$$

For most electric drive robots friction is the next most dominant joint force after gravity.

For the Puma robot joint friction varied from 10 to 47% of the maximum motor torque for the first three joints (Corke 1996b).

The Toolbox models friction within the `Link` object. The friction values are lumped and motor referenced, that is, they apply to the motor side of the gearbox. Viscous friction is a scalar that applies for positive and negative velocity. Coulomb friction is a 2-vector comprising (Q_C^+, Q_C^-). The dynamic parameters of the Puma robot's first link are shown on page 264 as link parameters `Bm` and `Tc`. The online documentation for the `Link` class describes how to set these parameters.

In practice some mechanisms have a velocity dependent friction characteristic.

9.2.5 Effect of Payload

Any real robot has a specified maximum payload which is dictated by two dynamic effects. The first is that a mass at the end of the robot will increase the inertia *experienced* by the joint motors and which reduces acceleration and dynamic performance. The second is that mass generates a weight force which all the joints need to support. In the worst case the increased gravity torque component might exceed the rating of one or more motors. However even if the rating is not exceeded there is less torque available for acceleration which again reduces dynamic performance.

As an example we will add a 2.5 kg point mass to the Puma 560 which is its rated maximum payload. The center of mass of the payload cannot be at the center of the wrist coordinate frame, that is inside the wrist, so we will offset it 100 mm in the z-direction of the wrist frame. We achieve this by modifying the inertial parameters of the robot's last link

This assumes that the last link itself has no mass which is a reasonable approximation.

```
>> p560.payload(2.5, [0 0 0.1]);
```

The inertia at the nominal pose is now

```
>> M_loaded = p560.inertia(qn);
```

and the *ratio* with respect to the unloaded case, computed earlier, is

```
>> M_loaded ./ M
ans =
    1.3363    0.9872    2.1490   49.3960   80.1821    1.0000
    0.9872    1.2667    2.9191    5.9299   74.0092    1.0000
    2.1490    2.9191    1.6601   -2.1092   66.4071    1.0000
   49.3960    5.9299   -2.1092    1.0647   18.0253    1.0000
   83.4369   74.0092   66.4071   18.0253    1.1454    1.0000
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
```

We see that the diagonal elements have increased significantly, for instance the elbow joint inertia has increased by 66% which reduces the maximum acceleration by nearly 40%. Reduced acceleration impairs the robot's ability to accurately follow a high speed path. The inertia of joint 6 is unaffected since this added mass lies on the axis of this joint's rotation. The off-diagonal terms have increased significantly, particularly in rows and columns four and five. This indicates that motion of joints 4 and 5, the wrist joints, which are swinging the offset mass give rise to large reaction forces that are *felt* by all the other robot joints.

The gravity load has also increased by some significant factors

```
>> p560.gravload(qn) ./ gravload
ans =
    0.3737    1.5222    2.5416   18.7826   86.8056       NaN
```

at the elbow and wrist. Note that the values for joints 1, 4 and 6 are invalid since they are each the quotient of numbers that are almost zero. We set the payload of the robot back to zero before proceeding

```
>> p560.payload(0)
```

9.2.6 Base Force

A moving robot exerts a wrench on its base – its weight as well as reaction forces and torques as the arm moves around. This wrench is returned as an optional output argument of the `rne` method, for example

```
>> [Q,Wb] = p560.rne(qn, qz, qz);
```

The wrench

```
>> Wb'
ans =
         0   -0.0000  230.0445  -48.4024  -31.6399   -0.0000
```

needs to be applied to the base to keep it in equilibrium. The vertical force of 230 N is the total weight of the robot which has a mass of

```
>> sum([p560.links.m])
ans =
   23.4500
```

There is also a moment about the x- and y-axes since the center of mass of the robot in this configuration is not over the origin of the base coordinate frame.

The base forces are important in situations where the robot does not have a rigid base such as on a satellite in space, on a boat, an underwater vehicle or even on a vehicle with soft suspension.

9.2.7 Dynamic Manipulability

In Sect. 8.2.2 we discussed a kinematic measure of manipulability, that is, how well configured the robot is to achieve velocity in any Cartesian direction. The force ellipsoid of Sect. 8.5.2 describes how well the manipulator is able to accelerate in different Cartesian directions but is based on the kinematic, not dynamic, parameters of the robot arm. Following a similar approach, we consider the set of generalized joint forces with unit norm

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

From Eq. 9.8 and ignoring gravity and assuming $\dot{\boldsymbol{q}} = 0$ we write

$$\boldsymbol{Q} = \boldsymbol{M}\ddot{\boldsymbol{q}}$$

Differentiating Eq. 8.2 and still assuming $\dot{\boldsymbol{q}} = 0$ we write

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

Combining these we write

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

or more compactly

$$\dot{\boldsymbol{\nu}}^T \boldsymbol{M}_x^{-1} \dot{\boldsymbol{\nu}} = 1$$

which is the equation of a hyperellipsoid in Cartesian acceleration space. For example, at the nominal pose

```
>> J = p560.jacob0(qn);
>> M = p560.inertia(qn);
>> Mx = (J * inv(M) * inv(M)' * J');
```

If we consider just the translational acceleration, that is the top left 3×3 submatrix of $\boldsymbol{M}_x$

```
>> Mx = Mx(1:3, 1:3);
```

this is a 3-dimensional ellipsoid

```
>> plot_ellipse( Mx )
```

which is plotted in Fig. 9.17. The major axis of this ellipsoid is the direction in which the manipulator has maximum acceleration at this configuration. The radii of the ellipse are the square roots of the eigenvalues

```
>> sqrt(eig(Mx))
ans =
    0.4412
    0.1039
    0.1677
```

and the direction of maximum acceleration is given by the first eigenvector. The ratio of the minimum to maximum radius

```
>> min(ans)/max(ans)
ans =
    0.2355
```

is a measure of the nonuniformity of end-effector acceleration.▶ It would be unity for isotropic acceleration capability. In this case acceleration capability is good in the x- and z-directions, but poor in the y-direction.

The 6-dimensional ellipsoid has dimensions with different units: m s^{-2} and rad s^{-2}. This makes comparison of all 6 radii problematic.

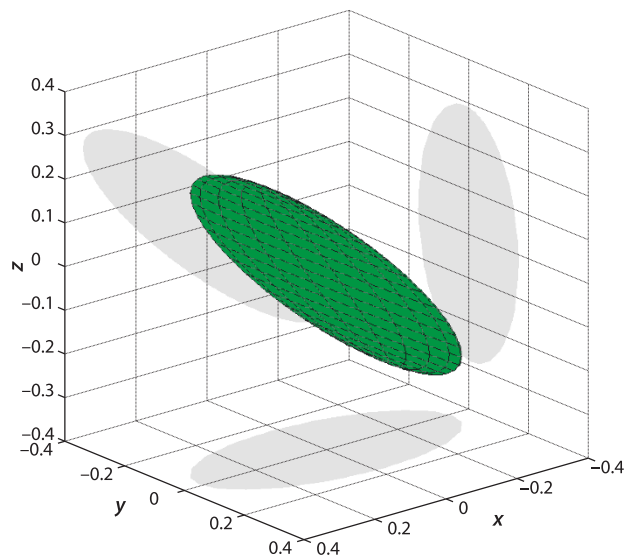


Fig. 9.17.
Spatial acceleration ellipsoid for Puma 560 robot in nominal pose

The scalar dynamic manipulability measure proposed by Asada is similar but considers the ratios of the eigenvalues of

$$\ddot{\boldsymbol{x}}^T \boldsymbol{J}^{-T} \boldsymbol{M} \boldsymbol{J}^{-1} \ddot{\boldsymbol{x}} = 1$$

and returns a uniformity measure $m \in [0, 1]$ where 1 indicates uniformity of acceleration in all directions. For this example

```
>> p560.maniplty(qn, 'asada')
ans =
    0.2094
```

9.3 Forward Dynamics

To determine the motion of the manipulator in response to the forces and torques applied to its joints we require the forward dynamics or integral dynamics. Rearranging the equations of motion Eq. 9.8 we obtain the joint acceleration

$$\ddot{\boldsymbol{q}} = \boldsymbol{M}^{-1}(\boldsymbol{q})\left(\boldsymbol{Q} - \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} - \boldsymbol{F}(\dot{\boldsymbol{q}}) - \boldsymbol{G}(\boldsymbol{q}) - \boldsymbol{J}(\boldsymbol{q})^T \boldsymbol{W}\right) \tag{9.10}$$

and $\boldsymbol{M}$ is always invertible. This function is computed by the `accel` method of the `SerialLink` class

```
qdd = p560.accel(q, qd, Q)
```

given the joint coordinates, joint velocity and applied joint torques. This functionality is also encapsulated in the Simulink block `Robot` and an example of its use is

```
>> sl_ztorque
```

which is shown in Fig. 9.18. The torque applied to the robot is zero and the initial joint angles is set as a parameter of the `Robot` block, in this case to the *zero-angle pose*. The simulation is run

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

and the joint angles as a function of time are returned in the object `r`

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

We can show the robot's motion in animation

```
>> p560.plot(q)
```

and see it collapsing under gravity since there are no torques to counter gravity and hold in upright. The shoulder falls and swings back and forth as does the elbow, while the waist joint rotates because of Coriolis coupling. The motion will slowly decay as the energy is dissipated by viscous friction.

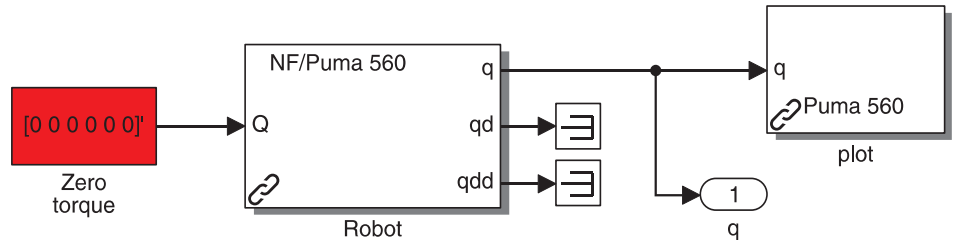


Fig. 9.18. Simulink model `sl_ztorque` for the Puma 560 manipulator with zero joint torques. This model removes Coulomb friction in order to simplify the numerical integration

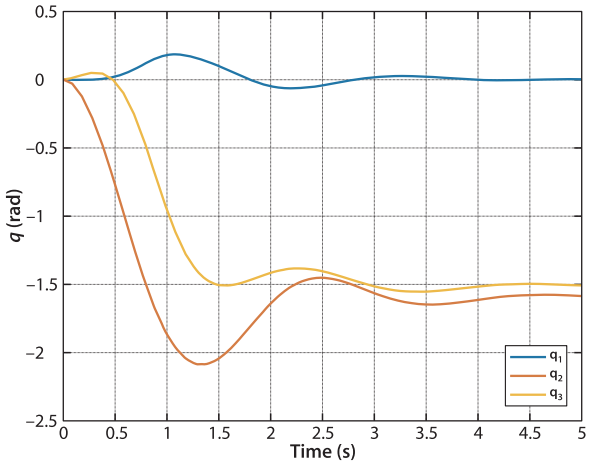


Fig. 9.19.
Joint angle trajectory for Puma 560 robot with zero Coulomb friction collapsing under gravity from initial joint configuration `qz`

Alternatively we can plot the joint angles as a function of time

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

and this is shown in Fig. 9.19. The method `fdyn` can be used as a nongraphical alternative to Simulink and is described in the online documentation.

This example is rather unrealistic and in reality the joint torques would be computed by some control law as a function of the actual and desired robot joint angles. This is the topic of the next section.

Coulomb friction is a strong nonlinearity and can cause difficulty when using numerical integration routines to solve the forward dynamics. This is usually manifested by very long integration times. Fixed-step solvers tend to be more tolerant, and these can be selected through the Simulink `Simulation+Model Configuration Parameters+Solver` menu item.

The default Puma 560 model, defined using `mdl_puma560`, has nonzero viscous and Coulomb friction parameters for each joint. Sometimes it is useful to zero the friction parameters for a robot and this can be achieved by

```
>> p560_nf = p560.nofriction();
```

which returns a copy of the robot object that is similar in all respects except that the Coulomb friction is zero. Alternatively we can set Coulomb *and* viscous friction coefficients to zero

```
>> p560_nf = p560.nofriction('all');
```

9.4 Rigid-Body Dynamics Compensation

In Sect. 9.1 we discussed some of the challenges for independent joint control and introduced the concept of feedforward to compensate for the gravity disturbance torque. Inertia variation and other dynamic coupling forces were not explicitly dealt with and were left for the feedback controller to handle. However inertia and coupling torques can be computed according to Eq. 9.8 given knowledge of joint angles, joint velocities and accelerations, and the inertial parameters of the links. We can incorporate these torques into the control law using one of two *model-based* approaches: feedforward control, and computed torque control. The structural differences are contrasted in Fig. 9.20 and Fig. 9.21.

9.4.1 Feedforward Control

The torque feedforward controller shown in Fig. 9.20 is given by

$$\begin{aligned} \boldsymbol{Q}^* &= \underbrace{\boldsymbol{M}\left(\boldsymbol{q}^*\right)\ddot{\boldsymbol{q}}^* + \boldsymbol{C}\left(\boldsymbol{q}^*, \dot{\boldsymbol{q}}^*\right)\dot{\boldsymbol{q}}^* + \boldsymbol{F}\left(\dot{\boldsymbol{q}}^*\right) + \boldsymbol{G}\left(\boldsymbol{q}^*\right)}_{\text{feedforward}} + \underbrace{\left\{\boldsymbol{K}_v\left(\dot{\boldsymbol{q}}^* - \dot{\boldsymbol{q}}^{\#}\right) + \boldsymbol{K}_p\left(\boldsymbol{q}^* - \boldsymbol{q}^{\#}\right)\right\}}_{\text{feedback}} \\ &= \mathcal{D}^{-1}\left(\boldsymbol{q}^*, \dot{\boldsymbol{q}}^*, \ddot{\boldsymbol{q}}^*\right) + \left\{\boldsymbol{K}_v\left(\dot{\boldsymbol{q}}^* - \dot{\boldsymbol{q}}^{\#}\right) + \boldsymbol{K}_p\left(\boldsymbol{q}^* - \boldsymbol{q}^{\#}\right)\right\} \end{aligned} \tag{9.11}$$

where $\boldsymbol{K}_p$ and $\boldsymbol{K}_v$ are the position and velocity gain (or damping) matrices respectively, and $\mathcal{D}^{-1}(\cdot)$ is the inverse dynamics function. The gain matrices are typically diagonal. The feedforward term provides the joint forces required for the desired manipulator state $(\boldsymbol{q}^*, \dot{\boldsymbol{q}}^*, \ddot{\boldsymbol{q}}^*)$ and the feedback term compensates for any errors due to uncertainty in the inertial parameters, unmodeled forces or external disturbances.

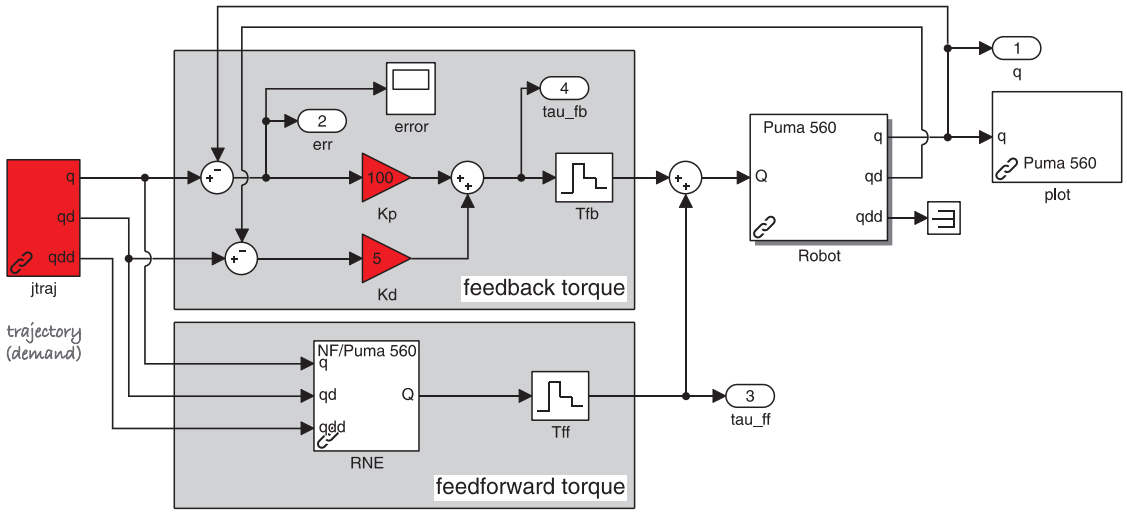
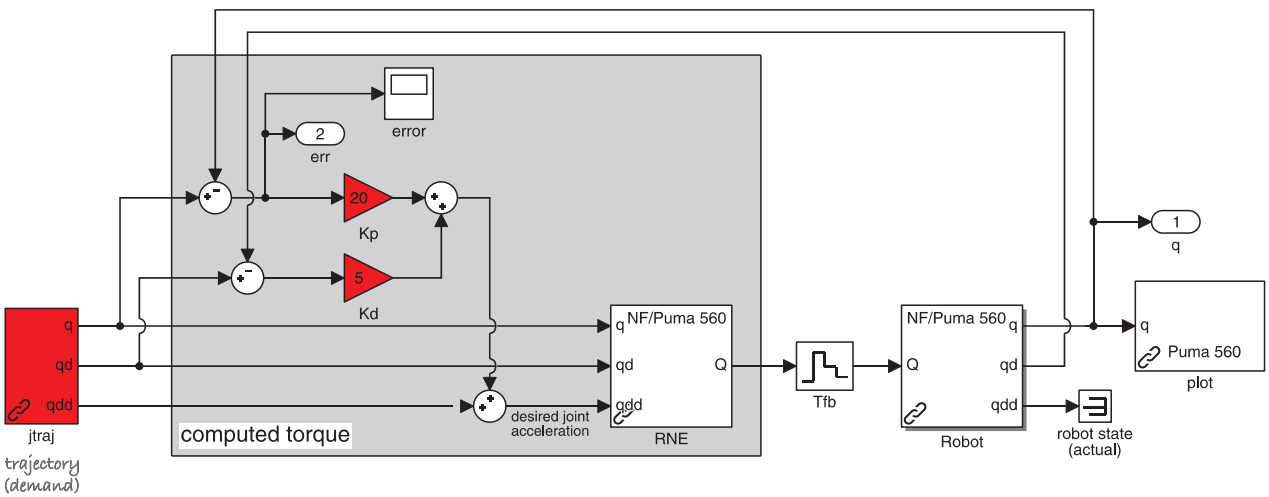


Fig. 9.20. The Simulink model `sl_fforward` for Puma 560 with torque feedforward control. The blocks with the staircase icons are zero-order holds

Fig. 9.21. Robotics Toolbox example `sl_ctorque`, computed torque control



We can also consider that the feedforward term linearizes the nonlinear dynamics about the operating point $(\boldsymbol{q}^*,\dot{\boldsymbol{q}}^*,\ddot{\boldsymbol{q}}^*)$. If the linearization is ideal then the dynamics of the error $\boldsymbol{e} = \boldsymbol{q}^* - \boldsymbol{q}^{\#}$ can be obtained by combining Eq. 9.8 and 9.11

$$\boldsymbol{M}\left(\boldsymbol{q}^*\right)\ddot{\boldsymbol{e}} + \boldsymbol{K}_v\dot{\boldsymbol{e}} + \boldsymbol{K}_p\boldsymbol{e} = 0 \tag{9.12}$$

For well chosen $\boldsymbol{K}_p$ and $\boldsymbol{K}_v$ the error will decay to zero but the joint errors are coupled▶ and their dynamics are dependent on the manipulator configuration.

Due to the nondiagonal matrix $\boldsymbol{M}$.

To test this controller using Simulink we first create a `SerialLink` object

```
>> mdl_puma560
```

and then load the torque feedforward controller model

```
>> sl_fforward
```

The feedforward torque is computed using the `RNE` block and added to the feedback torque computed from position and velocity error. The desired joint angles and velocity are generated using a `jtraj` block. Since the robot configuration changes relatively slowly the feedforward torque can be evaluated at a greater interval, T_{ff}, than the error feedback loops, T_{fb}. In this example we use a zero-order hold block sampling at the relatively low sample rate of 20 Hz.

We run the simulation by pushing the Simulink play button or

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

9.4.2 Computed Torque Control

The computed torque controller is shown in Fig. 9.21. It belongs to a class of controllers known as inverse dynamic control. The principle is that the nonlinear system is cascaded with its inverse so that the overall system has a constant unity gain. In practice the inverse is not perfect so a feedback loop is required to deal with errors.

The computed torque control is given by

$$\begin{aligned}\boldsymbol{Q} &= \boldsymbol{M}(\boldsymbol{q})\left\{\ddot{\boldsymbol{q}}^* + \boldsymbol{K}_v\left(\dot{\boldsymbol{q}}^* - \dot{\boldsymbol{q}}^{\#}\right) + \boldsymbol{K}_p\left(\boldsymbol{q}^* - \boldsymbol{q}^{\#}\right)\right\} + \boldsymbol{C}(\boldsymbol{q}^*,\dot{\boldsymbol{q}}^*)\dot{\boldsymbol{q}}^* + \boldsymbol{F}(\dot{\boldsymbol{q}}^*) + \boldsymbol{G}(\boldsymbol{q}^*) \\ &= \mathcal{D}^{-1}\left(\boldsymbol{q}^*,\dot{\boldsymbol{q}}^*,\left(\ddot{\boldsymbol{q}}^* + \boldsymbol{K}_v\left(\dot{\boldsymbol{q}}^* - \dot{\boldsymbol{q}}^{\#}\right) + \boldsymbol{K}_p\left(\boldsymbol{q}^* - \boldsymbol{q}^{\#}\right)\right)\right)\end{aligned} \tag{9.13}$$

where $\boldsymbol{K}_p$ and $\boldsymbol{K}_v$ are the position and velocity gain (or damping) matrices respectively, and $\mathcal{D}^{-1}(\cdot)$ is the inverse dynamics function.

In this case the inverse dynamics must be evaluated at each servo interval, although the coefficient matrices $\boldsymbol{M}$, $\boldsymbol{C}$, and $\boldsymbol{G}$ could be evaluated at a lower rate since the robot configuration changes relatively slowly. Assuming ideal modeling and parameterization the error dynamics of the system are obtained by combining Eq. 9.8 and 9.13

$$\ddot{\boldsymbol{e}} + \boldsymbol{K}_v\dot{\boldsymbol{e}} + \boldsymbol{K}_p\boldsymbol{e} = 0 \tag{9.14}$$

where $e = q^* - q^{\#}$. Unlike Eq. 9.12 the joint errors are uncoupled and their dynamics are therefore independent of manipulator configuration. In the case of model error there will be some coupling between axes, and the right-hand side of Eq. 9.14 will be a nonzero forcing function.

Using Simulink we first create a `SerialLink` object and then load the computed torque controller

```
>> mdl_puma560
>> sl_ctorque
```

The desired joint angles and velocity are generated using a `jtraj` block whose parameters are the initial and final joint angles. We run the simulation by pushing the Simulink play button or

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

9.4.3 Operational Space Control

The control strategies so far have been posed in terms of the robot's joint coordinates – its configuration space. Equation 9.8 describes the relationship between joint position, velocity, acceleration and applied forces or torques. However we can also express the dynamics of the end-effector in the Cartesian operational space where we consider the end-effector as a rigid body with inertia that actuator and disturbance forces and torques act on. We can reformulate Eq. 9.8 in operational space as

$$\boldsymbol{\Lambda}(\boldsymbol{x})\ddot{\boldsymbol{x}} + \boldsymbol{\mu}(\boldsymbol{x}, \dot{\boldsymbol{x}})\dot{\boldsymbol{x}} + \boldsymbol{p}(\boldsymbol{x}) = \boldsymbol{W} \tag{9.15}$$

where $\boldsymbol{x} \in \mathbb{R}^6$ is the manipulator Cartesian pose and $\boldsymbol{\Lambda}$ is the end-effector inertia which is subject to a gyroscopic and Coriolis force μ and gravity load $\boldsymbol{p}$ and an applied control wrench $\boldsymbol{W}$. These operational space terms are related to those we have already discussed by

$$\begin{aligned}
\boldsymbol{\Lambda}(\boldsymbol{x}) &= \boldsymbol{J}(\boldsymbol{q})^{-T}\boldsymbol{J}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q})^{-1} \\
\boldsymbol{\mu}(\boldsymbol{x}, \dot{\boldsymbol{x}}) &= \boldsymbol{J}(\boldsymbol{q})^{-T}\boldsymbol{C}(q, \dot{q}) - \boldsymbol{\Lambda}(\boldsymbol{q})\dot{\boldsymbol{J}}(\boldsymbol{q})\dot{\boldsymbol{q}} \\
p(\boldsymbol{x}) &= \boldsymbol{J}(\boldsymbol{q})^{-T}\boldsymbol{g}(\boldsymbol{q})
\end{aligned}$$

Imagine the task of wiping a table when the table's height is unknown and its surface is only approximately horizontal. The robot's z-axis is vertical so to achieve the task we need to move the end-effector along a path in the xy-plane to achieve coverage and hold the wiper at a constant orientation about the z-axis. Simultaneously we maintain a constant force in the z-direction to hold the wiper against the table and a constant torque about the x- and y-axes in order to conform to the orientation of the table top. The first group of axes are position controlled, and the second group are force controlled. Each Cartesian degree of freedom can be either position or force controlled. The operational space control allows independent control of position and forces along and about the axes of the operational space coordinate frame.

A Simulink model of the controller and a simplified version of this scenario can be loaded by

```
>> sl_opspace
```

and is shown in Fig. 9.22. It comprises a position-control loop and a force-control loop whose results are summed together and used to drive the operational space robot model – details can be found by opening that block in the Simulink diagram. In this simulation the operational space coordinate frame is parallel to the end-effector coordinate frame. Motion is position controlled in the x- and y-directions and about the x-, y- and z-axes of this frame – the robot moves from its initial pose to a nearby pose using 5 out of the 6 Cartesian DOF.◀

The robot model and the compliance specification are set by the model's `InitFcn` callback function. The setpoints are the red user adjustable boxes in the top-level diagram.

Motion is force controlled in the z-direction with a setpoint of –5 N. To achieve this the controller moves the end-effector downward in order to decrease the force. It moves in free space until it touches the surface at $z = -0.2$ which is modeled as a stiffness of 100 N m^{-1}. Results in Fig. 9.23 show the x- and y-position moving toward the goal and the z-position decreasing and the simulated sensed force decreasing after contact. The controller is able to simultaneously satisfy position and force constraints.

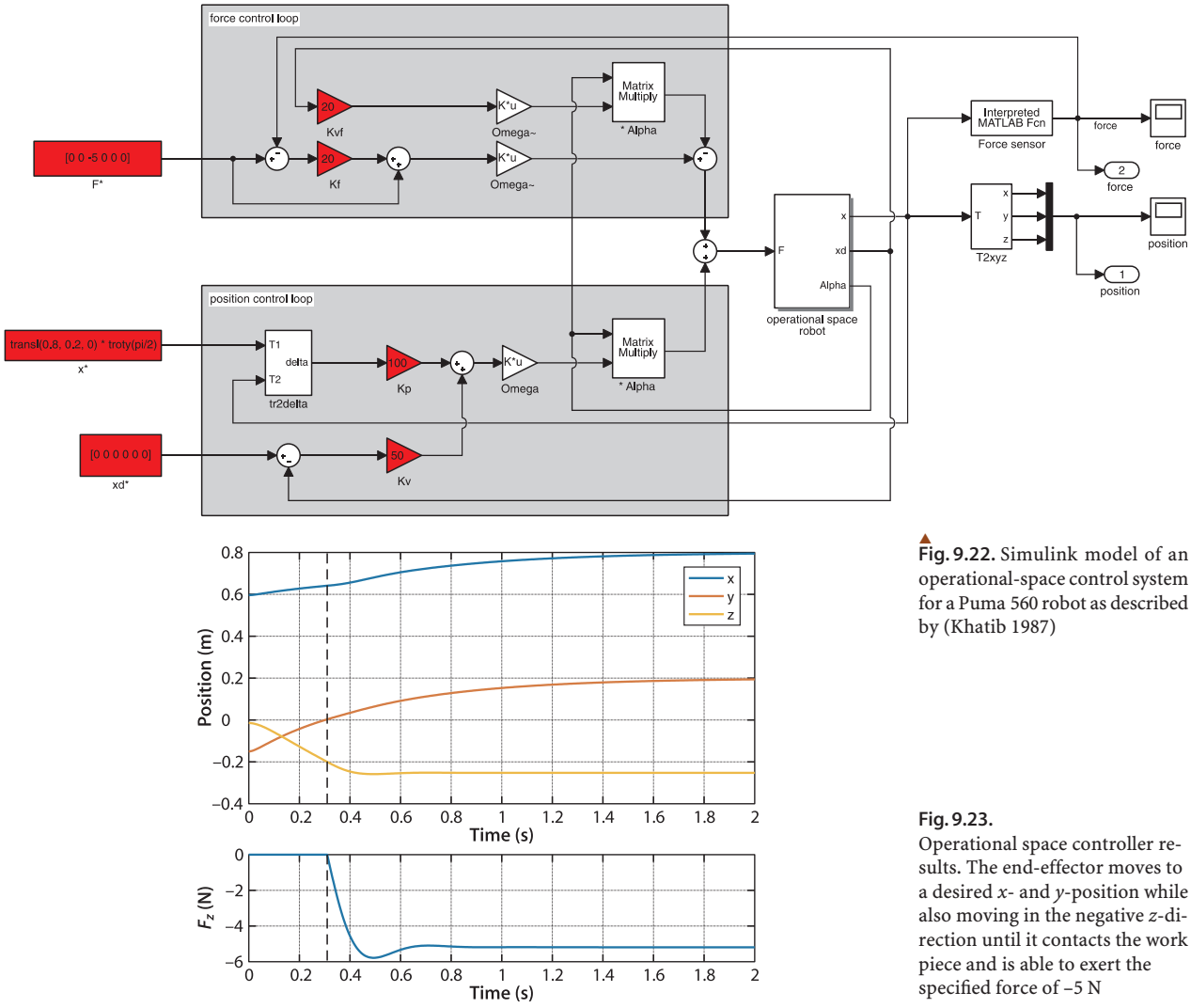


Fig. 9.22. Simulink model of an operational-space control system for a Puma 560 robot as described by (Khatib 1987)

Fig. 9.23. Operational space controller results. The end-effector moves to a desired x- and y-position while also moving in the negative z-direction until it contacts the work piece and is able to exert the specified force of –5 N

9.5 Applications

9.5.1 Series-Elastic Actuator (SEA)

For high-speed robots the elasticity of the links and the joints becomes a significant dynamic effect which will affect path following accuracy. Joint elasticity is typically caused by elements of the transmission such as: longitudinal elasticity of a toothed belt or cable drive, a harmonic gearbox which is inherently elastic, or torsional elasticity of a motor shaft. In dynamic terms, as shown schematically in Fig. 9.24, the problem arises because the force is applied to one side of an elastic element and we wish to control the position of the other side – the actuator and sensor are not colocated. More complex still, and harder to analyze, is the case where the elasticity of the links must be taken into account.

However there are advantages in having some flexibility between the motor and the load. Imagine a robot performing a task that involves the gripper picking an object off a table whose height is uncertain.▸ A simple strategy to achieve this is to move down until the gripper touches the table, close the gripper and then lift up. However at the instant of contact a large and discontinuous force will be exerted on the robot which has the potential to damage the object or the robot. This is particularly problematic for robots with large

▸ Or the robot is not very accurate.

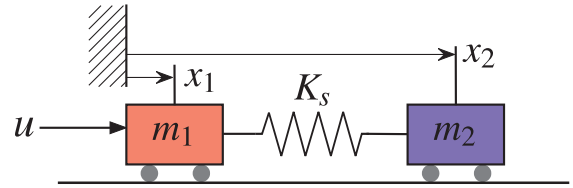


Fig. 9.24. Schematic of a series-elastic actuator. The two masses represent the motor and the load, and they are connected by an elastic element or spring

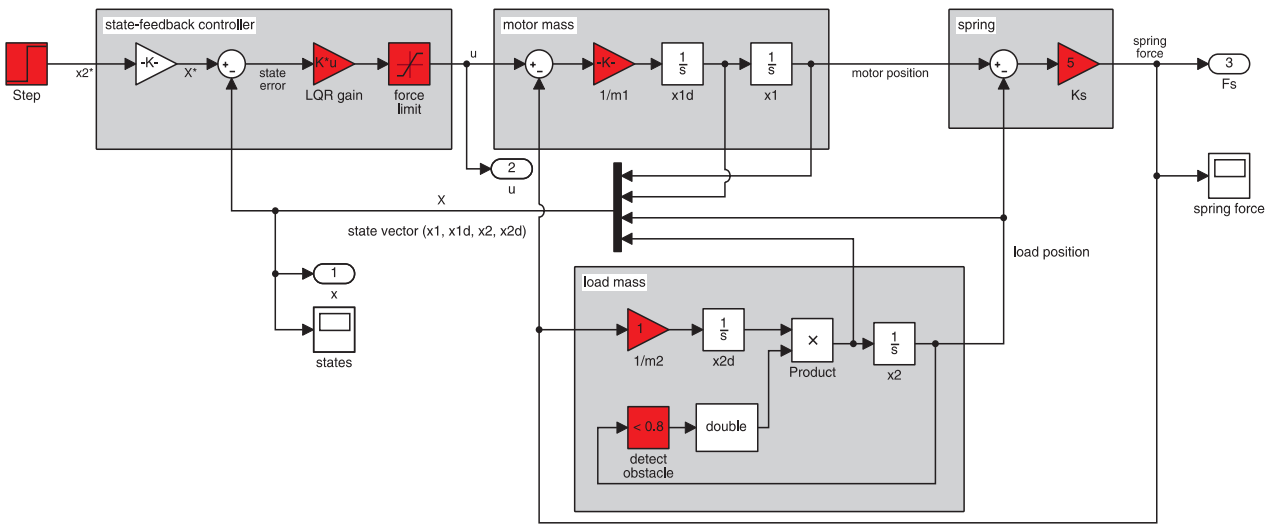


Fig. 9.25. Simulink model `sl_sea` of a series-elastic actuator colliding with an obstacle

inertia that are moving quickly – the kinetic energy must be instantaneously dissipated. An elastic element – a spring – between the motor and the joint would help here. At the moment of contact the spring would start to compress and the kinetic energy is transferred to potential energy in the spring – the robot control system has time to react and stop or reverse the motors. We have changed the problem from a damaging hard impact to a soft impact. In addition to shock absorption, the deformation of the spring provides a means of determining the force that the robot is exerting. This capability is particularly useful for robots that interact closely with people since it makes the robot less dangerous in case of collision, and a spring is simple technology that cannot fail. For robots that must exert a force as part of their task, this is a simpler approach than the operational space controller introduced in Sect. 9.4.3. However position control is now more challenging because there is an elastic element between the motor and the load.

Consider the 1-dimensional case shown in Fig. 9.24 where the motor is represented by a mass m_1 to which a controllable force u is applied.◂ It is connected via a linear elastic element or spring to the load mass m_2. If we apply a positive force to m_1 it will move to the right and compress the spring, and this will exert a positive force on m_2 which will also move to the right. Controlling the position of m_2 is not trivial since this system has no friction and is marginally stable. It can be stabilized by feedback of position and velocity of the motor *and* of the load – all of which are potentially measurable.

In a real robot this is a rotary system with a torsional spring.

In robotics such a system, built into a robot joint, is known as a series-elastic actuator or SEA. The Baxter robot of Fig. 7.1b includes SEAs in some of its joints.

A Simulink model of an SEA system can be loaded by

```
>> sl_sea
```

and is shown in Fig. 9.25. A state-feedback LQR controller has been designed using MATLAB and requires input of motor and load position and velocity which form a vector `x` in the Simulink model. Fig. 9.26 shows a simulation of the model moving the load m_2 to a position $x_2^* = 1$. In the first case there is no obstacle and it achieves the goal with minimal overshoot, but note the complex force profile applied to m_1. In the second case the load mass is stopped at $x_2 = 0.8$ and the elastic force changes to accomodate this.

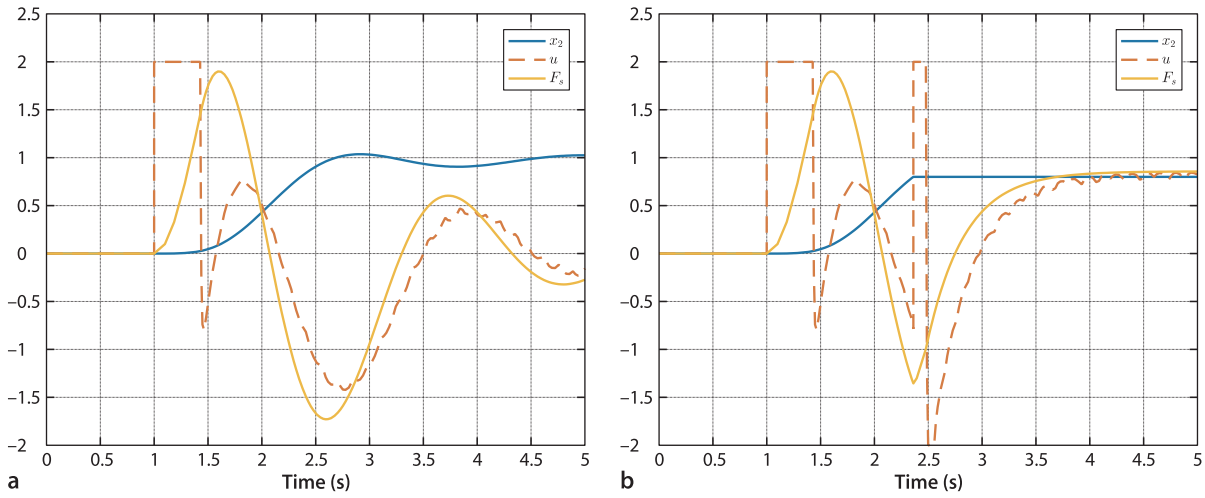


Fig. 9.26. Response of the series-elastic actuator to a unit-step demand at $t = 1$ s, showing load position (m), motor force (N) and spring force (N). **a** Moving to $x_2^* = 1$ with no collision; **b** moving to $x_2^* = 1$ with an obstacle at $x_2 = 0.8$ which is reached at $t \approx 2.3$

9.6 Wrapping Up

In this Chapter we discussed approaches to robot manipulator control. We started with the simplest case of independent joint control, and explored the effect of disturbance torques and variation in inertia, and showed how feedforward of disturbances such as gravity could provide significant improvement in performance. We then learned how to model the forces and torques acting on the individual links of a serial-link manipulator. The equations of motion or inverse dynamics compute the joint forces required to achieve particular joint velocity and acceleration. The equations have terms corresponding to inertia, gravity, velocity coupling, friction and externally applied forces. We looked at the significance of these terms and how they vary with manipulator configuration and payload. The equations of motion provide insight into important issues such as how the velocity or acceleration of one joint exerts a disturbance force on other joints which is important for control design. We then discussed the forward dynamics which describe how the configuration evolves with time in response to forces and torques applied at the joints by the actuators and by external forces such as gravity. We extended the feedforward notion to full model-based control using torque feedforward, computed torque and operational-space controllers. Finally we discussed series-elastic actuators where a compliant element between the robot motor and the link enables force control and people-safe operation.

Further Reading

The engineering design of motor control systems is covered in mechatronics textbooks such as Bolton (2015). The dynamics of serial-link manipulators is well covered by all the standard robotics textbooks such as Paul (1981), Spong et al. (2006), Siciliano et al. (2009) and the Robotics Handbook (Siciliano and Khatib 2016). The efficient recursive Newton-Euler method we use today is the culmination of much research in the early 1980s and described in Hollerbach (1982). The equations of motion can be derived via a number of techniques, including Lagrangian (energy based), Newton-Euler, d'Alembert (Fu et al. 1987; Lee et al. 1983) or Kane's method (Kane and Levinson 1983). However the computational cost of Lagrangian methods (Uicker 1965; Kahn 1969) is enormous, $O(N^4)$, which made it infeasible for real-time use on computers of that era and many simplifications and approximation had to be made. Orin et al. (1979) proposed an alternative approach based on the Newton-Euler (NE) equations of rigid-body motion applied to each link. Armstrong (1979) then showed how recursion could be applied resulting in

$O(N)$ complexity. Luh et al. (1980) provided a recursive formulation of the Newton-Euler equations with linear and angular velocities referred to link coordinate frames which resulted in a thousand-fold improvement in execution time making it practical to implement in real-time. Hollerbach (1980) showed how recursion could be applied to the Lagrangian form, and reduced the computation to within a factor of 3 of the recursive NE form, and Silver (1982) showed the equivalence of the recursive Lagrangian and Newton-Euler forms, and that the difference in efficiency was due to the representation of angular velocity.

The forward dynamics, Sect. 9.3, is computationally more expensive. An $O(N^3)$ method was proposed by Walker and Orin (1982) and is used in the Toolbox. Featherstone's (1987) articulated-body method has $O(N)$ complexity but for $N < 9$ is more expensive than Walker's method.

Critical to any consideration of robot dynamics is knowledge of the inertial parameters, ten per link, as well as the motor's parameters. Corke and Armstrong-Hélouvry (1994, 1995) published a meta-study of Puma parameters and provide a consensus estimate of inertial and motor parameters for the Puma 560 robot. Some of this data was obtained by painstaking disassembly of the robot and determining the mass and dimensions of the components. Inertia of components can be estimated from mass and dimensions by assuming mass distribution, or it can be measured using a bifilar pendulum as discussed in Armstrong et al. (1986).

Alternatively the parameters can be estimated by measuring the joint torques or the base reaction force and moment as the robot moves. A number of early works in this area include Mayeda et al. (1990), Izaguirre and Paul (1985), Khalil and Dombre (2002) and a more recent summary is Siciliano and Khatib (2016, § 6). Key to successful identification is that the robot moves in a way that is sufficiently exciting (Gautier and Khalil 1992; Armstrong 1989). Friction is an important dynamic characteristic and is well described in Armstrong's (1988) thesis. The survey by Armstrong-Hélouvry et al. (1994) is a very readable and thorough treatment of friction modeling and control. Motor parameters can be obtained directly from the manufacturer's data sheet or determined experimentally, without having to remove the motor from the robot, as described by Corke (1996a). The parameters used in the Toolbox Puma model are the best estimates from Corke and Armstrong-Hélouvry (1995) and Corke (1996a).

The discussion on control has been quite brief and has strongly emphasized the advantages of feedforward control. Robot joint control techniques are well covered by Spong et al. (2006), Craig (2005) and Siciliano et al. (2009) and summarized in Siciliano and Khatib (2016, § 8). Siciliano et al. have a good discussion of actuators and sensors as does the, now quite old, book by Klafter et al. (1989). The control of flexible joint robots is discussed in Spong et al. (2006). Adaptive control can be used to accomodate the time-varying inertial parameters and there is a large literature on this topic but some good early references include the book by Craig (1987) and key papers include Craig et al. (1987), Spong (1989), Middleton and Goodwin (1988) and Ortega and Spong (1989). The operational-space control structure was proposed in Khatib (1987). There has been considerable recent interest in series-elastic as well as variable stiffness actuators (VSA) whose position and stiffness can be independently controlled much like our own muscles – a good collection of articles on this technology can be found in the special issue by Vanderborght et al. (2008).

Dynamic manipulability is discussed in Spong et al. (2006) and Siciliano et al. (2009). The Asada measure used in the Toolbox is described in Asada (1983).

Historical and general. Newton's second law is described in his master work *Principia Nautralis* (mathematical principles of natural philosophy), written in Latin but an English translation is available on line at http://www.archive.org/details/newton-spmathema00newtrich. His writing on other subjects, including transcripts of his notebooks, can be found online at http://www.newtonproject.sussex.ac.uk.

Exercises

1. Independent joint control (page 258ff)
   a) Investigate different values of `Kv` and `Ki` as well as demand signal shape and amplitude.
   b) Perform a root-locus analysis of `vloop` to determine the maximum permissible gain for the proportional case. Repeat this for the PI case.
   c) Consider that the motor is controlled by a voltage source instead of a current source, and that the motor's impedance is 1 mH and 1.6 Ω. Modify `vloop` accordingly. Extend the model to include the effect of back EMF.
   d) Increase the required speed of motion so that the motor torque becomes saturated. With integral action you will observe a phenomena known as integral windup – examine what happens to the state of the integrator during the motion. Various strategies are employed to combat this, such as limiting the maximum value of the integrator, or only allowing integral action when the motor is close to its setpoint. Experiment with some of these.
   e) Create a Simulink model of the Puma robot with each joint controlled by `vloop` and `ploop`. Parameters for the different motors in the Puma are described in Corke and Armstrong-Hélouvry (1995).
2. The motor torque constant has units of N m A^{-1} and is equal to the back EMF constant which has units of V s rad^{-1}. Show that these units are equivalent.
3. Simple two-link robot arm of Fig. 9.4
   a) Plot the gravity load as a function of both joint angles. Assume $m_1 = 0.45$ kg, $m_2 = 0.35$ kg, $r_1 = 8$ cm and $r_2 = 8$ cm.
   b) Plot the inertia for joint 1 as a function of q_2. To compute link inertia assume that we can model the link as a point mass located at the center of mass.
4. Run the code on page 265 to compute gravity loading on joints 2 and 3 as a function of configuration. Add a payload and repeat.
5. Run the code on page 266 to show how the inertia of joints 1 and 2 vary with payload?
6. Generate the curve of Fig. 9.16c. Add a payload and compare the results.
7. By what factor does this inertia vary over the joint angle range?
8. Why is the manipulator inertia matrix symmetric?
9. The robot exerts a wrench on the base as it moves (page 269). Consider that the robot is sitting on a frictionless horizontal table (say on a large air puck). Create a simulation model that includes the robot arm dynamics and the sliding dynamics on the table. Show that moving the arm causes the robot to translate and spin. Can you devise an arm motion that moves the robot base from one position to another and stops?
10. Overlay the dynamic manipulability ellipsoid on the display of the robot. Compare this with the force ellipsoid from Sect. 8.5.2.
11. Model-based control (page 273ff)
    a) Compute and display the joint tracking error for the torque feedforward and computed torque cases. Experiment with different motions, control parameters and sample rate T_{fb}.
    b) Reduce the rate at which the feedforward torque is computed and observe its effect on tracking error.
    c) In practice the dynamic model of the robot is not exactly known, we can only invert our best estimate of the rigid-body dynamics. In simulation we can model this by using the `perturb` method, see the online documentation, which returns a robot object with inertial parameters varied by plus and minus the specified percentage. Modify the Simulink models so that the `RNE` block is using a robot model with parameters perturbed by 10%. This means that the inverse dynamics are computed for a slightly different dynamic model to the robot under control and shows the effect of model error on control performance. Investigate the effects on error for both the torque feedforward and computed torque cases.

d) Expand the operational-space control example to include a sensor that measures all the forces and torques exerted by the robot.on an inclined table surface. Move the robot end-effector along a circular path in the xy-plane while exerting a constant downward force – the end-effector should move up and down as it traces out the circle. Show how the controller allows the robot tool to conform to a surface with unknown height and surface orientation.

12. Series-elastic actuator (page 276)
    a) Experiment with different values of stiffness for the elastic element and control parameters. Try to reduce the settling time.
    b) Modify the simulation so that the robot arm moves to touch an object at unknown distance and applies a force of 5 N to it.
    c) Plot the frequency response function $X_2(s)/X_1(s)$ for different values of K_s, m_1 and m_2.
    d) Simulate the effect of a collision between the load and an obstacle by adding a step to the spring force.