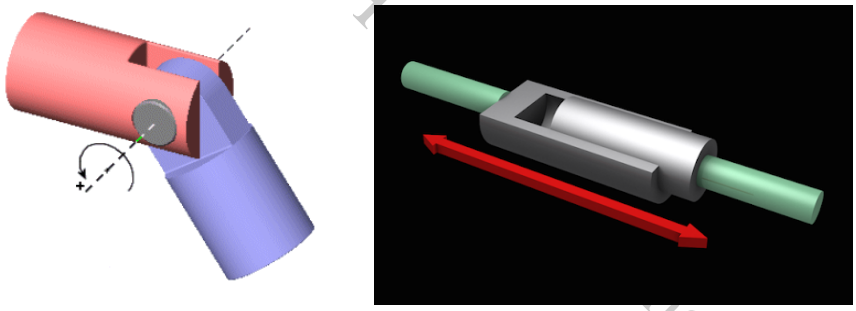
Contents

1 Serial Link Manipulators

1.1 Basic Definitions



(a) Revolute joint. (b) Prismatic joint.

Figure 1.1: Two main joint types in robot arms.

A **robot arm** (aka **serial link manipulator**) consists of a series of rigid links, connected by joints (motors), each of which has a single degree of freedom.:

- **Revolute** joint: the single degree of freedom is rotation about an axis.
- **Prismatic** joint: the single degree of freedom is translation along an axis.

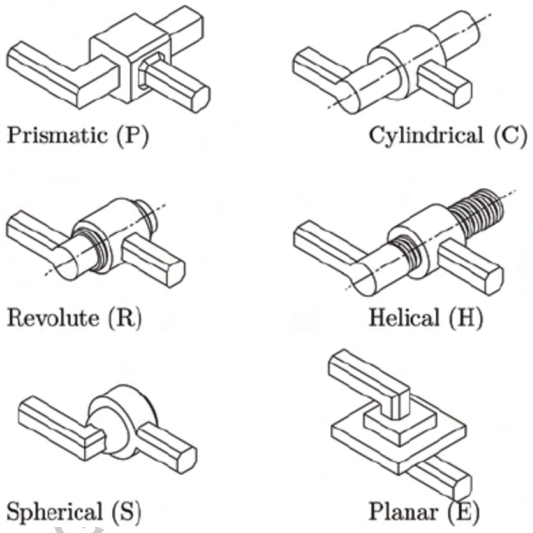


Figure 1.2: Other types of joints.

More complex joints can be described as combinations of these basic joints. There are several types of joint that have more than one degree of freedom – but we do not consider those in detail. In fact, all of the higher degree-of-freedom joints can be described by combinations of one degree-of-freedom joints, so there is no need to explicitly consider these.

A serial link manipulator has several **links**, numbered 0 to n, connected by **joints**, numbered 1 to n. Joint i connects link $i-1$ to link i. We will consider revolute joints with **joint angle** θ_i, or a prismatic joints with **link offset** d_i.

In summary, for for a robot with n joints:

- the base (which does not move) is Link 0;
- the end-effector or tool is attached to Link n;
- joint i connects Link i–1 to Link i.

We can treat both revolute and prismatic joins uniformly by introducing the concept of a **generalized joint coordinate** q_i, and specifying the joint type using a string, e.g., the classical Puma robot is RRRRRR, and the SCARA pick and place robot is RRRP. The vector $q \in Q$ of these generalized joint coordinates is also called the **pose** of the manipulator, where Q is called the **joint space** of the manipulator. In

summary, we define the joint variable q_i for joint i as:

$$\begin{cases} \theta_i & \text{if joint is revolute} \\ d_i & \text{if joint is prismatic} \end{cases}$$

1.2 An RRR Example

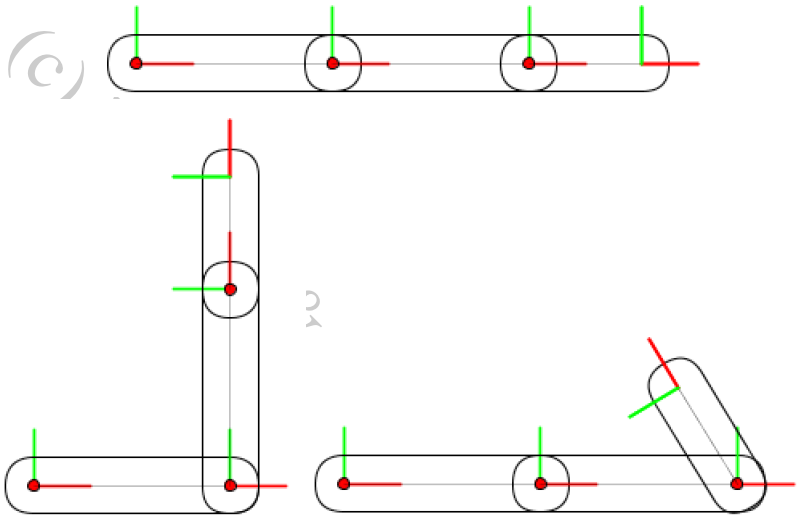


Figure 1.3: Top: rest state of a planar RRR serial manipulator, with the base frame on left. Bottom: actuating two degrees of freedom, respectively rotating θ_2 and θ_3.

All essential concepts can be easily developed for 2D or **planar manipulators** with revolute joints only. An example is shown in Figure 1.3. The top panel shows the manipulator at rest, along with four 2D coordinate frames: the base frame and one coordinate frame for each of the three links. For this RRR manipulator, the generalized joint coordinates are $q = \begin{bmatrix} \theta_1 & \theta_2 & \theta_3 \end{bmatrix}^T$, and the effect of changing individual joint angles θ_i is shown at the bottom of the figure.

1.3 Forward Kinematics

Kinematics describes the position and motion of a robot, without considering the forces required to cause the motion. Key questions we will answer are how

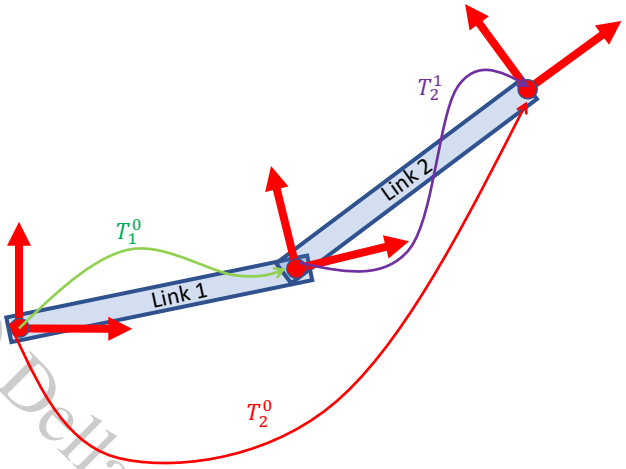


Figure 1.4: Transformations involved in the forward kinematics of a 2-link arm.

to determine the pose of the end-effector tool, given joint angles, and the reverse question: what joint angles should we command to get the tool to be at a desired pose? We start with the first question, which is known as forward kinematics.

The **forward kinematics problem** can be stated as follows:

> *Given generalized joint coordinates $q \in Q$, we wish to determine the pose $T_t^0(q)$ of the tool frame T relative to the base frame 0.*

The general approach we will take is this:

- we treat each Link j as a rigid body;
- we attach the reference coordinate frame 0 to Link 0, which is merely the fixed base;
- we describe the pose (position and orientation) of every other Link j by attaching a coordinate frame $T_i^0(q)$, expressed to the base frame 0;
- if two links, say link j−1 and link j are connected by a single joint, then the relationship between the two frames can be described by a homogeneous transformation matrix $T_i^{i-1}(q_i)$ which will depend only on the value of the joint variable q_i! Now, the trick is to express $T_i^{i-1}(q_i)$ as a function of q_i.

Before we do that, let us consider the example of a 2-link arm as shown in Figure 1.4. In the figure, the transform T_1^0 specifies the pose of Link 1 with respect to the base. We attached frame 1 at the end of Link 1, but it could be anywhere you like.

In turn, the transform T_2^1 specifies pose of Link 2 *with respect to Link* 1. To obtain the pose of Link 2 with respect to the base frame, we multiply both transforms, using simple matrix multiplication:

$$T_2^0 = T_1^0 T_2^1$$

We now generalize this to n links and an arbitrary end-effector or **Tool frame** in Link n. Since the tool frame T moves with link n, we have

$$T_t^0(q) = T_n^0(q) T_t^n$$

where T_t^n specifies the unchanging pose of the tool T in the frame of link n. Sometimes the tool frame is taken to be identical to frame n, and then X_t^n is simply the identity matrix. The link coordinate frame $T_n^0(q)$ itself can be expressed recursively as

$$T_n^0(q) = T_{n-1}^0(q_1 \dots q_{n-1}) T_n^{n-1}(q_n),$$

finally yielding

$$T_t^0(q) = T_1^0(q_1) \dots T_i^{i-1}(q_i) \dots T_n^{n-1}(q_n) T_t^n. \tag{1.1}$$

Exercise

Draw a simple two-link RP manipulator and provide the above forward kinematics formula.

1.4 Describing Serial Manipulators

Equation 1.1 is correct but we need to tie it to the robot's geometry. We can make everything easy by adopting the following strategy:

- We take frame 0 (the base frame) to have its origin at the center of Joint 1, i.e., on the axis of rotation.
- We rigidly attach Frame i is to Link i, in such a way that it has its origin at the center of Joint $i + 1$.
- The x_i-axis is chosen to be collinear with the origin of Frame $i - 1$.
- Define the **link length** a_i as the distance between the origins of Frames i and $i - 1$.

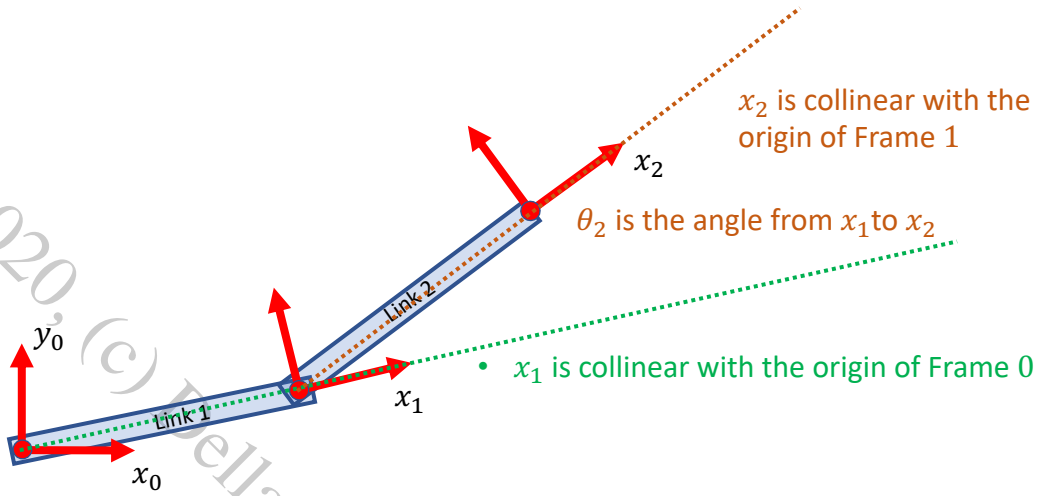


Figure 1.5: Example of assigning link frames for a 2-link arm.

This is illustrated for our 2-link RR-arm in Figure 1.5.

If we do all this, then the homogeneous transformation that relates adjacent frames is given by (for revolute joints):

$$T_i^{i-1} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i & a_i\sin\theta_i \\ 0 & 0 & 1 \end{bmatrix}$$

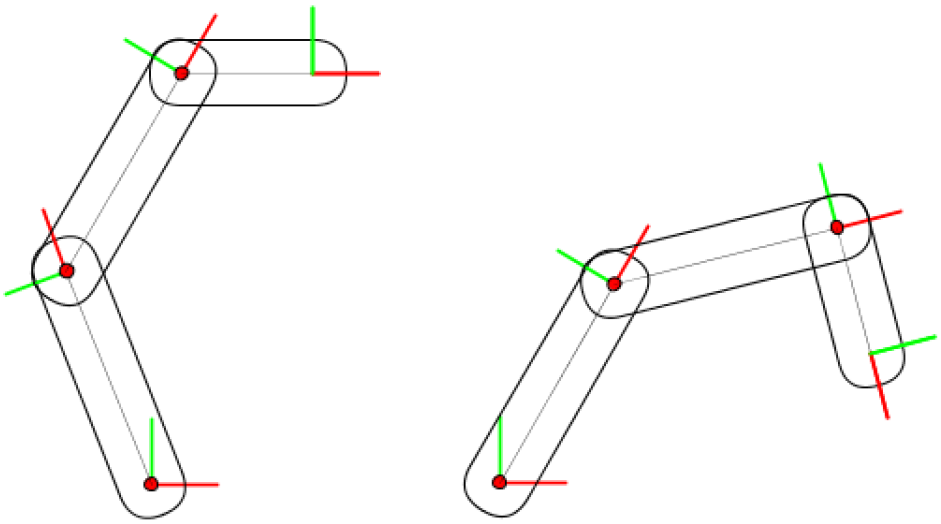
1.5 RRR Forward Kinematics: Worked Example

As an example, Figure 1.6 shows a planar RRR manipulator with $a_1 = 3.5$, $a_2 = 3.5$, and $a_3 = 2$, in two different joint configurations. We identified the tool frame with link frame 3, i.e., $X_t^3 = I$. We then have:

$$T_1^0 = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 3.5\cos\theta_1 \\ \sin\theta_1 & \cos\theta_1 & 3.5\sin\theta_1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T_2^1 = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 3.5\cos\theta_2 \\ \sin\theta_2 & \cos\theta_2 & 3.5\sin\theta_2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T_3^2 = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 2\cos\theta_3 \\ \sin\theta_3 & \cos\theta_3 & 2\sin\theta_3 \\ 0 & 0 & 1 \end{bmatrix}$$



(a) $\theta_1 = 112°$, $\theta_2 = -52°$, and $\theta_3 = -60°$ (b) $\theta_1 = 60°$, $\theta_2 = -45°$, and $\theta_3 = -90°$

Figure 1.6: Two example configurations for a planar RRR manipulator with all three joints actuated.

When multiplied out, we obtain

$$T_t^s(q) = \begin{pmatrix} \cos\beta & -\sin\beta & 3.5\cos\theta_1 + 3.5\cos\alpha + 2\cos\beta \\ \sin\beta & \cos\beta & 3.5\sin\theta_1 + 3.5\sin\alpha + 2\sin\beta \\ 0 & 0 & 1 \end{pmatrix} \tag{1.2}$$

with $\alpha = \theta_1 + \theta_2$ and $\beta = \theta_1 + \theta_2 + \theta_3$, the latter being the tool orientation.

Exercise

Provide the multiplied-out forward kinematics formula for your RP manipulator.

1.6 Joint-space Motion Control

Trajectory following is an important capability for manipulator robots, and three main approaches are common: (a) trajectory replay, (b) joint space motion control, and (c) cartesian space motion control.

Trajectory replay relies on an operator to perform the motion first, after which the robot simply replays the sequence, and is akin to motion-capture in movies. Even then, to interpolate between **waypoints** obtained by robot programming, one of the two other methods is needed.

Another scenario where motion control is useful is when we just want to move the end-effector T_t^0 to a particular pose. *Inverse kinematics*, which we will discuss in detail in a future section, allows us to calculate the joint angles q_d for a particular desired pose.

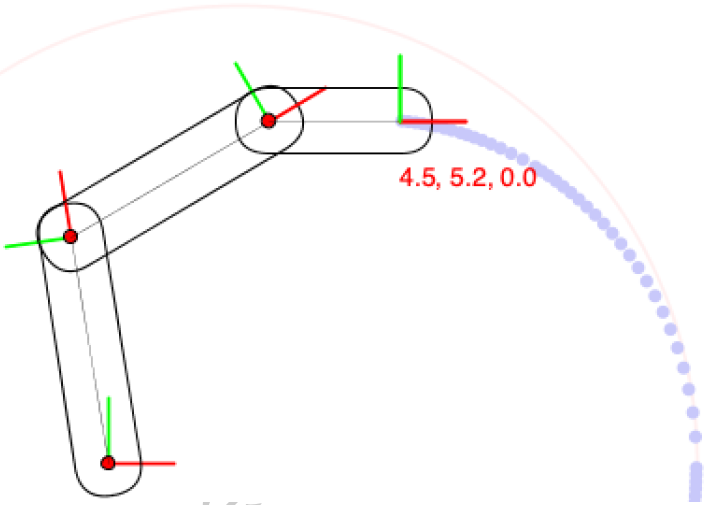


Figure 1.7: In joint-space motion control, the joint angles are linearly interpolated, but this leads to a curved path in Cartesian space.

Joint space motion control is the easiest, and applies linear interpolation or a simple control law in joint space to move from one waypoint to the other, e.g.,

$$q_{t+1} = q_t + K_p(q_d - q_t) \tag{1.3}$$

where q_t and q_d are the current and desired joint angles, and $K_p > 0$ is a gain parameter.

Before fully understanding Equation 1.3, let us consider an example of proportional joint space control in action, as shown in Figure 1.7 for the three-link manipulator. The robot started off in rest, with the end-effector at the bottom-right. The desired end-effector pose T_t^0 in this case is $(4.5, 5.2, 0.0)$ as shown in the figure. Using inverse kinematics the corresponding desired joint angle vector q_d was determined to be $q_d = [99.2, -68.8, -30.4]$, and the figure shows how proportional feedback control executes a curved trajectory upward to achieve the desired pose. The "trail" shows that the movement of the end-effector slows down considerably near the desired goal pose.

Now let us dive into what is happening in Equation 1.3: at any given moment t we calculate the **joint space error** $e_t = q_d - q_t$. Pretending for a moment that q

is scalar, then if the error e_t is positive we need to increase q_t to drive the error to zero. We can use a fixed amount Δq, but a better approach is to make the amount by which we increase q_t *proportional* to the error e_t, i.e.,

$$\Delta q = K_p e_t,$$

where K_p is the proportional gain parameter. The entire scheme is called **proportional feedback control**, and has two advantages:

- as the desired value q_d is approached, the adjustment Δq becomes smaller and smaller, slowly approaching q_d so it avoids overshooting.
- it automatically deals with the case in which the error e_t is negative, in which case we then decrease q_t;
- it generalizes directly to multiple dimensions, as is needed for joint angles.

Note that setting the value of the gain K_p needs some care. If we set it too large, we might overshoot the goal, and if we set it too small, convergence to the goal will be very slow. If you are familiar with gradient descent optimization, then K_p plays a similar role as the learning rate.

Other control schemes are possible, and a very popular approach is **PID control**, where in addition to the **P**roportional gain, we also allow feedback terms calculated from the **I**ntegral of the error and **D**erivative of the error, respectively.

Discussion

Joint-space control, while very simple, might seem a sub-optimal way to proceed. Indeed, if we are at a particular pose and want to move to another, desired pose, why not make a beeline for the goal pose? In the **cartesian workspace**, i.e., the regular 2D or 3D space in which the end effector lives, the shortest path should be a line, right? The answer is not so straightforward:

1. If you are talking about the position of the end-effector, then the shortest path is indeed a straight line, in this case in 2D.
2. However, if you take into account the orientation of the end effector, the answer is not so clearcut anymore: what is the shortest path in the space $SE(2)$ of rigid transformations?
3. The shortest path *in joint space* is exactly what joint-space control executes. Only, a straight line in joint space causes a curved path in cartesian space.

In many cases, there is a desire to control the trajectory in cartesian space rather than joint space, however, which leads to the next topic.

1.7 The Manipulator Jacobian

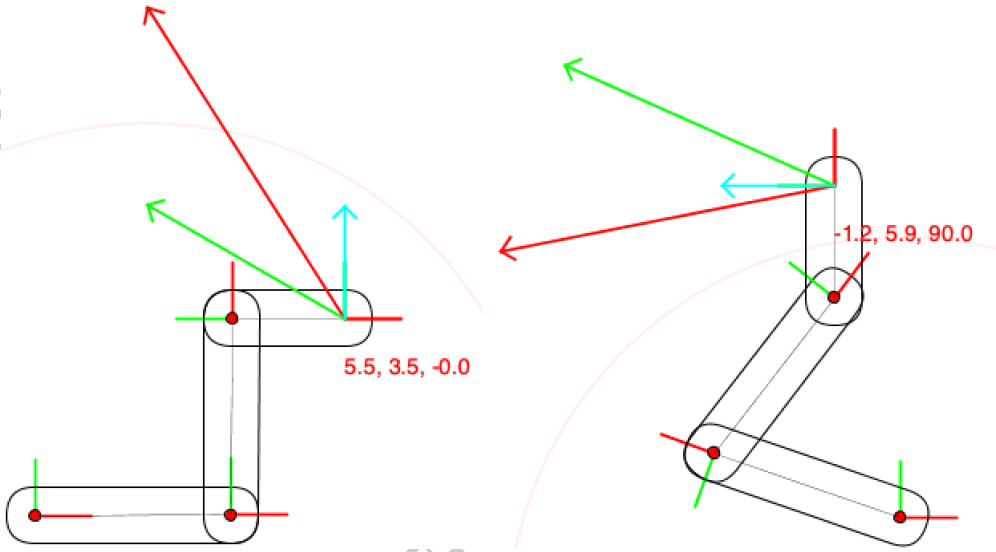


Figure 1.8: The velocities induced by a change in joint angle (red=θ_1, green=θ_2, blue=θ_3), for joint angles $q_{left} = (0°, 90°, 0°)$ and $q_{right} = (161°, -109°, 38°)$.

Because we eventually do need to control the joint angles q, the key is to derive a relationship between velocities $[\dot{x}, \dot{y}, \dot{\theta}]^T$ in pose space in response to commanded velocities in joint space $\dot{q}$. This relationship is *locally* linear, and hence we have the following expression at a given configuration q:

$$[\dot{x}, \dot{y}, \dot{\theta}]^T = J(q)\dot{q} \tag{1.4}$$

The quantity $J(q)$ above is the **manipulator Jacobian**. For planar manipulators, as $[\dot{x}, \dot{y}, \dot{\theta}]^T \in \mathbb{R}^3$, the Jacobian is a $3 \times n$ matrix, with n is the number of joints. Each column of the Jacobian $J(q)$ contains the velocity $[\dot{x}, \dot{y}, \dot{\theta}]^T \in \mathbb{R}^3$ corresponding a change in the joint angle q_i only, i.e.,

$$J(q) \triangleq \begin{bmatrix} J_1(q) & J_2(q) & \dots & J_n(q) \end{bmatrix}$$

where each column $J_i(q)$ is the vector of (three) partial derivatives of the pose with respect to joint angle q_i:

$$J_i(q) \triangleq \begin{bmatrix} \frac{\partial x(q)}{\partial q_i} \\ \frac{\partial y(q)}{\partial q_i} \\ \frac{\partial \theta(q)}{\partial q_i} \end{bmatrix}.$$

A graphical way to appreciate what a Jacobian means physically is to draw the 2D velocities in Cartesian space. For the three-link planar manipulator example, Figure 1.8 above shows the Jacobian $J(q)$ as a set of three velocities: red for joint 1, green for joint 2, and blue for joint 3. In other words, the red vector shows the instantaneous velocity vector for the origin of the end effector frame when joint velocity $\dot{\theta}_1 = 1$ and $\dot{\theta}_2 = \dot{\theta}_3 = 0$. These instantaneous velocities depend on the current joint angles q. The pattern is clear: these velocities are always perpendicular to the vector to the joint axis, and proportional to the distance to the joint axis.

Exercise

The Jacobian $J(q)$ varies with the configuration q. Is that always the case? Come up with a robot that has a constant Jacobian.

1.8 Cartesian Motion Control using the Inverse Jacobian

Cartesian motion control is a method that allows us to follow a well-defined path in Cartesian space, most often a straight line or some interpolating spline. One approach is to calculate an inverse kinematics solution at many intermediate waypoints and apply joint control again, to get from one to the other. However, there is a method by which we can avoid inverse kinematics altogether.

For a planar manipulator with three joints, i.e., $n = 3$, we can simply invert the 3×3 Jacobian $J(q)$ to calculate the joint space velocities $\dot{q}$ corresponding to a given end-effector velocity $[\dot{x}, \dot{y}, \dot{\theta}]^T$:

$$\dot{q} = J(q)^{-1}[\dot{x}, \dot{y}, \dot{\theta}]^T \tag{1.5}$$

Hence, to achieve a desired trajectory in Cartesian space, we need to calculate the desired direction *in cartesian space* at any given time. If our desired end effector pose is T_d and the current pose is $T(q_t)$ we can do this via

$$E_t(q) = \begin{bmatrix} e_x \\ e_y \\ e_\theta \end{bmatrix} = \begin{bmatrix} x_d - x(q_t) \\ y_d - y(q_t) \\ \theta_d \ominus \theta(q_t) \end{bmatrix}$$

where we extract $[x_d, y_d, \theta_d]^T$ from T_d and $[x(q_t), y(q_t), \theta(q_t)]^T$ from $T(q_t)$, respectively. Note that there are some subtleties around this: extracting a value for the orientation θ is not unique, because e.g., $\cos(\theta) = \cos(\theta + 2\pi)$. In addition, subtracting two θ values is fraught with danger because of this very same issue. To this end, we introduce the notation $\theta_d \ominus \theta(q_t)$ above to signify a "subtraction" that

yields the smallest angle between two different orientations, in absolute value. The resulting magnitude $|\theta_d \ominus \theta(q_t)|$ should always be less than or equal to π.

We then calculate the corresponding joint velocities $\dot{q}$ using (1.5), and apply simple proportional control, i.e.,

$$q_{t+1} = q_t + K_p J(q_t)^{-1} E_t(q). \tag{1.6}$$

What is going on in Equation 1.6? At any given moment, the cartesian or **workspace error** $E_t(q_t)$ is evaluated, and this is a three-dimensional error in x, y, and θ. This error gives us a direction in cartesian space that we want to move, and by multiplying it the inverse Jacobian we turn this into an error in joint space. But we already know how to resolve errors in joint space! Using a proportional gain K_p, we make progress in that joint space direction, and we repeat. The secret ingredient is that unlike in pure joint-space control, we re-compute the direction in joint space at each time t to obtain a straight-line trajectory in the workspace.

1.9 RRR Jacobian and Cartesian Control: Worked Example

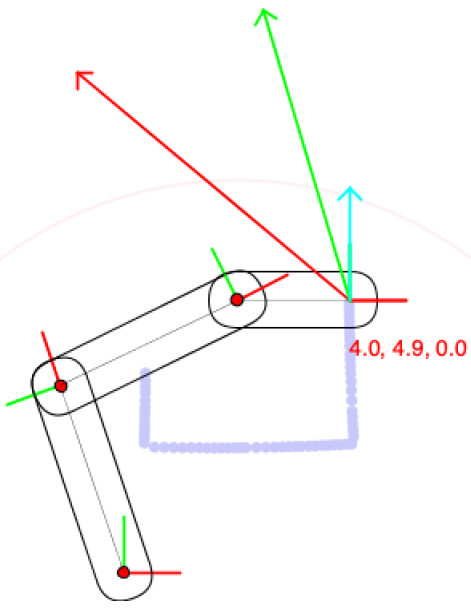


Figure 1.9: Cartesian space motion control, showing the resulting straight trajectories of the end-effector for three successive waypoints.

Let us calculate the Jacobian $J(q)$ for the three-link planar manipulator example. To analytically compute the Jacobian in this case, we can read off the pose

$T(q)$ components from the forward kinematics equation 1.2 on page 7, yielding

$$\begin{bmatrix} x(q) \\ y(q) \\ \theta(q) \end{bmatrix} = \begin{bmatrix} 3.5\cos\theta_1 + 3.5\cos\alpha + 2\cos\beta \\ 3.5\sin\theta_1 + 3.5\sin\alpha + 2\sin\beta \\ \beta \end{bmatrix}$$

where $\alpha = \theta_1 + \theta_2$ and $\beta = \theta_1 + \theta_2 + \theta_3$. Hence, the 3×3 Jacobian $J(q)$ can be computed as

$$\begin{pmatrix} -3.5\sin\theta_1 - 3.5\sin\alpha - 2.5\sin\beta & -3.5\sin\alpha - 2.5\sin\beta & -2\sin\beta \\ 3.5\cos\theta_1 + 3.5\cos\alpha + 2.5\cos\beta & 3.5\cos\alpha + 2.5\cos\beta & 2\cos\beta \\ 1 & 1 & 1 \end{pmatrix} \tag{1.7}$$

Note that for a planar manipulator, all entries in the third row will always be 1 the way we defined things: the rotation rates of the joints can just be added up to obtain the rotation rate of the end effector.

An example of Cartesian motion control with proportional control is shown in Figure 1.9 for the three-link planar robot. As shown by the trail in the figure, the inverse Jacobian managed to follow straight lines in cartesian space.