Chapter 3

Kinematics

In order to plan a robot's movements, we have to understand the relationship between our control variables (i.e. the input to the motors that we can control at any given time) and the effect of these control variables on the motion of the robot. The simplest models of such relationships can be built by looking at the geometry of our robot, known as the field of *kinematics*. For simple arms in static configurations, a kinematic model is rather straightforward: if we know the generalized position / configuration angle of each joint, we can calculate the generalized position of its end-effectors using trigonometry—a process known as *forward kinematics*. This process is usually more involved for mobile robots, as the speeds of the wheels need to be integrated to determine changes in robot pose, which we refer to as *odometry*. Roboticists are often concerned with trying to compute the inverse relationship: the position each joint must be at for the end-effector to be in a desired position or pose. This is generally a far more complex, underdetermined problem, known as *inverse kinematics*.

As we will see below, kinematics is the simplest and most fundamental level of abstraction that a roboticist can use to model the motion of a robot and its geometry: it deals exclusively with positional quantities, and considers the robot as if it was frozen in time. Although this simplification is far from being realistic, we will see that a lot can be done through kinematics alone! However, a more expressive tool at our disposal is to do a similar modeling in a second level of abstraction that operates in velocity space; this domain is called *differential kinematics* and is introduced in Section 3.3. In all, the goals of this chapter are to:

- Introduce the forward kinematics of simple arms and mobile robots, and understand the concept of holonomy
- Provide an intuition on the relationship between inverse kinematics and path-planning
- Become familiar with differential kinematics and the Jacobian technique

Within the scope of a kinematic analysis, the term *generalized position* or *generalized configuration* means "any position-equivalent quantity needed to describe the element". For what concerns joint space, it depends on the type of actuation: a revolute joint imparts a rotational motion around its axis and its configuration is fully described by an angle; a prismatic joint commands a translational motion along its axis and its configuration is represented by a distance. Conversely, generalized position in task space depends on the specific task; in its most general case, a generalized position equates to the end-effector pose, which is comprised of a $3D$ position and a $3D$ orientation—as we will see below.

> Remember: configuration space $\equiv$ joint space; cartesian space $\equiv$ task space. Forward kinematics maps from joint space to task space, and inverse kinematics does the opposite. The number of mechanical degrees-of-freedom n (i.e. DoFs in task space) depends on the robot, while the number of Cartesian degrees-of-freedom m (i.e. DoFs in task space) depends on the task. In general, $n \neq m$!

3.1. Forward Kinematics

Now that we have introduced the notion of local coordinate frames, we are interested in calculating the pose and speed of these coordinate frames as a function of the robot's actuators and joint configuration. That is, we are interested in computing a function f that allows us to map a joint configuration to its corresponding end-effector pose:

$$r = f(q) \ , \qquad f : \mathbb{R}^n \to \mathbb{R}^m \ , \tag{3.1}$$

where r is the task-space (end-effector) configuration and q is the joint-space configuration. It is important to remember that the choice of q and

r (and, consequently, the complexity of f) depends on your specific robot platform and the specific task you are investigating. q generally refers to the actuators/joints that you can control on your robot; it is of size n, where n is the number of degrees of freedom in joint space (see also Section 2.3). Conversely, r depends on the task and its dimensionality is m, where m is the number of DoFs in task space.

We will focus on the problem of computing the forward kinematics mapping f for a variety of robot arms to build intuition. While it is always possible to compute the forward kinematics *analytically* (i.e. by inspecting the arm mechanism and the relationship between joint and task configuration, see Section 3.1.1), in Section 3.1.2 we will introduce a scalable, *geometric* technique to compute forward kinematics with more complex arms composed of many mechanical Degrees of Freedom.

3.1.1. Forward Kinematics of a simple robot arm

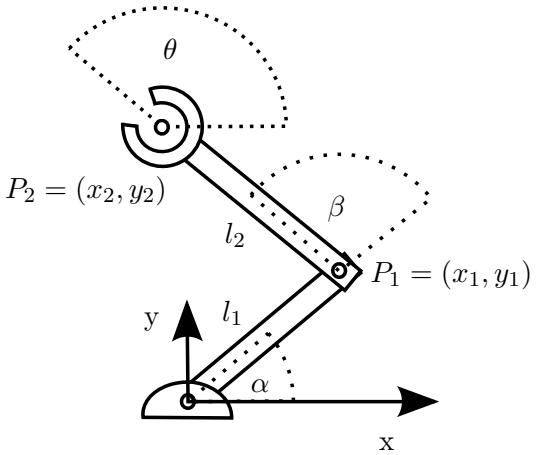

Figure 3.1. A simple 2-DOF arm.

Consider the robot arm in Figure 3.1; it is mounted to a table, and is composed of two links and two joints. Let the length of the first link be l_1 and the length of the second link be l_2. You could specify the position of the link closer to the table by the angle α and the angle of the second link relative to the first link using the angle β. Therefore, $q = [\alpha, \beta]^T$ specifies the two degrees of freedom that we can control. Our goal is to calculate the position $[x, y]^T$ and orientation θ of the end-effector given the values of q;

consequently, f will map to $r = [x, y, \theta]^T$.

Let's now calculate the position $P_1 = (x_1, y_1)$ of the joint between the first and the second link using simple trigonometry:

$$\begin{aligned} x_1 &= l_1 \cos\alpha \\ y_1 &= l_1 \sin\alpha \end{aligned} \tag{3.2}$$

Similarly, the position of the end-effector $P_2 = (x_2, y_2)$ is given by:

$$\begin{aligned} x_2 &= x_1 + l_2 \cos(\alpha + \beta) \\ y_2 &= y_1 + l_2 \sin(\alpha + \beta) \end{aligned} \tag{3.3}$$

For what concerns the orientation of the arm's end-effector θ, we know it is just the sum of $\alpha + \beta$. Altogether, the configuration r of the end-effector is given by:

$$\begin{aligned} x &= l_1 \cos\alpha + l_2 \cos(\alpha + \beta) \\ y &= l_1 \sin\alpha + l_2 \sin(\alpha + \beta) \\ \theta &= \alpha + \beta \end{aligned} \tag{3.4}$$

The above equations represent *the forward kinematic equations* of the robot—as they relate its control parameters α and β (also known as joint configuration) to the pose of its end-effector in the local coordinate system spanned by x and y with the origin at the robot's base. Note that both α and β shown in the figure are positive: both links rotate around the $z-$axis. Using the right-hand rule, the direction of positive angles is defined to be counter-clockwise.

The *configuration space* of the robot—i.e. the set of angles each actuator can be set to—is given by $0 < \alpha < \pi$ as it is not supposed to run into the table, and $-\pi < \beta < \pi$. The configuration space is defined with respect to the robot's joints and allows us to use the forward kinematics equations to calculate the *workspace* of the robot, i.e. the physical space it can move to. This terminology will be identical for mobile robots. An example of configuration and workspace for both a manipulator and a mobile robot is shown in Figure 3.4.

We can now write down a transformation that includes a rotation around

the z-axis:

$$f(q) = \begin{bmatrix} c_{\alpha\beta} & -s_{\alpha\beta} & 0 & c_{\alpha\beta}l_2 + c_{\alpha}l_1 \\ s_{\alpha\beta} & c_{\alpha\beta} & 0 & s_{\alpha\beta}l_2 + s_{\alpha}l_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

The notation $s_{\alpha\beta}$ and $c_{\alpha\beta}$ are shorthand for $sin(\alpha + \beta)$ and $cos(\alpha + \beta)$, respectively. This transformation now allows us to transform from the robot's base to the robot's end-effector configuration $r = [x, y, \theta]^T$ as a function of the joint configuration $q = [\alpha, \beta]^T$. This transformation will be helpful if we want to calculate suitable joint angles in order to reach a certain pose (i.e., inverse kinematics) or if we want to convert measurements taken relative to the end-effector back into the base's coordinate system (e.g., when we have sensors mounted on the end effector whose output needs to be mapped back to the world reference frame).

3.1.2. The Denavit-Hartenberg notation

So far, we have considered the forward kinematics of a simple arm and derived relationships between actuator parameters and end-effector positions using basic trigonometry. In the case of multi-link arms (the vast majority of robot manipulators in existence), the approach detailed in Section 3.1.1 is difficult to scale, and alternative solutions are needed. Interestingly, we can think of the forward kinematics as a chain of homogeneous transformations with respect to a coordinate system mounted at the base of a manipulator (or a fixed position in the room). Deriving these transformations can be confusing and can be facilitated by following a "recipe" such as the one conceived by Denavit and Hartenberg in 1955 (see (Hartenberg & Denavit 1955) and (Craig 2009)). The so-called Denavit-Hartenberg (DH) representation has since evolved as a *de-facto* standard.

A manipulator consists of a series of typically rigid links that are connected by joints. In the vast majority of cases, a joint can either be revolute (i.e. change its angle/orientation) or prismatic (i.e. change its length). Knowing the robot's kinematic properties (e.g. the length of all rigid links, similarly to l_1 and l_2 in Figure 3.1), the pose of its end-effector is fully described by its joint configuration (angle for revolute joints, length for prismatic joints).

In order to use the DH convention, we first need to define a coordinate system at each joint. With reference to Figure 3.2, we choose the $z-$axis to be the axis of rotation for a revolute joint and the axis of translation for a

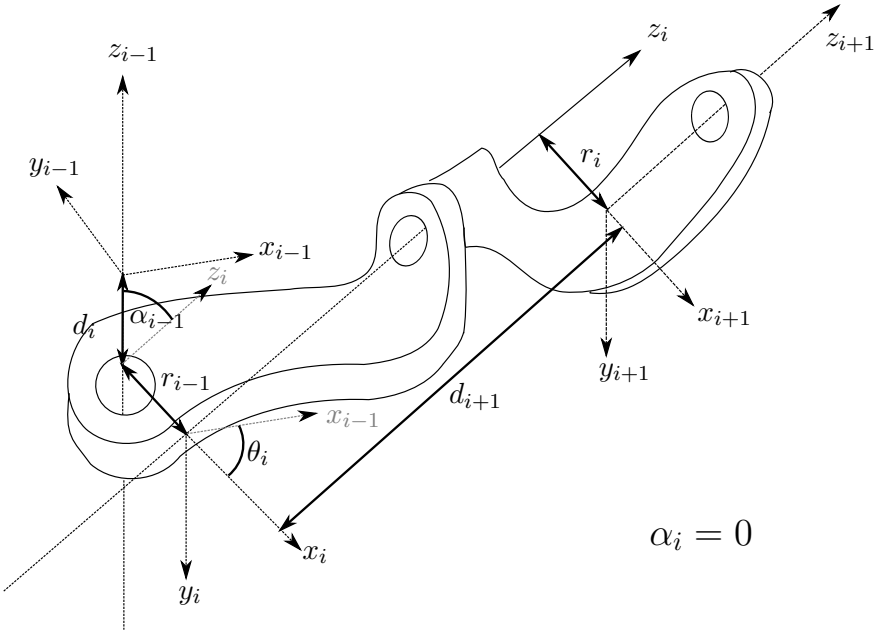

Figure 3.2. Example of selected Denavit-Hartenberg parameters for three sequential revolute joints. The z-axes of joint i and $i+1$ are parallel, which results in $\alpha_i = 0$.

prismatic joint. We can now find the common normal between the $z-$axes of two subsequent joints, i.e. a line that is orthogonal to each $z-$axis and intersects both. While the direction of the $x-$axis at the base can be chosen arbitrarily, subsequent $x-$axes are chosen such that they lie on the common normal shared between two joints. Whereas the direction of the $z-$axis is given by the positive direction of rotation (right-hand rule), the $x-$axis points away from the previous joint. This allows defining the $y-$axis using the right-hand rule. Note that these rules, in particular the requirement that x-axes lie along the common normal, might result in coordinate systems with their origins outside the joint. This is not problematic as the kinematics of a manipulator is a mathematical representation that need only represent the geometric and kinematic properties of the robot, and does not need to bear any physical correspondence to the system. The transformation between two joints is then fully described by the following four parameters:

1. The length r (sometimes, a is used) of the common normal between the z-axes of two joints i and $i-1$ (link length).

2. The angle α between the z-axes of the two joints with respect to the common normal (link twist), i.e. the angle between the old and the new z-axis, measured about the common normal.

3. The distance d between the joint axes (link offset), i.e. the offset along the previous z-axis to the common normal.

4. The rotation θ around the common axis along which the link offset is measured (joint angle), i.e. the angle from the old x-axis to the new x-axis, about the previous z-axis.

Two of the above D-H parameters describe the link between the joints, and the other two describe the link's connection to a neighboring link. Depending on the link/joint type, these numbers are fixed by the specific mechanical instance of the robot or can be controlled. For example, in a revolute joint θ is the varying joint angle, while all other quantities are fixed. Similarly, for a prismatic joint d is the joint variable. An example of two revolute joints is shown in Figure 3.2.

The final coordinate transform from one link $(i-1)$ to another (i) can now be constructed by concatenating the four steps above, which are nothing but a series of rotations and translations, one for each DH parameter:

$$ {}^{n}_{n-1}T = T'_z(d_n)R'_z(\theta_n)T_x(r_n)R_x(\alpha_n) \tag{3.6} $$

with

$$ T'_z(d_n) = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_n \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \quad R'_z(\theta_n) = \left[\begin{array}{ccc|c} \cos\theta_n & -\sin\theta_n & 0 & 0 \\ \sin\theta_n & \cos\theta_n & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{3.7} $$

and

$$ T_x(r_n) = \left[\begin{array}{ccc|c} 1 & 0 & 0 & r_n \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \quad R_x(\alpha_n) = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_n & -\sin\alpha_n & 0 \\ 0 & \sin\alpha_n & \cos\alpha_n & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{3.8} $$

These are a translation of d_n along the previous z-axis ($T'_z(d_n)$), a rotation of θ_n about the previous z-axis ($R'_z(\theta_n)$), a translation of r_n along the new x-axis ($T_x(r_n)$)and a rotation of α_n around the new x-axis ($R_x(\alpha_n)$). By

replacing each element in Equation (3.6), the following matrix is created:

$$
{}^{n}_{n-1}T = \left[\begin{array}{ccc|c} \cos\theta_n & -\sin\theta_n\cos\alpha_n & \sin\theta_n\sin\alpha_n & r_n\cos\theta_n \\ \sin\theta_n & \cos\theta_n\cos\alpha_n & -\cos\theta_n\sin\alpha_n & r_n\sin\theta_n \\ 0 & \sin\alpha_n & \cos\alpha_n & d_n \\ \hline 0 & 0 & 0 & 1 \end{array}\right] = \left[\begin{array}{ccc|c} & R & & t \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{3.9}
$$

where R is the 3×3 rotation matrix and t is the 3×1 translation vector.

Like for any homogeneous transform, the inverse ${}^{n}_{n-1}T^{-1}n$ is given by

$$
{}^{n-1}_{n}T = \left[\begin{array}{ccc|c} & R^{-1} & & -R^{-1}T \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{3.10}
$$

with the inverse of R simply being its transpose, $R^{-1} = R^T$.

Similar to the concatenation of transformations detailed in Section 2.4.3, ${}^{n}_{n-1}T$ in Equation (3.6) can be concatenated with the other transformation matrices relative to the remaining links in order to compute a the full kinematics of the robot arm from the base reference frame up to the end-effector.

3.2. Inverse Kinematics

The forward kinematics of a system are computed by means of a transformation matrix from the base of a manipulator (or fixed location, such as the corner of a room) to the end-effector of a manipulator (or a mobile robot). As such, they are an exact description of the pose of the robot and they fully characterize its kinematic state. Inverse Kinematics deal with the opposite problem: finding a joint configuration that results in a desired pose at the end effector. To achieve this goal, we will need to solve the forward kinematics equations for joint angles as a function of the desired pose. With reference to Equation (3.1), inverse kinematics aims to solve the following:

$$
q = f^{-1}(r) \ , \qquad f^{-1} : \mathbb{R}^m \to \mathbb{R}^n \ , \tag{3.11}
$$

with a notation similar to Equation (3.1). For a mobile robot, we can do this only for velocities in the local coordinate system, and need more sophisticated methods to calculate appropriate trajectories. We will discuss this in depth in Section 3.3.

3.2.1. Solvability

Equation (3.11) is the inverted version of Equation (3.1), and is heavily non-linear except for trivial mechanisms. Therefore, it makes sense to briefly think about whether we can solve it at all for specific parameters before trying. Here, the workspace of a robot becomes important. The workspace is the sub-space that can be reached by the robot in any configuration. Clearly, there will be no solutions for the inverse kinematic problem outside of the workspace of the robot.

A second question to ask is how many solutions we actually expect to exist and what it means to have multiple solutions *geometrically*. Multiple solutions to achieve a desired pose correspond to multiple ways in which a robot can reach a target (i.e., joint configurations). For example a three-link arm that wants to reach a point that can be reached without fully extending all links (which would have only a single solution) can do this by either folding its links in a concave or a convex fashion. Reasoning about how many solutions will exist for a given mechanism and desired pose quickly becomes non-intuitive. For example, a 6-DOF arm can reach certain points with up to 16 different configurations!

3.2.2. Inverse Kinematics of a Simple Manipulator Arm

We will now look at the inverse kinematics of the 2−link arm that we introduced in Figure 3.1. We need to solve the equations determining the robot's forward kinematics by solving for α and β. This is tricky, however, as we have to deal with more complicated trigonometric expressions than the forward kinematics case.

To build an intuition, assume there to be only one link, l_1. Solving (3.2) for α yields to two distinct solutions:

$$\alpha = \pm \cos^{-1} \frac{x_1}{l_1}, \tag{3.12}$$

as cosine is symmetric for positive and negative values. Indeed, for any possible position on the $x-$axis ranging from $-l_1$ to l_1, there exist two solutions: the first one with the arm above the table, and the other one with the arm below it. At the extremes of the workspace, both solutions are the same.

Solving 3.4 for α and β adds two additional solutions that are cumbersome to reproduce here, involving terms of x and y to the sixth power, and is left as an exercise to the reader, for example using an online symbolic solver.

What will drastically simplify this problem, is to not only specify the desired position, but also the orientation θ of the end-effector. In this case, a desired pose can be specified in the following form

$$\begin{bmatrix} cos\theta & -sin\theta & 0 & x \\ sin\theta & cos\theta & 0 & y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.13}$$

A solution can now be found by simply equating the individual entries of the transformation (3.5) with those of the desired pose. Specifically, we can observe:

$$\begin{aligned} cos\theta &= cos(\alpha + \beta) \\ x &= c_{\alpha\beta} l_2 + c_\alpha l_1 \\ y &= s_{\alpha\beta} l_2 + s_\alpha l_1 \end{aligned} \tag{3.14}$$

These can be reduced to

$$\begin{aligned} \theta &= \alpha + \beta \\ c_\alpha &= \frac{c_{\alpha\beta} l_2 - x}{l_1} = \frac{c_\theta l_2 - x}{l_1} \\ s_\alpha &= \frac{s_{\alpha\beta} l_2 - y}{l_1} = \frac{s_\theta l_2 - y}{l_1} \end{aligned} \tag{3.15}$$

Providing the orientation of the robot in addition to the desired position therefore allows solving for α and β just as a function of x, y and θ.

The main issue with the geometric approach detailed above is that it does not scale easily with an increase of DoF at the joints, and it quickly becomes unwieldy with more dimensions. For higher-DoF platforms, we can calculate a *numerical solution* using an approach that we will later see is very similar to path planning in mobile robotics. To this end, we will take an optimization-based approach: first we calculate a measure of error between the current solution and the desired one, and then change the joint configuration in a way that minimizes this error. In our example, the measure of error is the Euclidean distance between the current end-effector pose (given by the forward kinematics equations in Section 3.1.1) and the desired solution $[x, y]$ in configuration space, i.e. (assuming $l_1 = l_2 = 1$ for simplicity):

$$f_{x,y}(\alpha, \beta) = \sqrt{(s_{\alpha\beta} + s_\alpha - y)^2 + (c_{\alpha\beta} + c_\alpha - x)^2} \tag{3.16}$$

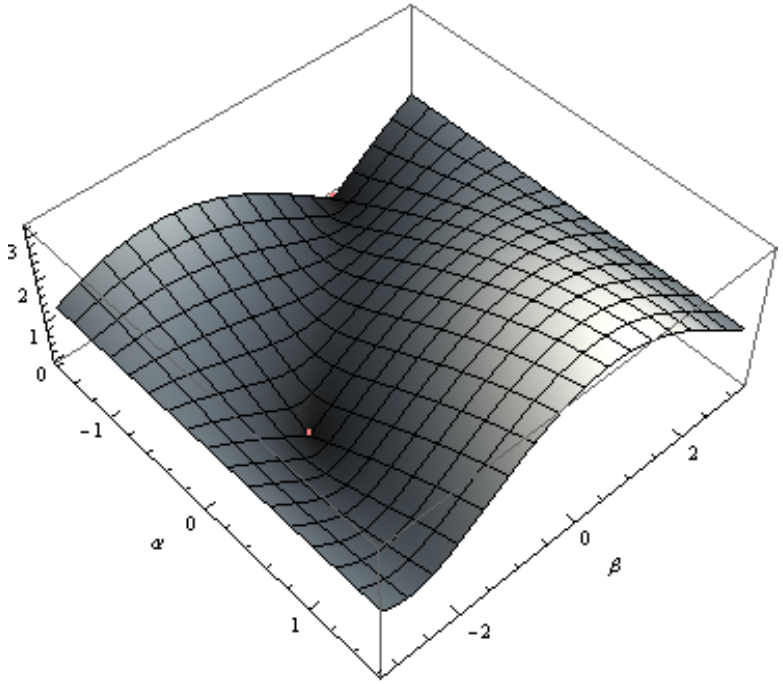

Figure 3.3. Distance to $(x = 1, y = 1)$ over the configuration space of a two-arm manipulator. Minima corresponds to exact inverse kinematic solutions.

Here, the first two terms in parentheses are given by the forward kinematics of the robot, whereas the third term in the parentheses is the desired y and x position, respectively. Equation (3.16) can be plotted as a 3D function of α and β, our joint-space variables. As shown in Figure 3.3 this function has a minima, in this case zero, for values of α and β that bring the manipulator to $(1, 1)$. These values are $(\alpha \to 0, \beta \to -\frac{\pi}{2})$ and $(\alpha \to -\frac{\pi}{2}, \beta \to \frac{\pi}{2})$.

You can now think about inverse kinematics as a path finding problem from anywhere in the configuration space to the nearest minima. A more formalized approach to this will be discussed in Section 3.4.2. How to find the shortest path in space, that is finding the shortest route for a robot to get from A to B, is one of the subjects covered within Chapter 13.

3.3. Differential Kinematics

The two-link arm in Figure 3.1 involved only two free parameters, but was already pretty complex to solve analytically if the end-effector pose was not specified. One can imagine that things become very hard with more degrees of freedom or more complex geometries (mechanisms in which some axes intersect are simpler to solve than others, for example). It is worth noting that, so far, we have analyzed the geometry of motion of a robot at its simplest level of abstraction, i.e. in the space of positions. This abstraction becomes useless as soon as the order of motions matters. For example, in a differential wheel robot, turning the left wheel first and then the right wheel will lead to a very different position than turning the right wheel first and then the left wheel. This is not the case in a robotic arm with two links, which will arrive at the same position no matter which joint will be moved first.

In order to include a notion of temporal evolution of the robot configuration, it is convenient to shift toward a slightly more complex abstraction, that is the space of generalized velocities. This modeling is called *differential kinematics*, as velocities are the time derivative (i.e. the differential) of positions. Similarly to before, with "generalized velocities" we mean "any velocity-equivalent quantity needed to describe the element", as we will detail below.

3.3.1. Forward Differential Kinematics

Forward differential kinematics deals with the problem of computing an expression that relates the generalized velocities at the joints (i.e. the "speed" of our motors) to the generalized velocity of the robot's end-effector. In all, it is the corresponding differential version of Equation (3.1). Let the translational speed of a robot be given by:

$$v = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}. \tag{3.17}$$

As the robot can potentially not only translate, but also rotate, we also need to specify its angular velocity. Let these velocities be given as a vector ω:

$$\omega = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}. \tag{3.18}$$

We can now write translational and rotational velocities in a 6×1 generalized velocity vector as $\nu = [v\ \omega]^T$. This notation is also called *velocity twist*. Let the generalized configuration in joint space (i.e. joint angles/positions) be $q = [q_1, \ldots, q_n]^T$; therefore, we can define the set of joint speeds as $\dot{q} = [\dot{q}_1, \ldots, \dot{q}_n]^T$. We now want to compute the differential kinematics version of Equation (3.1), and in this case relate joint velocities $\dot{q}$ with end-effector velocities $[v\ \omega]^T$. A simple derivation of Equation (3.1) with respect to time gives:

$$\nu = [v \quad \omega]^T = J(q) \cdot [\dot{q}_1, \ldots, \dot{q}_n]^T = J(q) \cdot \dot{q}\ , \tag{3.19}$$

which is our forward differential kinematics equation. $J(q)$ is known as the *Jacobian matrix*; it is a function of the joint configuration q, and contains all of the partial derivatives of f, relating every joint angle to every velocity. In practice, J looks like the following:

$$\nu = \begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial q_1} & \frac{\partial x}{\partial q_2} & \cdots & \frac{\partial x}{\partial q_n} \\ \frac{\partial y}{\partial q_1} & \frac{\partial y}{\partial q_2} & \cdots & \frac{\partial y}{\partial q_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial \omega_z}{\partial q_1} & \frac{\partial \omega_z}{\partial q_2} & \cdots & \frac{\partial \omega_z}{\partial q_n} \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \vdots \\ \dot{q}_n \end{bmatrix} = J(q) \cdot \dot{q} \tag{3.20}$$

This notation is important as it tells us how small changes in joint space will affect the end-effector's position in Cartesian space. It may be helpful to think of each column of this matrix as telling us something about how each component of velocity changes when the configuration (i.e., angle) of a particular joint changes, or each row of the matrix as showing how movement in each joint affects a particular component of velocity. The forward kinematics of a mechanism and its analytical derivative can always be calculated, which allows us to calculate numerical values for the entries of matrix J for every possible joint angle/position.

3.3.2. Forward Kinematics of a Differential Wheeled Robot

After abstractly considering differential kinematics in the previous section, we will now study a mechanism for which general non-differential kinematic models do not exist. We recall that the pose of a robotic manipulator is uniquely defined by its joint angles, which can be estimated using encoders. However, this is not the case for a mobile robot. Here, the encoder values simply refer to wheel orientations, which need to be integrated over time

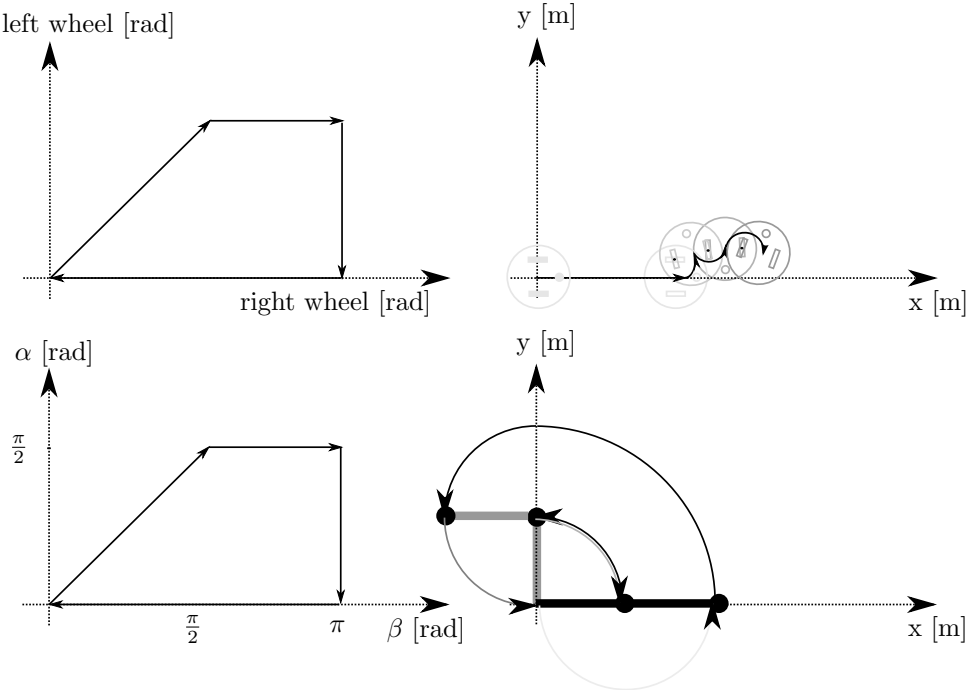

Figure 3.4. Configuration or joint space (left) and workspace or operational space (right) for a non-holonomic mobile robot (top) and a holonomic manipulator (bottom). Closed trajectories in configuration space result in closed trajectories in the workspace if the robot's kinematics is holonomic.

in order to assess the robot's position with respect to the world's frame of reference. As we will later see, this is a source of great uncertainty. What complicates matters is that for so-called *non-holonomic* systems, it is not sufficient to simply measure the distance that each wheel traveled, we must also measure *when* each movement was executed.

A system is non-holonomic when closed trajectories in its configuration space may not return it to its original state. A robot arm is holonomic because each joint position corresponds to a unique position in space (Figure 3.4, bottom): a generic joint-space trajectory that comes back to the starting point will position the robot's end-effector at the exact same position in operational space. A train on a track is holonomic too: moving its wheels backwards by the same amount they have been moving forward brings the train to the exact same position in space. Conversely, a car and

a differential-wheel robot are non-holonomic vehicles (Figure 3.4, top): performing a straight line and then a right-turn leads to the same amount of wheel rotation as doing a right turn first and then going in a straight line; however, getting the robot to its initial position requires not only rewinding both wheels by the same amount, but also getting their relative speeds right. The configuration and corresponding workspace trajectories for a non-holonomic and a holonomic robot are shown in Figure 3.4. Here, a robot first moves on a straight line, meaning both wheels turn an equal amount. Then, the left wheel remains fixed and only the right wheel turns forward. Then, the right wheel remain fixed and the left wheel turns backward. Finally, the right wheel turns backwards, arriving at the initial encoder values (zero). Yet, the robot does not return to its origin. Performing a similar trajectory in the configuration space of a two-link manipulator makes the robot return to its initial position.

It should be clear by now that for a mobile robot, not only does traveled distance per wheel matter, but also the *speed* of each wheel as a function of time. Interestingly, this information was not required to uniquely determine the pose of a manipulating arm. We will establish a world coordinate system $\{I\}$—which is known as the inertial frame by convention (see Figure 3.5). We establish a coordinate system $\{R\}$ on the robot and express the robot's speed ${}^{R}\dot{\xi}$ as a vector ${}^{R}\dot{\xi} = [{}^{R}\dot{x}, {}^{R}\dot{y}, {}^{R}\dot{\theta}]^{T}$. Here ${}^{R}\dot{x}$ and ${}^{R}\dot{y}$ correspond to the speed along the x and y directions in $\{R\}$, whereas ${}^{R}\dot{\theta}$ corresponds to the rotation velocity around the $z-$axis, that you can imagine to be sticking out of the ground. We denote speeds with dots over the variable name, as speed is simply the derivative of distance. Now, let's think about the robot's position in $\{R\}$. It is always zero, as the coordinate system is fixed on the robot itself. Therefore, velocities are the only interesting quantities in this coordinate system and we need to understand how velocities in $\{R\}$ map to positions in $\{I\}$, which we denote by ${}^{I}\xi = [{}^{I}x, {}^{I}y, {}^{I}\theta]^{T}$. These coordinate systems are shown in Figure 3.5.

Notice that the positioning of the coordinate frames and their orientations are arbitrary, meaning it is a choice that we can make. Here, we choose to place the coordinate system in the center of the robot's axle and align ${}^{R}x$ with its default driving direction. In order to calculate the robot's position in the inertial frame, we need to first find out how speed in the robot coordinate frame maps to speed in the inertial frame. This can be done again by employing trigonometry. There is only one complication: a movement into

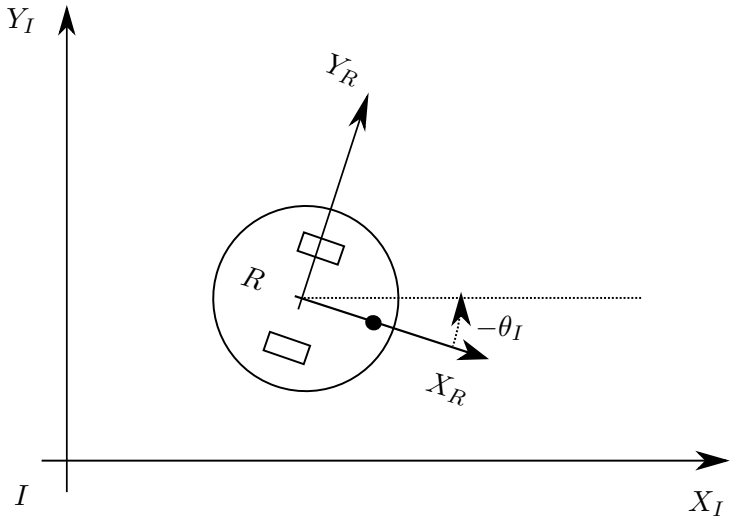

Figure 3.5. Mobile robot with local coordinate system {R} and world frame {I}. The arrows indicate the positive direction of position and orientation vectors.

the robot's $x-$axis might lead to movement along both the $x-$axis and the $y-$axis of the world coordinate frame I. By looking at Figure 3.5, we can derive the following components to $\dot{x}_I$. First,

$$\dot{x}_{I,x} = cos(\theta)\dot{x}_R. \tag{3.21}$$

There is also a component of motion coming from $\dot{y}_R$ (ignoring the kinematic constraints for now, see below). For negative θ, as in Figure 3.5, a move along y_R would let the robot move into positive X_I direction. The projection from $\dot{y}_R$ is therefore given by

$$\dot{x_{I,y}} = -sin(\theta)\dot{y_R}. \tag{3.22}$$

We can now write

$$\dot{x_I} = cos(\theta)\dot{x_R} - sin(\theta)\dot{y_R}. \tag{3.23}$$

Similar reasoning leads to:

$$\dot{y_I} = sin(\theta)\dot{x_R} + cos(\theta)\dot{y_R} \tag{3.24}$$

and

$$\dot{\theta_I} = \dot{\theta_R} \quad , \tag{3.25}$$

which is the case because both the robot's and the world coordinate system share the same $z-$axis in this example. We can now conveniently write:

$$\dot{\xi}_I = {}^I_R T(\theta) \dot{\xi}_R \tag{3.26}$$

with ${}^I_R T(\theta)$ being a rotation around the $z-$axis:

$${}^I_R T(\theta) = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad . \tag{3.27}$$

Maybe unsurprisingly, this is simply the well-known equation for a generic rotation of θ around the $z-$axis, which applies to both velocity vectors as well as poses.

We are now left with the problem of how to calculate the speed $\dot{\xi}_R$ in robot coordinates. For this, we make use of the *kinematic constraints* of the robotic wheels. For a standard wheel in an ideal case scenario, the kinematic constraints are that every rotation of the wheel leads to strictly forward or backward motion and does not allow sideways motion or sliding. We can therefore calculate the forward speed of a wheel $\dot{x}$ using its rotational speed $\dot{\phi}$ (assuming the encoder value/angle is expressed as ϕ) and radius r by

$$\dot{x} = \dot{\phi} r. \tag{3.28}$$

This becomes apparent when considering that the circumference of a wheel with radius r is $2\pi r$. The distance a wheel rolls when turned by the angle ϕ (in radians) is therefore $x = \phi r$, see also Figure 3.6, right. Taking the derivative of this expression on both sides leads to the above expression.

How each of the two wheels in our example contributes to the speed of the robot's center—where its coordinate system is anchored—requires the following trick: we calculate the contribution of each individual wheel while assuming all other wheels remaining un-actuated (see Figure 3.6, left). In this example, the left wheel will move of $r\phi_l$, and the right wheel will move of $r\phi_r$, which in the space of velocities will become $r\dot{\phi}_l$ and $r\dot{\phi}_r$ respectively. Then, the distance traveled by the center point is exactly half of that traveled by each individual wheel (Figure 3.6). We can therefore write:

$$\dot{x_R} = \frac{1}{2}\left(r\dot{\phi}_l + r\dot{\phi}_r\right) = \frac{r\dot{\phi}_l}{2} + \frac{r\dot{\phi}_r}{2} \tag{3.29}$$

given the speeds $\dot{\phi}_l$ and $\dot{\phi}_r$ of the left and the right wheel, respectively.

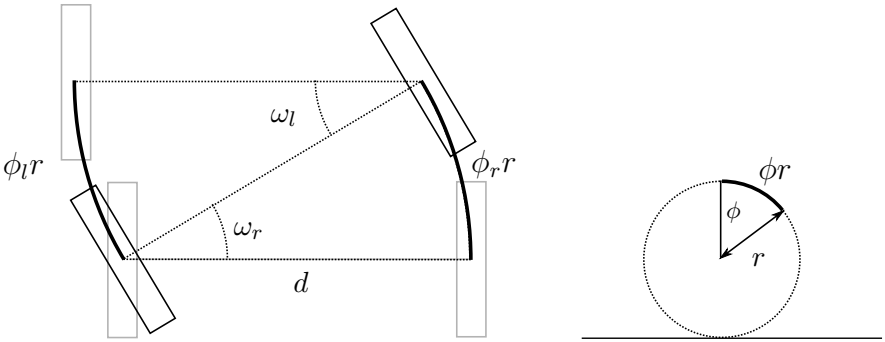

Figure 3.6. Left: Differential wheel robot pivoting around its left wheel first and its right wheel next. For infinitesimal motion, it is possible to decouple left and right wheel to simplify computation of the forward kinematics. Right: A wheel with radius r moves by ϕr when rotated by ϕ degrees.

> Think about how the robot's speed along its y-axis is affected by the wheel speed given the coordinate system in Figure 3.5. Think about the kinematic constraints that the standard wheels impose.

It may be unintuitive at first, but the speed of the robot along its y-axis is always zero. This is because the constraints of the standard wheel tell us that the robot can never slide. We are now left with calculating the rotation of the robot around its z-axis. This rotation can be seen when imagining the robot's wheels spinning in opposite directions. In this case, the robot does not move forward but rather spins in place. We will again consider each wheel independently. Assuming the left wheel to be non-actuated, spinning the right wheel forwards will lead to counter-clockwise rotation. Given an axle diameter (distance between the robot's wheels) d, we can now write

$$\omega_r d = \phi_r r \tag{3.30}$$

with ω_r the angle of rotation around the left wheel (Figure 3.6, left). Taking the derivative on both sides yields speeds and we can write

$$\dot{\omega_r} = \frac{\dot{\phi_r} r}{d} \tag{3.31}$$

Adding the rotation speeds up (with the one around the right wheel being

negative based on the right-hand grip rule), leads to:

$$\dot{\theta} = \frac{\dot{\phi_r} r}{d} - \frac{\dot{\phi_l} r}{d} \tag{3.32}$$

Putting it all together, we can finally write:

$$\begin{bmatrix} \dot{x_I} \\ \dot{y_I} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{r\dot{\phi_l}}{2} + \frac{r\dot{\phi_r}}{2} \\ 0 \\ \frac{\dot{\phi_r} r}{d} - \frac{\dot{\phi_l} r}{d} \end{bmatrix} \tag{3.33}$$

The interested reader might want to compare this form with Equation (3.20), the general Jacobian form of differential kinematics. For this, we ignore the rotation matrix in Equation (3.33) and rewrite its second term in matrix notation:

$$\begin{bmatrix} \dot{x_R} \\ \dot{y_R} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ 0 & 0 \\ \frac{r}{d} & -\frac{r}{d} \end{bmatrix} \begin{bmatrix} \dot{\phi_l} \\ \dot{\phi_r} \end{bmatrix} = \begin{bmatrix} \frac{\partial x_R}{\partial \dot{\phi_l}} & \frac{\partial x_R}{\partial \dot{\phi_r}} \\ \frac{\partial y_R}{\partial \dot{\phi_l}} & \frac{\partial y_R}{\partial \dot{\phi_r}} \\ \frac{\partial \theta}{\partial \dot{\phi_l}} & \frac{\partial \theta}{\partial \dot{\phi_l}} \end{bmatrix} \begin{bmatrix} \dot{\phi_l} \\ \dot{\phi_r} \end{bmatrix} \quad , \tag{3.34}$$

with $X_R = \left(\frac{r\dot{\phi_l}}{2} + \frac{r\dot{\phi_r}}{2}\right) t$ and similar expressions for θ, we observe the validity of the Jacobian approach.

From Forward Kinematics to Odometry

Equation (3.33) only provides us with the relationship between the robot's wheel speed and its speed in the inertial frame. Calculating its actual pose in the inertial frame is known as *odometry*. Technically, it requires integrating Equation (3.33) from 0 to the current time T. As this is not possible but for very special cases, one can approximate the robot's pose by summing up speeds over discrete time intervals, or more precisely:

$$\begin{bmatrix} x_I(T) \\ y_I(T) \\ \theta(T) \end{bmatrix} = \int_0^T \begin{bmatrix} \dot{x_I}(t) \\ \dot{y_I}(t) \\ \dot{\theta}(t) \end{bmatrix} dt \approx \sum_{k=0}^{k=T} \begin{bmatrix} \Delta x_I(k) \\ \Delta y_I(k) \\ \Delta \theta(k) \end{bmatrix} \Delta t \tag{3.35}$$

which can be calculated incrementally as

$$x_I(k+1) = x_I(k) + \Delta x(k) \Delta t \tag{3.36}$$

using $\Delta x(k) \approx \dot{x}_I(t)$ and similar expressions for y_I and θ. Note that Equation (3.36) is just an approximation. The larger Δt becomes, the more inaccurate this approximation becomes as the robot's speed might change during the interval.

> Don't let the notion of an integral worry you! As robots' computers are fundamentally discrete, integrals usually turn into sums, which are nothing more complex than for-loops.

3.3.3. Forward kinematics of Car-like steering

Differential wheel drives are very popular in mobile robotics as they are very easy to build, maintain, and control. Although not holonomic, a differential drive can approximate the function of a fully holonomic robot by first rotating in place to achieve a desired heading and then driving straight. The primary drawbacks of a differential drive are its reliance on a caster wheel, which performs poorly at high speeds, and difficulties in driving straight lines as it requires both motors to drive at the exact same speed.

These drawbacks are mitigated by car-like mechanisms, which are driven by a single motor and can steer their front wheels. This mechanism is known as "Ackermann steering". Ackermann steering should not be confused with "turntable" steering where the front wheels are fixed on an axis with central pivot point. Instead, in Ackermann steering each wheel has its own pivot point and the system is constrained in such a way that all wheels of the car drive on circles with a common center point, avoiding skid. As the Ackermann mechanism lets all wheels drive on circles with a common center point, its kinematics can be approximated by those of a tricycle with rear-wheel drive, or even simpler by a bicycle. This is shown in Figure 3.7.

Consider a car with the shape of a box with length L between its front and rear axis. Let the center point of the common circle described by all wheels be distance R from the car's longitudinal center line. Then, the steering angle ϕ is given by

$$\tan \phi = \frac{L}{R} \tag{3.37}$$

The angles of the left and the right wheel, ϕ_l and ϕ_r can be calculated using the fact that all wheels of the car rotate around circles with a common

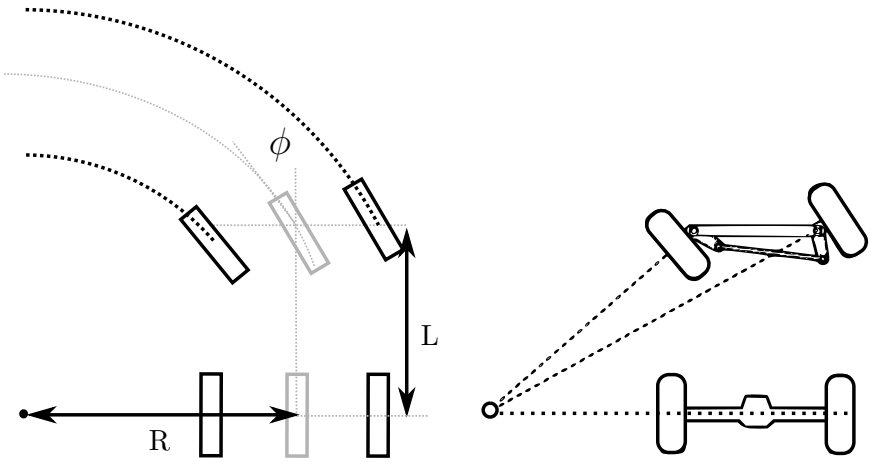

Figure 3.7. Left: Kinematics of car-like steering and the equivalent bicycle model. Right: Mechanism of an Ackermann vehicle.

center point. With the distance between the two front wheels l, we can write:

$$\begin{aligned} \frac{L}{R - l/2} &= \tan(\phi_r) \\ \frac{L}{R + l/2} &= \tan(\phi_l) \end{aligned} \tag{3.38}$$

This is important, as it allows us to calculate the resulting wheel angles resulting from a specific angle ϕ and test whether they are within the constraints of the actual vehicle.

Assuming the wheel speed to be $\dot{\omega}$ and the wheel radius r, we can calculate the speeds in the robot's coordinate frame as:

$$\begin{aligned} \dot{x}_r &= \dot{\omega} r \\ \dot{y}_r &= 0 \\ \dot{\theta}_r &= \frac{\dot{\omega} r \tan\phi}{L} \end{aligned} \tag{3.39}$$

using (3.37) to calculate the circle radius R.

3.4. Inverse Differential Kinematics

It would now be desirable to just invert J in Equation (3.20) in order to calculate the necessary joint speeds for every desired end-effector speeds—a

problem known as *Inverse Differential Kinematics*. Unfortunately, J is only invertible if the matrix is quadratic (i.e. the number of degrees of freedom in joint space n equals the number of degrees of freedom in task space m) and full rank. In the example detailed in Section 3.2.2, the velocity wrench $[v\ \omega]^T$ is 6−dimensional, which means that n should be equal to 6: therefore, inversion of J is only possible if the robot under consideration is equipped with exactly 6 actuators/joints. If this is not the case, we can use the pseudo-inverse computation:

$$J^+ = \frac{J^T}{JJ^T} = J^T(JJ^T)^{-1} \tag{3.40}$$

As you can see, J^T cancels from the equation leaving $1/J$, while being applicable to non-quadratic matrices. We can now write a simple feedback controller that drives our error e, defined as the difference between desired and actual position, to zero:

$$\Delta q = -J^+ e \tag{3.41}$$

That is, we will move each joint a tiny bit in the direction that minimizes our error e. It can be easily seen that the joint speeds will only be zero if e has become zero.

This solution might or might not be numerically stable, depending on the current joint values. If the inverse of J is mathematically not feasible, we speak of a *singularity* of the mechanism. One case where this can happen is when two joint axes line up, therefore effectively removing a degree of freedom from the mechanism, or when the robot is at the boundary of its workspace. As it happens very often in robotics, the concept of singularity has both a strong mathematical justification (the joint configuration is such that the Jacobian is not full rank any more), and a direct physical consequence: singularity configurations are to be avoided as no solution for the inverse differential kinematics problem exists and the robot might become unsafe to operate. In a singularity, the solution to J^+ "explodes" and joint speeds go to infinity. In order to work around this, we can introduce damping to the controller.

In this case, we do not only minimize the error, but also the joint velocities. The minimization problem then becomes:

$$\|J\Delta q - e\| + \lambda^2\|\Delta q\|^2 \tag{3.42}$$

where λ is a constant. One can show that the resulting controller that achieves this has the form:

$$\Delta q = (J^T J + \lambda^2 I)^{-1} J^+ e \tag{3.43}$$

This is known as the *Damped Least-Squares* method. Problems that can arise when taking this approach include the existence of local minima and singularities of the mechanism, which might render solutions suboptimal or infeasible.

3.4.1. Inverse Kinematics of Mobile Robots

There is no unique relationship between the amount of rotation of a robot's individual wheels and its position in space for non-holonomic robots. Instead, we can treat the inverse kinematics problem as solving for the velocities within the local robot coordinate frame. Let's first establish how to calculate the necessary speed of the robot's center given a desired speed $\dot{\xi}_I$ in world coordinates. We can transform the expression $\dot{\xi}_I = T(\theta)\dot{\xi}_R$ by multiplying both sides with the inverse of $T(\theta)$:

$$T^{-1}(\theta)\dot{\xi}_I = T^{-1}(\theta)T(\theta)\dot{\xi}_R \tag{3.44}$$

which leads to $\dot{\xi}_R = T^{-1}(\theta)\dot{\xi}_I$. Here

$$T^{-1} = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.45}$$

which can be computed by performing the matrix inversion or by deriving the trigonometric relationships from the drawing. Similarly to Section 3.2.2, we can now solve Equation (3.33) for ϕ_l, ϕ_r:

$$\begin{aligned} \dot{\phi}_l &= (2\dot{x}_R - \dot{\theta}d)/2r \\ \dot{\phi}_r &= (2\dot{x}_R + \dot{\theta}d)/2r \end{aligned} \tag{3.46}$$

allowing us to calculate the robot's wheel speed as a function of a desired $\dot{x}_R$ and $\dot{\theta}$, which can be calculated using (3.44).

Note that this approach does not allow us to deal with $\dot{y}_R \neq 0$, which might result from a desired speed in the inertial frame. Non-zero values for translation in y-direction are simply ignored by the inverse kinematic solution, and driving toward a specific point either requires feedback control (Section 3.4.2) or path planning (Chapter 13).

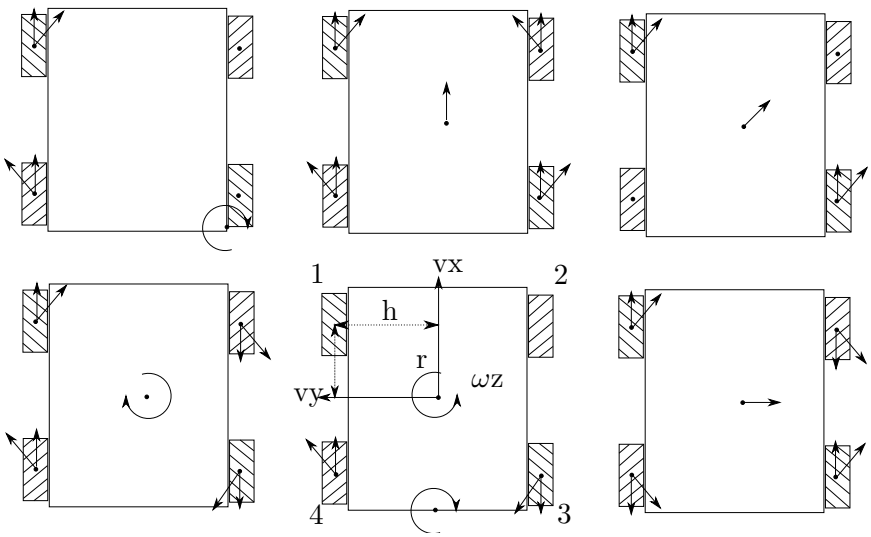

Figure 3.8. Omni-directional robot using "swedish wheels" in different configurations. Each wheel has two velocity components, speed perpendicular to the wheel's main axis and speed of the rollers. Arrows on the robot body indicate the resulting direction of motion and rotation.

Inverse kinematics of an omnidirectional robot

Omnidirectional robots using "Swedish wheels" or "Meccano wheels" are common in factories and educational settings. A drawing of a swedish wheel is shown in Table 2.1. It consists of an actuated wheel with non-actuated rollers around its circumference that are attached in a 45 degree angle. We recall that, similarly to the caster wheel, the Swedish wheel has full three degrees of freedom in the plane, but can enable omnidirectional motion of a robotic platform without the need to rotate. A typical four-wheel configuration is shown in Figure 3.8. Notice the arrangment of the wheels, and in particular the orientation of the rollers, which is critical for the operation as shown.

When actuated by itself, the wheel will perform a sideways motion that is perpendicular to the main axis of its rollers. When used in pairs, opposite directions of motions cancel out, resulting for example into forward motion as shown in Figure 3.8, top, center, or sideways motion, bottom, right.

Similar to a differential wheel platform, each wheel also imparts a rotation

on the robot body. As the wheels are mounted off the center axle, each wheel contributes to two angular moments. One is around the horizontal axis with distance h_i to the robot's center, and the other is around the vertical axis with distance r_i to the robot's center (Figure 3.8, bottom, center). All combined, the rotation of each wheel will add up to the robot moving with velocities v_x, v_y and ω_z.

The velocity at each wheel has two components, the velocity of the i-th wheel perpendicular to its main axis $v_{i,m}$, and the velocity of the rollers $v_{i,r}$ that is either $+45\,\text{deg}$ or $-45\,\text{deg}$ to the wheel axis. Note that for the system to work, diagonally opposite wheels need to have the same angle. Let the angle of the roller of wheel i be γ_i. We can now derive the following equations following (Maulana, Muslim & Hendrayawan 2015):

$$v_{i,m} + v_{i,r}\cos(\gamma_i) = v_x - h_i * \omega_z \tag{3.47}$$

That is, the velocity components perpendicular to the wheel axis are equivalent to the forward velocity of the robot plus the velocity component at the wheel resulting from the robot's angular velocity. Positive angular velocity would result in backward motion, by definition of the robot coordinate system. Similarly, we can write

$$v_{i,r}\sin(\gamma_i) = v_y + r_i * \omega_z \tag{3.48}$$

Note that there is no lateral component to the main wheel's motion, as lateral motion can only be achieved via the rollers.

Dividing (3.48) by (3.47) and solving for v_i results into

$$v_i = v_x - h_i\omega_z - \frac{v_y + r_i\omega_z}{\tan\gamma_i} \tag{3.49}$$

With $h_i \in h = \{h, -h, h, -h\}$, $r_i \in r = \{r, r, -r, -r\}$ and $\gamma_i \in \gamma = \{-45\,\text{deg}, +45\,\text{deg}, +45\,\text{deg}, -45\,\text{deg}\}$ to reflect the different configuration of each wheel, we can derive an expression for the controllable wheel velocities $v_{i,m}$

$$\begin{aligned} v_{1,m} &= v_x + v_y + r\omega_z - h\omega_z \\ v_{2,m} &= v_x - v_y - r\omega_z + h\omega_z \\ v_{3,m} &= v_x - v_y + r\omega_z - h\omega_z \end{aligned} \tag{3.50}$$

$$v_{4,m} = v_x + v_y - r\omega_z + h\omega_z$$

With $v_{i,m} = R\omega_i$ and R the radius of each Swedish wheel, we can now compute the required wheel velocity for any desired robot velocity v_x, v_y, and ω_z.

3.4.2. Feedback Control for Mobile Robots

Assume the robot's position to be given by x_r, y_r and θ_r and the desired pose as x_g, y_g and θ_g—with the subscript g indicating "goal". We can now calculate the error in the desired pose by:

$$\begin{aligned} \rho &= \sqrt{(x_r - x_g)^2 + (y_r - y_g)^2} \\ \alpha &= \tan^{-1} \frac{y_g - y_r}{x_g - x_r} - \theta_r \\ \eta &= \theta_g - \theta_r \ , \end{aligned} \tag{3.51}$$

which is illustrated in Figure 3.9. These errors can be converted directly into robot speeds, for example using a simple proportional controller with gains p_1, p_2 and p_3:

$$\dot{x} = p_1 \rho \tag{3.52}$$

$$\dot{\theta} = p_2 \alpha + p_3 \eta \tag{3.53}$$

which will let the robot drive in a curve until it reaches the desired pose.

3.4.3. Under-actuation and Over-actuation

As detailed at the beginning of this Chapter, kinematics is concerned with analyzing the mapping between our control variables (i.e. the robot's motors represented by the n DoFs in joint space) and their effect on the motion of the robot (our m DoFs in task/configuration space). These two spaces might have different dimensionality, and the relation between these two dimensions greatly affects how we can solve the kinematic problem. It is convenient to analyze these differences by looking at the Jacobian J, since the size of the matrix is $m \times n$; in all, we have three different conditions:

- $n = m \rightarrow$ The robot is *fully actuated*. The Jacobian J is square and full rank, and the forward kinematics equation is directly invertible;

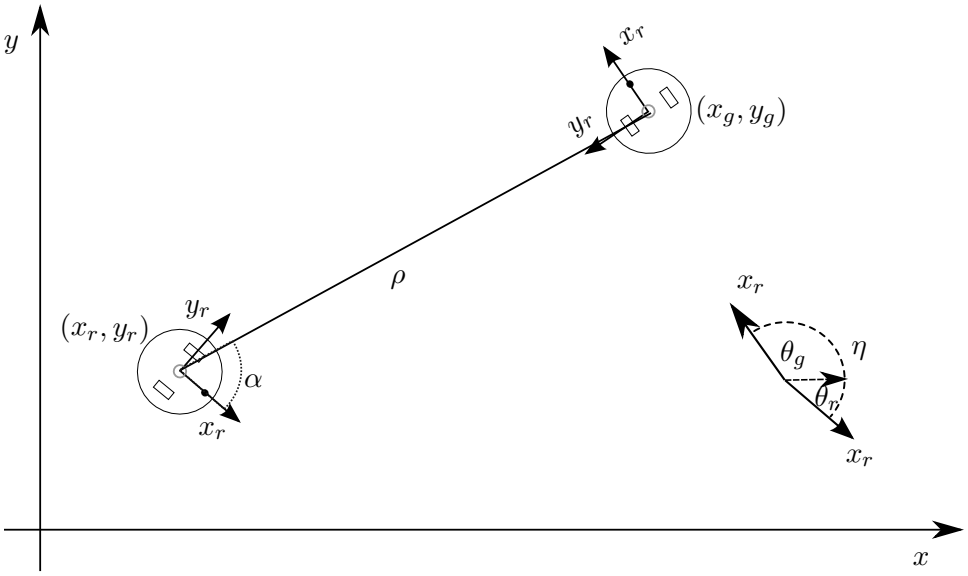

Figure 3.9. Difference in desired and actual pose as a function of distance ρ, bearing α and heading η.

- $n \leq m \rightarrow$ The robot is *under actuated*, and the kinematics problem is *kinematically deficient*. The Jacobian J is "wide", because there are more columns m than rows n, and not invertible any more; the only way to solve the inverse kinematics problem is through the pseudo-inverse J^+ (and similar/more advanced approaches).

- $n \geq m \rightarrow$ The robot is *over actuated*, and the kinematics problem is *kinematically redundant*. The Jacobian J is "tall", because there are more rows n than columns m, and not invertible any more; the only way to solve the inverse kinematics problem is through the pseudo-inverse J^+ (and similar/more advanced approaches). In this scenario, it is useful to determine the redundancy coefficient $n-m$ which affects the space of solutions of the inverse kinematics problem.

Over- and under-actuation are important design considerations to keep in mind when choosing a robot for a particular task. In a *kinematically deficient* scenario, the robot is not capable of full motion in task space, as it does not have sufficient degrees of freedom in joint space to "cover" every possible

configuration in task space. This does not mean that the robot is useless! It can still perform tasks—just not *every* possible task you might ask it to perform. Conversely, if the problem is *kinematically redundant*, the robot has more joint DoFs available than it needs, and there exist an infinite number of inverse kinematics solutions in non-singular configurations. Contrary to what one may think, redundancy is actually a great feature to have in a robot system, in that it enables flexibility and versatility in solving the kinematic problem: that is, it is possible to choose *the best* solution among many, and one that satisfies additional constraints and requirements. A human arm (without considering the hand) is a good example of a kinematically redundant manipulator, as it is equipped with seven DoFs in joint space (three at the shoulder, one at the elbow and three at the wrist), whereas the task space is of dimension six (i.e. the three positions and three orientations of the wrist). This additional degree of mobility allows humans to reach for objects in multiple configurations, choose motions that are energy efficient, avoid obstacles, and more!

Take-home lessons

- Forward kinematics deals with finding a coordinate transform from a world coordinate system into a coordinate system on the robot. Such a transform is a combination of a (3×1) translation vector and a (3×3) rotation matrix that consists of the unit vectors of the robot coordinate system. Both translation and rotation can be combined into a 4×4 homogeneous transform matrix.
- Forward and Inverse Kinematics of a mobile robot are performed with respect to the speed of the robot and not its position.
- To calculate the effect of each wheel on the speed of the robot, you need to consider the contribution of each wheel independently.
- The inverse kinematics of a robot involves solving the forward kinematics equations for the joint angles. Calculating the inverse kinematics analytically becomes quickly infeasible, and is impossible for complicated mechanisms.
- A simple numerical solution is provided by taking all partial derivatives of the forward kinematics in order to get an easily invertible expression that relates joint speeds to end-effector speeds.

- The inverse kinematics problem can then be formulated as feedback control problem, which will move the end-effector towards its desired pose using small steps. Problems with this approach are local minima and singularities of the mechanism, which might render this solution infeasible.
- Redundancy allows a robot to solve a kinematic problem in multiple different ways, thus providing better dexterity and versatility in its motion.

Exercises

Coordinate systems

1. a) Write out the entries of a rotation matrix ${}^{A}_{B}R$ assuming basis vectors X_A, Y_A, Z_A, and X_B, Y_B, Z_B.
   b) Write out the entries of rotation matrix ${}^{B}_{A}R$.
2. Assume two coordinate systems that are co-located in the same origin, but rotated around the $z-$axis by the angle α. Derive the rotation matrix from one coordinate system into the other and verify that each entry of this matrix is indeed the scalar product of each basis vector of one coordinate system with every other basis vector in the second coordinate system.
3. Consider two coordinate systems $\{B\}$ and $\{C\}$, whose orientation is given by the rotation matrix ${}^{C}_{B}R$ and have distance ${}^{B}P$. Provide the homogenous transform ${}^{C}_{B}T$ and its inverse ${}^{B}_{C}T$.
4. Consider the frame $\{B\}$ that is defined with respect to frame $\{A\}$ as $\{B\} = \{{}^{A}_{B}R, {}^{A}P\}$. Provide a homogeneous transform from $\{A\}$ to $\{B\}$.
5. Program a simple application that displays a 2D (or 3D) coordinate system and add the ability to move and turn the coordinate system using your keyboard.

Forward and inverse kinematics

1. Consider a differential wheel robot with a broken motor, i.e., one of the wheels cannot be actuated anymore. Derive the forward kinematics of this platform. Assume the right motor is broken.
2. Consider a tri-cycle with two independent standard wheels in the rear and the steerable, driven front-wheel. Choose a suitable coordinate system and use ϕ as the steering wheel angle and wheel-speed $\dot{\omega}$. Provide forward and inverse kinematics.

3. Program an application that displays a differential wheel platform and allows you to control forward and rotational speed with your keyboard. Output the robot's pose after every step.
4. Program an application that displays a two-link robotic arm moving in the plane and lets you change both joint angles using your keyboard.
5. Derive the forward kinematics of a two-link robotic arm as well as its Jacobian. Implement its inverse kinematic using the inverse Jacobian technique.
6. Program an application that displays a two-link robotic arm moving in the plane and lets you change the position of its end-effector using your keyboard.
7. Explore the internet for toolkits that allow you to manage forward and inverse kinematics for a robotic arm. What kind of tools do you find and what kind of input do they require to model the robot's geometry?
8. Download the manual of a commercially available robot arm of your choice. What kind of input does it take from its user? Does it allow you to control its position directly?
9. Use a robot simulator of your choice to access a simulated vehicle. What kind of actuator input can you provide and what are the sensors that are available? Drive the car using your keyboard and try to estimate its pose by implementing odometry.
10. Can you provide an example of a kinematically deficient and a kinematically redundant robot manipulator?