8803MM: Trajectory Control and IK

Frank Dellaert
Center for Robotics and Intelligent Machines
Georgia Institute of Technology

April 12, 2020

Contents

1 Review: Forward Kinematics

The problem of forward kinematics (FK) is [2]:

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

One way to define FK of planar arms (in 2D) is to adopt 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 $i - 1$.

Defining T_t^n to be the unchanging pose of the tool T in the frame of link n, and concatenating all transforms we obtain the following recursive equation for FK:

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

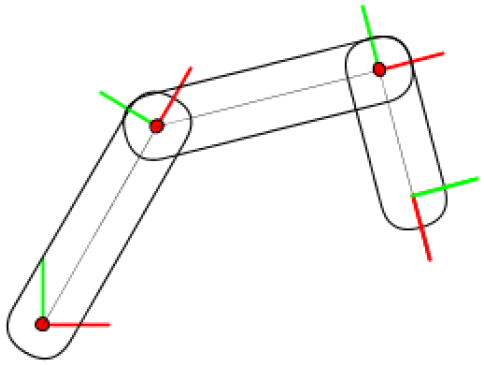

Figure 1.1: Example of simple RRR manipulator with all three joints actuated: $\theta_1 = 60^{\text{o}}$, $\theta_2 = -45^{\text{o}}$, and $\theta_3 = -90^{\text{o}}$, respectively.

As an example, Figure 1.1 shows a planar RRR manipulator with $a_1 = 3.5$, $a_2 = 3.5$, and $a_3 = 2$. We identified the tool frame with link frame 3, i.e., $X_t^3 = I$. We then have:

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

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

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

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.

2 Trajectory Control and the Manipulator Jacobian

Trajectory following is an important capability for manipulator robots, and three main approaches are common:

1. Trajectory replay
2. Joint space motion control
3. Cartesian space motion control.

The first 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.

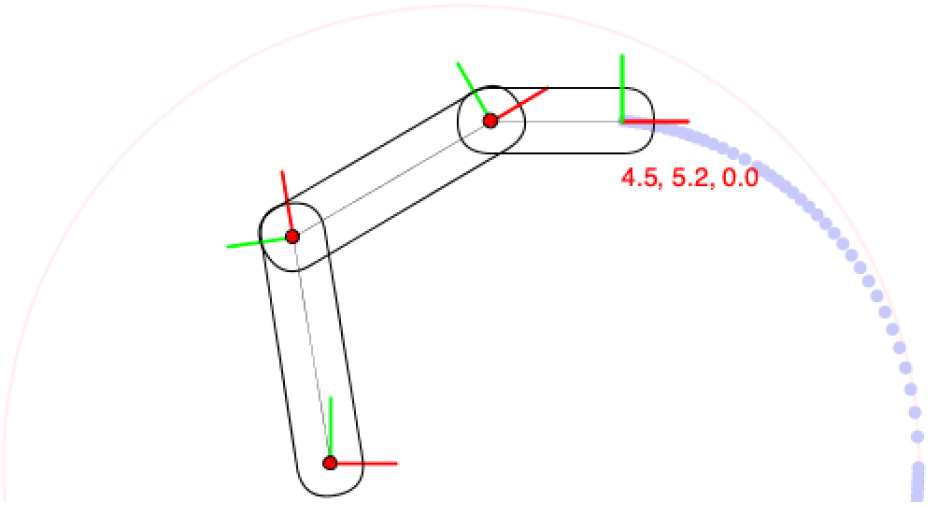

Figure 2.1: In joint-space motion control, the joint angles are linearly interpolated, but this leads to a curved path in Cartesian space.

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.,

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

where q_t and q_d are the current and desired joint angles, and K_p is a proportional gain. An example of this is shown in Figure 2.1 for the three-link manipulator.

More difficult is **Cartesian motion control**, where we want the robot 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 (see Section 3) 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. Because we eventually do need to control the joint angles q, the key is to derive a relationship between velocities $(\dot{x}, \dot{y}, \dot{\theta})$ 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:

$$(\dot{x}, \dot{y}, \dot{\theta}) = J(q)\dot{q} \tag{2.1}$$

The quantity $J(q)$ above is the **manipulator Jacobian**. For planar manipulators, as $(\dot{x}, \dot{y}, \dot{\theta}) \in \mathbb{R}^3$, 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 $(\dot{x}, \dot{y}, \dot{\theta}) \in \mathbb{R}^3$ corresponding a change in the joint angle q_j only, i.e.,

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

where

$$J_i(q) \triangleq \frac{\partial(x(q), y(q), \theta(q))}{\partial q_j} = \begin{bmatrix} \frac{\partial x(q)}{\partial q_j} \\ \frac{\partial y(q)}{\partial q_j} \\ \frac{\partial \theta(q)}{\partial q_j} \end{bmatrix}$$

Example. 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 2.2 shows the Jacobian $J(q)$ as a set of three velocities: red for joint 1, green for joint 2, and blue for joint 3. Clearly, these 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.

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 3, 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

$$\begin{pmatrix} -3.5\sin\theta_1 - 3.5\sin\alpha - 2.5\sin\beta & -3.5\sin\alpha - 2.5\sin\beta & -2\sin\beta \\ 3.5\cos\theta_1 + 3.5\cos\alpha + 2.5\cos\beta & 3.5\cos\alpha + 2.5\cos\beta & 2\cos\beta \\ 1 & 1 & 1 \end{pmatrix} \tag{2.2}$$

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.

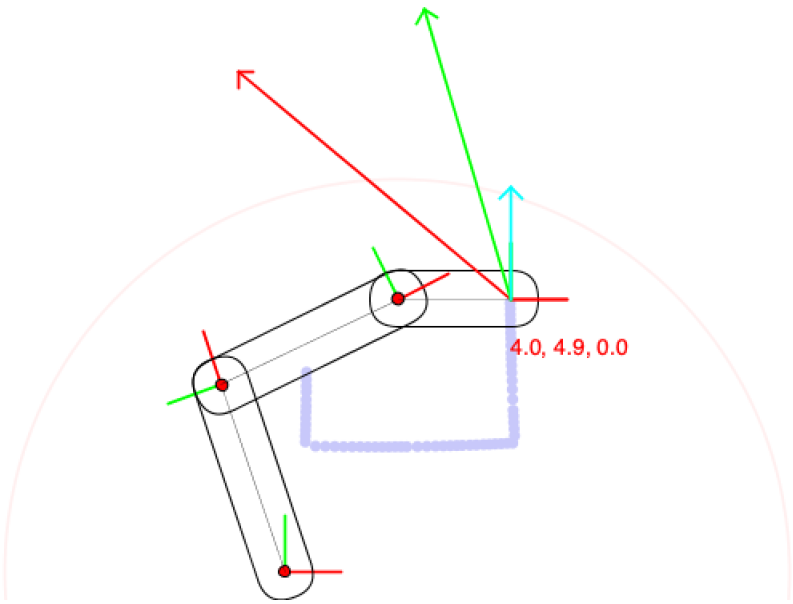

Figure 2.2: Cartesian space motion control, showing the resulting straight trajectories of the end-effector for three successive waypoints.

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 $(\dot{x}, \dot{y}, \dot{\theta})$:

$$\dot{q} = J(q)^{-1}(\dot{x}, \dot{y}, \dot{\theta}) \tag{2.3}$$

Hence, to achieve a desired trajectory in Cartesian space, we can calculate the desired velocity $\mathcal{V}$ at any given time, calculate the corresponding joint velocities $\dot{q}$ using (2.3), and apply simple proportional control, i.e.,

$$q_{t+1} = q_t + K_p J(q_t)^{-1}(T_d - T(q_t)),$$

Example. An example of Cartesian motion control with proportional control is shown in Figure 2.2 for the three-link planar robot, where we followed a sequence of straight trajectories of the end-effector, for three successive waypoints.

3 Inverse Kinematics

Inverse kinematics (IK) is the process of finding joint angles given a desired end-effector pose $T_{desired}$, i.e., solve Equation 1.1 for q:

$$T_t^s(q) = T_{desired}$$

If $T_{desired}$ is outside the workspace of the robot, there is no solution, otherwise there might be a unique solution or multiple solutions.

The essential concepts can be explained by using a two-link planar manipulator. In this simple case, the forward kinematics are given by

$$\begin{cases} x(q) = l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2) \\ y(q) = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2) \end{cases} \tag{3.1}$$

The inverse kinematics problem is then to find the joint angles $q = (\theta_1, \theta_2)$ such that $(x(q), y(q)) = (x_d, y_d)$, the desired end-effector position.

3.1 Closed-Form Solutions

Many industrial manipulators have **closed-form solutions**, and there are several ways to derive these. In this simple 2-link case above, a closed form IK solution is possible, although generally non-unique, within a radius $(l_1 + l_2)$ of the origin. I adapted a solution from John Hollerbach's 2008 lecture notes, for $l_1 = l_2 = L$: we first compute the second joint angle

$$\theta_2(x_d, y_d) = \pm 2 \arctan \sqrt{\frac{(2L)^2}{x_d^2 + y_d^2} - 1} \tag{3.2}$$

after which we compute the first joint angle

$$\theta_1(x_d, y_d, \theta_2) = atan2(y_d, x_d) - atan2\left(L\sin\theta_2, L(1 + \cos\theta_2)\right) - \frac{\pi}{2} \tag{3.3}$$

Note that Equation 3.2 is not defined if $x_d^2 + y_d^2 > (2L)^2$, which corresponds to a desired position that is out-of-range.

To extend this to the three-link example, we add a desired tool orientation, i.e., we have a desired 2D pose $T_d = (x'_d, y'_d, \theta'_d)$, where I used primes to distinguish

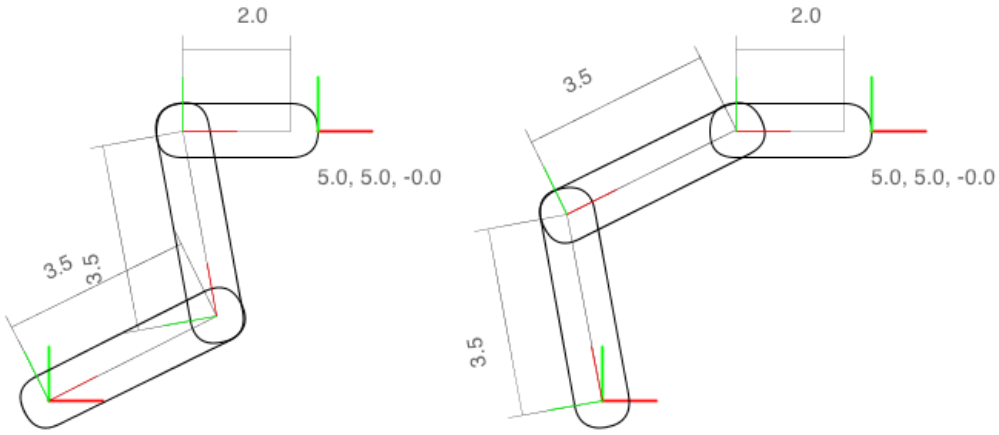

Figure 3.1: Two IK solutions for the planar three-link arm, for tool pose $(x'_d, y'_d, \theta'_d) = (5, 5, 0)$, with joint angles $q = (-63.6^{\text{o}}, 74.0^{\text{o}}, -100.4^{\text{o}})$ and $q = (10.4^{\text{o}}, -74.0^{\text{o}}, -26.4^{\text{o}})$, corresponding respectively to choosing a positive and negative sign in Equation 3.2.

from the 2-link example. Note from the forward kinematics Equation 1.2 that $\beta = \theta'_d - \pi/2$, and we can adapt Equations3.3 and 3.2 to accommodate for the joint 2 axis offset:

$$x_d = x'_d + 2.5 \sin\beta \text{ and } y_d = y'_d - 2.5 \cos\beta$$

$$\theta'_2 = (x_d, y_d) \text{ and } \theta'_1 = (x_d, y_d, \theta'_2)$$

Then, the wrist joint angle is easily computed as

$$\theta'_3 = \theta'_d - \theta'_1 - \theta'_2 - \pi/2$$

Figure 3.1 shows two examples corresponding to choosing a different sign in Equation 3.2 for the same desired pose $(x'_d, y'_d, \theta'_d) = (5, 5, 0)$, illustrating the fact that inverse kinematics is not a one-on-one mapping, but that multiple solutions might exist, even for a non-redundant manipulator.

3.2 Iterative Methods

As discussed, many industrial manipulators have closed-form solutions, but they are still an area of active research and general solutions are as yet elusive. There exist, however, iterative methods to solve the IK problem.

One approach is to find the joint angles q that minimize the error between desired end-effector pose and computed end-effector pose. For example, for the simple 2-link arm we would try to minimize

$$E(q) \triangleq \frac{1}{2}\|p_d - p(q)\|^2 = \frac{1}{2}\left(x_d - x(q)\right)^2 + \frac{1}{2}\left(y_d - y(q)\right)^2 \tag{3.4}$$

where $p_d \in \mathbb{R}^2$ and $p(q) \in \mathbb{R}^2$ are the desired and computed 2D position. However, this is a non-linear minimization problem, as $x(q)$ and $y(q)$ are typically non-linear functions of the joint-angles (at least for rotational joints). Linear least-squares problems are easier to solve, which motivates us to start with an initial guess q for the joint angles, and to linearize the forward kinematics around this pose,

$$p(q + \delta q) \approx p(q) + J(q)\delta q \tag{3.5}$$

where $J(q)$ is once again the manipulator Jacobian, derived in Section 2. For the simple 2-link planar manipulator we can only hope achieve a desired position $p_d \in \mathbb{R}^2$, and hence the Jacobian $J(q)$ is only a 2×2 matrix, easily calculated as

$$\begin{aligned} J(q) &= \begin{bmatrix} \frac{\partial x(q)}{\partial \theta_1} & \frac{\partial x(q)}{\partial \theta_2} \\ \frac{\partial y(q)}{\partial \theta_1} & \frac{\partial y(q)}{\partial \theta_2} \end{bmatrix} \\ &= \begin{bmatrix} -y(q) & -l_2 \sin\left(\theta_1 + \theta_2\right) \\ x(q) & l_2 \cos\left(\theta_1 + \theta_2\right) \end{bmatrix} \end{aligned}$$

Substituting the approximation 3.5 into the objective 3.4 we obtain

$$E(q + \delta q) \approx \frac{1}{2}\|\left(p_d - p(q)\right) - J\delta q\|^2 \tag{3.6}$$

where the dependance of J on q is implied, for notational simplicity. The above says that we can make the position error $e(q) \triangleq p_d - p(q)$ go to zero by calculating a change δq in joint angles, such that

$$J\delta q = p_d - p(q)$$

For a non-redundant manipulator the Jacobian $J(q)$ is square, and its inverse generally exists (except at the boundaries of the workspace). Hence, we could try to invert it immediately:

$$\delta q = J^{-1}\left(p_d - p(q)\right) \tag{3.7}$$

However, because the forward kinematics are non-linear, we might have to iterate this a few times, and the process might in fact diverge.

3.3 Damped Least-Squares

A safer approach is to impose some penalty for taking steps δq that are too large, which can be done by adding a term to the objective function 3.6:

$$E(q+\delta q) \approx \frac{1}{2}\| \left(p_d - p(q)\right) - J\delta q\|^2 + \frac{1}{2}\|\lambda \delta q\|^2 \tag{3.8}$$

This can be solved for δq by setting the derivative of E in Equation 3.8 to 0,

$$-J^T \left(\left(p_d - p(q)\right) - J\delta q\right) + \lambda^2 \delta q = 0$$

which, after simply re-arranging, leads to a damped least-squares iteration:

$$\delta q = \left(J^T J + \lambda^2 I\right)^{-1} J^T \left(p_d - p(q)\right) \tag{3.9}$$

The value of λ is chosen as to make the iterative process converge, and can even be increased automatically when a low value is seen to lead to divergence. On the other hand, too high a value might lead to slow convergence.

One strategy is to proceed very cautiously, i.e., make λ very large, in which case Equation 3.9 essentially becomes gradient descent:

$$\delta q = \lambda^{-2} J^T \left(p_d - p(q)\right) \tag{3.10}$$

Because it only involves J^T, this is also called the **transpose Jacobian method**, but it has the disadvantage of converging very slowly, see Fig. 4.1.

A method that picks λ automatically is the **Levenberg-Marquardt** method: start with a high λ, and then reduce it by a constant factor at every iteration until the error goes up instead of down: in that case undo the change in λ and try again.

3.4 Iterative IK Methods Summary

In summary, all iterative inverse kinematics approaches share the same structure:

Algorithm 1 The basic iterative IK algorithm.

1: **function** ITERATIVEIK(x_d)
2: $q \leftarrow q_0$ ▷ Guess an initial value for joint angles
3: **while** $E(q) \neq 0$ **do** ▷ While not converged
4: $e \leftarrow p_d - p(q)$ ▷ Calculate error between desired and computed pose
5: Calculate $\delta q = f(J(q), e)$ ▷ Using Eqn. (3.7), (3.9), (3.10)
6: $q \leftarrow q + \delta q$ ▷ Update the joint angles q
7: **return** q ▷ Joint angles yielding p_d

4 Redundant Manipulators

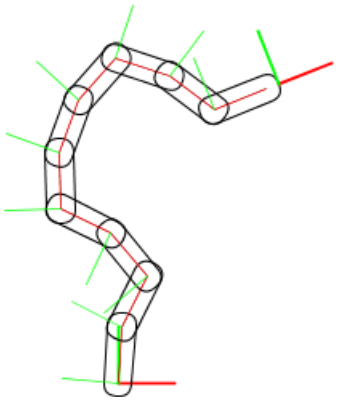

Figure 4.1: A highly redundant planar manipulator.

One advantage of the least-squares approaches above is that they also work for $n > m$, i.e., for **redundant manipulators**. An example of a highly redundant planar manipulator is shown in Figure 4.1. In this case the inverse in Equation 3.7 does not exists (as J is non-square), but we can use the **pseudo-inverse** $J^\dagger$,

$$\delta q = J^\dagger \left(p_d - p(q) \right) \tag{4.1}$$

where $J^\dagger \triangleq J^T \left(J J^T \right)^{-1}$ for $n > m$.

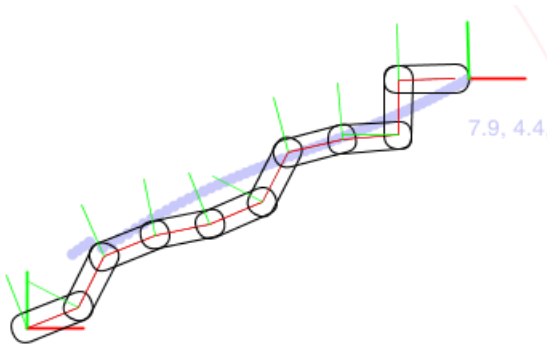

Figure 4.2: Convergence of iterative IK with the transpose Jacobian method.

Note that for redundant manipulators the pseudo-inverse is equivalent to damped least squares with $\lambda = 0$. Of course, even in the redundant case we can take λ to

be non-zero and just apply gradient descent (3.9), or make λ large in which case we recover the transpose method (3.10). An example of the latter is shown as a trajectory in Figure 4.2.

In a redundant manipulator there are typically an infinite number of ways to attain a desired end-effector pose. The methods above arbitrarily pick one of the solutions. However, the extra degrees of freedom might be put to other uses, as well: for example, we might want to favor configurations that require less energy to maintain, or avoid singularities, or even - in the case of using IK for animation - follow a certain style [1]. This can be done by adding an additional, user-defined penalty term $E_{user}(q)$ to the error function that penalizes certain joint configurations and favors others:

$$E(q) \stackrel{\Delta}{=}^2 \frac{1}{2}\|p_d - p(q)\|^2 + E_{user}(q)$$

As long as the derivative of $E_{user}(q)$ is available, it is easy to incorporate this extra information.

A simple example is to make the joint angles as small as possible, i.e., penalizing a deviation from the rest state, leading to

$$E(q) \stackrel{\Delta}{=}^2 \frac{1}{2}\|p_d - p(q)\|^2 + \frac{1}{2}\|\beta q\|^2$$

After linearizing, we have

$$E(q + \delta q) \approx \frac{1}{2}\| \left(p_d - p(q)\right) - J\delta q\|^2 + \frac{1}{2}\|\beta(q + \delta q)\|^2$$

which yields the following update,

$$\delta q = \left(J^T J + \beta^2 I\right)^{-1} \left(J^T \left(p_d - p(q)\right) - \beta^2 q\right)$$

which looks very much like the damped least-squares iteration (3.9), except that now there is an extra error term that drives the joint angles to zero.

Finally, a user could also impose hard equality or inequality constraints. One such constraint is that the robot should never self-intersect or collide with objects in its environment. In dynamic environments, then, these constraints define a **configuration space** that can change over time. We then get into the realm of fully fledged **motion planning**, which is a prolific and active area of research.

References

[1] K. Grochow, S.L. Martin, A. Hertzmann, and Z. Popović. Style-based inverse kinematics. In *SIGGRAPH*, volume 23, pages 522–531. ACM, 2004.

[2] R.M. Murray, Z. Li, and S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.