Inverse Kinematics II

Glen Henshaw
Craig Carignan

March 6, 2025

1 Numerical Inverse Kinematics

It's often difficult to impossible to find a closed–form solution for the inverse kinematics of a given manipulator. But it's always possible to solve the inverse kinematics problem numerically. This has become more popular in the last decade or so because embedded computers have become powerful enough to enable it.

Numerical techniques are also often more flexible and powerful than closed–form techniques. A good numerical approach can be agnostic to the manipulator kinematics, for instance; you can give it the DH parameters of whatever arm you are using, and it will work with few changes. You can even add or subtract degrees of freedom.

2 Direct Numerical Search

You might be tempted to just set this up as a problem in joint space:

$$\min_{\underline{q}} \ \| {}^{O}_{W}T - {}^{0}T(\underline{q}) \|$$

or similar. This minimization problem could in theory be solved by gradient descent or a quasi–Newton method.

This **can** work, but it often doesn't. The issue is that the minimization problem is nonlinear, and hence may have many locally minima. In general it has no special structure that lends itself to being solved by nonlinear optimization solvers. It is also difficult to bound the number of iterations needed to find the solution, and hence in realtime applications it can be problematic.

3 Jacobian–based Techniques: Resolved–motion Rate Control (RMRC)

A better idea is to think about the problem in terms of Cartesian velocities and rates instead of positions and orientations. That lets us use the fact that the

Jacobian relationship:

$$\underline{\dot{x}} = J(\underline{q})\underline{\dot{q}}$$

is linear with respect to the joint rates $\underline{\dot{q}}$. The problem, of course, is that the Jacobian relationship goes the wrong way. We want to be able to fix $\underline{\dot{x}}$ and calculate $\underline{\dot{q}}$.

You could consider just inverting the Jacobian:

$$\underline{\dot{q}} = J^{-1}(\underline{q})\underline{\dot{x}}$$

This is also, generally speaking, a bad idea. The most obvious problem is that the Jacobian may not even be square: it is $N \times 6$, where N is the number of degrees of freedom of the manipulator. So if your manipulator is not 6–DOF, you *can't* take an inverse. But even if you do have a 6–DOF manipulator, simply inverting the Jacobian is a bad idea, because square matrices can be singular. So we'll talk about what it means, both mathematically and mechanically, to have a singular Jacobian.

3.0.1 An aside: some concepts from linear algebra

Consider the system

$$Ax = 0$$

where A is any matrix, square or nonsquare. If A is M by N, the set of all vectors X that satisfy this equation form a subset of $\mathbb{R}^N$. This subset is nonempty, since it clearly contains the zero vector: $x = 0$ always satisfies $Ax = 0$. This subset actually forms a subspace of $\mathbb{R}^N$, called the *nullspace* of the matrix A and denoted $N(A)$. If x_1 is in $N(A)$, then, by definition, $A(x_1 + x_2) = Ax_2$ for any value of x_2.

In terms of robotic manipulators, the Jacobian will always have a nullspace. For a 6–DOF manipulator, in general the nullspace will contain only the zero vector. But for a manipulator with more than 6 DOF, the nullspace will contain a subspace of joint rates **that do not cause the end effector to move**. In other words, for nearly any arm pose $\underline{q}$, you will be able to find an infinite number of joint rate vectors $\underline{\dot{q}}$ such that $J(\underline{q})\underline{\dot{q}} = 0$.

This is called "self–motion". It is perhaps easiest to visualize with a 7–DOF revolute arm: in this case, you can independently choose to rotate the elbow without moving the end effector.

If A is square, it may or may not be invertible. A square matrix A that is not invertible is called *singular*. This means that there exists at least some vector $\underline{x} \in \mathbb{R}^N$ not equal to zero such that $A\underline{x} = 0$. A matrix that is singular may also be referred to as a reduced–rank matrix, as distinguished from a full-rank matrix. Although the formal mathematical concept of singularity is only defined for square matrices, in robotics we use this term somewhat loosely. We will say that a nonsquare $M \times N$ matrix that has a row rank less than N is also singular.

Physically, when we refer to a singular Jacobian, what this means is that the arm is in a pose where it cannot produce a Cartesian tooltip velocity in some

direction. Recall that the standard (read: bad) way of calculating an inverse that is taught in linear algebra courses involves dividing by the matrix determinant. A singular matrix has a determinant of 0, so attempting to calculate the inverse of a singular matrix is the linear algebra equivalent of a divide–by–zero.

Consequently, if you blindly stuff a singular Jacobian into any sort of numerical inversion routine, in the best case you will end up with infinite values for at least some of the matrix entries (assuming your code doesn't just blow up!), and if you then multiply it by a desired Cartesian tooltip velocity vector, you may end up with infinitely large joint rates.

4 Manipulability and Singularities

The *manipulability* of a manipulator is (usually) a function of its pose, and refers to how close the Jacobian is to becoming reduced–rank. A manipulator is in a singular pose when the Jacobian at that pose does, in fact, become reduced–rank. There are several ways of calculating the manipulability of a robotic arm. One way is to examine the determinant of the Jacobian; there are at least three or four other widely used techniques.

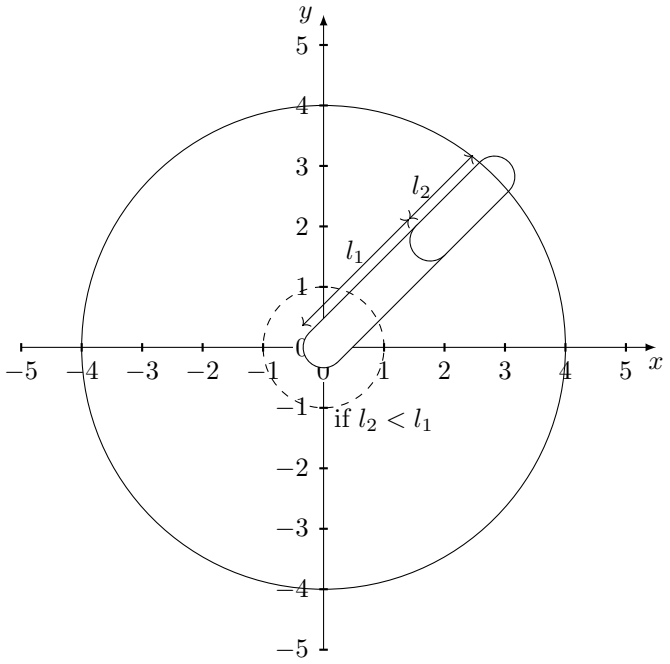
Singular poses have the following characteristics:

- mobility of the manipulator is reduced
- infinite numbers of solutions to the inverse kinematics may exist
- small desired end effector velocities may result in very large joint velocities
- small joint torques may cause very large tool tip forces

There are two types of singularities: *boundary singularities* and *internal singularities*. Boundary singularities are poses where the tooltip lies on the boundary of the workspace. Remember our diagram from last lecture: except here the arm is full stretched out so that the end effector lies exactly on the outer workspace boundary. This arm is in a singular pose, and cannot produce a Cartesian velocity along the direction orthogonal to the normal of the workspace (either inward or outward). All robotic arms have singular poses at their workspace boundaries.

For many manipulators there are also poses that are not on a workspace boundary where the manipulator cannot produce a velocity in some direction. It's important to avoid these situations. Some ways to locate such poses:

- Set the determinant to zero, $|J(\underline{q})| = 0$, and solve for $\underline{q}$
  - only works for 6–DOF arms (or other arms with square Jacobians)
  - this is essentially an exhaustive search; there may be many such poses, and you have to somehow find all of them.



- Examine the rank degeneracy of J. If we express the Jacobian as

$$J = \begin{bmatrix} | & | & | & | & | \\ c_1 & c_2 & c_3 & \cdots & c_N \\ | & | & | & | & | \end{bmatrix}$$

then J becomes deficient when the c vectors become linearly dependent.

Note that for an arm with a spherical wrist, this can be split into two subproblems: $J_{tran}(\theta_{arm})$ and $J_{rot}(\theta_{wrist})$.

4.1 Example: two axis planar arm

We're going to use the simplest J, which we derived a couple of lectures ago, in order to make life easier on ourselves. Jacobians expressed in different frames all have exactly the same singularities, so we are free to use the Jacobian expressed in any frame we like.

$$^3J = \begin{bmatrix} l_1 s_2 & 0 \\ l_1 c_2 + l_2 & l_2 \end{bmatrix}$$

where $M = 2$ (x, y) and $N = 2$ (θ_1, θ_2).

Setting the determinant to zero:

$$|J| = l_1 l_2 s_2 = 0 \Rightarrow \theta_2 = 0^\circ \text{ or } 180^\circ$$

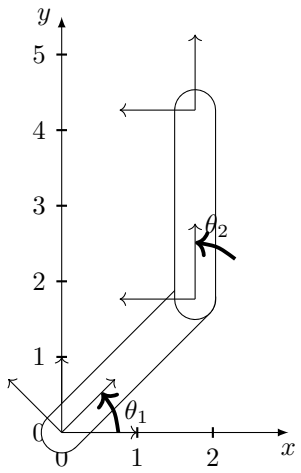


Figure 1: Two–link planar manipulator

in which case,

$$ {}^3J = \begin{bmatrix} 0 & 0 \\ l_1 + l_2 & l_2 \end{bmatrix} $$

and the two columns are not linearly dependent.

4.1.1 Joint velocities near the singularity

$$ \underline{\dot{q}} = {}^3J^{-1}\,{}^3\underline{\dot{x}} \Rightarrow {}^3J^{-1} = \begin{bmatrix} \frac{1}{l_1 s_2} & 0 \\ -(\frac{1}{l_1 s_2} + \frac{c_2}{l_2 c_2}) & \frac{1}{l_2} \end{bmatrix} $$

so

$$ \begin{aligned} \dot{\theta_1} &= \frac{1}{l_1 s_2}\dot{x} \\ \dot{\theta_2} &= -\left(\frac{1}{l_1 s_2} + \frac{c_2}{l_2 s_2}\right)\dot{x} \end{aligned} $$

which implies that the arm cannot move in the y direction, and an attempt to set $\underline{\dot{x}}$ will result in the denominators of the fractions becoming zero, hence the joint rates will become infinite.

NOTE, however that even if θ_2 is not exactly 0° or 180° but is merely close to one of these values, the denominators of the expressions will be quite large. Hence, the singularity is "felt" for some distance away; exactly how far depends on the maximum joint rates that are acceptable and/or achievable.

4.1.2 End Effector forces near the singularity

$$ {}^3\underline{f} = {}^3J^{-T}\underline{\tau} $$

so

$$\begin{aligned} f_x &= \frac{1}{l_1 s_2}\tau_1 - \left(\frac{1}{l_1 s_2} + \frac{c_2}{l_2 s_2}\right)\tau_2 \\ f_y &= \frac{1}{l_2}\tau_2 \end{aligned}$$

which implies that $f_x \to \infty$ near the singularity.

4.2 Example: PUMA RRR Wrist

The D–H table for the PUMA robot wrist is

i	α_{i-1}	a_{i-1}	d_i	θ_i
4	-90°	α_3	d_4	θ_4
5	$+90^\circ$	0	0	θ_5
6	-90°	0	0	θ_6

$$^3J_{rot} = \begin{bmatrix} 0 & s_4 & -c_4 s_5 \\ 1 & 0 & c_5 \\ 0 & c_4 & s_4 s_5 \end{bmatrix}$$

$$|\,^3J| = -\sin\theta_5 \Rightarrow \theta_5 = 0^\circ, \pm 180^\circ$$

At the singularity,

$$^3J_{rot} = \begin{bmatrix} 0 & s_4 & 0 \\ 1 & 0 & 1 \\ 0 & c_4 & 0 \end{bmatrix}$$

so the first and third columns are linearly dependent. At the singularity, the first roll axis is aligned with the third roll axis, and essentially they have become redundant.

This is an important point: interior singularities occur at poses where degrees of freedom that are normally independent become redundant.

4.3 Pseudo–inverses

OK, so if taking a direct inverse of the Jacobian is a bad idea, what else can we do? A better idea is to use a pseudo–inverse. Contrary to popular opinion, there are in general an infinite number of pseudo–inverses for a matrix, but the one everyone knows about is the Moore–Penrose pseudo–inverse:

$$J^\dagger \triangleq J^T \underbrace{(JJ^T)}_{square!}{}^{-1}$$

You can always take a Moore–Penrose pseudo–inverse, even for a nonsquare Jacobian. In addition, the Moore–Penrose pseudo–inverse has the nice property that it minimizes the 2–norm:

$$\|J^\dagger(\underline{q})\underline{\dot{x}}\|_2^2 = \min \|\underline{\dot{q}}\|_2^2 \;\text{ s.t. } J(\underline{q})\underline{\dot{q}} = \underline{\dot{x}}$$

What this means is that of all the vectors of joint rates that lie in the nullspace of the Jacobian, the Moore–Penrose pseudo–inverse finds the one that is smallest, e.g. the joints rates are the lowest.

To tie this into our aside above: any given pseudo–inverse will choose exactly one joint rate vector $\underline{\dot{q}}_2$ that satisfies $\underline{\dot{x}} = J(\underline{q})\underline{\dot{q}}_2$. But if the Jacobian has a nontrivial nullspace $N(J)$, we can find an infinite number of vectors $\underline{\dot{q}}_1$ such that $\underline{\dot{x}} = J(\underline{q})(\underline{\dot{q}}_2 + \underline{\dot{q}}_1)$. The Moore–Penrose pseudo–inverse inherently chooses $\underline{\dot{q}}_1$ such that $\|\underline{\dot{q}}_1 + \underline{\dot{q}}_2\|_2^2$ is the smallest it can be.

There are more clever ways to pick $\underline{\dot{q}}_1$. You could, for instance, choose a $\underline{\dot{q}}_1$ that moves the elbow of your arm away from a potential collision hazard or that improved its manipulability.

One easy way to improve the robustness of any technique that uses a Moore–Penrose pseudo–inverse is to add what is called a *damped least–squares* term:

$$J^\dagger \triangleq J^T(JJ^T + \gamma I)^{-1}$$

where $\gamma \ll 0$. Equivalently, this adds a small positive value to the diagonal of the square matrix that is being inverted. The effect of this is to limit joint rates in the vicinity of a singularity, so it adds singularity robustness. The tradeoff for this is that the trajectory the arm takes will deviate somewhat from the (impossible to follow) trajectory that you specified.

A different, more mathematically justifiable, way to robustly take a pseudo–inverse is based on the singular–value decomposition (SVD). SVDs are beyond the scope of this course, but if you already know about SVDs, the idea is that the "singular values" of a matrix, which are sort of like eigenvalues, can be used to directly compute the matrix inverse. A near–singular matrix will have a number of singular values that are very small. You can simply drop these singular values and only use the ones that are above some threshold to calculate a pseudo–inverse.

4.4 Problems with pseudo–inverses

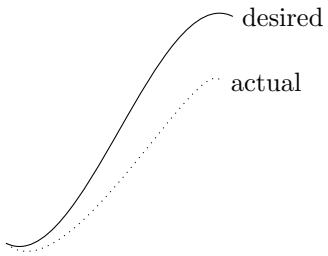
First, it's important to bear in mind that a manipulator singularity is a mechanical phenomenon. Better math won't let us get rid of singularities. So even if you use a Moore–Penrose pseudo–inverse, if the arm is in a singular pose, you still won't be able to move it along its singular direction. Your best bet is to avoid singular poses somehow, normally by examining the rank of the Jacobian — or some other manipulability index — along the trajectory you want and if it starts to become singular, modify the trajectory.

A more subtle problem stems from the fact that we're operating in joint velocity space. In order to get back to joint angle space, i.e. to have something useful to send to whatever control laws are controlling the actuators: we have to perform an integration.

Numerical integrators are susceptible to drift. Simple ones that are suitable for realtime use are especially so. If you do something like

$$\underline{q}_{i+1} = \underline{q}_i + J^\dagger(\underline{q}_i)\underline{\dot{x}}$$

i.e. use a straight forward Euler integration scheme, you will get trajectories that look like this:



There are two common ways to solve this.

4.4.1 Iterated Jacobian method

One is called the "iterated Jacobian method". The idea is that you do a Newton–Raphson iteration on *position*.

$$\begin{aligned} \underline{x} &= \underline{f}(\underline{q}) \\ \underline{x}_0 &= \underline{f}(\underline{q}_0) \end{aligned}$$

Take a Taylor series about $\underline{q}_0$:

$$\begin{aligned} \underline{f}(\underline{q}_0 + \Delta\underline{q}) &= \underline{f}(\underline{q}_0) + \left.\frac{\partial \underline{f}}{\partial \underline{q}}\right|_{\underline{q}=\underline{q}_0} \Delta\theta + O(\Delta\underline{q}^2) \\ \underline{q} &= \underline{q}_0 + J^\dagger(\underline{q}_0) \underbrace{[\underline{x} - \underline{x}_0]}_{\Delta\underline{x}} \qquad \text{for } N = M \end{aligned} \tag{1}$$

Note that Δx must be small for this to converge.

OK, what is $\underline{x}$ here?

$$\underline{x} = \begin{bmatrix} \underline{p} \\ ? \end{bmatrix}$$

$\underline{p}$ is straightforward: It's your Cartesian position vector. But what is in the rotation spot?

It depends on how you have chosen to represent orientation, **but it must be a three–parameter representation**. If you are using Euler or fixed angles, it can be simply those. But if it's a quaternion, you have a bit of a problem, because quaternions have four parameters. Remember, though, that quaternions are defined as

$$\underset{\sim}{q} = \begin{bmatrix} \psi \\ \hat{n} \end{bmatrix}$$

where ψ is a scalar and $\hat{n}$ is a 3–vector. **In a small angle situation**, ψ is close to unity and $\hat{n}$ is close to zero, and exactly zero if the angle is zero. In fact, for small angles, $\hat{n}$ approximates 1/2 time the 3–2–1 Euler angles. In general we use $\hat{n}$ as the rotation part.

So what's $\Delta\underline{x}$? Again, for the positional part it's straightforward: it's $\underline{p}-\underline{p}_0$. And again, for rotation... it's complicated.

If you are using an Euler angle or fixed angle set, you can subtract them. This hand–waves over a bunch of math but as a small angle approximation it holds, and only extreme pedants (or possibly those cramming fo their doctoral quals) need go any further. So if you are using the $[\alpha, \beta, \gamma]^T$ set:

$$\underline{x} = \begin{bmatrix} \underline{p} \\ \alpha - \alpha_0 \\ \beta - \beta_0 \\ \gamma - \gamma_0 \end{bmatrix}$$

If you are using a quaternion representation, you really shouldn't just subtract $\hat{n}$ from $\hat{n}_0$. The proper way to calculate the difference between two quaternions $\underset{\sim}{q}$ and $\underset{\sim}{q}_0$ is:

$$\underset{\sim}{q}_0 * \Delta\underset{\sim}{q} = \underset{\sim}{q} \;\Rightarrow\; \Delta\underset{\sim}{q} = \underset{\sim}{q} * \underset{\sim}{q}_0^{-1}$$

and it turns out that the quaternion inverse is

$$\underset{\sim}{q}^{-1} = \begin{bmatrix} \psi \\ -\hat{n} \end{bmatrix}$$

and quaternion multiplication is defined as

$$\begin{bmatrix} \psi_1 \\ -\hat{n}_1 \end{bmatrix} \times \begin{bmatrix} \psi_2 \\ -\hat{n}_2 \end{bmatrix} = \begin{bmatrix} \psi_1\psi_2 - \hat{n}_1 \cdot \hat{n}_2 \\ \psi_1\hat{n}_2 + \psi_2\hat{n}_1 + \hat{n}_1 \times \hat{n}_2 \end{bmatrix}$$

So the algorithm is then:

1 $\underline{q}_0 = \underline{q}$ and $\underline{x}_0 = \underline{f}(\underline{q})$

2 $\Delta\underline{x} = \underline{x}$"-"$\underline{x}_0$

3 $J = J(\underline{q})$

4 $\Delta\underline{q} = J^\dagger\Delta\underline{x}$

5 $\underline{q} = \underline{q}_0 + \Delta\underline{q}$

6 $\|\Delta\underline{q}\| < \epsilon$? If yes, stop

7 go to 1

4.4.2 Direct Error Correction

Another way to solve the integral drift problem is simply to build an error correction term into the RMRC law, e.g. to implement

$$\underline{\dot{q}} = J^{\dagger}(\underline{q})\left(\underline{\dot{x}} + \gamma \Delta \underline{x}\right)$$

where γ is a gain that has to be selected for good performance, and everything else is defined as we did above.

4.4.3 Constrained quadratic optimization

It turns out that we can derive a straightforward quadratic optimization problem in the spirit of RMRC:

$$\begin{aligned}
& \min_{\underline{\dot{q}}} \quad \|\underline{\dot{x}} - J(\underline{q})\underline{\dot{q}}\|^2 \\
= & \min_{\underline{\dot{q}}} \quad \left(\underline{\dot{x}} - J(\underline{q})\underline{\dot{q}}\right)^T \left(\underline{\dot{x}} - J(\underline{q})\underline{\dot{q}}\right) \\
= & \min_{\underline{\dot{q}}} \quad \underline{\dot{x}}^T\underline{\dot{x}} - 2\underline{\dot{x}}^T J(\underline{q})\underline{\dot{q}} + \underline{\dot{q}}^T J^T(\underline{q}) J(\underline{q})\underline{\dot{q}} \\
= & \min_{\underline{\dot{q}}} \quad -2\underline{\dot{x}}^T J(\underline{q})\underline{\dot{q}} + \underline{\dot{q}}^T J^T(\underline{q}) J(\underline{q})\underline{\dot{q}}
\end{aligned}$$

This is in a form for which there are many good, realtime–capable quadratic optimization solvers. unlike the two previous techniques, though, in this form it's easy to add constraints, e.g.

$$\begin{aligned}
& \min_{\underline{\dot{q}}} \quad -2\underline{\dot{q}}^T J(\underline{q}) + \underline{\dot{q}}^T J^T(\underline{q}) J(\underline{q})\underline{\dot{q}} \\
& \text{s.t.} \\
& -\theta_{1,min} \leq \theta_1 \leq \theta_{1,max} \\
& -\theta_{2,min} \leq \theta_2 \leq \theta_{2,max} \\
& \vdots
\end{aligned}$$

and with a little cleverness you can even add constraints that depend on $\underline{q}$, not $\underline{\dot{q}}$, such as collision constraints. You can also correct for integrator drift by solving

$$\min_{\underline{\dot{q}}} \quad \|\underline{\dot{x}} - J(\underline{q})\underline{\dot{q}}\|^2 + \gamma \|\underline{q}_d - \left(\underline{q} + \Delta t \underline{\dot{q}}\right)\|^2$$

Thinking along these lines eventually leads you to a technique called model–predictive control, which if you've ever seen YouTube videos of robots doing parkour, you've seen in action. It is a very powerful technique for robotic motion control.