Operational Space Control

Glen Henshaw
Craig Carignan

November 12, 2023

1 Backtracking

Last time we talked about computed torque control, the most common form of "true" nonlinear control used in robotics (as opposed to gain scheduling, which is a form of nonlinear control that treats the robot as if it has linear dynamics, and then uses *ad hoc* techniques to make up for the fact that it doesn't.

Computed torque control is in fact a specific type of a more general control technique called "alpha–beta partitioning". The term refers to the fact that when dealing with a nonlinear dynamic system, we often want to have a controller with two parts: a model part that attempts to cancel the natural dynamics, and a feedback part that then instituted new, stable, linear dynamics and in the process takes care of any inaccuracies in the model. This basic idea appears all over nonlinear control and is not restricted to robots.

We can easily see this distinction if we think about controlling a nonlinear spring–mass–damper:

$$m\ddot{x} + b\dot{x} + qx^3 = f \tag{1}$$

where we have a nonlinear spring in that third term. The control law is going to have the form

$$f = \alpha f' + \beta$$

where

$$\begin{aligned} \alpha &= m \\ \beta &= b\dot{x} + qx^3 \end{aligned}$$

and the servo portion is

$$f = \ddot{x}_d + k_v\dot{e} + k_p e$$

If we assume we know the dynamics perfectly, and can therefore cancel them exactly, then we're just left with a dynamic system where the error is clearly stable, by inspection:

$$\begin{aligned} m\ddot{x} + b\dot{x} + qx^3 &= m\left(\ddot{x}_d + k_v\dot{e} + k_p e\right) + b\dot{x} + qx^3 \\ \Rightarrow 0 &= \ddot{e} + k_v\dot{e} + k_p e \end{aligned}$$

If we assume that we don't know the model perfectly, then we need to resort to Lyapunov theory (or some other form of nonlinear analysis). NOTE: there is no ironclad guarantee that such a controller is stable (although it often is – the presence of a linear feedback servo term makes this framework *extremely* robust to model uncertainties assuming the gains chosen are "big enough" in some informal sense that has to do with both the amount of uncertainty in the model and the bandwidth of the desired trajectory).

Recalling the computed torque control from last week:

$$\underline{\tau} = M(\underline{\theta})(\ddot{\underline{\theta}}_d - K_p(\underline{\theta} - \underline{\theta}_d) - K_v(\dot{\underline{\theta}} - \dot{\underline{\theta}}_d)) + C(\underline{\theta}, \dot{\underline{\theta}})\dot{\theta} + E(\underline{\theta}, \dot{\underline{\theta}}) \tag{2}$$

you can see that in this case

$$\begin{aligned} \alpha &= M(\underline{\theta}) \\ \beta &= C(\underline{\theta}, \dot{\underline{\theta}})\dot{\theta} + E(\underline{\theta}, \dot{\underline{\theta}}) \\ f' &= \ddot{\underline{\theta}}_d - K_p(\underline{\theta} - \underline{\theta}_d) - K_v(\dot{\underline{\theta}} - \dot{\underline{\theta}}_d) \end{aligned}$$

2 Intro

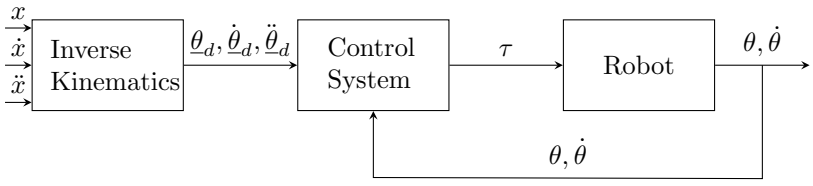
Remember from last time that we can write the dynamics of our robot in the form

$$\underline{\tau} = M(\underline{\theta})\ddot{\underline{\theta}} + C(\underline{\theta}, \dot{\underline{\theta}})\dot{\theta} + E(\underline{\theta}, \dot{\underline{\theta}}) \tag{3}$$

This is, essentially, a joint–space approach to control. The fundamental problem with joint–space (also known as configuration space) control is that we almost always want to specify the motions of our robotic arm in terms of Cartesian moves of the end effector. We have good inverse kinematics schemes, which let us do this, but by now we're building up a fairly complex set of algorithms, and sometimes it's easier and better to just design our control laws directly in operational space.

3 Joint–Based Control Architecture

One approach, of course, is to just specify the trajectory in Cartesian space, use inverse kinematics to convert it to joint space, and then do the control as we've already discussed. Here's a block diagram of our robotic system:



where the inverse kinematics equations look like:

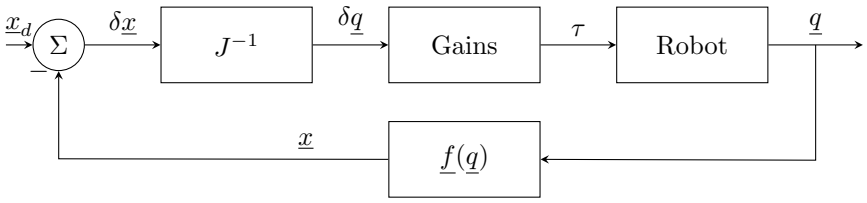
$$\begin{aligned} \underline{q}_d &= f^{-1}(\underline{x}_d) \\ \dot{\underline{q}}_d &= J^{-1}(\underline{q}_d)\dot{\underline{x}}_d \\ \ddot{\underline{q}}_d &= \dot{J}^{-1}\dot{\underline{x}}_d + J^{-1}\ddot{\underline{x}}_d \end{aligned}$$

...and finding that second derivative might be tricky. In practice, it's often done using a numerical difference scheme, which works in most cases because we're differencing something that has no noise. But it isn't ideal.

4 Operational Space Control

4.1 Inverse Jacobian Control

Here's one way to approach Cartesian space control:

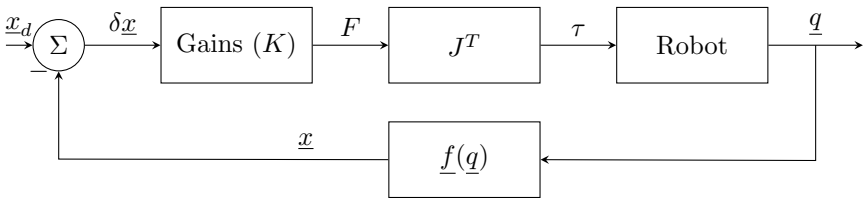


Note that $\underline{x}_d - \underline{x} \approx J(\underline{q}_d - \underline{q})$ for small errors.
This has two problems:

1. It's unstable near singularities
2. That inverse calculation can be computationally complex. This used to be a bigger problem, when embedded computers were a lot slower at floating point computations, but it still can crop up, e.g. if you're trying to operate a rover on the surface of Venus or something similarly insane.

4.2 Jacobian Transpose Control

If you *are* trying to build a rover for the surface of Venus, and you just can't deal with modern CPUs, there's a fascinating approach that doesn't require taking an inverse. It's also very, very stable.



...where the idea is that we use the fact that the Jacobian transpose converts Cartesian forces to joint torques:

$$\underline{\tau} = J^T \underline{F}$$

so what we're doing here is using our gains to calculate a fictional Cartesian force, and then converting that force to joint torques, thus approximating those forces.

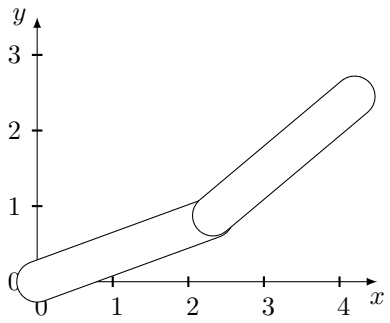
This also has some issues:

1. $\underline{\tau} = J^T \underline{F}$ ignores dynamics. It "behaves" as if the arm is an ideal kinematic system; and
2. $\underline{F} = K\ \delta\underline{x}$ is a static relationship.

4.2.1 Example

Here's an example, doing it both ways.

Our old friend the two–link planar manipulator:



where

$$J = \begin{bmatrix} -l_1 s_1 - l_2 s_{12} & -l_2 s_{12} \\ l_1 c_1 + l_2 c_{12} & l_2 c_{12} \end{bmatrix}$$

Let's examine it at $\theta_1 = \theta_2 = 0$:

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

1. J^{-1} control:

$$\underline{\dot{x}} = J\underline{\dot{\theta}}$$

so

$$\begin{aligned} \dot{x} &= 0 \\ \dot{y} &= (l_1 + l_2)\dot{\theta_1} + l_2\dot{\theta_2} \end{aligned}$$

which indicates that motion in the x direction is not possible. Which makes sense, this is a singular pose. When we try to do the Jacobian inversion, then, J^{-1} explodes.

2. J^T Control

$$\begin{aligned} = \underline{\tau} &= J^T \underline{F} \\ \underline{F} &= K\ \delta\underline{x} \end{aligned}$$

where

$$\begin{aligned} \tau_1 &= (l_1 + l_2) f_y \\ \tau_2 &= l_2 f_y \end{aligned}$$

...so in this case, commands to move in the $x-$ direction are simply ignored. Which is good, because the arm is physically incapable of moving in the $x-$direction. And you get singularities management, basically for free.

5 Cartesian Computed Torque Control

It turns out that it's possible to write the full nonlinear dynamics of a robot in Cartesian terms:

$$H_x(\underline{\theta})\underline{\ddot{x}} + C_x(\underline{\theta}, \underline{\dot{\theta}}) + E_x(\underline{\theta}, \underline{\dot{\theta}}) = \underline{F}$$

which you can convince yourself enough with the right insertion of Jacobians and derivatives of Jacobians in the right places:

$$\begin{aligned} H_x(\underline{\theta}) &= J^{-T}(\underline{\theta}) M(\underline{\theta}) J^{-1}(\underline{\theta}) \\ C_x(\underline{\theta}, \underline{\dot{\theta}}) &= J^{-T}(\underline{\theta}) \left(C(\underline{\theta}, \underline{\dot{\theta}}) - H(\underline{\theta}) J^{-1}(\underline{\theta}) \dot{J}(\underline{\theta}) \underline{\dot{\theta}} \right) \\ E_x(\underline{\theta}, \underline{\dot{\theta}}) &= J^{-T}(\underline{\theta}) E(\underline{\theta}, \underline{\dot{\theta}}) \end{aligned}$$

where J^{-T} refers to the transpose of the Jacobian inverse.

You can then contemplate doing what looks like a Cartesian computed torque controller using the same $\alpha - \beta$ partitioning scheme as before:

$$\begin{aligned} \underline{F} &= H_x(\underline{\theta})\underline{F}' + \beta \\ \beta &= C_x(\underline{\theta}, \dot{\underline{\theta}}) + E_x(\underline{\theta}, \dot{\underline{\theta}}) \\ \underline{f}' &= \ddot{\underline{x}}_d + K_{p_x}(\underline{x}_d - \underline{x}) + K_{v_x}(\dot{\underline{x}}_d - \dot{\underline{x}}) \end{aligned}$$

Note that finding the derivative of the Jacobian is a fairly onerous calculation, which goes a long way towards explaining why these schemes are not more widely used. There are now programming languages that support automatic differentiation, though, so maybe we'll see them become more popular in the future.