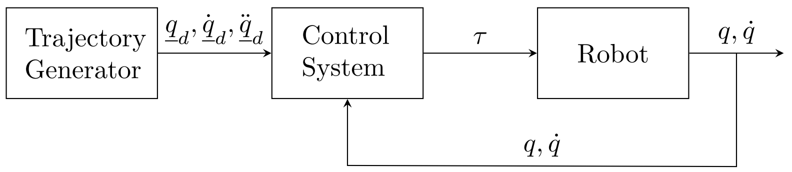
# Computed Torque Control

Glen Henshaw
Craig Carignan

November 12, 2023

## 1 Intro

Here's a block diagram of our robotic system:



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

$$\underline{\tau} = M(\underline{q})\underline{\ddot{q}} + C(\underline{q}, \underline{\dot{q}})\dot{q} + E(\underline{q}, \underline{\dot{q}}) \tag{1}$$

## 2 Gain scheduling

Remember last time we decided to pretend that our dynamics were uncoupled and linear:

$$\tau_i = m_i\ddot{q}_i + c_i\dot{q}_i + e_i(q_i) \tag{2}$$

and we talked about some limitations of that approach. It really only works when either the mass matrix is mostly diagonal and when the joint velocities are relatively low. But it has another problem that we didn't talk about, which is that the full dynamics given in Equation 1 vary with arm pose, whereas the simplified dynamics given in Equation 2 don't. As a consequence, EVEN IF you assume that the dynamics really are uncoupled, the $m_i$, $c_i$, and $e(\cdot)$ terms in Equation 2 change depending on where you are in the workspace. If you go through the exercise we did last lecture where we designed an independent PD control law with a desired stiffness and critical damping, the values of $m_i$ and $c_i$ show up in the calculations for $k_v$ and $k_p$. If you calculate $k_v$ and $k_p$ for some given values of $m_i$ and $c_i$, but then servo the arm to a pose significantly different from the one you started with, you will suddenly not have a system with the desired stiffness or damping. What can you do about that?

One thing you can do is called “robust controller design”, which in intuitive terms is a set of design techniques that help you choose gains that work for your problem over the entire range of $m_i$ and $c_i$ values your robot will assume. For instance, you could calculate both the minimum and maximum values of $m_i$ and $c_i$, pick a minimum and maximum stiffness and that you always want critical damping or overdamping – in other words, that your damping constant $\zeta$ must always be less than 1 – and then find a single set of gains that works.

With serial link manipulators, the range of effective masses you may ancounter can change so dramatically, though, that this is often a futile exercise. You will often be able to find a set of gains that works for your worst case, but it will be so overdamped for your “average” case that the robot can’t really do what you need it to.

One very common way to handle this difficulty is through *gain scheduling*. The idea here is that you pick different operating poses and/or different masses you will encounter – e.g different values of $m_i$ and $c_i$ – and calculate different values of $k_v$ and $k_p$ for each pose. You then switch gains in real time as you get closer to or farther away from different poses and/or as you pick up or drop different payloads.

Formally, this is a table:

| $m_i$ | $c_i$ | $k_p$ | $k_p$ |
|---|---|---|---|
| 1 | 2 | 5 | 3 |
| 5 | 3 | 7 | 4 |
| 3 | 1 | 6 | 3.5 |

This works pretty well for systems where the degrees of freedom are low, or where you don’t need extremely high performance. It relies on the fact – a surprisingly difficult to formally prove fact – that linear PD control is unreasonably effective when applied to a nonlinear Hamiltonian system like a serial joint manipulator.

The major way in which gain scheduling fails is that the number of gain sets you need scales exponentially with the degrees of freedom. So, whereas it may be straightforward if somewhat tedious to find a complete gain schedule (e.g. one that gives you similar performance across the entire dexterous workspace) for a 3 DOF arm, it may become completely intractible for a 7 DOF arm. For instance, if you decided you need, say, ten gain sets for your three DOF arm, you would need 10,000 gain sets for a 6 DOF arm and 100,000 for a 7 DOF arm in order to get roughly equivalent performance.

Nevertheless, especially for manufacturing cell automation where the number of poses you will actually see are small, gain scheduling can be quite effective.

## 3 Computed Torque Control

An alternative is to artificially try to "cancel" the arm's natural dynamics and replace them with ones you like better.

Recall that last time we talked about "open–loop" control, where our dynamics were

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

and our controller was

$$\underline{\tau} = M(\underline{q})\underline{\ddot{q}}_d + C(\underline{q}, \underline{\dot{q}})\dot{q}_d + E(\underline{q}, \underline{\dot{q}})$$

and we said that trying this was a bad idea, because we never really knew the exact values of the mass, Coriolis, and centripetal matrices, and also because there was always noise in our system. But let's play with this idea for a minute. What if we actually did know $M$, $C$, and $E$? In that case we could consider a **closed–loop** control law of the form

$$\underline{\tau} = M(\underline{q})(\underline{\ddot{q}}_d - K_p(\underline{q} - \underline{q}_d) - K_v(\underline{\dot{q}} - \underline{\dot{q}}_d)) + C(\underline{q}, \underline{\dot{q}})\dot{q} + E(\underline{q}, \underline{\dot{q}}) \qquad (3)$$

If we plug Equation 3 into Equation 1 we get:

$$M(\underline{q})\underline{\ddot{q}} + C(\underline{q}, \underline{\dot{q}})\dot{q} + E(\underline{q}, \underline{\dot{q}}) = M(\underline{q})(\underline{\ddot{q}}_d - K_p(\underline{q} - \underline{q}_d) - K_v(\underline{\dot{q}} - \underline{\dot{q}}_d)) + C(\underline{q}, \underline{\dot{q}})\dot{q} + E(\underline{q}, \underline{\dot{q}})$$

or

$$M(\underline{q})(\underline{\ddot{q}} - \underline{\ddot{q}}_d + K_p(\underline{q} - \underline{q}_d) + K_v(\underline{\dot{q}} + \underline{\dot{q}}_d)) + C(\underline{q}, \underline{\dot{q}})(\underline{\dot{q}} - \underline{\dot{q}}) = 0$$

and if we define

$$\begin{aligned} \underline{\tilde{q}} &= \underline{q}_d - \underline{q} \\ \underline{\dot{\tilde{q}}} &= \underline{\dot{q}}_d - \underline{\dot{q}} \\ \underline{\ddot{\tilde{q}}} &= \underline{\ddot{q}}_d - \underline{\ddot{q}} \end{aligned} \qquad (4)$$

then we get

$$\underline{\ddot{\tilde{q}}} + K_v\underline{\dot{\tilde{q}}} + K_p\underline{\tilde{q}} = 0$$

e.g. we would get perfect spring–mass–damper dynamics. And since this is a second–order linear equation, we know how it behaves – as long as we pick $K_p$ and $K_v$ correctly, it asymptotically approaches the origin. Note, also, that because we have complete control over $K_p$ and $K_v$, we can (and almost always do) choose them to be diagonal. Therefore, the dynamics actually are decoupled, e.g. they look like

$$\ddot{\tilde{q}}_i + k_{v,i}\dot{\tilde{q}}_i + k_{p,i}\tilde{q}_i = 0.$$

**It is very important to note** that this equation we have derived is in fact a differential equation that describes the dynamics of the SERVO ERROR. This is not the dynamics of the actual robot; this is the equation that describes how the tracking error of the robot evolves over time. This is a game we often play in nonlinear controls: we try to come up with a control law that gives us a differential equation describing the evolution of the tracking error.

Furthermore, calculating the "model" portion of this controller,

$$M(\underline{q})\ddot{\underline{q}}_d + C(\underline{q}, \dot{\underline{q}})\dot{q}_d + E(\underline{q}, \dot{\underline{q}})$$

is straightforward – it just uses the recursive Newton–Euler dynamics equations from Lectures 13 and 14. This control law is therefore often referred to as a "computed torque" control law, and it is the second most frequently used controller design in robotics, after independent joint control. it is particularly valuable in cases where the robot needs to move quickly and accurately, possibly while handling large masses.

There are some issues with computed torque control, though, the primary one being that we never exactly know the model parameters, and so we can't ever perfectly cancel out the system's natural dynamics. So in practice a computed torque control law actually calculates

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

where $\hat{M}$ is our best guess at the true $M$ matrix, adn so on. If you plug this controller into Equation 1, you end up with the following servo error dynamics:

$$\ddot{\tilde{q}} + K_v\dot{\tilde{q}} + k_p\tilde{q} = \hat{M}^{-1}\left[(M - \hat{M})\ddot{q} + (C - \hat{C})\dot{q} + (E - \hat{E})\right]$$

and it's not immediately obvious what the solutions to this equation look like. In point of fact, as long as the right–hand side is small enough – which is to say, of our model isn't terribly inaccurate – then you can sort of imagine this system as a perturbed spring–mass–damper system, and you can intuitively see that such a system won't stay at the origin, but it *will* stay near the origin. How near depends on the spring constant and the size of the perturbation. Calculating a bound on this requires *Lyapunov theory.*

## 4 Lyapunov Theory

Lyapunov was a 19th century Russian mathematician who pioneered ways of understanding the behavior of nonlinear systems without needing to actually solve them, e.g. without needing to write down closed–form solutions to their differential equations. This is good, because in general we *can't* solve nonlinear differential equations.

The fundamental insight here is that if you can write down an equation for the energy of a system, and if you can prove that the system always dissipates energy, then at the end of the day the system has to be "well behaved". A mechanical system with no energy, for instance, is stationary.

Lyapunov generalized this insight slightly. Lyapunov theory posits that if you can write down **any** function of the state of the system which is always positive except at the origin, and if the time derivative of this equation is always negative except at the origin, then the state has to asymptotically converge to zero. Such a function is called a Lyapunov function. If you think of a Lyapunov function as the energy of the system, or a sort of pseudo–energy of the system, then you'll have the right intuition.

So let's consider the following simplified system plus controller:

$$m\ddot{x} = -b\dot{x} - kx \tag{5}$$

our old friend the spring–mass–damper system. The total energy of this system is

$$v = \frac{1}{2}m\dot{x}^2 + \frac{1}{2}kx^2 \tag{6}$$

and the derivative of this is

$$\dot{v} = m\dot{x}\ddot{x} + kx\dot{x}$$

we can substitute for $m\ddot{x}$ from Equation 5 to get

$$\dot{v} = -b\dot{x}^2 \tag{7}$$

Note that by inspection, Equation 6 is always positive except at $x = 0$, and Equation 7 is always negative except at $\dot{x} = 0$. This implies that the energy in the system must always leave the system until it reaches exactly zero, after which it must remain there. Therefore, without every writing down an explicit solution to Equation 5 we have proven that it must continuously lose energy until it reaches the origin, which is an attractive stable point.

Congratulations, you have just done your first nonlinear stability proof.

OK, let's scale up a bit and see if we can use this approach for the manipulator described by Equation 1 and a simplified computed–torque controller where we assume we know exactly the Coriolis and centripetal forces and the environmental forces but not the mass matrix. We will also assume that we want the manipulator to settle at $\underline{q} = 0$, its home position. And in order to make the math easier, we'll also assume that our mass estimate $\hat{M}$ is constant. This is basically a linear PD control law with an extra term that tries to cancel out environmental forces like gravity. This is an extremely common control design in industrial manipulators.

So our controller is:

$$\tau_c = \hat{M}\left(-K_p\underline{q} - K_v\underline{\dot{q}}\right) + E \tag{8}$$

Substituting Equation 8 into Equation 1, we get

$$\begin{aligned} M\underline{\ddot{q}} + C\underline{\dot{q}} + E &= \hat{M}\left(-K_p\underline{q} - K_v\underline{\dot{q}}\right) + E \\ M\underline{\ddot{q}} + \hat{M}K_p\underline{q} + \hat{M}K_v\underline{\dot{q}} + C\underline{\dot{q}} &= 0 \end{aligned} \tag{9}$$

Because of our godlike intelligence, we're going to pull a candidate Lyapunov function out of the air:

$$V = \frac{1}{2}\underline{\dot{q}}^T M\underline{\dot{q}} + \frac{1}{2}\underline{q}^T\hat{M}K_p\underline{q}$$

which is positive definite.

Taking the derivative, we get

$$\begin{aligned}
\dot{V} &= \frac{1}{2}\underline{\dot{q}}^T \dot{M}\underline{\dot{q}} + \underline{\dot{q}}^T M \underline{\ddot{q}} + \underline{q}^T \hat{M} K_p \underline{\dot{q}} \\
&= \frac{1}{2}\underline{\dot{q}}\dot{M}\underline{\dot{q}} + \underline{\dot{q}}^T \left(-\hat{M} K_p \underline{q} - \hat{M} K_v \underline{\dot{q}} - C\underline{\dot{q}}\right) + \underline{q}^T \hat{M} K_p \underline{\dot{q}} \\
&= \underline{\dot{q}}^T \left(\frac{1}{2}\dot{M} - C - \hat{M} K_v\right)\underline{\dot{q}} \\
&= \underline{\dot{q}}^T \left(\frac{1}{2}\dot{M} - C\right)\underline{\dot{q}} - \underline{\dot{q}}^T \hat{M} K_v \underline{\dot{q}} \qquad (10)
\end{aligned}$$

and, huh, we've got a weird term involving the derivative of the mass matrix and the Coriolis/centripetal forces.

It turns out we can prove that this term is equal to zero. The kinetic energy of the manipulator system can be written in vector form as

$$\underline{\dot{q}}^T M \underline{\dot{q}}$$

Recall that power is the time derivative of kinetic energy. The power for this system can be written in vector form as

$$P = \underline{\dot{q}}^T (\tau - E)$$

which is simply the joint velocities times the external torques on the system. In this case, the external torques are the torques exerted by the controller plus the gravitational and other environmental forces. Therefore:

$$\begin{aligned}
\frac{1}{2}\frac{d}{dt}\left(\underline{\dot{q}} M \underline{\dot{q}}\right) &= \underline{\dot{q}}^T (\tau - E) \\
\frac{1}{2}\underline{\dot{q}}^T \dot{M}\underline{\dot{q}} + \underline{\dot{q}}^T M \underline{\ddot{q}} &= \underline{\dot{q}}^T (\tau - E) \\
\frac{1}{2}\underline{\dot{q}}^T \dot{M}\underline{\dot{q}} + \underline{\dot{q}}\left[\tau - C\underline{\dot{q}} - E\right] &= \underline{\dot{q}}^T (\tau - E) \\
\frac{1}{2}\underline{\dot{q}}^T \dot{M}\underline{\dot{q}} + \underline{\dot{q}}\left[\tau - C\underline{\dot{q}} - E\right] &= \underline{\dot{q}}^T (\tau - E) \\
\underline{\dot{q}}^T \left(\frac{1}{2}\dot{M} - C\right)\underline{\dot{q}} &= 0
\end{aligned}$$

and this is the result we need. We can now cancel the term in Equation 10 to get

$$\begin{aligned}
\dot{V} &= \underline{\dot{q}}^T \left(\frac{1}{2}\dot{M} - C\right)\underline{\dot{q}} - \underline{\dot{q}}^T \hat{M} K_v \underline{\dot{q}} \\
&= -\underline{\dot{q}}^T \hat{M} K_v \underline{\dot{q}}
\end{aligned}$$

which completes the proof. Note that the validity of this proff depends on the choice of gains. In particular, we must choose $K_p$ to be positive definite, and we must choose $\hat{M}K_v$ to be positive definite. It is not immediately obvious whether a given matrix is positive definite, although one way to know for sure is to design it such that it is diagonal with positive values. This suffices for $K_p$. Another way to verify positive definite–ness is to verify that all of the eigenvalues of a matrix are positive.