16-299 Lecture 7: Feedforward Control

So far we have focused on feedback control in this course. We will now shift emphasis to include "feedforward" control, in which some function of the goal is "fed forward" to the plant, independent of the plant state or any feedback.

Consider holding up your arm against gravity. If you just used feedback control, there would have to be an error in order to generate any force to hold your arm up. How could you actually reach the target?

- **Integral Control:** Gradually add force to lift your arm until it reaches the target. If you overshoot, reduce the added force.
- **Model-based Feedforward Control:** Use a model of the arm to predict the added force necessary to hold your arm up at the target, and add that force to the output of a feedback controller.
- **Model Learning:** Remember what effect each added force had, and use this information to create, alter, or correct the model used in model-based feedforward control.
- **Policy Learning:** Remember what added force worked for each target, and based on the target, interpolate to find the amount of added force necessary. The difference between model and policy learning becomes clearer when the model is complicated or expensive to evaluate, and just remembering what command or action to apply skips any model evaluation.
- **Exploration-based learning:** Search for the best added force by trying out different added forces. Use your favorite optimization algorithm to decide what force to try next.

Integral Control

We have focused on regulators so far, with the goal of keeping the plant at an equilibrium point. In this case integral control is often useful:

$$u_{\text{ff}_{next}} = u_{\text{ff}} - k_i * \text{error} \tag{1}$$

However, when the system is moving or changing, integral control is less useful because the correction is based on past rather than current errors. In many cases integral control is turned off during a movement or state change. Integral control is only gradually turned back on, or u_{ff} is set to zero, so there is no discontinuity in the command when integral is turned back on after the movement or change is over. There are also situations where u_{ff} increases to unsafe levels because movement or change is temporarily blocked, or the actuation is not turned on or is saturated. When the problem stops, the large u_{ff} causes a large action. This is known as integrator windup. Careful management of u_{ff} is necessary. The amount of safety code (software) is often much larger than the control code. Ironically, often the safety code itself is a source of bugs, errors, and accidents. I also note that user interface code often dwarfs other types of code, and control experts spend a lot of their effort on safety and user interface programming.

A simple way to manage integral control is to 1) not integrate new errors if the velocity magnitude is above a low threshold of a blockage of the system is detected, and 2) use a leaky integrator so the feedforward command decays over time. On each tick of the controller:

$$u_{\text{ff}_{next}} = 0.99 * u_{\text{ff}} - k_i * \text{error} \tag{2}$$

where the error is set to zero when $|\text{velocity}| > \epsilon$ or blockage is detected.

PID Control

PID (or actually PD) control is a good place to start to get a system up and running. At that point one can start collecting data to indentify what the state dimensions are, build a model, and being optimizing a controller.

The first task is to get the signs right on all the controls. It is nice to have positive force or torque move a degree of freedom (DOF) in the positive direction. Initially one often has to use numerically differentiated position to estimate velocity.

I would gradually turn up the position gain until the system oscillates at the end of a step reponse. I would then try to add enough damping to reduce the oscillation to tolerable levels. I would repeat this process until either A) I couldn't get rid of the oscillations or B) The step response is fast enough. In this initial (PD) phase, a key idea is "Don't be greedy." All you are trying to do is collect enough data to make a model. It is a good idea to use data from different controllers (different control gains) in makeing a model, so that the model does not inadvertently model the controller rather than the plant.

As long as we are talking about PID control, I should mention that in cases where the derivative of the state or the error is not measured directly, one should ultimately use some kind of observer, ideally a Kalman Filter, to estimate velocity. Numerically differentiating the position signal amplifies sensor noise. Adding a low pass filter cleans up the velocity estimate, but also delays it, which degrades controller performance. It is much better to take a state space approach, use a Kalman Filter, and use optimization to choose the controller and state estimator parameters.

State space approaches provide a natural way to handle delayed feedback as well.

Model-based control

A typical attitude towards model-based control is revealed by this quote from the textbook. “It is much better to achieve zero steady-state error by integral action than by feedforward [ie. a prediction from a model], which requires a precise knowledge of process parameters.” (page 11-4)

I must add that not only must the model parameters be known, but the model structure needs to be sufficiently correct as well. People are lazy, and would like to avoid thinking about, identifying, or correctly applying models. The complexity of some models is daunting, and leads many roboticists and AI researchers to view robots as stochastic systems that just don’t do what one wants because of random perturbations.

I feel your pain. However, almost all of the progress in areas like bipedal legged locomotion have been due to more accurate models and better state estimation. Modeling effort pays off. Using some form of learning (which is what integral control is) to make up for a lack of effort on modeling usually turns out to be a bad decision, and not much progress is made. Learning complements modeling, and the “more the agent knows, the better and faster it can learn.”

Robot dynamics

Robot dynamics equations generally have the following form:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \dot{\mathbf{q}}^{\mathrm{T}}\mathbf{C}(\mathbf{q})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{F}(\dot{\mathbf{q}}) = \tau + \mathbf{J}(\mathbf{q})\mathbf{f} \tag{3}$$

where

- $\mathbf{q}$ are the position or configuration variables of the robot.
- $\dot{\mathbf{q}}$ are the velocity variables of the robot.
- $\ddot{\mathbf{q}}$ are the acceleration variables of the robot.
- $\mathbf{M}(\mathbf{q})$ is the inertial matrix of the robot, which depends only on its configuration. It is the multidimensional equivalent of a the one dimensional mass or moment of inertia. You see what it is for a TWIP robot in the assignment.
- $\dot{\mathbf{q}}^{\mathrm{T}}\mathbf{C}(\mathbf{q})\dot{\mathbf{q}}$ represents the Coriolis and centripetal forces, which are quadratic in velocity. $\mathbf{C}()$ only depends on configuration variables.
- $\mathbf{G}(\mathbf{q})$ are the gravitational forces, which depend only on configuration variables.
- $\mathbf{F}(\dot{\mathbf{q}})$ are the frictional forces, which ideally depend only on velocities.
- τ are the joint torques.
- $\mathbf{J}(\mathbf{q})\mathbf{f}$ uses a Jacobian matrix $\mathbf{J}(\mathbf{q})$ to map contact forces $\mathbf{f}$ into joint torques.

My thesis was about learning models

I started graduate school in 1981. At the time, it was thought that learning (identifying) accurate models of robot dynamics was not practical since there were so many parameters. Even if such models could be created, they could not be used for real time control since they were very complex, and thus too computationally expensive. Colleagues worked on learning kinematic models and creating recursive models of robot dynamics which could be evaluated in real time. Now, when computers are a million times faster, it seems silly to worry about computation time, but it was a real issue then.

In my thesis work I wanted to show that accurate dynamic models could be identified. Humans can do amazing things (just watch the Olympics) that would seem to require models (or some stored information), given the delays in the nervous system. I initially set out to identify a global (works for any behavior) model of inertial forces, and then worked on local models for specific behaviors.

A large part of a robot dynamics model tries to capture the rigid body dynamics. My key insight is that for rigid body dynamics the inertial parameters of mass, moments of inertia, and mass moment (the product of the mass and the center of mass ofset from a reference point) appear linearly in the rigid body dynamics equations. Each rigid link has 10 parameters (1 for mass, 3 for mass moment, and 6 for the 3x3 symmetric moment of inertia matrix). Since rigid body dynamics is based on

$$\text{force} = \text{mass} * \text{acceration} \tag{4}$$

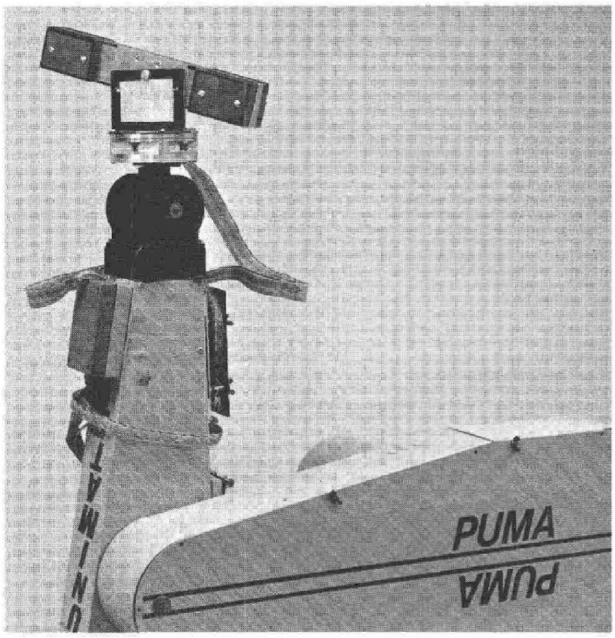
we can see that mass appears linearly, and forces sum linearly. The complexity of these equations comes from expressing the linear and angular accelerations of mass "particles" or "bodies", which are combinations of lengths (offset vectors), trigonometric functions of angles, and quadratic velocity terms. For a single rigid

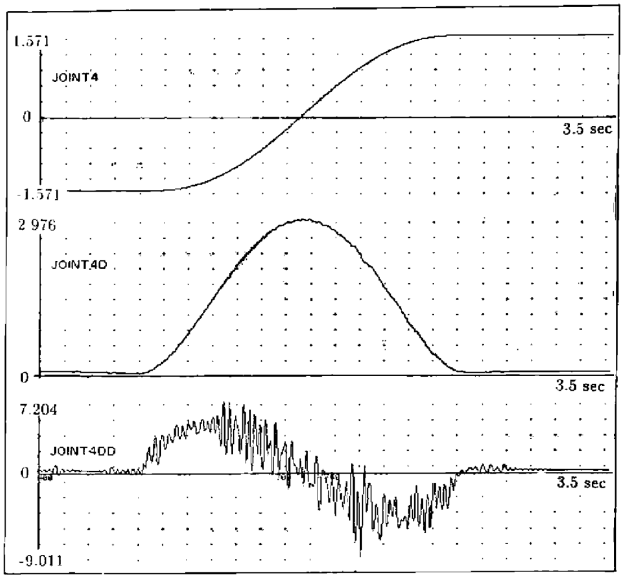
link, we can write the dynamics as

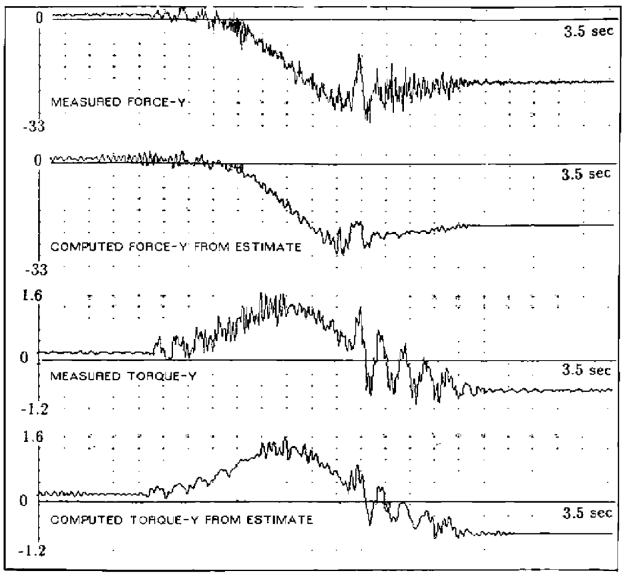
$$\mathrm{MatrixOfAccelerations}(\mathbf{q},\dot{\mathbf{q}})*\begin{pmatrix} m \\ m\mathbf{c}_x \\ m\mathbf{c}_y \\ m\mathbf{c}_z \\ \mathbf{I}_{xx} \\ \mathbf{I}_{xy} \\ \mathbf{I}_{xz} \\ \mathbf{I}_{yy} \\ \mathbf{I}_{yz} \\ \mathbf{I}_{zz} \end{pmatrix} = \mathrm{VectorOfForcesAndTorques}(\mathbf{q}) \quad (5)$$

For a robot we combine equations like these with unknown inertial parameters for all the robot's (rigid) parts. Note that the parameter $m\mathbf{c}_{xx}$ is a single parameter, rather than a product of two parameters, m and $\mathbf{c}_{xx}$. That is what makes the unknown parameters appear linearly in these equations.

Because the motions of the robot's proximal links have limited degrees of freedom (the first link can only rotate around the base joint) if the robot is bolted down, it is not possible to identify all parameters. However, the parameters that can't be identified don't affect the dynamics, and thus don't matter. Similarly, sometimes only linear combinations of parameters can be identified but not the individual parameters. This is not a problem either, since only the linear combination of parameters affect the dynamics. The individual parameters don't independently affect the dynamics.


PUMA
UNIMATION


1.571
JOINT4
0
3.5 sec
-1.571
2.976
JOINT4D
0
3.5 sec
7.204
JOINT4DD
0
3.5 sec
-9.011


0
3.5 sec
MEASURED FORCE-Y
-33
0
3.5 sec
COMPUTED FORCE-Y FROM ESTIMATE
-33
1.6
0
MEASURED TORQUE-Y
3.5 sec
-1.2
1.6
0
COMPUTED TORQUE-Y FROM ESTIMATE
3.5 sec
-1.2

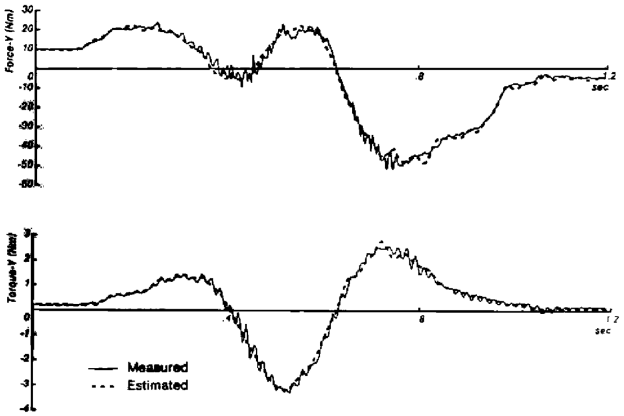
TWIP example.

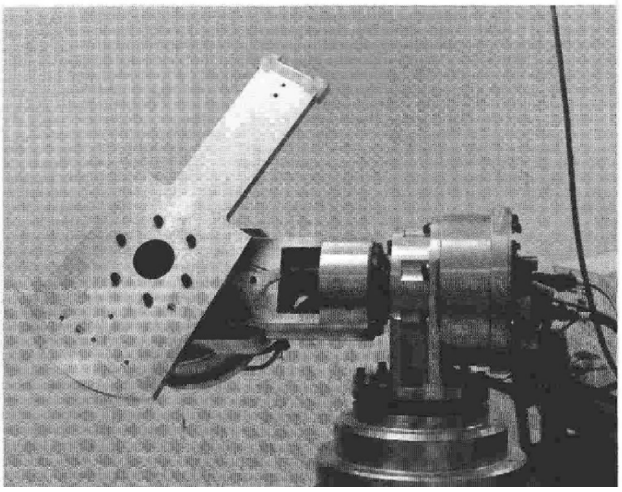
$$\begin{pmatrix} \frac{I_w}{r_w^2} + m_p + m_w & l_p * m_p * \cos(\theta) \\ l_p * m_p * \cos(\theta) & I_p + l_p^2 * m_p \end{pmatrix} \begin{pmatrix} \ddot{x} \\ \ddot{\theta} \end{pmatrix} = \begin{pmatrix} r_w * (-\tau + \dot{\theta}^2 * l_p * m_p * r_w * \sin(\theta)) \\ \tau + \mathrm{G} * l_p * m_p * \sin(\theta) \end{pmatrix} \tag{6}$$

- m_w: $\ddot{x}$
- $\mathbf{I}_w$: $\ddot{x}/r_w^2$
- m_p: $\ddot{x}$
- $l_p m_p$: $(\ddot{x} + \ddot{\theta})\cos(\theta) - \dot{\theta}^2 * r_w^2 * \sin(\theta) - \mathrm{G} * \sin(\theta)$
- $\mathbf{I}_p^*$: $\ddot{\theta}$
- right hand side: $\begin{pmatrix} -r_w * \tau \\ \tau \end{pmatrix}$

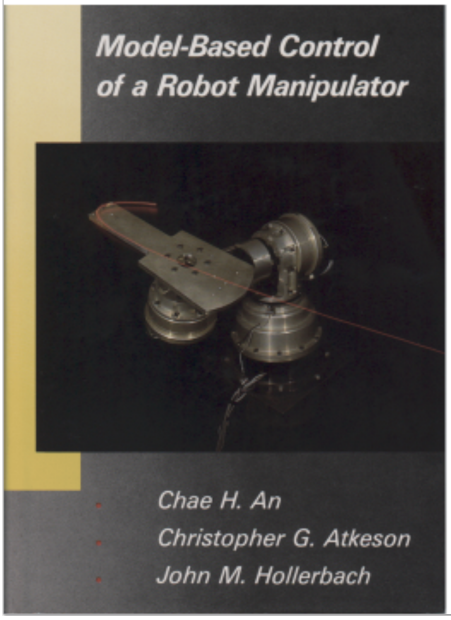
Learning trajectories from practice

Another part of my thesis focused on how robots could learn to execute trajectories perfectly by practicing them. I assumed we had already done the best job we could to identify a global robot model. I believe the secret to human competence is task specific models, models that only attempt to capture a small and local region of behavior space.





Editing Motor Tapes:



Commands

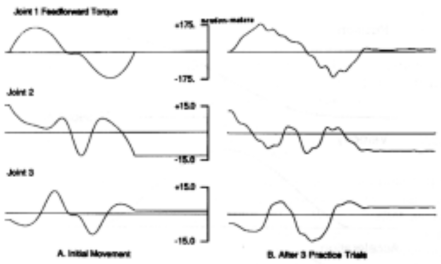


Figure 7.5: Feedforward Torques.

Before

After

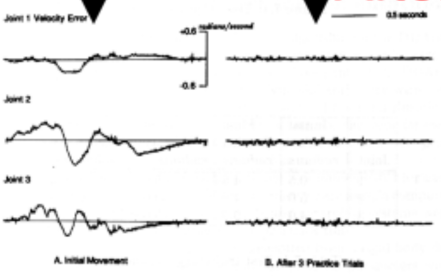


Figure 7.6: Velocity Errors.

Errors

MPC

What models are necessary?

Heavily geared motors isolates joints PID fine.
Actuator dynamics?