Optimal and Hierarchical Control

Travis DeWolf

CNRG - Waterloo

October 16, 2009

What is optimal control?

Optimal control deals with the problem of finding a control law for a given system such that a certain optimality criterion is achieved. A control problem includes a cost functional that is a function of state and control variables.

State variable: $\mathbf{x} = [\mathrm{x} \; ; \; \dot{x}]$

Control variable: **u**

System model

The simplest system model is linear.
Set up a system of linear equations that describe the next state, $\dot{\mathbf{x}}$, and group into system, $\mathbf{A}$, and control, $\mathbf{B}$, matrices.

$$\dot{\mathbf{x}} = \mathbf{Ax} + \mathbf{Bu} \tag{1}$$

$$\mathbf{y} = \mathbf{Cx} \tag{2}$$

In our simple linear model we will be using time-invariant first order dynamics. That is to say, $\mathbf{A}$ and $\mathbf{B}$ are not dependent on time t.

Cost function

To obtain optimality, a cost function is minimized. If you can formulate your cost function in the following way, you have a linear quadratic optimal control problem, which is great.

$$\mathbf{J} = \frac{1}{2}\mathbf{x}^T(t_f)\mathbf{S}_f\mathbf{x}(t_f) + \frac{1}{2}\int_{t_0}^{t_f}(\mathbf{x}^T(t)\mathbf{Q}\mathbf{x}(t) + \mathbf{u}^T(t)\mathbf{R}\mathbf{u}(t))\,dt \tag{3}$$

Why would you want this? Because oodles of work has been done with problems of this form, so you can skip the hard stuff! :D

Where is it applied?

Everywhere!!

Ex: Find the way to drive the car so as to minimize its fuel consumption, given that it must complete a given course in a time not exceeding some amount.

Infinite horizon cost function

What if we have no time limit? Our goal is to simply track a input signal? ie cruise *control*
Then our cost function becomes

$$\mathbf{J} = \frac{1}{2}\int_0^\infty (\mathbf{x}^T(t)\mathbf{Q}\mathbf{x}(t) + \mathbf{u}^T(t)\mathbf{R}\mathbf{u}(t))\,dt \qquad (4)$$

This is known as the infinite horizon cost function.

Why use it to model motor control?

Think of evolution. OK, now doesn't it make sense?

What is hierarchical control?

Hiearchical control is pretty much exactly what you would think. It's a hiearchy of controllers.
The reason this is useful is that as you go up higher and higher you can operate on more and more abstract encompassing or simple dimensions.

For example: Some of Emo Todorov's arm models work from muscle activation control is muscle space to torque control in joint space to end-effector force control in 3D space.

It's way easier to control the end-effector optimally than all the arm muscles.

Where is it applied?

Well, here's the thing: It's pretty difficult, and optimal (or approximately optimal) hierarchical control has just started to be analytically explored.

Why use it to model motor control?

Do you want to control musce activations or end-effector force? GOD.

Optimal control of linear systems

So, remember our linear system?

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \\ \mathbf{y} &= \mathbf{C}\mathbf{x} \end{aligned}$$

Well it's back. And now we're going to use feedback to control it! Oh yes.
So we set up our control signal $\mathbf{u}$ as

$$\mathbf{u} = -\mathbf{F}\mathbf{x} \tag{5}$$

and optimize the feedback gain, $\mathbf{F}$, such that our cost function $\mathbf{J}$ is minimized! Which we can do because $\mathbf{J}$ is *quadratic*.

Oh goodness, that seems like some mathwork

It is! BUT, people have already done it for you, thank heavens. And people will tell you:

$$\mathbf{F}_{opt} = -\mathbf{R}^{-1}\mathbf{B}^T\mathbf{P} \tag{6}$$

where that delightful matrix $\mathbf{P}$ is the positive semi-definite solution of the algebraic Riccati equation:

$$0 = \mathbf{A}^T\mathbf{P} + \mathbf{P}\mathbf{A} + \mathbf{Q} - \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{P} \tag{7}$$

of course! And I'm going to need you to do me a big favor and RTFM if you have any questions about this math right here.

Input signal tracking

I will work through this with you though (kind of)! What if, what if we define a new matrix $\mathbf{T}$ and set that up like this:

$$\mathbf{T} = -\mathbf{C}(\mathbf{A} + \mathbf{BF})^{-1}\mathbf{B}; \tag{8}$$

and now we set up our control signal as

$$\mathbf{u} = \mathbf{F}_{opt}\mathbf{x} + \mathbf{T}^{-T}(\mathbf{TT}^{-1})^{-1}ref \tag{9}$$

Why that would be very sneaky!

A second leap of faith or RTFM

It turns out that when you have

$$\dot{\mathbf{x}} = (\mathbf{A} + \mathbf{B}\mathbf{F}_{opt})\mathbf{x} + \mathbf{B}\mathbf{T}^T(\mathbf{T}\mathbf{T}^T)^{-1}ref \tag{10}$$

what you get for $\mathbf{x}$ is

$$\mathbf{x} = (\mathbf{A} + \mathbf{B}\mathbf{F}_{opt})^{-1}\mathbf{B}\mathbf{T}^T(\mathbf{T}\mathbf{T}^T)^{-1}ref \tag{11}$$

so for $\mathbf{y}$ you get

$$\mathbf{y} = \mathbf{C}(\mathbf{A} + \mathbf{B}\mathbf{F}_{opt})^{-1}\mathbf{B}\mathbf{T}^T(\mathbf{T}\mathbf{T}^T)^{-1}ref \tag{12}$$

Does that first chunk look gd familiar? That's because it's $\mathbf{T}$! Oh my. So then what we have is in fact

$$\mathbf{y} = ref \tag{13}$$

Phew

So this is good. I think we're about ready to look at the first arm model.
Onward ho!

Single-link arm model

Figure: Single-link simple arm model

Single-link physics

Let's design the system! We have a single joint and an arm segment length 1, mass 1.
So we set up our state

$$\mathbf{x} = [\theta\,;\,\dot{\theta}] \tag{14}$$

and using the rotational equivalent of $F = ma$, which is $\tau = I\alpha$, we will derive our **A** and **B** matrices.

System derivation!

So we have

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{\theta} \\ \ddot{\theta} = \frac{\tau}{I=Lm^2} \end{bmatrix} \tag{15}$$

Which we can rearrange, setting $u = \tau$, into

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 \\ \frac{1}{Lm^2} \end{bmatrix} \mathbf{u} \tag{16}$$

Where L and m both equal 1 we then have

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 \\ \frac{1}{Lm^2} \end{bmatrix}$$

The linear controller

Alright, so what we want our system to do is track the input signal, that sounds a lot like that nifty input signal tracker we set up a while ago! Remember that? Let's use it.

So we will find our $\mathbf{F}_{opt}$ by solving the algebraic Riccati equation for $\mathbf{P}$ using MATLABs `are()` command and setting

$$\mathbf{F}_{opt} = R^{-1}B^{T}P$$

as discussed before. Also we create that sneaky $\mathbf{T}$ matrix and set

$$\mathbf{u} = \mathbf{F}_{opt}\mathbf{x} + \mathbf{T}^{T}(\mathbf{T}\mathbf{T}^{T})^{-}1ref$$

so that the system tracks the input signal *ref*.

Single-link arm model linear controller in action

Now I will try to run MATLAB on my desktop from here.

Things to note

$$\mathbf{J} = \frac{1}{2}\int_0^\infty (\mathbf{x}^T(t)\mathbf{Q}\mathbf{x}(t) + \mathbf{u}^T(t)\mathbf{R}\mathbf{u}(t))\,dt$$

In case I didn't show you, a couple things about choosing $\mathbf{Q}$ and $\mathbf{R}$ matrices:

- the larger $\mathbf{Q}$ is relative to $\mathbf{R}$, the more energy is used to quickly arrive at the desired state
- the larger $\mathbf{R}$ is relative to $\mathbf{Q}$, the slower the system arrives at the desired state
- larger $\mathbf{Q}$'s result in overshooting goal and correcting, larger $\mathbf{R}$'s result in slow convergence to goal

Hierarchical control

One of the beautiful things about hierarchical control is that it allows you to abstract the state space to something much prettier than your messy low level system model.

So what happens in your system is that the high level controller works on a more desirable state space and send a control signal down which the low level controller attempts to match.

Depending on how this is done the system may not be actually optimal, but may only be approximately optimal.

Application to motor control

As mentioned before, this applies to motor control nicely in that we can have a system that controls muscle activation or joint torque but be working to optimize the end-effector force.

Pretty nifty I'd say. And I do. Say it.

Two-link arm model

Alright, so, it's a little useless to have a one-link arm model with hierarchical control, so let's look at the two-link model. LET US.

Figure: Two-link simple arm model

The physics

Okeydoke. We're using a very basic model of the two-link arm physics. We'll start by defining our low level system model.

$$\mathbf{x} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} \tag{17}$$

and our control signal will be

$$\mathbf{u} = [\tau_1 \ \tau_2] \tag{18}$$

and now that makes our **A** and **B** matrices

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{19}$$

Low level system model

So our low level system dynamics are

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \\ \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} [\tau_1 \ \tau_2] \quad (20)$$

written compactly as

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$

of course.

Linear low level controller

Again our low level model is linear, and again we can set up a linear controller for it using the same approach as in the single-link arm model: Set up

$$\mathbf{F}_{opt} = R^{-1}B^T P$$

as discussed before. And again create that sneaky $\mathbf{T}$ matrix and set

$$\mathbf{u} = \mathbf{F}_{opt}\mathbf{x} + \mathbf{T}^T(\mathbf{T}\mathbf{T}^T)^-1ref$$

This time $\mathbf{u}$ will be a 2 x 1 vector instead of a scalar value. $\mathbf{x}$ and $\dot{\mathbf{x}}$ are 4 x 1 vectors.

Very simple high level controller

So for our very simple high level controller, all we're going to do is give a command to move the end-effector in a straight line to the target.

This will be converted into joint-angles and become the *ref* signal used by the low level controller.

Another MATLAB example attempt!

Good luck!

A couple things to note

This is a horrible high level controller.

- there is no cost function optimization
- there is no consideration for the underlying system
- very petty; said it hated you one time

-

Let's build some character!

Alright! We'll start by building a model of the lower level cost function.

Boring. Why would we want to do that?

Oh ho! You might think it's a waste of time, BUT

- a cost function model will allow us to realistically model the actual control cost of a high level command
- this allows to take into account all those low level things like going against gravity or other external forces modeled on the low level
- so we get to choose our control signal in a lower-dimensional state space but still take into account important features of the lower level

Control cost

The important thing for us will be to model the control cost.

Wait a tic...

So what about the low level state cost?, you might be saying to yourself, if you're really on the ball. ARE YOU?

Good question!
In this setup though we have *online access* to the low level state model. ie the high level controller working in end-effector space knows what the muscle activations are

Now we will look at changing our low level state to the high level state.

So how do we do it?

Good question!
First let's define our high level state as:

$$\mathbf{y} = \begin{bmatrix} p_x \\ p_y \\ \dot{p_x} \\ \dot{p_y} \end{bmatrix} \tag{21}$$

and our high level control signal will be denoted $\mathbf{v}$, and our high level dynamics will be:

$$\dot{\mathbf{y}} = f(\mathbf{y}) + G(\mathbf{y})\mathbf{v} \tag{22}$$

What are f and G doing?

So now we're not necessarily dealing with a linear system anymore. That is why we use f and G, which are functions of $\mathbf{y}$ instead of matrix representations.

The low level can also be written as a($\mathbf{x}$) and B($\mathbf{x}$)u, but since we're keeping the low level linear we'll stick with the $\mathbf{Ax} + \mathbf{Bu}$ representation we've already set up.

Jacobians!

NOW we are going to define the transition from low level state **x** to high level state **y** and use a Jacobian matrix $J(\theta)$ to represent it. So

$$p_x = L_1 * cos(\theta_1) + L_2 * cos(\theta_1 + \theta_2) \quad (23)$$

$$p_y = L_1 * sin(\theta_1) + L_2 * sin(\theta_1 + \theta_2) \quad (24)$$

Jacobians!

So define our Jacobian

$$J(\theta) = \partial p / \partial \theta \tag{25}$$

explicity written out

$$J(\theta) = \begin{bmatrix} \frac{\partial p_x}{\partial \theta_x} & \frac{\partial p_x}{\partial \theta_y} \\ \\ \frac{\partial p_y}{\partial \theta_x} & \frac{\partial p_y}{\partial \theta_y} \end{bmatrix} \tag{26}$$

Jacobians!

Which gives us:

$$J(\theta) = \begin{bmatrix} -L_1 sin(\theta_1) - L_2 sin(\theta_1 + \theta_2) & -L_2 sin(\theta_1 + \theta_2) \\ L_1 cos(\theta_1) + L_2 cos(\theta_1 + \theta_2) & L_2 cos(\theta_1 + \theta_2) \end{bmatrix}$$

We need to get both p and $\dot{p}$ from the low level to calculate the $f(\mathbf{y})$ of our $\dot{\mathbf{y}}$ equation.

We have our Jacobian $J(\theta)$ and we know

$$J(\theta)\dot{\theta} = \dot{p} \tag{27}$$

and so it follows that

$$\dot{J(\theta)}\dot{\theta} + J(\theta)\ddot{\theta} = \ddot{p} \tag{28}$$

Low to high transformation matrix

We will condense everything needed to transform low state to high state in one matrix **H**

$$\mathbf{H} = \begin{bmatrix} J(\theta) & \mathbf{0} \\ \dot{J}(\theta) & J(\theta) \end{bmatrix} \tag{29}$$

Rewrite $\dot{\mathbf{y}}$

The matrix $\mathbf{H}$ allows us to rewrite the high level dynamics

$$\dot{\mathbf{y}} = f(\mathbf{y}) + G(\mathbf{y})\mathbf{v}$$

as

$$\begin{aligned} \dot{\mathbf{y}} = &\quad \mathbf{H}(a(\mathbf{x} + B(\mathbf{x})\mathbf{u}) && (30) \\ = &\quad \mathbf{HAx} + \mathbf{HBu} && (31) \end{aligned}$$

So what do we have?

Alright, now we have the low level state represented well at the high level, so we can obtain a state cost at the high level representative of the low leve state cost.

NOW we get to model the low level control cost on the high level! Oh man!

Control cost too?!

BUT Travis, you're saying, we've already had so much fun, can't we call it a day?

No you gd cannot. The control cost model is a very important part of your high level model.

BUT

But don't worry, it's not that bad.

Alright, giddyup

Let $r(\mathbf{u})$ represent our low level cost function. The high level will attempt to model the low level control cost written as:

$$r(\mathbf{u}) = \frac{\gamma}{2}\mathbf{u}^T\mathbf{u} \tag{32}$$

with its own approximation $\tilde{r}(\mathbf{v})$.

If we can represent the low level controls, $\mathbf{u}$, using the high level controls, $\mathbf{v}$, then we're golden! So set up something like exactly this:

$$\mathbf{u} = \mathbf{K}(\mathbf{x})\mathbf{v} \tag{33}$$

With this lovely thing we could then set up our high level control cost as

$$\tilde{r}(\mathbf{v}) = \frac{\gamma}{2}\mathbf{v}^T\mathbf{K}(\mathbf{x})^T\mathbf{K}(\mathbf{x})\mathbf{v} \tag{34}$$

For reasons unclear perhaps

We have already defined the relationship between **u** and **v**, did you know?

$$\mathbf{HBu} = G(\mathbf{y})\mathbf{v} \tag{35}$$

For our model we have $\mathbf{G} = \mathbf{B}$; what we want our G(**y**) to do is mimic what the **A** matrix adds to the controlled equations in our low level model. BUT the **A** matrix is all zeros in our low level, so we don't have to add anything. So G(**y**) will just be $\mathbf{G} = \mathbf{B}$

Moving along

So what (35) gives us then is

$$J(\theta)\mathbf{u} = \mathbf{v} \tag{36}$$

Now take the SVD of $J(\theta)$!

$$svd(J(\theta)) = U\Lambda V^T \tag{37}$$

If we remember (or learn) how SVD breaks up, then we know that V^T is the set of operations that takes our input and projects it into task-relevant and task-irrelevant spaces.

When we have $\mathbf{u}$ projected into task-ir and relevant spaces, to minimize our control cost $r(\mathbf{u})$ we want to set the task-irrelevant space equal to 0 and work on minimizing the task-relevant space.

back to K!

So at a high level, we're going to, well, what we're going to do is, we're going to set

$$\mathbf{K} = (U\tilde{\Lambda})^{-1} \tag{38}$$

where $\tilde{\Lambda}$ is the first n_V columns of Λ.

And now our $\tilde{r}(\mathbf{v})$ cost function is going to do a bang up job of approximating r($\mathbf{u}$)! Yeah!

Full low level model on the high level!

Hurray! Now we have our low level modeled in the high level dynamics and cost function. Now we can go about optimizing the high level!

AHA! Thought we were done! Not even close. Oh yeah. Lots more to do still.

High level optimization

How on earth are we going to optimize the high level? We're going to the iterative Linear Quadratic Gaussian (iLQG) optimization techniques developed by Emo Todorov. Fortunately he's got code on his site so after you figure out how the hell it works and adapt it for a hierarchy you're all set!

Two-link with iLQG on top, linear on bottom

Example! maybe

linear on bottom kind of sucks

So the linear on bottom is not very generalizable if we want to upgrade our low level model to say, second order dynamics. So we switch to the MATLAB quadprog function and things get better.

some things

There are oodles of other things to talk about still

- implement constraints on model
- other optimization techniques
- better models
- other stuff

But this is already a long talk, so I'm going to cut it here. If you want to know more after the presentation please don't hesitate to RTFM. I am gd busy.

Questions?