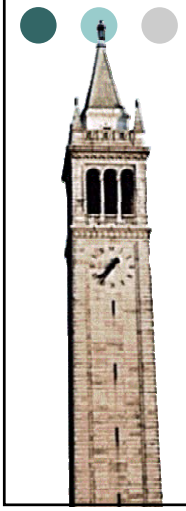


Introduction to Embedded Systems

Edward A. Lee & Sanjit Seshia
UC Berkeley
EECS 124
Spring 2008

Lecture 2: Model-Based Design

Key Concepts in Model-Based Design

- Models describe physical dynamics.
- Specifications are executable models.
- Models are composed to form designs.
- Models evolve during design.
- Deployed code may be generated from models.
- Modeling languages have semantics.
- Modeling languages themselves may be modeled (meta models)

For embedded systems, this is about

- Time
- Concurrency
- Dynamics

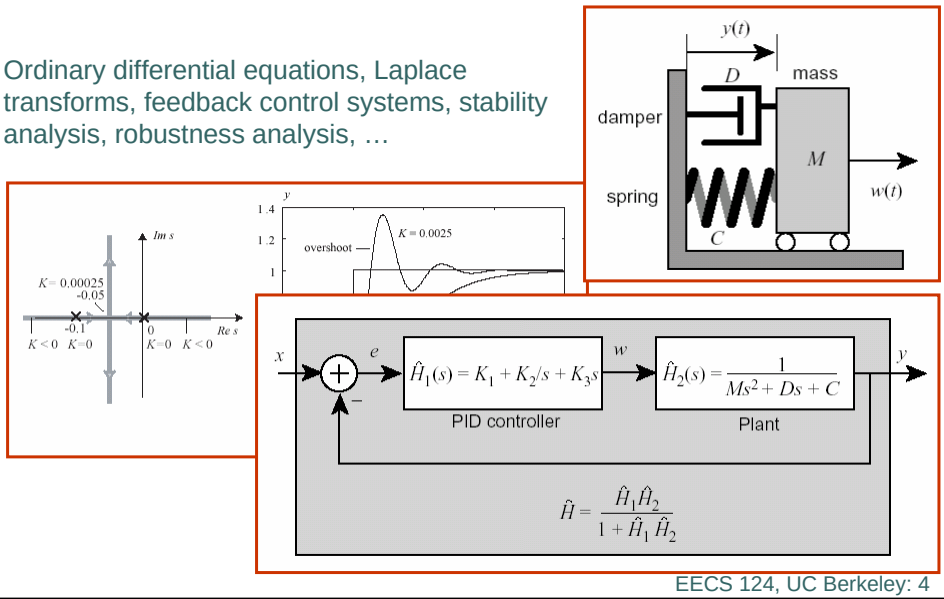
Modeling

Abstraction of physical and cyber dynamics:
(how things change)

- Modeling physical motion
- Modeling physical dynamics
- Feedback control systems
- Modeling modal behavior
- Modeling sensors and actuators
- Modeling software behavior
- Modeling networks

Modeling of Continuous Dynamics

Ordinary differential equations, Laplace transforms, feedback control systems, stability analysis, robustness analysis, …



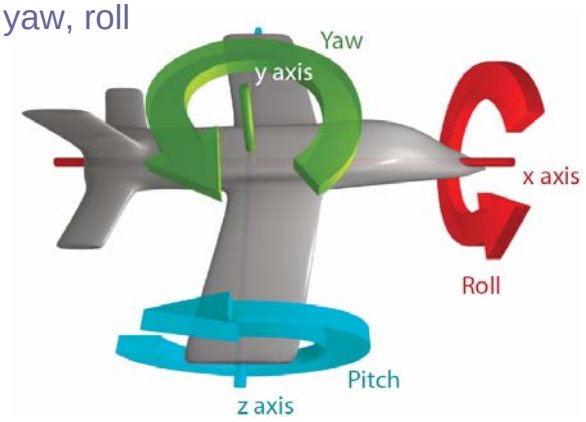
$$\hat{H}_1(s) = K_1 + K_2/s + K_3 s \qquad \hat{H}_2(s) = \frac{1}{Ms^2 + Ds + C}$$

$$\hat{H} = \frac{\hat{H}_1 \hat{H}_2}{1 + \hat{H}_1 \hat{H}_2}$$

Modeling Physical Motion

Six degrees of freedom:

- Position: x, y, z
- Orientation: pitch, yaw, roll



Notation

Position is given by three functions:

$$x\colon \mathbb{R} \to \mathbb{R}$$
$$y\colon \mathbb{R} \to \mathbb{R}$$
$$z\colon \mathbb{R} \to \mathbb{R}$$

where the domain $\mathbb{R}$ represents time and the co-domain (range) $\mathbb{R}$ represents position along the axis. Collecting into a vector:

$$\mathbf{x}\colon \mathbb{R} \to \mathbb{R}^3$$

Position at time $t \in \mathbb{R}$ is $\mathbf{x}(t) \in \mathbb{R}^3$.

Notation

Velocity

$$\dot{\mathbf{x}}\colon \mathbb{R} \to \mathbb{R}^3$$

is the derivative, $\forall\, t \in \mathbb{R}$,

$$\dot{\mathbf{x}}(t) = \frac{d}{dt}\mathbf{x}(t)$$

Acceleration $\ddot{\mathbf{x}}\colon \mathbb{R} \to \mathbb{R}^3$ is the second derivative,

$$\ddot{\mathbf{x}} = \frac{d^2}{dt^2}\mathbf{x}$$

Force on an object is $\mathbf{F}\colon \mathbb{R} \to \mathbb{R}^3$.

Newton's Second Law

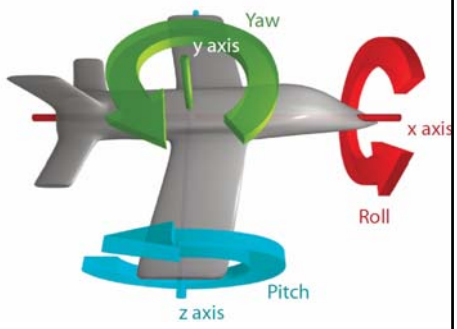
Newton's second law states $\forall\, t \in \mathbb{R}$,

$$\mathbf{F}(t) = M\ddot{\mathbf{x}}(t)$$

where M is the mass. To account for initial position and velocity, convert this to an integral equation

$$\begin{aligned}\mathbf{x}(t) &= \mathbf{x}(0) + \int_0^t \dot{\mathbf{x}}(\tau)d\tau \\ &= \mathbf{x}(0) + t\dot{\mathbf{x}}(0) + \frac{1}{M}\int_0^t\int_0^\tau \mathbf{F}(\alpha)d\alpha d\tau,\end{aligned}$$

Orientation

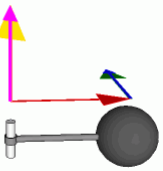


- Orientation: $\theta : \mathbb{R} \rightarrow \mathbb{R}^3$
- Angular velocity: $\dot{\theta} : \mathbb{R} \rightarrow \mathbb{R}^3$
- Angular acceleration: $\ddot{\theta} : \mathbb{R} \rightarrow \mathbb{R}^3$
- Torque: $\mathbf{T} : \mathbb{R} \rightarrow \mathbb{R}^3$

$$\theta(t) = \begin{bmatrix} \dot{\theta}_x(t) \\ \dot{\theta}_y(t) \\ \dot{\theta}_z(t) \end{bmatrix} = \begin{bmatrix} \text{roll} \\ \text{yaw} \\ \text{pitch} \end{bmatrix}$$

Angular version of force is torque.
For a point mass rotating around a fixed axis:

- radius of the arm: $r \in \mathbb{R}$
- force orthogonal to arm: $f \in \mathbb{R}$
- mass of the object: $m \in \mathbb{R}$



$$T_y(t) = r\, f(t)$$

angular momentum, momentum

Just as force is a push or a pull, a torque is a twist.

Units: newton-meters/radian, Joules/radian

Note that radians are meters/meter (2π meters of circumference per 1 meter of radius), so as units, are optional.

Rotational Version of Newton's Second Law

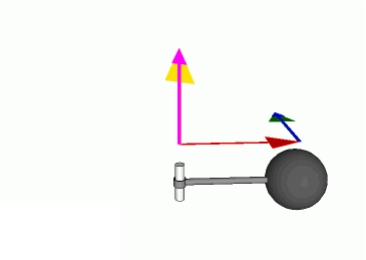
$$\mathbf{T}(t) = \frac{d}{dt}\left(I(t)\dot{\theta}(t)\right),$$

where $I(t)$ is a 3×3 matrix called the moment of inertia tensor.

$$\begin{bmatrix} T_x(t) \\ T_y(t) \\ T_z(t) \end{bmatrix} = \frac{d}{dt}\left(\begin{bmatrix} I_{xx}(t) & I_{xy}(t) & I_{xz}(t) \\ I_{yx}(t) & I_{yy}(t) & I_{yz}(t) \\ I_{zx}(t) & I_{zy}(t) & I_{zz}(t) \end{bmatrix}\begin{bmatrix} \dot{\theta}_x(t) \\ \dot{\theta}_y(t) \\ \dot{\theta}_z(t) \end{bmatrix}\right)$$

Here, for example, $T_y(t)$ is the net torque around the y axis (which would cause changes in yaw), $I_{yx}(t)$ is the inertia that determines how acceleration around the x axis is related to torque around the y axis.

Simple Example



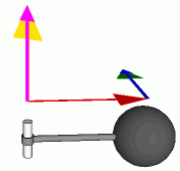
Yaw dynamics:

$$T_y(t) = I_{yy}\ddot{\theta}_y(t)$$

To account for initial angular velocity, write as

$$\dot{\theta}_y(t) = \dot{\theta}_y(0) + \frac{1}{I_{yy}}\int_0^t T_y(\tau)d\tau.$$

Feedback Control Problem

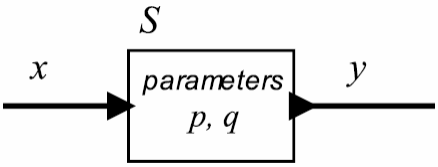


A helicopter without a tail rotor, like the one below, will spin uncontrollably due to the torque induced by friction in the rotor shaft.

Control system problem: Apply torque using the tail rotor to counterbalance the torque of the top rotor.

Actor Model of Systems

A *system* is a function that accepts an input *signal* and yields an output signal.



The domain and range of the system function are sets of signals, which themselves are functions.

$$x: \mathbb{R} \to \mathbb{R}, \quad y: \mathbb{R} \to \mathbb{R}$$

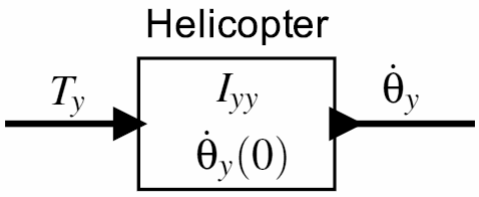
$$S: X \to Y$$

$$X = Y = (\mathbb{R} \to \mathbb{R})$$

Parameters may affect the definition of the function *S*.

Actor model of the helicopter

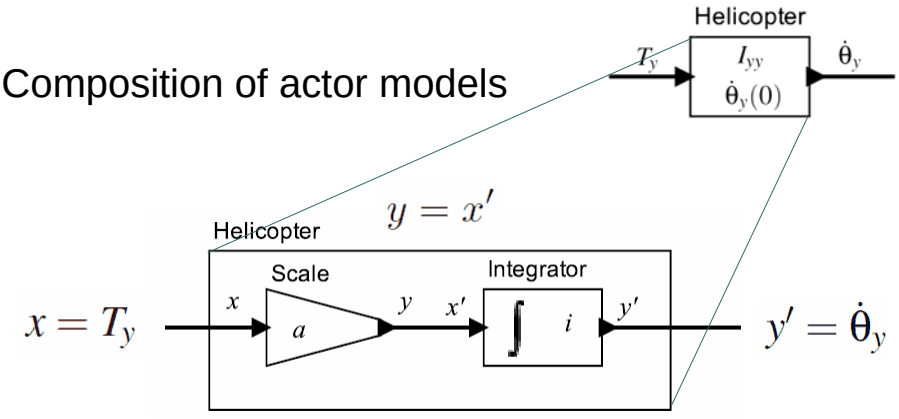
Input is the net torque of the tail rotor and the top rotor. Output is the angular velocity around the *y* axis.



Parameters of the model are shown in the box. The input and output relation is given by the equation to the right.

$$\dot{\theta}_y(t) = \dot{\theta}_y(0) + \frac{1}{I_{yy}} \int_0^t T_y(\tau) d\tau$$

Composition of actor models



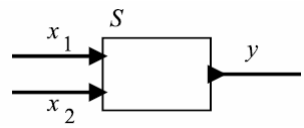
$$\forall\, t \in \mathbb{R}, \quad y(t) = a x(t) \qquad y'(t) = i + \int_0^t x'(\tau) d\tau$$

$$y = ax$$

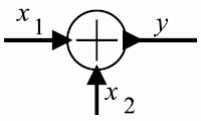
$$a = 1/I_{yy}$$

$$i = \dot{\theta}_y(0)$$

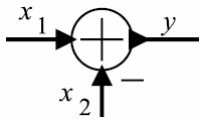
Actor models with multiple inputs



$$S: (\mathbb{R} \to \mathbb{R})^2 \to (\mathbb{R} \to \mathbb{R})$$

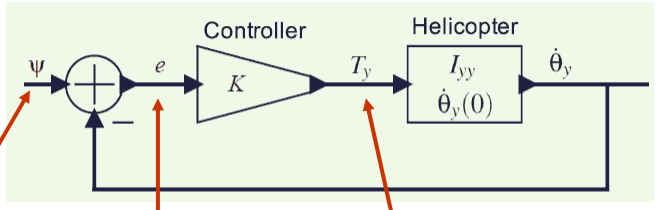


$$\forall\, t \in \mathbb{R}, \quad y(t) = x_1(t) + x_2(t)$$



$$(S(x_1, x_2))(t) = y(t) = x_1(t) - x_2(t)$$

Proportional controller



desired angular velocity

error signal

net torque

$$e(t) = \psi(t) - \dot{\theta}_y(t)$$

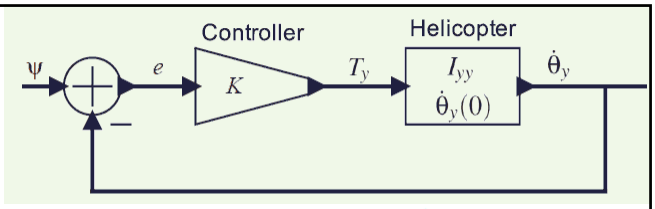
$$T_y(t) = Ke(t)$$

$$\dot{\theta}_y(t) = \dot{\theta}_y(0) + \frac{1}{I_{yy}} \int_0^t T_y(\tau) d\tau$$

$$= \dot{\theta}_y(0) + \frac{K}{I_{yy}} \int_0^t (\psi(\tau) - \dot{\theta}_y(\tau)) d\tau$$

Note that the angular velocity appears on both sides, so this equation is not trivial to solve.

Behavior of the controller



$$\dot{\theta}_y(t) = \dot{\theta}_y(0) + \frac{K}{I_{yy}} \int_0^t (\psi(\tau) - \dot{\theta}_y(\tau)) d\tau$$

Assume that helicopter is initially at rest,

$$\dot{\theta}(0) = 0,$$

and that the desired signal is

$$\psi(t) = au(t)$$

for some constant a.
By calculus (see notes), the solution is

$$\dot{\theta}_y(t) = au(t)(1 - e^{-Kt/I_{yy}})$$

Exercise

Reformulate the helicopter model so that it has two inputs, the torque of the top rotor and the torque of the tail rotor.

Show (by simulation) that if the top rotor applies a constant torque, then our controller cannot keep the helicopter from rotating. Increasing the feedback gain, however, reduces the rate of rotation.

A better controller would include an integrator in the controller. Such controllers are studied in EECS 128.

Questions

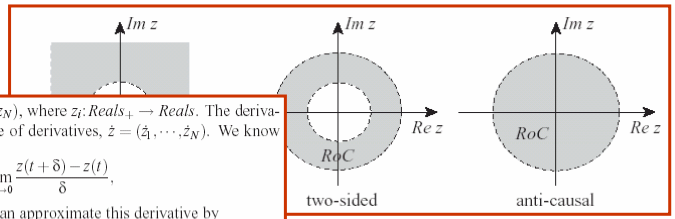
- How do we measure the angular velocity?
- Can this controller be implemented in software?
- How does the behavior change when it is implemented in software?

Discretized Model
A Step Towards Software

Numerical integration techniques provided sophisticated ways to get from the continuous idealizations to computable algorithms.

Discrete-time signal processing techniques offer the same sophisticated stability analysis as continuous-time methods.

But it's not accurate for software controllers (fails on correctness)



In general, z is an N-tuple, $z = (z_1, \cdots, z_N)$, where z_i: $Reals_+ \rightarrow Reals$. The derivative of an N-tuple is simply the N-tuple of derivatives, $\dot{z} = (\dot{z}_1, \cdots, \dot{z}_N)$. We know from calculus that

$$\dot{z}(t) = \frac{dz}{dt} = \lim_{\delta \to 0} \frac{z(t+\delta) - z(t)}{\delta},$$

and so, if $\delta > 0$ is a small number, we can approximate this derivative by

$$\dot{z}(t) \approx \frac{z(t+\delta) - z(t)}{\delta}.$$

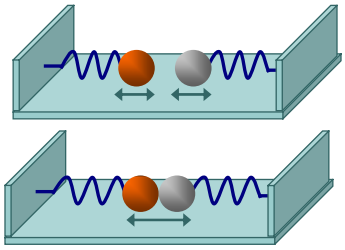
Using this for the derivative in the left-hand side of (5.50) we get

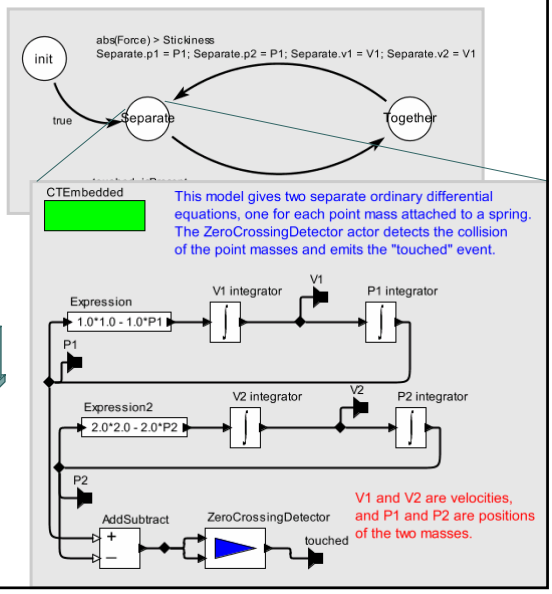
$$z(t+\delta) - z(t) = \delta g(z(t), v(t)). \quad (5.51)$$

Hybrid Systems – Union of Continuous & Discrete

A good starting point, but has limitations.

E.g. Consider building a hybrid system model for software running under a multitasking real-time OS.

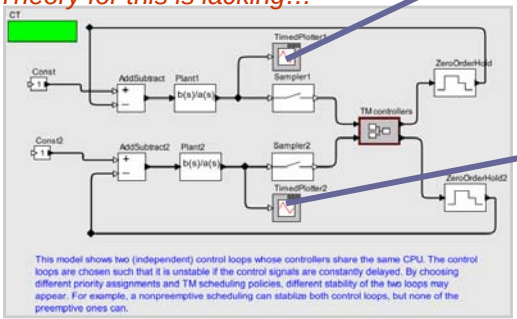


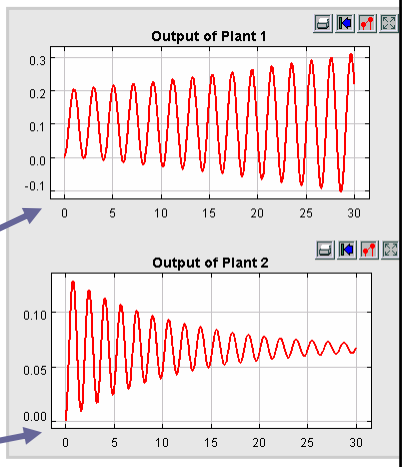


The Timing of Software is the Wrong Thing to Model

An example, due to Jie Liu, has two controllers sharing a CPU under an RTOS. Under preemptive multitasking, only one can be made stable (depending on the relative priorities). Under non-preemptive multitasking, both can be made stable.

Theory for this is lacking…





Model-based design techniques we will talk about

- State machines
  - sequential decision logic
  - amenable to reachability analysis
- Synchronous/reactive concurrent composition
  - concurrent computation
  - composes well with state machines
- Dataflow models
  - exploitable parallelism
  - well suited to signal processing
- Discrete-event models
  - explicit about time
- Time-driven
  - suitable for periodic, timed actions
- Continuous-time models
  - models of physical dynamics
  - extended to “hybrid systems” to embrace computation