Lecture 2:
Optimization-Based Control

Richard M. Murray
Caltech Control and Dynamical Systems
16 March 2009

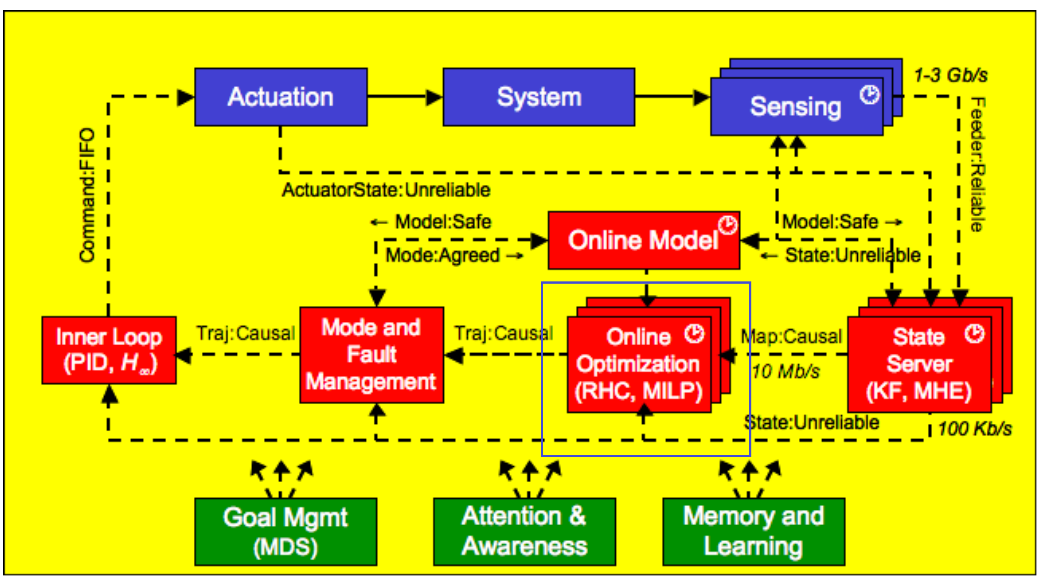
Goals:

- Review trajectory generation and receding horizon control (online optimization)
- Review the Kalman filtering problem for state estimation and sensor fusion
- Describe some implementation tools (spread, pthreads) for optimization-based control

Reading:

- AM08 supplement: *Optimization-Based Control*, Murray, 2009

Networked Control Systems

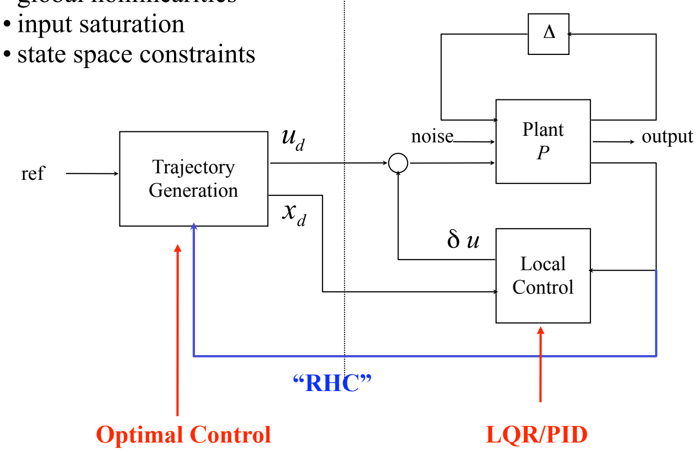


Murray, Hauser et al
SEC chapter (IEEE, 2002)

Control Architecture: Two DOF Design

Nonlinear design
- global nonlinearities
- input saturation
- state space constraints

Local design

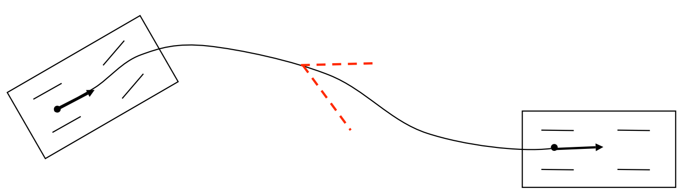


- Use nonlinear trajectory generation to construct (optimal) feasible trajectories
- Use local control to handle uncertainty and small scale (fast) disturbances
- Receding horizon control: iterate trajectory generation during operation

Trajectory Generation Using Differential Flatness

$$\begin{aligned} \dot{x} &= f(x, u) \\ z &= h(x, u, \dot{u}, \ldots, u^{(p)}) \\ |u| &< L \end{aligned} \quad \longleftrightarrow \quad \begin{aligned} x &= x(z, \dot{z}, \ldots, z^{(q)}) \\ u &= u(z, \dot{z}, \ldots, z^{(q)}) \end{aligned}$$

Complicated (algebraic) constraints



$$\bar{z}_0 = \begin{bmatrix} z(0) \\ \dot{z}(0) \\ \ddot{z}(0) \\ \vdots \\ z^{(q)}(0) \end{bmatrix} \quad \overset{z}{\rightsquigarrow} \quad \bar{z}_f = \begin{bmatrix} z(T) \\ \dot{z}(T) \\ \ddot{z}(T) \\ \vdots \\ z^{(q)}(T) \end{bmatrix}$$

$$z = \sum \alpha_i \psi^i(t)$$

$$M\alpha = \begin{bmatrix} \bar{z}_0 \\ \bar{z}_f \end{bmatrix}$$

- Use basis functions to parameterize output ⇒ linear problem in terms of coefficients

Optimal Control Using Differential Flatness

Can also solve constrained optimization problem via flatness

$$\min J = \int_{t_0}^{T} L(x,u)\,dt + V(x(T),u(T))$$

subject to $\quad \dot{x} = f(x,u) \qquad g(x,u) \le 0$

- Input constraints
- State constraints

If system is flat, once again we get an *algebraic* problem:

$$\left.\begin{aligned} x &= x(z, \dot{z}, \ldots, z^{(q)}) \\ u &= u(z, \dot{z}, \ldots, z^{(q)}) \\ z &= \sum \alpha_i \psi^i(t) \end{aligned}\right\} \Rightarrow \left\{\begin{aligned} &\min J = \int_{t_0}^{T} L(\alpha,t)\,dt + V(\alpha) \\ &g(\alpha,t) \le 0 \\ &\text{Finite parameter } \textit{optimization} \text{ problem} \end{aligned}\right.$$

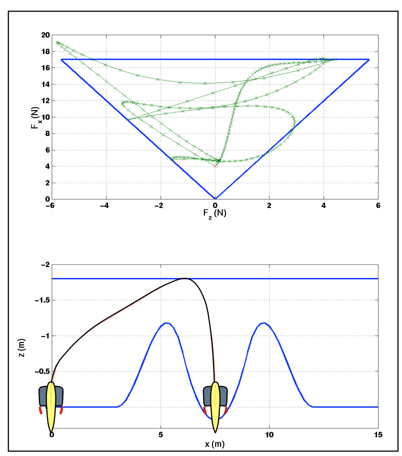
- Constraints hold at all times $\Rightarrow$ potentially over-constrained optimization
- Numerically solve by discretizing time (collocation)

Petit, Milam, Murray
NOLCOS, 2001

NTG: Nonlinear Trajectory Generation

Flatness-based optimal control package

- B-spline representation of (partially) flat outputs
- Collocation based optimization approach
- Built on NPSOL optimization pkg (requires license)
- Warm start capability for receding horizon control



Solves general nonlinear optimization problem

$$\min J = \int_{t_0}^{T} q(x,u)\,dt + V(x(T),u(T))$$

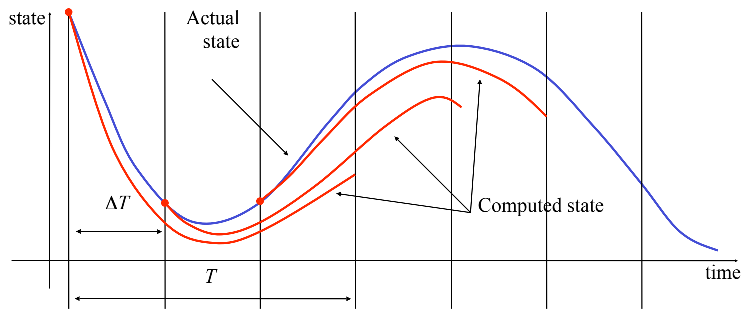
$$\dot{x} = f(x,u) \qquad lb \le g(x,u) \le ub$$

- Assumes x and u are given in terms of (partially) flat outputs
- Constraints are enforced at a user-specified set of collocation points
- Gives *approximate* solution; need to use w/ feedback to ensure robustness (2 DOF)

http://www.cds.caltech.edu/~murray/software/2002a_ntg.html

Murray, Hauser et al
SEC chapter (IEEE, 2002)

Receding Horizon Control



Solve finite time optimization over *T* seconds and implement first Δ*T* seconds

$$u_{[t,t+\Delta T]} = \arg\min \int_t^{t+T} L(x(\tau), u(\tau))\, d\tau + V(x(t+T))$$

$$x_0 = x(t) \quad x_f = x_d(t+T)$$

$$\dot{x} = f(x,u) \quad g(x,u) \le 0$$

Finite horizon optimization

Terminal cost

Requires that computation time be small relative to time horizons

- Initial implementation in process control, where time scales are fairly slow
- Real-time trajectory generation enables implementation on faster systems

Stability of Receding Horizon Control

RHC can destabilize systems if not done properly

- For properly chosen cost functions, get stability with *T* sufficiently large
- For shorter horizons, counter examples show that stability is trickier

Thm (Jadbabaie & Hauser, 2002). Suppose that the terminal cost *V*(*x*) is a control Lyapunov function such that

$$\min_u (\dot{V} + L)(x,u) < 0$$

for each $x \in \Omega_r = \{x\colon V(x) < r^2\}$, for some $r > 0$. Then, for every $T > 0$ and $\Delta T \in (0; T]$, the resulting receding horizon trajectories go to zero exponentially fast.

Remarks

- Earlier approach used terminal trajectory constraints; hard to implement in real-time
- CLF terminal cost is difficult to find in general, but LQR-based solution at equilibrium point often works well - choose $V = x^T P x$ where $P =$ Riccati soln

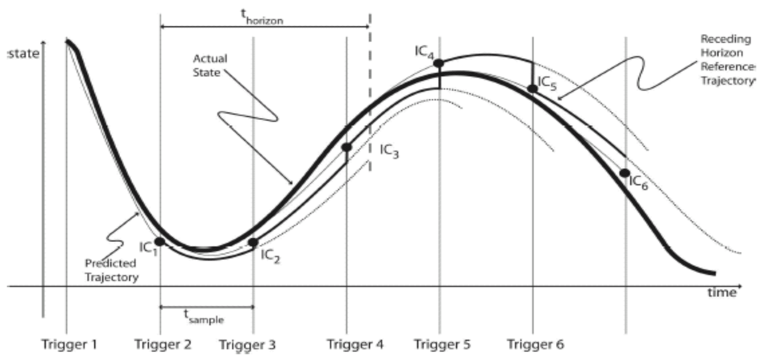
From Real-Time Trajectory Generation to RHC

Three key elements for making RHC fast enough for motion control apps

- Fast computation to optimize over many variables quickly
- Differential flatness to minimize the number of dynamic constraints
- Optimized algorithms including B splines, colocation, and SQP solvers

Use of feedback allows substantial approximation

- Approximate computations since result will be recomputed using actual state
- NTG exploits this principle through the use of collocation

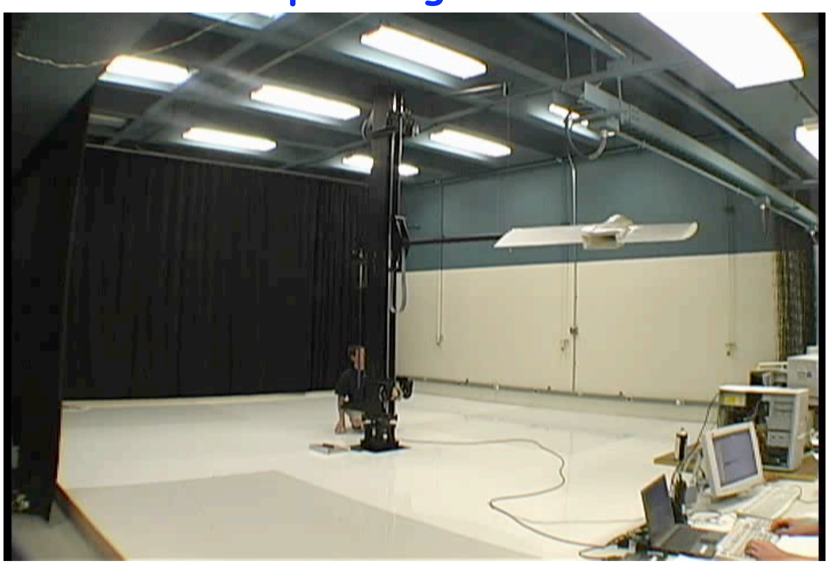


Tuning tricks

- Compute predicted state to account for computation times
- Optimize collocation times and optimization horizon
- Choose sufficiently smooth spline basis

Franz, Milam et al
ACC 2002

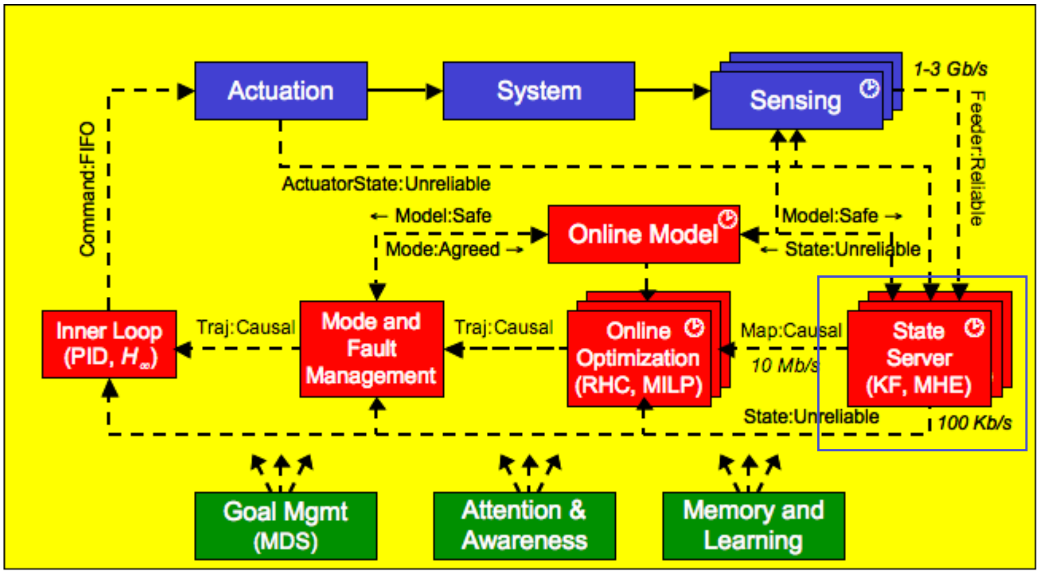
Example: Flight Control



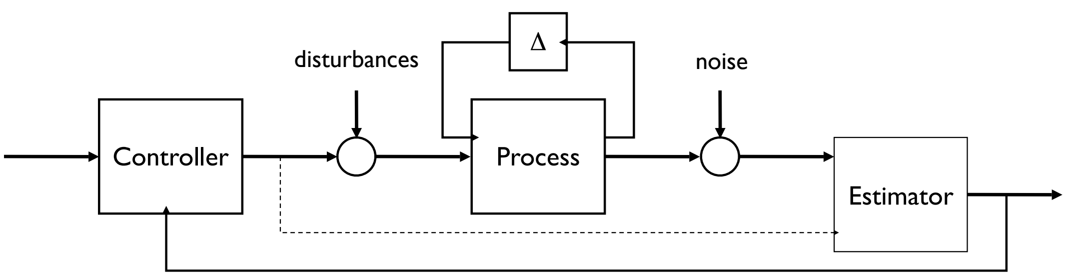
dSPACE-based control system

- Two C30 DSPs + two 500 MHz DEC/Compaq/HP Alpha processors
- Effective servo rates of 20 Hz (guidance loop)

Networked Control Systems



The State Estimation Problem



Problem Setup

- Given a dynamical system with noise and uncertainty, estimate the state

$$\begin{aligned} \dot{x} &= Ax + Bu + Fv \\ y &= Cx + Du + Gw \end{aligned} \longrightarrow \quad \begin{aligned} &\dot{\hat{x}} = \alpha(\hat{x}, y, u) \\ &\lim_{t\to\infty} E\{x - \hat{x}\} = 0 \end{aligned}$$

estimator

expected value

- $\hat{x}$ is called the *estimate* of x

Discrete-time systems

estimator gain

$$\begin{aligned} x[k+1] &= Ax[k] + Bu[k] + Fv[k] \\ y[k] &= Cx[k] + w[k], \end{aligned} \qquad \hat{x}[k+1] = \underbrace{A\hat{x}[k] + Bu[k]}_{\text{prediction}} + \underbrace{L(y[k] - C\hat{x}[k])}_{\text{correction}}$$

Optimal Estimation

System description

$$
\begin{aligned}
x[k+1] &= Ax[k] + Bu[k] + Fv[k] \\
y[k] &= Cx[k] + w[k],
\end{aligned}
$$

$$
E\{v[k]\} = 0
$$

$$
E\{v[k]v[j]^T\} = \begin{cases} 0 & k \neq j \\ R_v & k = j \end{cases}
$$

- Disturbances and noise are multi-variable Gaussians with covariance R_v, R_w

Problem statement: Find the estimate that minimizes the mean square error $E\{(x[k] - \hat{x}[k])(x[k] - \hat{x}[k])^T\}$

Proposition

- For Gaussian noise, optimal estimate is the expectation of the random process x given the *constraint* of the observed output:

$$
\hat{x}[k] = E\{X[k] \,|\, Y[l], l \leq k\}
$$

- Can think of this as a *least squares* problem: given all previous $y[k]$, find the estimate $\hat{x}[k]$ that satisfies the dynamics and minimizes the square error with the measured data.

Kalman Filter

Thm (Kalman, 1961) The observer gain *L* that minimizes the mean square error is given by

$$
L[k] = AP[k]C^T(R_w + CP[k]C^T)^{-1}
$$

where

$$
\begin{aligned}
P[k+1] &= (A - LC)P[k](A - LC)^T + R_v + LR_wL^T \\
P_0 &= E\{X(0)X^T(0)\}.
\end{aligned}
$$

Proof (easy version). Let $P[k] = E\{(\hat{x}[k] - x[k])(\hat{x}[k] - x[k])^T\}$ By definition,

$$
\begin{aligned}
P[k+1] &= E\{x[k+1]x[k+1]^T\} \\
&= AP[k]A^T - AP[k]C^TL^T - LCA^T + L(R_w + CP[k]C^T)L^T.
\end{aligned}
$$

Letting $R_\epsilon = (R_w + CP[k]C^T)$,

$$
\begin{aligned}
P[k+1] = AP[k]A^T &+ \big(L - AP[k]C^TR_\epsilon^{-1}\big)R_\epsilon\big(L - AP[k]C^TR_\epsilon^{-1}\big)^T \\
&- AP[k]C^TR_\epsilon^{-1}CP[k]^TA^T + R_w.
\end{aligned}
$$

to minimize covariance, choose $L = AP[k]C^TR_\epsilon^{-1}$

Kalman Filtering with Intermittent Data

Kalman filter has "predictor-corrector" form

$$\hat{x}[k+1] = A\hat{x}[k] + Bu[k] + L(y[k] - C\hat{x}[k])$$

$$P[k+1] = \underbrace{AP[k]A^T + R_w}_{\text{prediction}} \underbrace{- AP[k]C^T R_\epsilon^{-1} CP[k]^T A^T}_{\text{correction}}$$

- Key idea: updated prediction on each iteration; apply correction when data arrives

Alternative formulation

- Prediction:

$$\hat{x}[k+1|k] = A\hat{x}[k|k] + Bu[k]$$
$$P[k+1|k] = AP[k|k]A^T + FR_v[k]F^T$$

- Correction:

$$\hat{x}[k|k] = \hat{x}[k|k-1] + L[k](y[k] - C\hat{x}[k|k-1])$$
$$P[k|k] = P[k|k-1] - P[k|k-1]C^T(CP[k|k-1]C^T + R_w[k])^{-1}CP[k|k-1]$$

Extension: Information Filter

Idea: rewrite Kalman filter in terms of inverse covariance

$$I[k|k] := P^{-1}[k|k], \qquad \hat{Z}[k|k] := P^{-1}[k|k]\hat{X}[k|k]$$
$$\Omega_i[k] := C_i^T R_{W_i}^{-1}[k] C_i, \qquad \Psi_i[k] := C_i^T R_{W_i}^{-1}[k] C_i \hat{X}[k|k]$$

Resulting update equations become *linear*:

$$\hat{X}[k|k-1] = (1 - \Gamma[k]F^T)A^{-T}\hat{X}[k-1|k-1] + I[k|k-1]Bu$$
$$I[k|k-1] = M[k] - \Gamma[k]\Sigma[k]\Gamma^T[k]$$
$$\boxed{\begin{aligned} I[k|k] &= I[k|k-1] + \sum_{i=1}^{q} \Omega_i[k] \\ \hat{Z}[k|k] &= \hat{Z}[k|k-1] + \sum_{i=1}^{q} \Psi_i[k] \end{aligned}}$$

$$M[k] = A^{-T}P^{-1}[k-1|k-1]A^{-1}$$
$$\Gamma[k] = M[k]F\sigma^{-1}[k]$$
$$\Sigma[k] = F^T M[k] F + R_v^{-1}$$

Remarks

- Information form allows simple addition for correction step: "additional measurements add information"
- Sensor fusion: each additional sensor increases the information
- Multi-rate sensing: whenever new information arrives, add it to the scaled estimate, information matrix; no date => prediction update only
- Derivation of the information filter is non-trivial; not easy to derive from Kalman filter

Henrik Sandberg, 2005

Extension: Moving Horizon Estimation

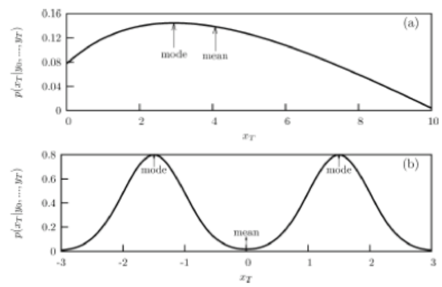
System description:

$$x_{k+1} = f_k(x_k, w_k)$$
$$y_k = h_k(x_k) + v_k \qquad\qquad x_k \in \mathbb{X}_k, \quad w_k \in \mathbb{W}_k, \quad v_k \in \mathbb{V}_k.$$

The problem: *Given* the data

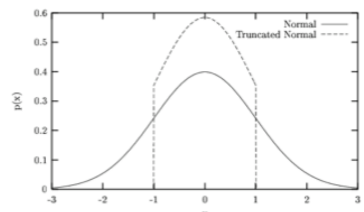
$$Y_k = \{y_i : 0 \le i \le k\},$$

find the "best" (to be defined) estimate $\hat{x}_{k+m}$ of x_{k+m}.
($m = 0$ filtering, $m > 0$ prediction, and $m < 0$ smoothing.



Pose as optimization problem:

$$\{\hat{x}_0, \ldots, \hat{x}_T\} = \arg \max_{\{x_0,\ldots,x_T\}} p(x_0, \ldots, x_T | Y_{T-1})$$



Remarks:

- Basic idea is to compute out the "noise" that is required for data to be consistent with model and penalize noise based on how well it fits its distribution

Extension: Moving Horizon Estimation

Solution: write out probability and maximize

$$\begin{aligned}
&\arg \max_{\{x_0,\ldots,x_T\}} p(x_0, \ldots, x_T | y_0, \ldots, y_{T-1}) \\
&\qquad = \arg \max_{\{x_0,\ldots,x_T\}} p_{x_0}(x_0) \prod_{k=0}^{T-1} p_{v_k}(y_k - h(x_k)) p(x_{k+1}|x_k) \\
&\qquad = \arg \max_{\{x_0,\ldots,x_T\}} \sum_{k=0}^{T-1} \log p_{v_k}(y_k - h_k(x_k)) + \log p(x_{k+1}|x_k) + \log p_{x_0}(x_0)
\end{aligned}$$

Special case: Gaussian noise

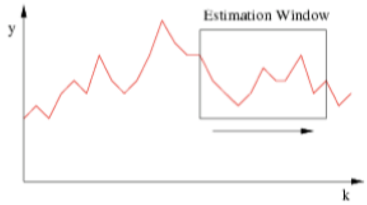
$$\min_{x_0, \{w_0, \ldots, w_{T-1}\}} \sum_{k=0}^{T-1} \|y_k - h_k(x_k)\|^2_{R_k^{-1}} + \|w_k\|^2_{Q_k^{-1}} + \|x_0 - \bar{x}_0\|^2_{P_0^{-1}}$$

- Log of the probabilities sum of squares for noise terms
- Note: switched use of w and v from Friedland (and course notes)

Extension: Moving Horizon Estimation

Key idea: estimate over a finite window in the past

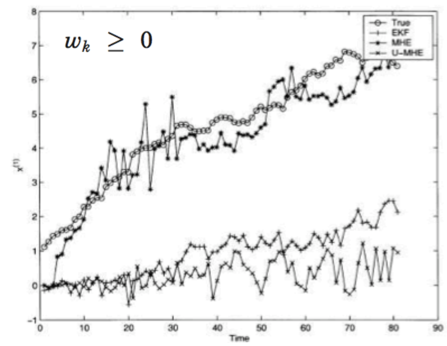
$$\Phi_T^* = \min_{x_0, \{w_k\}_{k=0}^{T-1}} \left(\sum_{k=T-N}^{T-1} L_k(w_k, v_k) + \sum_{k=0}^{T-N-1} L_k(w_k, v_k) + \Gamma(x_0) \right)$$
$$= \min_{z \in \mathcal{R}_{T-N}, \{w_k\}_{k=T-N}^{T-1}} \left(\sum_{k=T-N}^{T-1} L_k(w_k, v_k) + Z_{T-N}(z) \right).$$



Example (Rao et al, 2003): nonlinear model with positive disturbances

$$x_{1,k+1} = 0.99 x_{1,k} + 0.2 x_{2,k}$$
$$x_{2,k+1} = -0.1 x_{1,k} + \frac{0.5 x_{2,k}}{1 + x_{2,k}^2} + w_k$$
$$y_k = x_{1,k} - 3 x_{2,k} + v_k$$

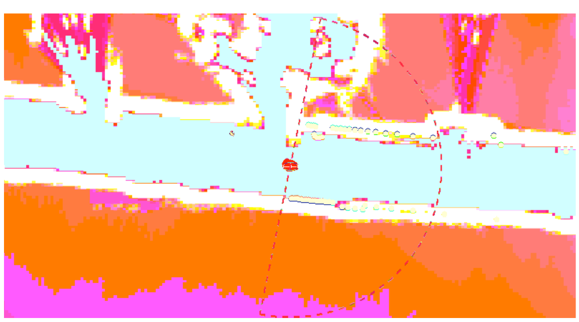
- EKF handles nonlinearity, but assumes noise is zero mean => misses positive drift

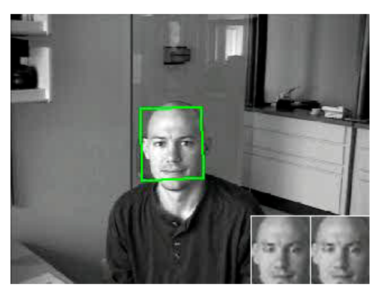


Extension: Particle Filters

Sequential Monte Carlo

- Rough idea: keep track of many possible states of the system via individual “particles”
- Propogate each particle (state estimate + noise) via the system model with noise
- Truncate those particles that are particularly unlikely, redistribute weights

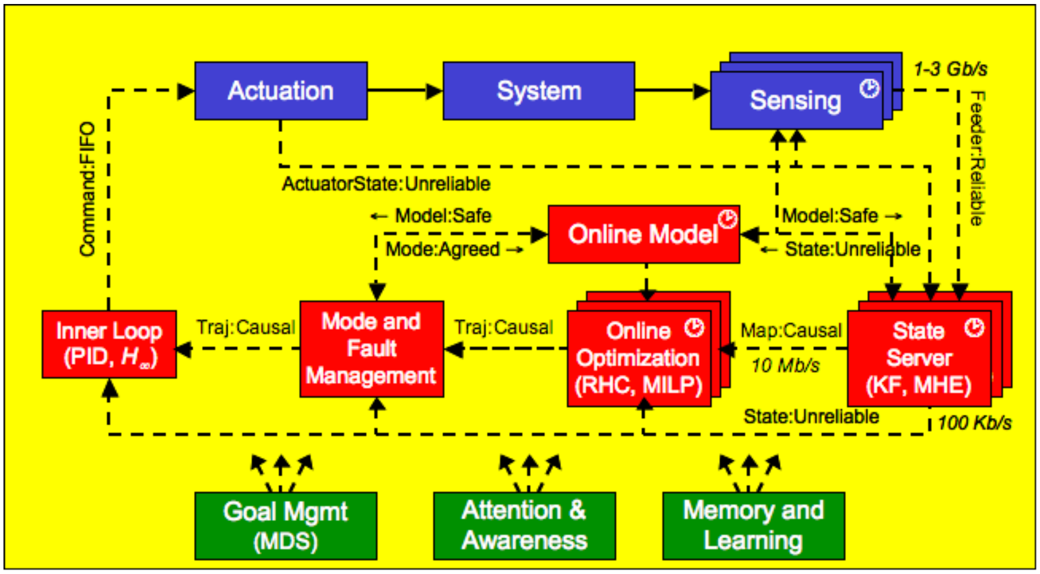




Remarks

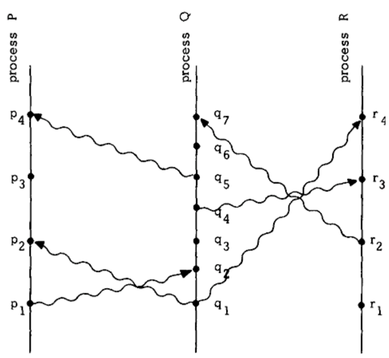
- Can handle nonlinear, non-Gaussian processes
- Very computationally intensive; typically need to exploit problem structure
- Being explored in many application areas (eg, SLAM in robotics)
- Lots of current debate about information filters versus MHE versus particle filters

Networked Control Systems



Next: effects of the *network*...

Causality in Distributed Communications (Lamport, '78)



Partial ordering: *a* → b

- If *a* and *b* are events in the same process, then *a* → *b*
- If *a* is the sending of a message by one process and *b* is the receipt of the same message by another process, then *a* → *b*
- If *a* → *b* and *b* → *c* then *a* → *c*
- *a* → *b* means "*a* can causally effect *b*"

Logical Clocks

- Let $C_i\langle a\rangle$ be a clock for process P_i that assigns a number to an event
- Define $C\langle b\rangle = C_j\langle b\rangle$ if b is an event in process P_j
- *Clock condition*: for any two events *a*, *b*: if *a* → *b* then $C\langle a\rangle < C\langle b\rangle$

Remarks

- Events are *partially* ordered: can compare some events but not all events
- Example: $p_1 \rightarrow q_3$ but p_3 and q_3 are no related
- Clocks are not unique (can choose any set of integers with appropriate relations)

Group Messaging Systems

Group

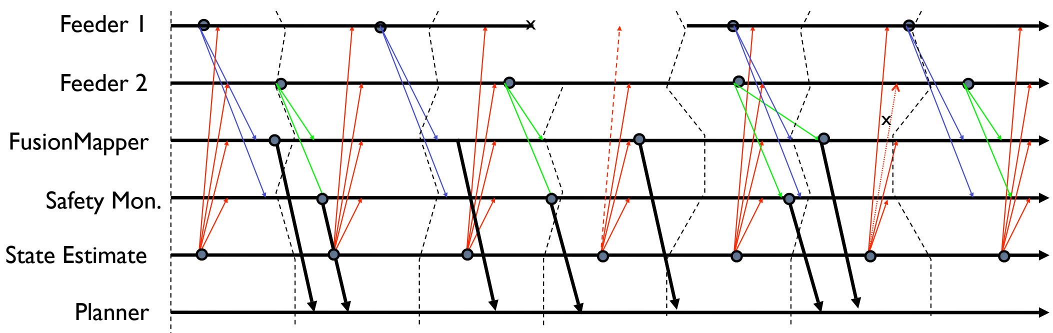
- Collections of processes that can send messages back and forth to everyone
- Messaging system has to keep track of people joining and leaving groups
- Goal: deliver packets reliably and *causally*

Ex: Alice NCS group message types

- Modules receive certain message types

Issues

- Need to track membership over time
- Need to provide different levels of reliability (at the group level)
- Need to provide different levels of ordering (or causality)
- Also need to keep track of the fact that time may be different on different computers (no global clock)



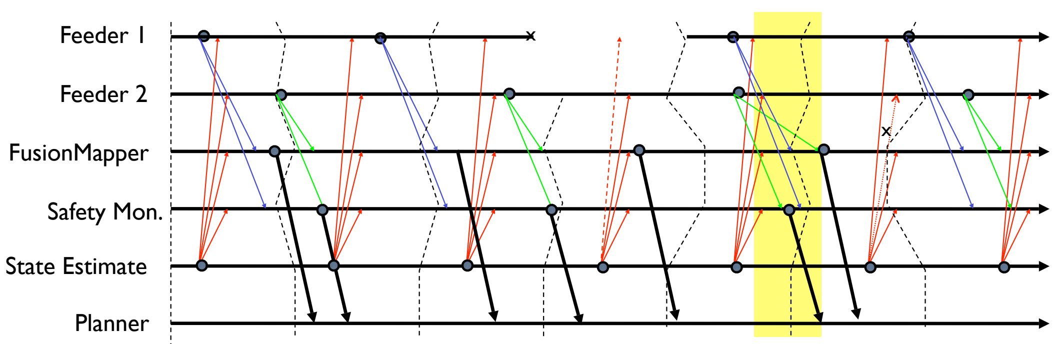
Message Ordering ("Virtual Synchrony")

Ordering

- *None* - No ordering guarantee.
- *Fifo by Sender-* All messages sent by this sender are delivered in FIFO order.
- *Causal* - All messages sent by all senders are delivered in Lamport causal order.
- *Total Order* - All messages sent by all senders are delivered in the exact same order to all recipients

Remarks

- Imposing causality increases message overhead; need to make sure that everyone has the message
- Things get interesting with multiple groups - everyone in same collection of groups should receive all messages in same order
- HW: figure out an example where causal and total order are different



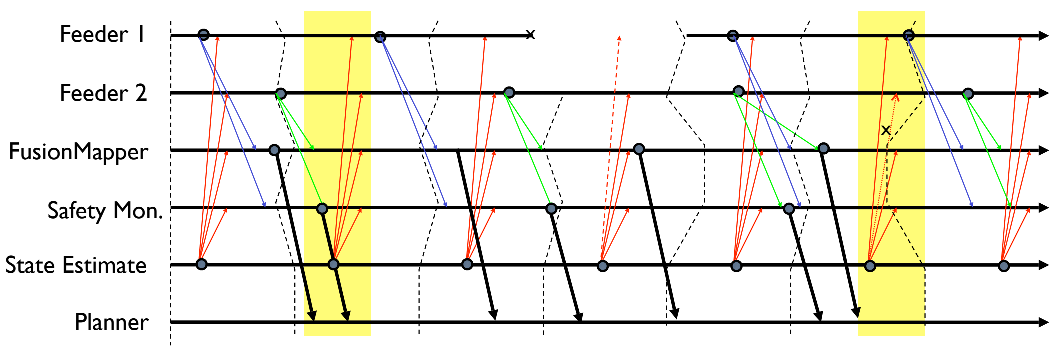
Message Reliability ("Extended Virtual Synchrony")

Reliability

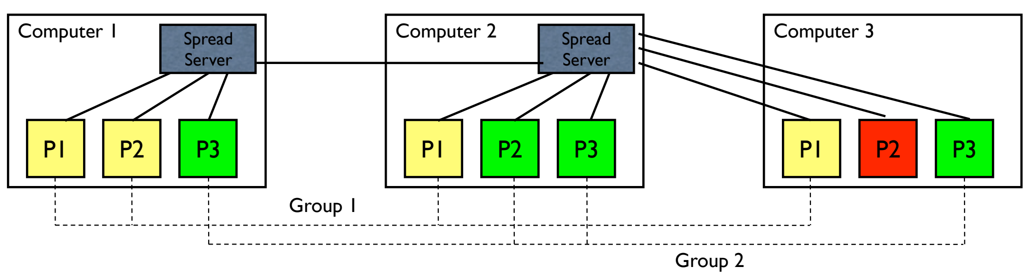
- *Unreliable* - Message may be dropped or lost and will not be recovered.
- *Reliable* - Message will be reliably delivered to all recipients who are in group to which message was sent.
- *Safe* - The message will ONLY be delivered to a recipient if everyone currently in the group definitely has the message

Remarks

- Key issue is keeping track of reliability in *groups*. Reliable messages should be received by everyone (eventually).
- Requires agreement algorithm across computers (who has what)
- HW: find an example where reliable messages are not safe.



Spread Toolkit (Stanton '02)



Spread Functions

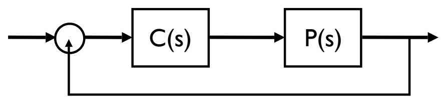
- SP_connect: establish a connection with the spread daemon
- SP_disconnect: terminate connection
- SP_join(mbox. group): join a group
- SP_leave(mbox. group): leave a group
- SP_multicast(…, group, message, type): send a message to everyone in group of given type
- SP_receive: receive a message

Message types

- Unreliable - no order, unreliable
- Reliable - no order, reliable
- FIFO - FIFO by sender, reliable
- Causal - Causal (Lamport), reliable
- Agreed - Totally ordered, reliable
- Safe - Totally ordered, safe

- Note: each message has a type; these can be mixed within groups

Traditional Control Systems Implementation (Sparrow)



$$C(s) \xrightarrow{\text{Tustin}} C(z)$$

Simplest case: interrupt driven loop

- Use HW/SW interrupts to run control routine at an accurate and fixed rate
- "Servo loop" overrides normal program operation
- Need to be careful about interaction of variables in servo loop with main pgm

Variations:

- Time-triggered protocols - scheduling of events to allow multiple "servos"

Sample program

- Discrete time implementation

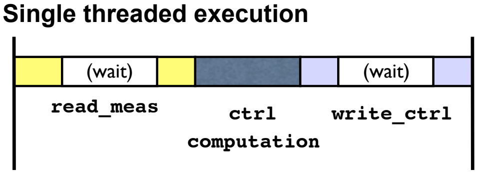
$$z_{k+1} = A_c z_k + B_c e$$
$$y_k = C_c z_k + D_c e$$

- Uses quasi-sparrow implementation

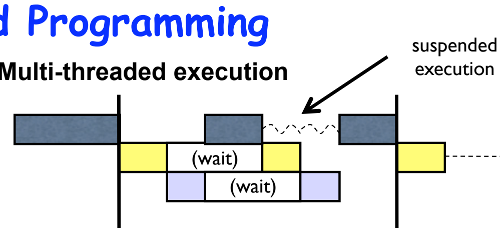
```
load_controller(file)
servo_setup(loop, rate, flags);
servo_enable();

loop()
{
  y = read_measurement();
  r = read_reference();
  xnew = Ac * x + Bc * (r - y);
  u = Cc * x + Dc * (r - y);
  write_control(u);

  x = xnew;
}
```


Multi-Threaded Programming

Single threaded execution



Multi-threaded execution



Basic Idea

- Separate code into independent segments ("threads")
- Switch between threads, allowing each to run "simultaneously"
- Threads share memory and devices; allows rapid sharing of information

Threads vs Processes

- Processes have separate memory space and device handles
- Requires interprocess communication to share data

Advantages

- Avoid manual coding to eliminate pauses due to hardware response
- Multiple control loops become separate threads; OS insures execution
- Allows messages (or signals) to be received in middle of long computation

Issues

- Race conditions
- Dead locks ("deadly embrace")
- Asynchronous operations

Verifying Multi-Threaded Programs

SPIN (Holzmann)

- Model system using PROMELA (Process Meta Language)
  - Asynchronous processes
  - Buffered and unbuffered message channels
  - Synchronizing statements
  - Structured data
- Simulation: Perform random or iterative simulations of the modeled system's execution
- Verification: Generate a C program that performs a fast exhaustive verification of the system state space
- Check for deadlocks, livelocks, unspecified receptions, and unexecutable code, correctness of system invariants, non-progress execution cycles
- Also support the verification of linear time temporal constraints

TLA/TLC (Lamport et al)

- Temporal Logic of Actions (TLA): Leslie Lamport, 1980's
- Behavior (a sequence of states) is described by an initial predicate and an action

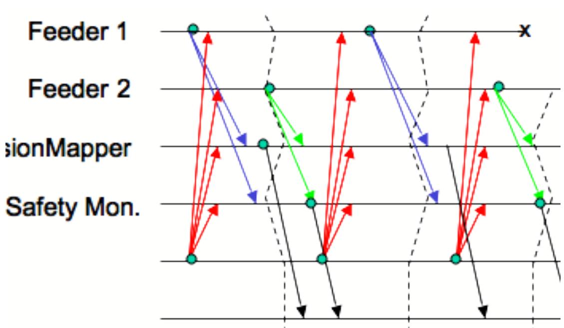
  Spec ≡ Init $\wedge$ $\Box$Action
- Specify a system by specifying a set of possible behaviors
- Theorem: A temporal formula satisfied by every behavior

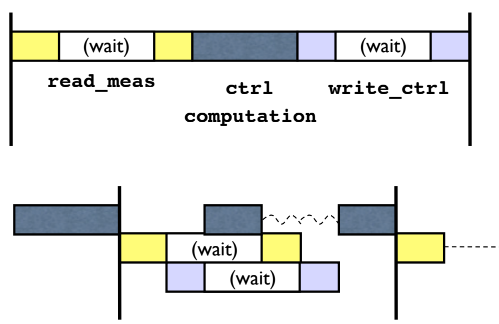
  Theorem ≡ Spec ⇒ $\Box$Properties

TLA+

- Can be used to write a precise, formal description of almost any sort of discrete system
- Especially well suited to describing asynchronous systems
- Tools: Syntactic Analyzer, TLC model checker

Summary: Embedded Systems Programming





Advantages

- Increased modularity
- Simplified programming*

Cautions

- Asynchronous execution
- Race conditions
- Deadlocking
- Debugging

Open Issues for Control Theory

- How do we best implement controllers in this setting?
- How do we verify that programs satisfy the specifications and design intent
- How do we implement multi-rate controllers using threaded process and distributed computing?