Analysis and Design of Hybrid AI/Control Systems

Glen Henshaw, PhD  
(formerly)  
Space Systems Laboratory  
University of Maryland, College Park

13 May 2011
Dynamically Complex Vehicles

- Increased deployment of complex autonomous systems
  - Unpiloted Aerial Vehicles
  - Autonomous Underwater Vehicles
  - Spacecraft
  - Robotic Manipulators (possibly mounted to one of the above)

- Dynamic are much much more complicated than standard laboratory (wheeled) mobile vehicles
Tiered software architectures are useful for automating these systems:

- “High–level” intelligent agents produce trajectories/goals
- “Low–level” control algorithms execute trajectories
  - Controller simplifies the input/output behavior of the dynamical system as seen by the intelligent agent
  - Potentially makes agent’s job easier — reduces scope of behaviors to be accounted for
Hybrid Systems Can Have Nontrivial Dynamics

- Control systems are often treated as a black box
- But the dynamics behavior of a coupled control algorithm/vehicle can be nontrivial
  - Nonzero settling time
  - Overshoot
  - Steady-state offsets
  - Imperfect tracking of complex trajectories

Claim: As a result, undesirable behavior can occur when the controller and agent are linked in a feedback loop even though each module functions correctly in isolation
Insert stories here.
Analysis of a Hybrid System

- Research aim: develop an autonomous proximity operations spacecraft
- Testbed: MPOD, a neutral buoyancy spacecraft simulator
- Facility: University of Maryland’s Neutral Buoyancy Research Facility

Recently named one of US’ “five most awesome college labs” by Popular Science
MPOD Control Algorithms

Standard “PD” linear attitude control algorithm:

\[ \tau_{PD} = -K_d \tilde{\omega} - K_p \tilde{\epsilon} \]

\[ \equiv -K_d \sigma \]

\[ \sigma \triangleq \tilde{\omega} + \lambda \tilde{\epsilon} \equiv \tilde{\omega} + \frac{K_p}{K_d} \tilde{\epsilon} \]

with

\[ \tilde{\epsilon} \] configuration error (here, attitude error)

\[ \tilde{\omega} \] velocity tracking error (here, angular velocity error)

\[ K_p, K_d \] proportional and derivative gains

Three gainsets of low, medium, and high stiffness
MPOD Control Algorithm Performance

- Gainsets tuned to provide progressively faster responses
- Progressively lower tracking error
- Similar overshoot (≈7%)
- Good performance, as measured by classic linear controller performance metrics
Piloted Docking Task

- Pilots instructed to fly MPOD from start position to hard dock with a rigid target
- Desired trajectories generated by human pilots
- Two 3-DOF joysticks, “smoothed” by 1st order low pass filter
- Each pilot flew MPOD to hard dock six times using each controller
Pilot Performance

- Pilot performance strongly dependent on gainset
- “Stiff” PD controller exhibited worst pilot performance; pilot physical and mental workload far higher
- Strange oscillatory episodes observed with stiff PD controller
Pilot–Induced Oscillation
Predicting Undesirable Interactions

- Similar problem seen in aircraft industry: pilot-induced oscillations (PIO)
- Result of the interaction between the pilot (a non–mathematical system!) and a vehicle/control system
- Rapid oscillatory motion, often with catastrophic results
- Often seen with
  - Experienced pilots (including test pilots and astronauts)
  - Variety of aircraft: F—15, YF—22, MD—11, C—17, Space Shuttle, etc
  - Often when performing high precision operations such as landing
- Tools for analyzing system dynamical behavior are mathematical
- Most intelligent agents (including humans) are inherently non–mathematical

How do you mathematically analyze a non–mathematical system?
Mathematical Analysis of MPOD PIO

- Analysis of PIO requires mathematical models of vehicle, controller, and pilot

- Most widely accepted pilot model: the “crossover model”*
  - Represents actions of pilot performance setpoint maintenance task (e.g. maintaining a heading)
  - Pilot acts as a linear PD controller with time delay

PD Controllers Can Lead To PIO

- Root locus plot indicates instability for stiff PD system with large pilot gains
- Apparent C.L. pole locations (represented by '*'s) indicate highly oscillatory system
Preventing Undesirable Interactions

What can be done to prevent undesirable interactions?

Idea —
Linear control algorithms are simple:

\[ \tau_{PD} = -K_d \sigma \]

But their closed-loop dynamic are complex:

\[ G_{CL} (s) = \frac{K}{s (s + 2)} \times \frac{\omega_n^2 \left( \frac{2}{\lambda} s + 1 \right)}{s^2 + 2\zeta \omega_n s + \omega_n^2} \]

Is there any way to simplify the closed-loop behavior?
Control Algorithms With Better Performance

Some nonlinear control algorithms exhibit better performance:

\[
\tau_{NL} = -K_d \sigma + H(\epsilon) \dot{\omega}_r + C(\epsilon, \omega) \omega_r + E(\epsilon, \omega)
\]

\[
\omega_r = \omega - \sigma
\]

where

- \(H\) — Inertia matrix
- \(C\) — Coriolis and centripetal forces
- \(E\) — Environmental forces (drag, gravity, etc.)

(theoretically) guarantees asymptotically perfect tracking of arbitrarily complex desired trajectories (assuming continuous second derivatives).
Nonlinear Controller Seems to Eliminate PIO

- Asymptotically perfect tracking (theoretically) implies that controller causes closed--loop dynamics to become trivial
- Root--locus plot always stable

![Nonlinear Root Locus](image-url)
Nonlinear controllers reduce tracking error moderately...

But they improve pilot performance a lot.
What’s The Difference?

Linear control algorithms are simple:

\[ \tau_{PD} = -K_d \sigma \]

But they lead to complex closed-loop dynamics:

\[ G_{CL}(s) = \frac{K}{s(s+2)} \times \frac{\omega_n^2 \left( \frac{2}{\lambda} s + 1 \right)}{s^2 + 2\zeta \omega_n s + \omega_n^2} \]

Nonlinear control algorithms are more complex:

\[ \tau_{NL} = -K_d \sigma + H(\epsilon) \dot{\omega}_r + C(\epsilon, \omega) \omega_r + E(\epsilon, \omega) \]

But they exhibit much simpler closed-loop dynamics:

\[ G_{CL}(s) = \frac{K}{s(s+2)} \]
Conclusions

* Humans are the “gold standard” of intelligent agents, but unexpected behaviors can emerge even in piloted systems

* Similar behaviors can certainly emerge from more autonomous tiered systems

* Initial claim is verified: validating hardware/software components in isolation is not sufficient to guarantee desired performance
Lessons Learned

* Tools for analyzing complex software systems are not well developed
* But...
* Taking PIO analysis as inspiration, mathematically approximating non–mathematical systems can be effective and give important insights into system behavior
* System behavior is greatly simplified when individual component input/output behavior is as simple as possible
  * Simple components often do not lead to simple behaviors or simple interactions
* Differential equations are your friends