

Basics of DHP type Adaptive Critics / Approximate Dynamic Programming and some application issues

TUTORIAL

George G. Lendaris

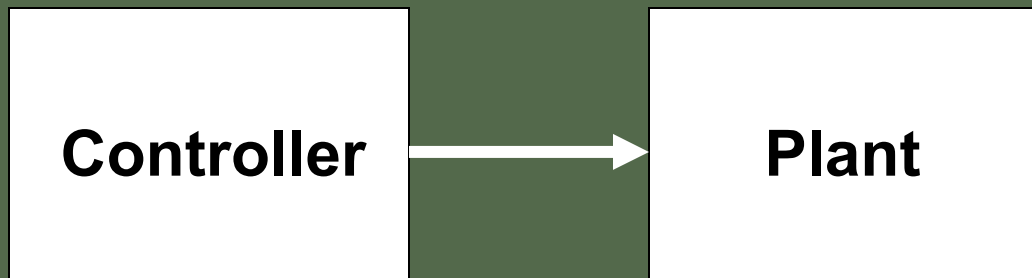
NW Computational Intelligence Laboratory

Portland State University, Portland, OR

April 3, 2006



Basic Control Design Problem



Controller designer needs following:

- **Problem domain specifications**
- **Design objectives / Criteria for “success”**
- **All available *a priori* information about Plant and Environment**

High Level Design Objective: “Intelligent” Control

In this tutorial, we focus on
Adaptive Critic Method (ACM)

A methodology for designing an (approximately) optimal controller for a given plant according to a stated criterion, via a **learning process**.

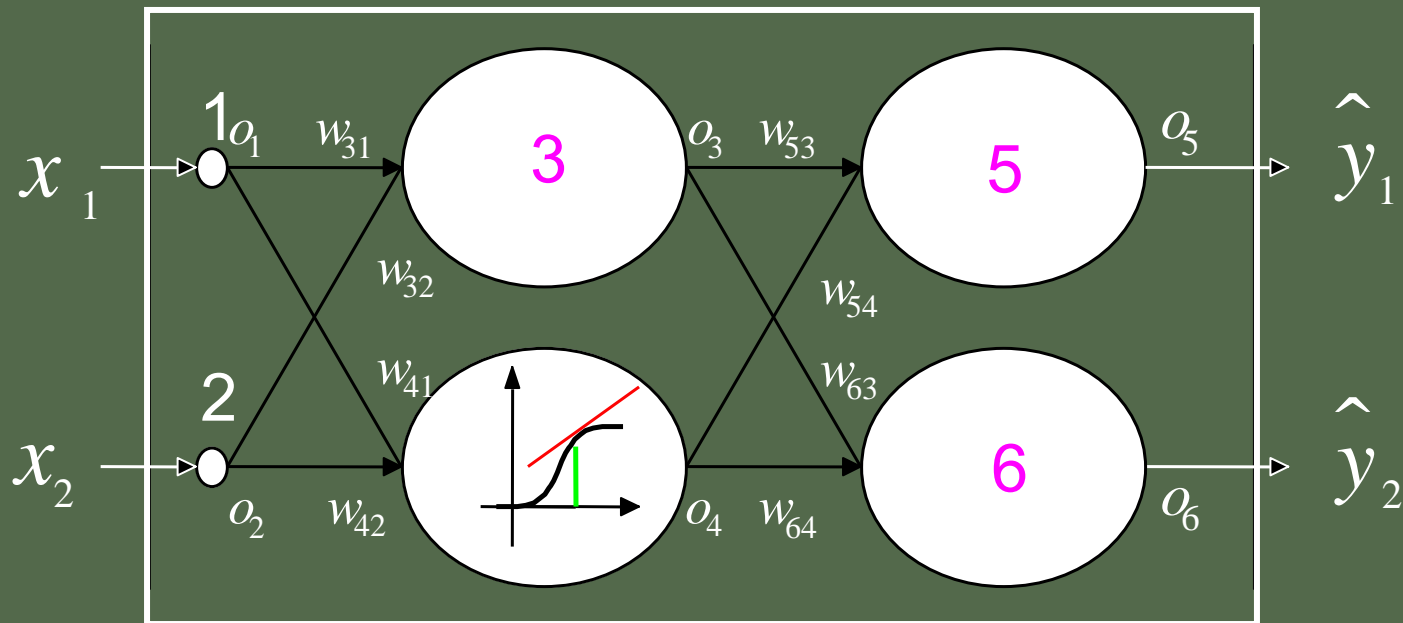
ACM may be implemented using two **neural networks** (also *Fuzzy systems*):

---> one in role of **controller**, and

---> one in role of **critic**.



Simple, 1-Hidden Layer, Feedforward Neural Network [to implement $\underline{y}=f(\underline{x})$]



Can obtain partial derivatives via Dual of trained NN



Overview of Adaptive Critic method

User provides the

Design objectives / Criteria for “success”

through a Utility Function, $U(t)$ (local cost).

Then, a new utility function is defined (**Bellman Eqn.**),

$$J(t) = \sum_{k=0}^{\infty} \gamma_k U(t+k) \quad [\text{“cost to go”}]$$

which is to be minimized [**~ Dynamic Programming**].

[We note: $J(t) = U(t) + \gamma J(t+1)$ ← **Bellman Recursion**]



Family of Adaptive Critic Methods:

The *critic* approximates either $J(t)$ or the gradient of $J(t)$ wrt state vector $R(t)$ [$\nabla J(R)$]

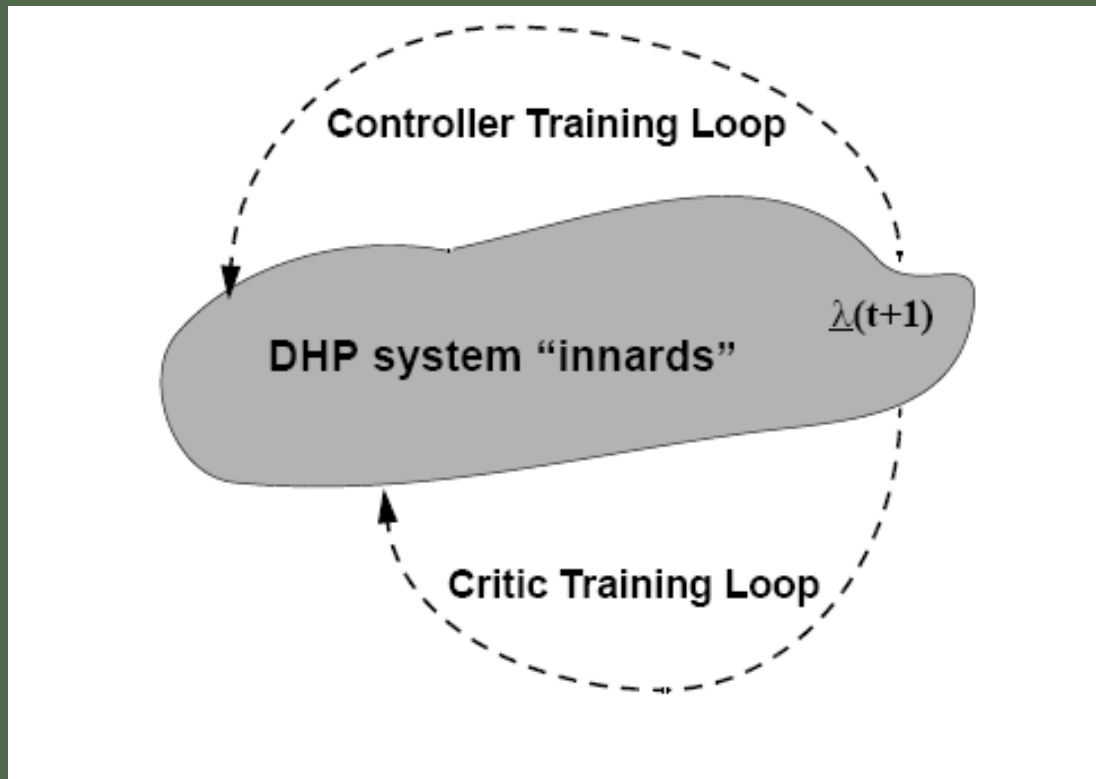
Two members of this “family”:

- Heuristic Dynamic Programming (HDP)
Critic approximates $J(t)$
(cf. “Q Learning”)
- Dual Heuristic Programming (DHP)
Critic approximates $\nabla J(R(t)) \equiv \underline{\lambda}(t)$

[Today, focus on DHP]



May describe DHP training process via two primary feedback loops:



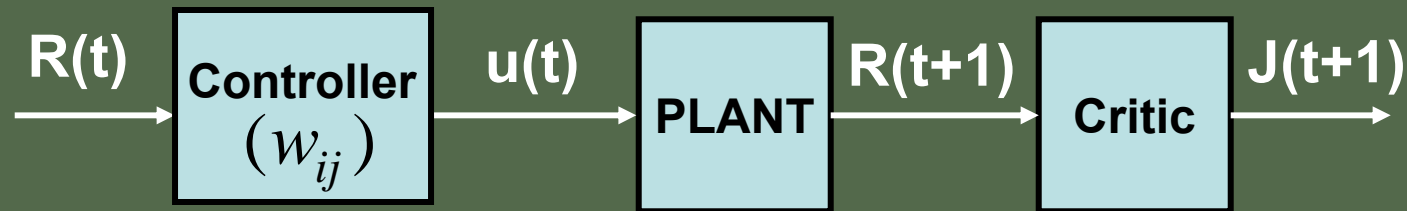
“System Innards” comprise:

*controller*NN, Plant, *critic*NN, Utility Function, etc.

1. The ***controller training loop***. A Supervised Learning process in which the controller is trained to minimize the performance measure $J(t)$ of the control problem, based on data from the critic [called $\underline{\lambda}(t + 1)$].

2. The ***critic training loop***. The process wherein the *critic* neural network learns to approximate the derivatives of the performance measure $J(t)$, which are used in the *controller training loop*.

To develop feel for the weight update rule, consider a *partial* block diagram and a little math (discrete time):



Desire a training “Delta Rule” for w_{ij} to minimize cost-to-go $J(t)$.

Obtain this via $\frac{\partial J(t)}{\partial w_{ij}(t)}$ and the chain rule of differentiation.



The weights in controller NN are updated with objective of minimizing $J(t)$:

$$\Delta w_{ij}(t) = -lcoef \cdot \frac{\partial J(t)}{\partial w_{ij}(t)}$$

where

$$\frac{\partial J(t)}{\partial w_{ij}(t)} = \sum_{k=1}^a \frac{\partial J(t)}{\partial u_k(t)} \cdot \frac{\partial u_k(t)}{\partial w_{ij}}$$

and

$$\frac{\partial J(t)}{\partial u_k(t)} = \frac{\partial U(t)}{\partial u_k(t)} + \frac{\partial J(t+1)}{\partial u_k(t)}$$

and

$$\frac{\partial J(t+1)}{\partial u_k(t)} = \sum_{s=1}^n \frac{\partial J(t+1)}{\partial R_s(t+1)} \cdot \frac{\partial R_s(t+1)}{\partial u_k(t)}$$

Call this term $\lambda_s(t+1)$

(to be output of critic)



It follows that **Controller** training is based on:

$$\frac{\partial J(t+1)}{\partial u_k(t)} = \frac{\partial U(t)}{\partial u_k(t)} + \sum_{s=1}^n \frac{\partial J(t+1)}{\partial R_s(t+1)} \cdot \frac{\partial R_s(t+1)}{\partial u_k(t)}$$

Via CRITIC

Via Plant Model

Similarly, **Critic** training is based on:

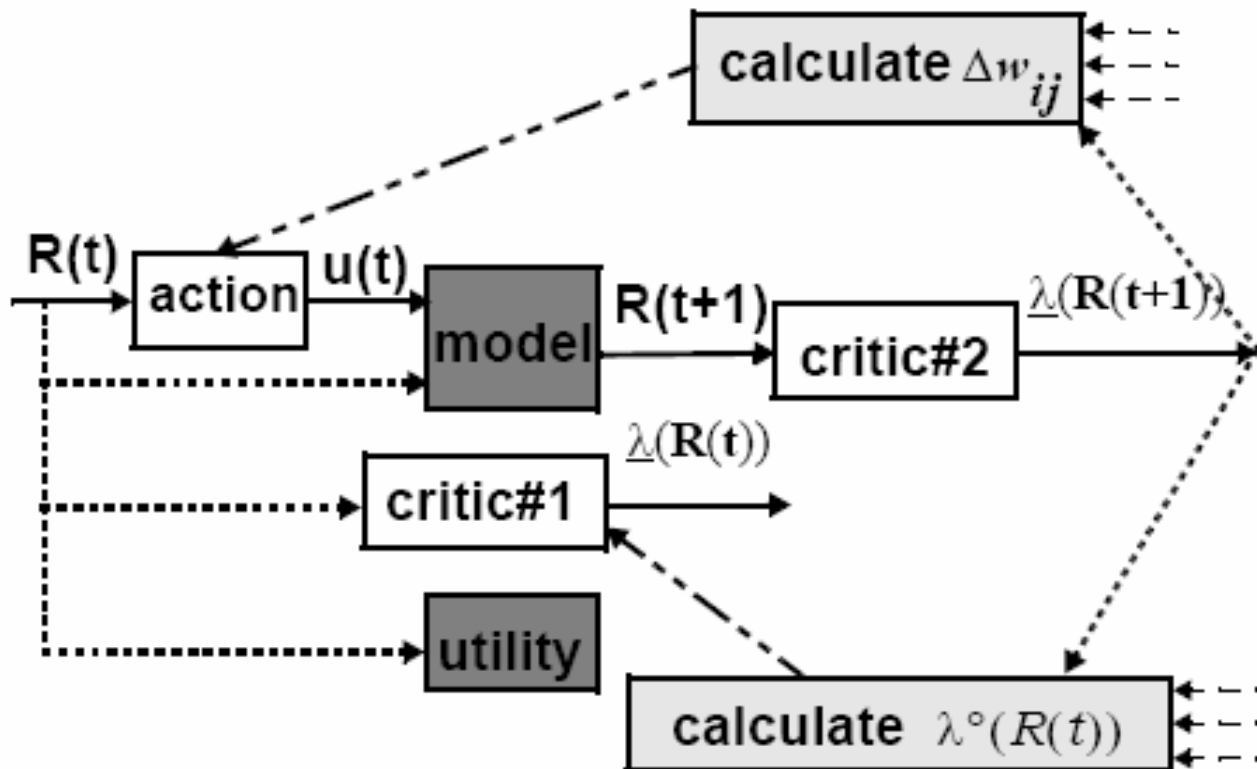
$$\frac{\partial J(t)}{\partial R_s(t)} = \frac{dU(t)}{dR_s(t)} + \sum_{k=1}^n \frac{\partial J(t+1)}{\partial R_k(t+1)} \cdot \left[\frac{\partial R_k(t+1)}{\partial R_s(t)} + \sum_m \frac{\partial R_k(t+1)}{\partial u_m(t)} \cdot \frac{\partial u_m(t)}{\partial R_s(t)} \right]$$

Via Plant Model

[Bellman Recursion & Chain Rule used in above.]

Plant model is needed to calculate partial derivatives for DHP ...

Components of DHP type Adaptive Critic



[Dark boxes: analytic expressions; medium boxes: critical calculations; clear boxes: neural networks.]



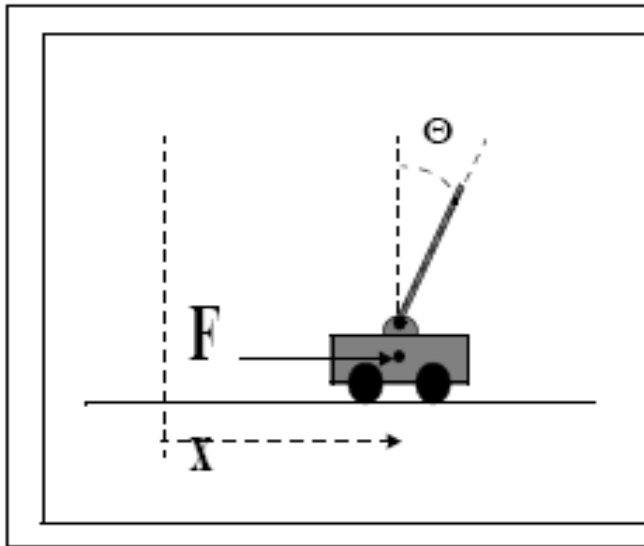
Key ADP Process Parameters

Specification of:

- **state variables**
- **size of NN structure, connection pattern, and type of activation functions**
- **learn rate of weight update rules**
- **discount factor (gamma of Bellman Equation)**
- **scaling of variable values (unit hypersphere)**
- **“lesson plan” for training**
-
-



Pole-Cart Benchmark Problem



$$R_1(t) = x(t) \quad R_4(t) = \Theta(t)$$

$$R_2(t) = \dot{x}(t) \quad R_5(t) = \dot{\Theta}(t)$$

$$R_3(t) = \ddot{x}(t) \quad R_6(t) = \ddot{\Theta}(t)$$

\dot{x} \ddot{x} $\dot{\Theta}$ $\ddot{\Theta}$ [derivatives-wrt time]

For this example, we defined

$$U(t) = -0.25\theta^2 - 0.25x^2$$

$$R_6(t+1) = \frac{g \sin \Theta + \cos \Theta \left[\frac{-F - ml \ddot{\Theta} \sin \Theta + \mu_c \operatorname{sgn}(\dot{x})}{m_c + m} \right] - \frac{\mu_p \dot{\Theta}}{ml}}{l \left[\frac{4}{3} - \frac{m(\cos \Theta)^2}{m_c + m} \right]}$$

$$R_3(t+1) = \frac{F + ml \left[\ddot{\Theta} \sin \Theta - \dot{\Theta}^2 \cos \Theta \right] - \mu_c \operatorname{sgn}(\dot{x})}{m_c + m}$$

$$R_1(t+1) = R_1(t) + \tau \bullet R_2(t); \quad R_2(t+1) = R_2(t) + \tau \bullet R_3(t)$$

$$R_4(t+1) = R_4(t) + \tau \bullet R_5(t); \quad R_5(t+1) = R_5(t) + \tau \bullet R_6(t)$$



Experimental Procedure (“lesson plan”):

A. Train 3 passes through sequence

(5, -10, 20, -5, -20, 10) [degrees from vertical].

Train 30 sec. on each angle.

B. Accumulate absolute values of U: C(1), C(2), C(3).

C. Perform TEST pass through train sequence

(30 sec. each angle). Accumulate U values: C(4).

D. Perform GENERALIZE pass through sequence

(-23, -18, -8, 3, 13, 23) [degrees from vertical] and

accumulate U values: C(5).

E. Perform GENERALIZE pass through sequence

(-38, -33, 23, 38) [degrees from vertical] and

accumulate U values: C(6).



STRATEGIES TO SOLVE EARLIER EQUATIONS:

Strategy 1. Straight application of the equation.

Strategy 2. Basic 2-stage process [“flip/flop”].

[e.g., Santiago/Werbos, Prokhorov/Wunsch]

During stage 1, train criticNN, not actionNN;

During stage 2, train actionNN, not criticNN.

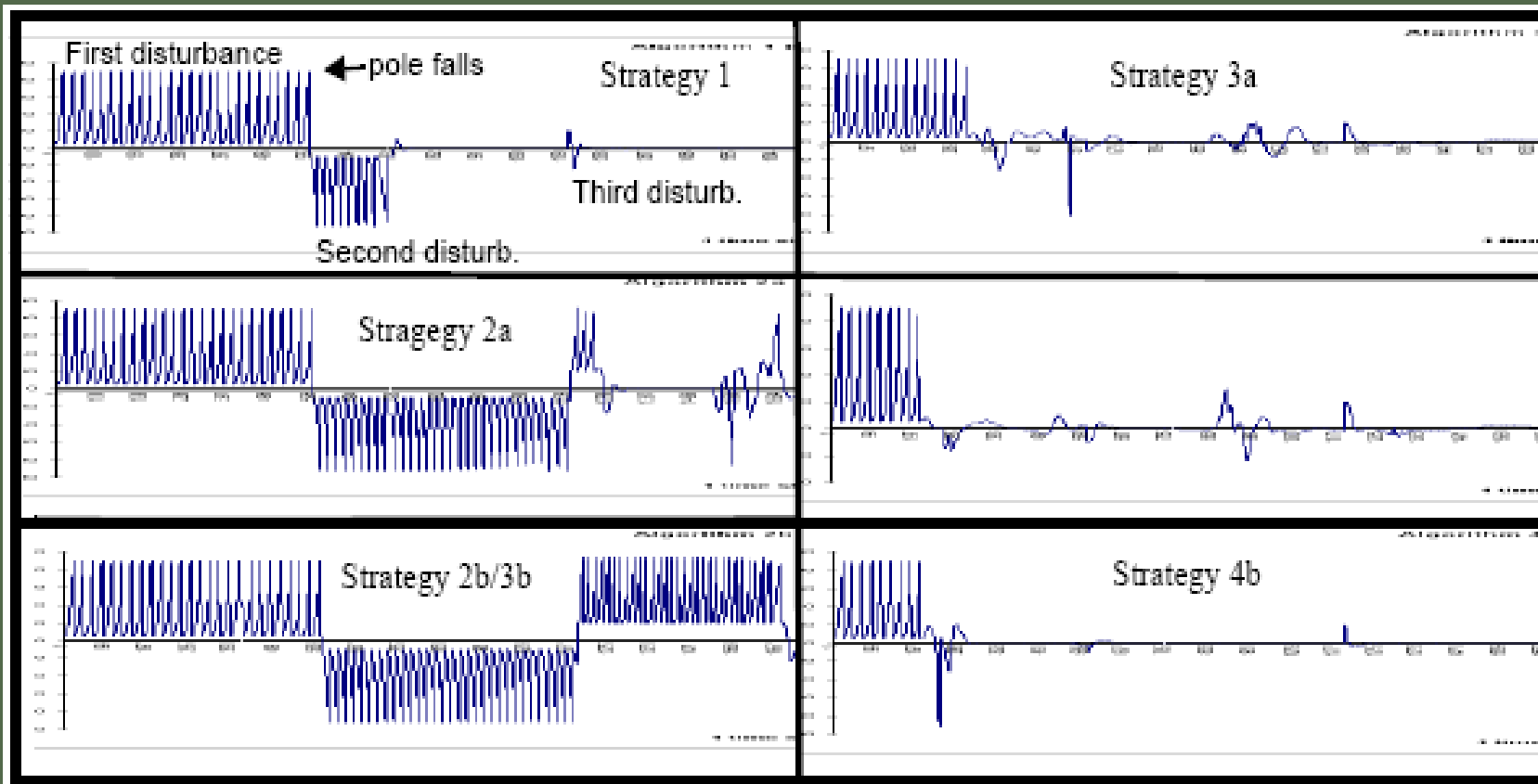
Strategy 3. Modified 1st stage of 2-stage process.

While train criticNN during stage 1, keep parameters constant in module that calculates critic’s desired output (R).

Then adjust weights all at once at end of stage 1.

Strategy 4. Single-stage process, using modifications introduced in Strategy 3.

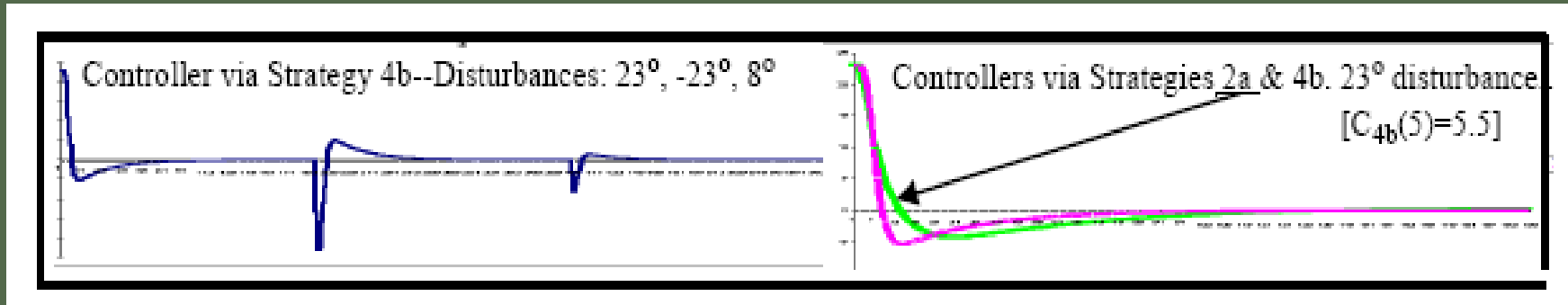




**Progress of training process under each of the strategies.
 [Pole angles during the first 80 sec. of training. Note how
 fast Strategies 4a & 4b learn.]**



Controller Responses to Disturbances



Pole angles during part of test sequence.
 [Markings below and parallel to axis are plotting artifacts.]



The following table lists the value of the performance measure for each DHP train strategy, *averaged* over 4 separate training runs.

	1	2a	2b /3b	3a	4a	4b
C(1)	368	740	883	209	107	160
C(4)	3.2	3.1	2.9	2.8	2.4	2.2
C(5)	5.1	7.9	7.4	7.5	6.2	5.5
C(6)	30.5	24.1	25.4	20.7	14.5	14.0

We observe in this table that **strategies 4a & 4b converge the fastest** [lowest value of C(1)], and also **appear to yield the best controllers** [lowest values in C(4), C(5) & C(6) rows]. We note separately that **strategy 1 (when it converges)** yields controllers on par with the better ones.



S1: Simultaneous/Classical Strategy

S2: Flip-Flop Strategy

S4: Shadow Critic Strategy

Introduce a copy of the *critic*NN in the lower loop.

Run both loops simultaneously. Lower loop: **train the copy** for an epoch; upload the weight values from copy (called *shadow* critic) into the active *critic*NN; repeat.

S5: Shadow Controller Strategy

Introduce a copy of the *action*NN in the upper loop.

Run both loops simultaneously. Upper loop, **train the copy** for an epoch; upload the weight values from copy (called *shadow* controller) into the active *action*NN, repeat.

S6: Double Shadow Strategy

Make use of the NN copies in both training loops.

Run both loops simultaneously. Both loops: **train the copies** for an epoch; upload the weight values from the copies into their respective active NNs.

Lendaris, G.G., T.T. Shannon & A. Rustan (1999), "A Comparison of Training Algorithms for DHP Adaptive Critic Neuro-control," *Proceedings of International Conference on Neural Networks'99 (IJCNN'99)*, Washington,DC IEEE Press



Autonomous Vehicle Example (Two-axle terrestrial vehicle)

Objective: Demonstrate design via DHP (“from scratch”) of a steering controller which effects a Change of Lanes (on a straight highway).

Controller Task: Specify sequence of steering angles to effect accelerations needed to change orientations of vehicle velocity vector.

Design Scenarios -- general:

Maintain approximately constant forward velocity during lane change.

Control angular velocity of wheel.

Thus, Controller needs 2 outputs:

1.) Steering angle

2.) Wheel rotation velocity

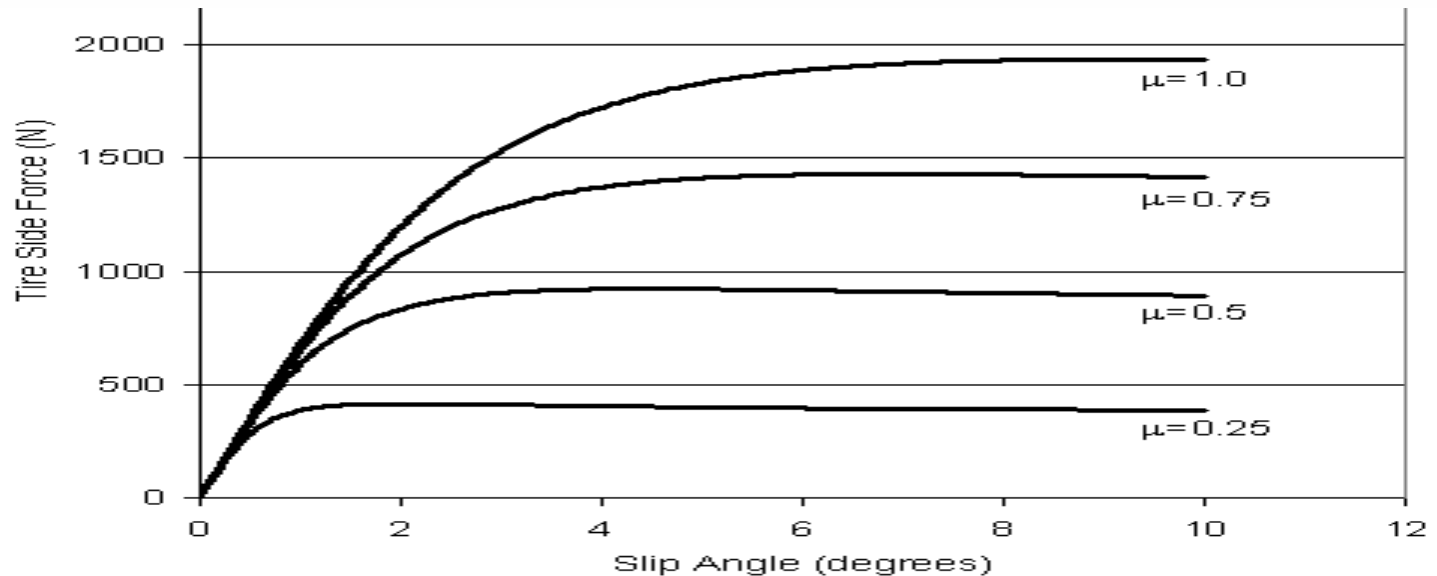
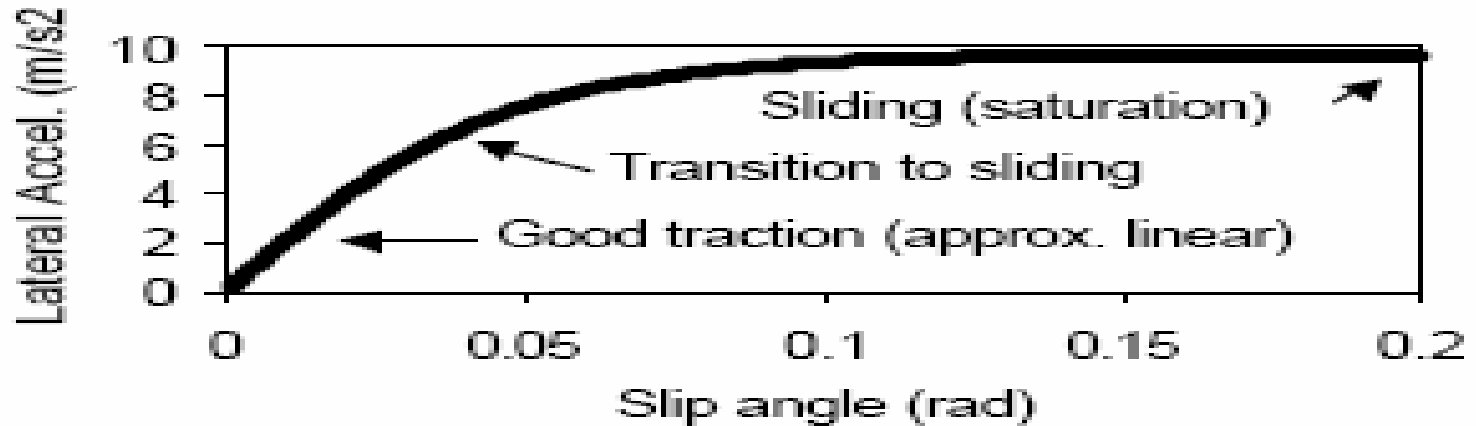


Major unknown for controller:

**Coefficient of Friction (cof)
between tire/road at any point in time.**

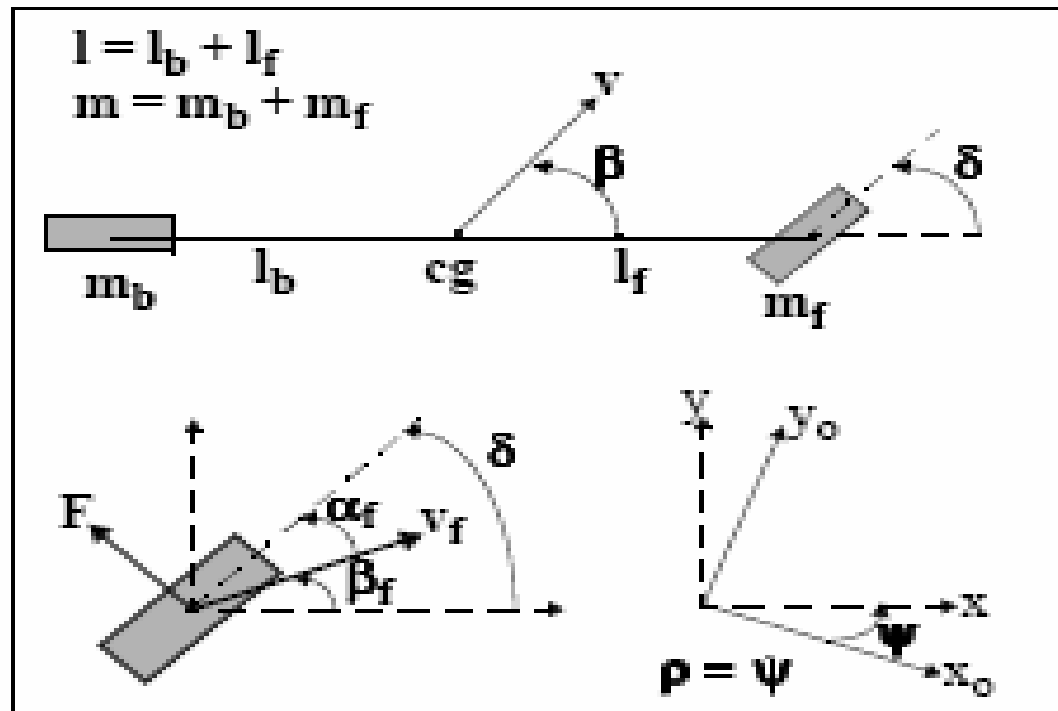
**It can change abruptly
(based on road conditions).
Requires Robustness and/or fast
on-line adaptation.**

Tire side-force/side-slip-angle as a function of COF



Vehicle Simulation

Based on “bicycle model”



Equations used:

$$\begin{bmatrix} mv(\beta + p) \\ m\dot{v} \\ ml_f l_b \dot{\beta} \end{bmatrix} = \begin{bmatrix} -\sin(\beta) & \cos\beta & 0 \\ \cos\beta & \sin\beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f_x \\ f_y \\ m_z \end{bmatrix}$$

$$\begin{bmatrix} f_x \\ f_y \\ m_z \end{bmatrix} = \begin{bmatrix} -\sin(\delta_f) & 0 & 1 \\ \cos(\delta_f) & 1 & 0 \\ l_f \cos(\delta_f) & -l_b & 0 \end{bmatrix} \begin{bmatrix} F_f(\alpha_f) \\ F_b(\alpha_b) \\ F_x(\kappa) \end{bmatrix}$$

$$a_{f_y} = \frac{F_f \cos(\delta_f)}{m_f}$$



Assume on-board sensors for constructing state information

Only 3 accelerometers needed:

- x direction accel. of chassis**
- lateral accel. at rear axle**
- lateral accel. at front axle**

**Plus load cells at front & rear axles
(to give estimate of c.g. location)**

[Simulation used analytic proxies]



Criteria for various Design Scenarios:

1. **Velocity Error**
 - reduce to zero.
2. **Velocity Rate**
 - limit rate of velocity corrections.
3. **Y-direction Error**
 - reduce to zero.
4. **Lateral front axle acceleration**
 - re. “comfort” design specification.
5. **Friction Sense**
 - estimate closeness to knee of cof curves.



Utility Functions for three Design Scenarios:

[different combinations of above criteria]

1. $U(1,2,3)$

2. $U(1,2,3,5)$

3. $U(1,2,3,4,5)$

All applied to task of designing controller
for autonomous 2-axle terrestrial vehicle.

Design Scenario 1.

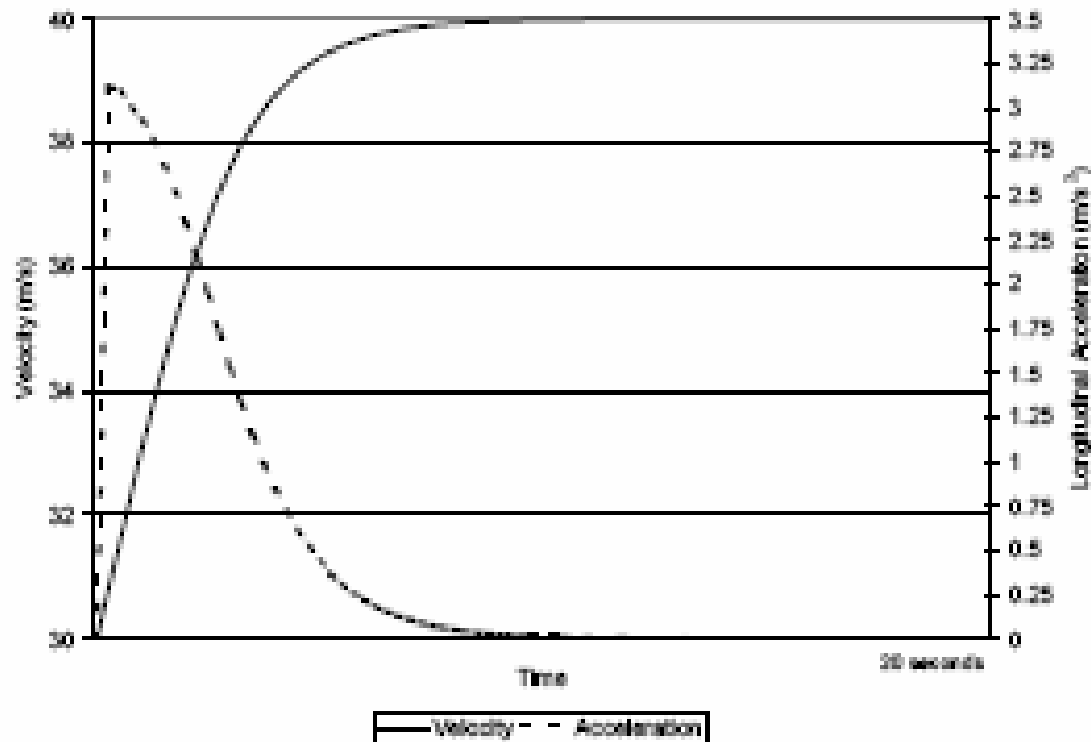
Fewest restrictions on controller design.

$$U_1 = -\frac{1}{2}(y_{err})^2 - \frac{1}{8}(v_{err})^2 - \frac{1}{16}(\dot{v})^2$$

Velocity terms given lower coefficients, based on assumed specification that forward velocity be *approximately* constant, with explicit specification for Y-error to become zero.

Design Scenario 1 Experimental Result

--> Step Velocity Change, via U_1 :

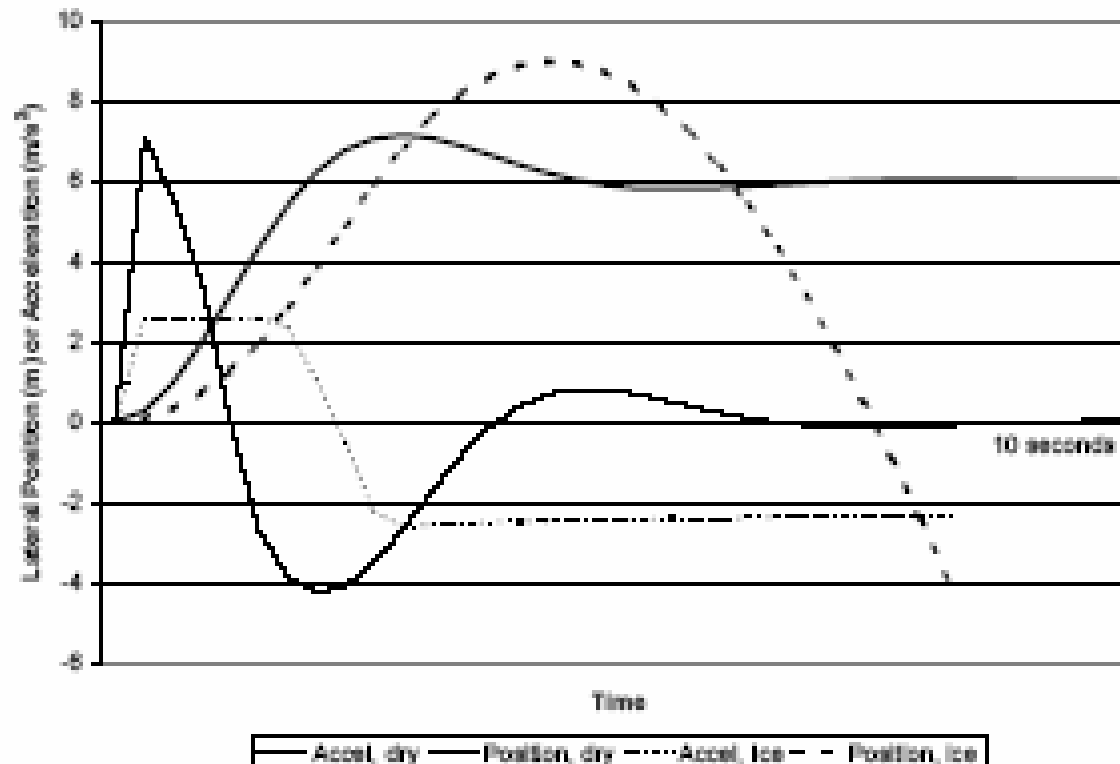


Commanded change from 30 m/s to 40 m/s.



Design Scenario 1 Experimental Result

--> Lane Change [$\Delta y = 6m$], via U_1 :



Tested with COF: Icy --> Car goes off the road!

[was trained on dry road, thus steering commands too aggressive.]

Design Scenario 2.

Add Criterion 5 (“friction sense”) in U2.
This is intended to allow aggressive lane changes on dry pavement,

PLUS

make lane changes on icy road conditions as aggressively as the icy road will allow.

Design Scenario 2.

Implement Criterion 5 via “Sliding Index”:

$$SI = -10 \left(\frac{\frac{\partial a_y}{\partial \alpha_f} - \left(\frac{\partial a_y}{\partial \alpha_f} \right)_{base}}{\left(\frac{\partial a_y}{\partial \alpha_f} \right)_{base}} \right), \text{ where } \left(\frac{\partial a_y}{\partial \alpha_f} \right)_{base} \text{ is}$$

slope at very small slip angle (i.e., the linear portion of the curves).

SI takes on value of ~10 when sliding, and ~0 at no sliding.

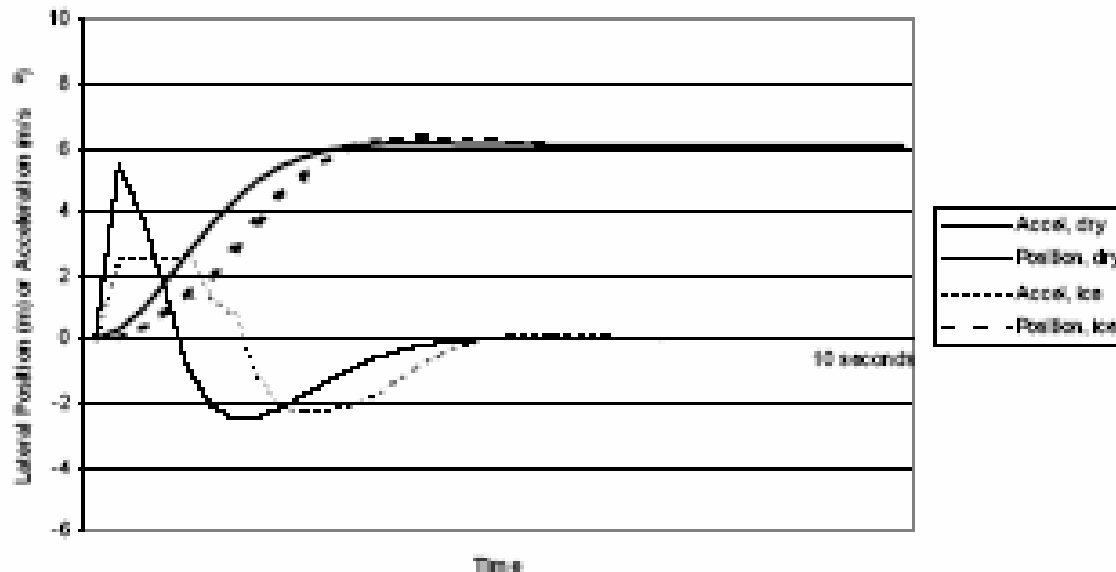


Design Scenario 2. Resulting Utility Function U_2 :

$$U_2 = \begin{cases} U_1 & \text{for } SI < 3 \\ U_1 - \frac{1}{4}(SI)^2 & \text{for } SI \geq 3 \end{cases}$$

Design Scenario 2 Experimental Result

--> Lane Change [$\Delta y = 6m$], via U_2 :



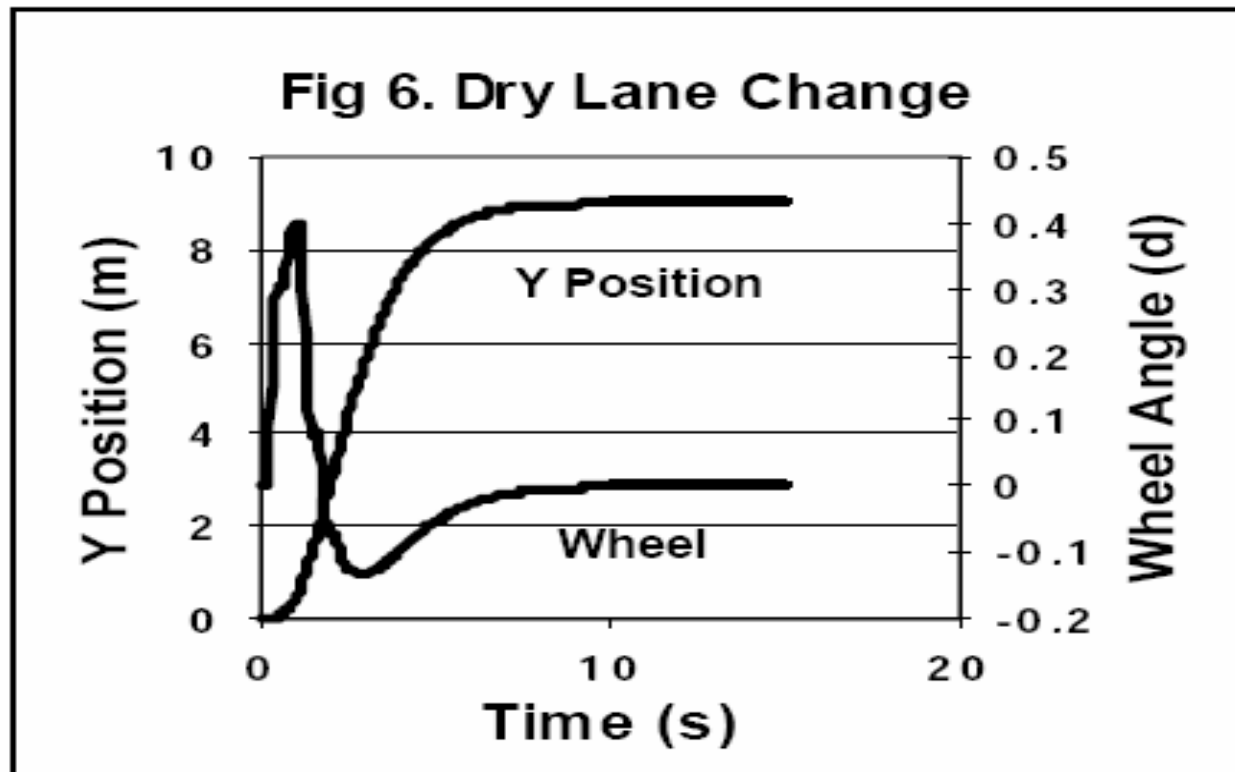
Acceptable lane changes on *both* dry & icy pavement.

Dry pavement response slightly slower than U_1 , via SI penalty.

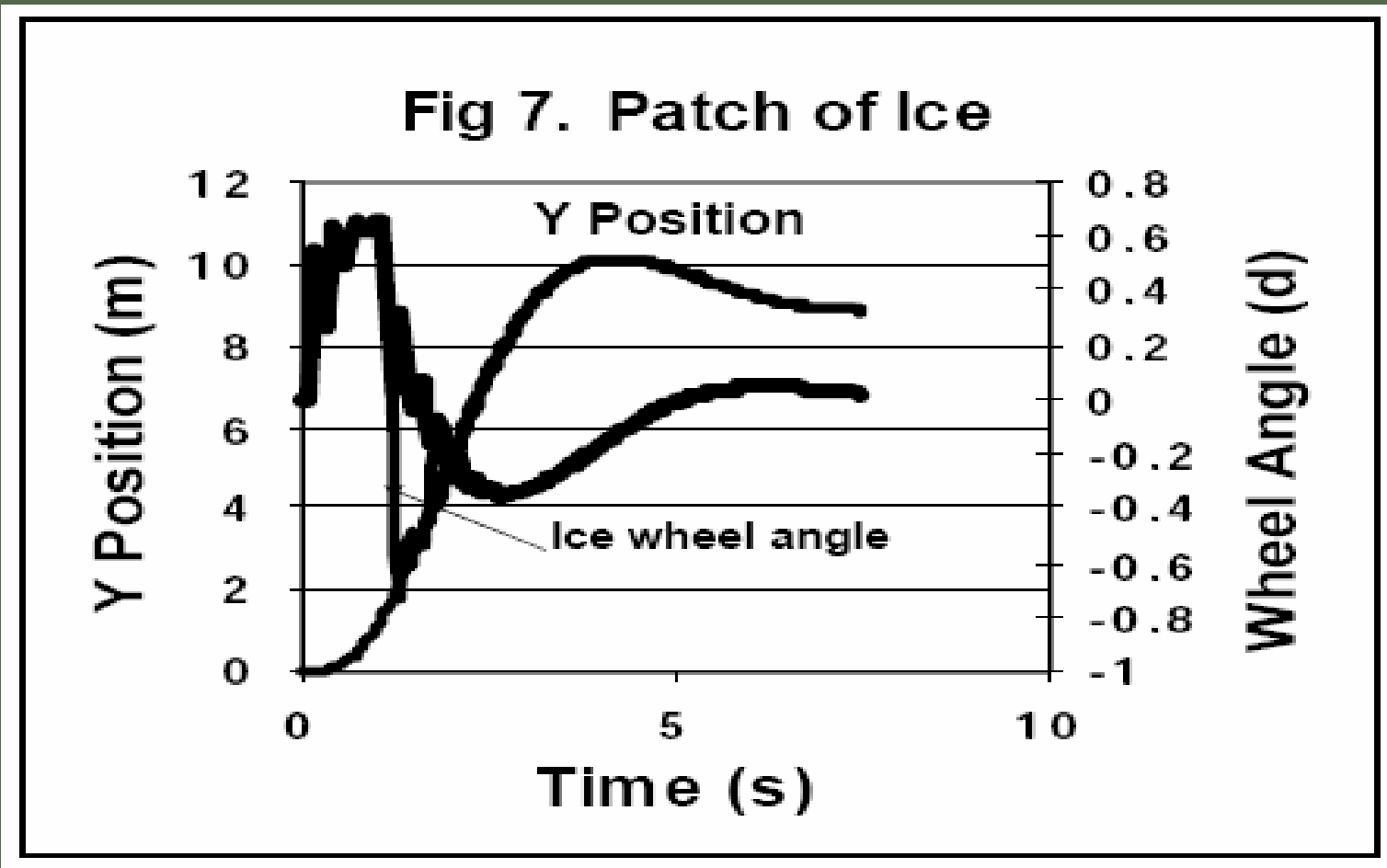
Peak acceleration on dry pavement: $\sim 0.5g$

On ice, controller keeps request to lower limit allowed on ice.

Dry Lane Change to compare for Generalize test



Generalize Test: Ice patch at point where wheel angle is greatest in dry pavement case, retrained with coefficient modified.



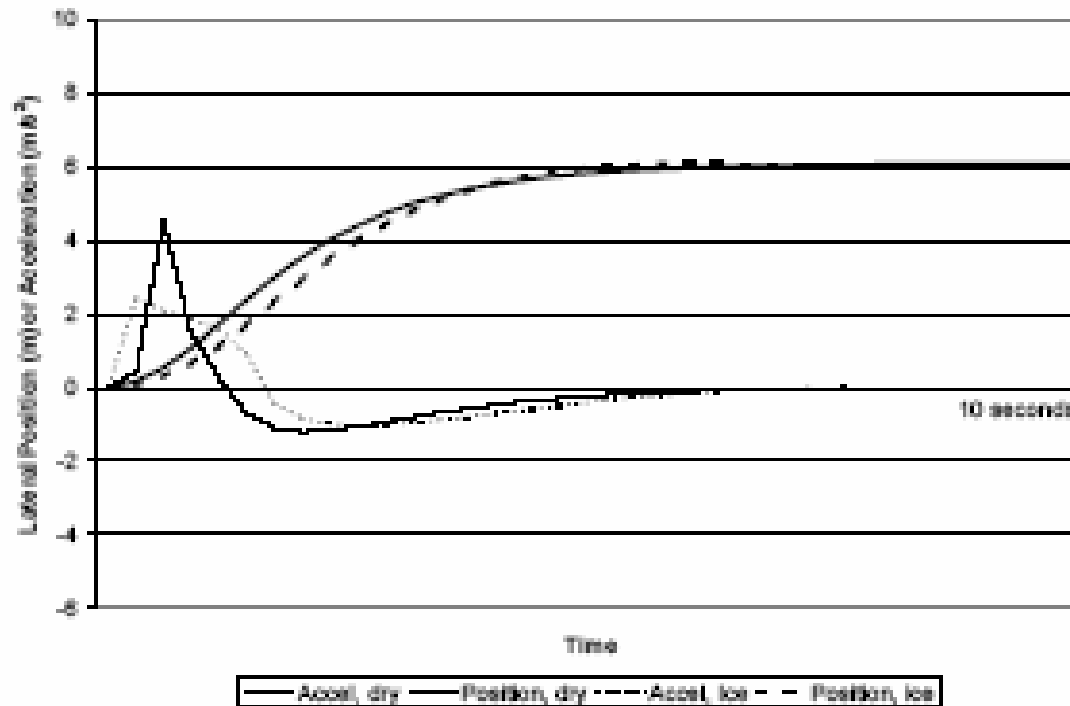
Design Scenario 3.
Add Criterion 4 (“comfort”) in U_3 .
[Limits Lateral front axle acceleration.]

$$U_3 = U_2 - \frac{1}{8}(a_f)^2$$



Design Scenario 3 Experimental Result

--> Lane Change [$\Delta y = 6m$], via U_3 :



Lane change response for both road conditions slower, due to acceleration limiting term.



Conclusions from Utility Function Expts.

Controller Designs resulting via DHP satisfy intuitive sense of being “good”.

Control Engineer knows that controller design requires careful specification of objective, and that as change design criteria, controllers change.

For DHP, control objectives are contained in the Utility Function.

The DHP process embodied the different requirements for the three design scenarios in qualitatively distinct controllers -- all yielding intuitively “good” results, according to the design constraints.



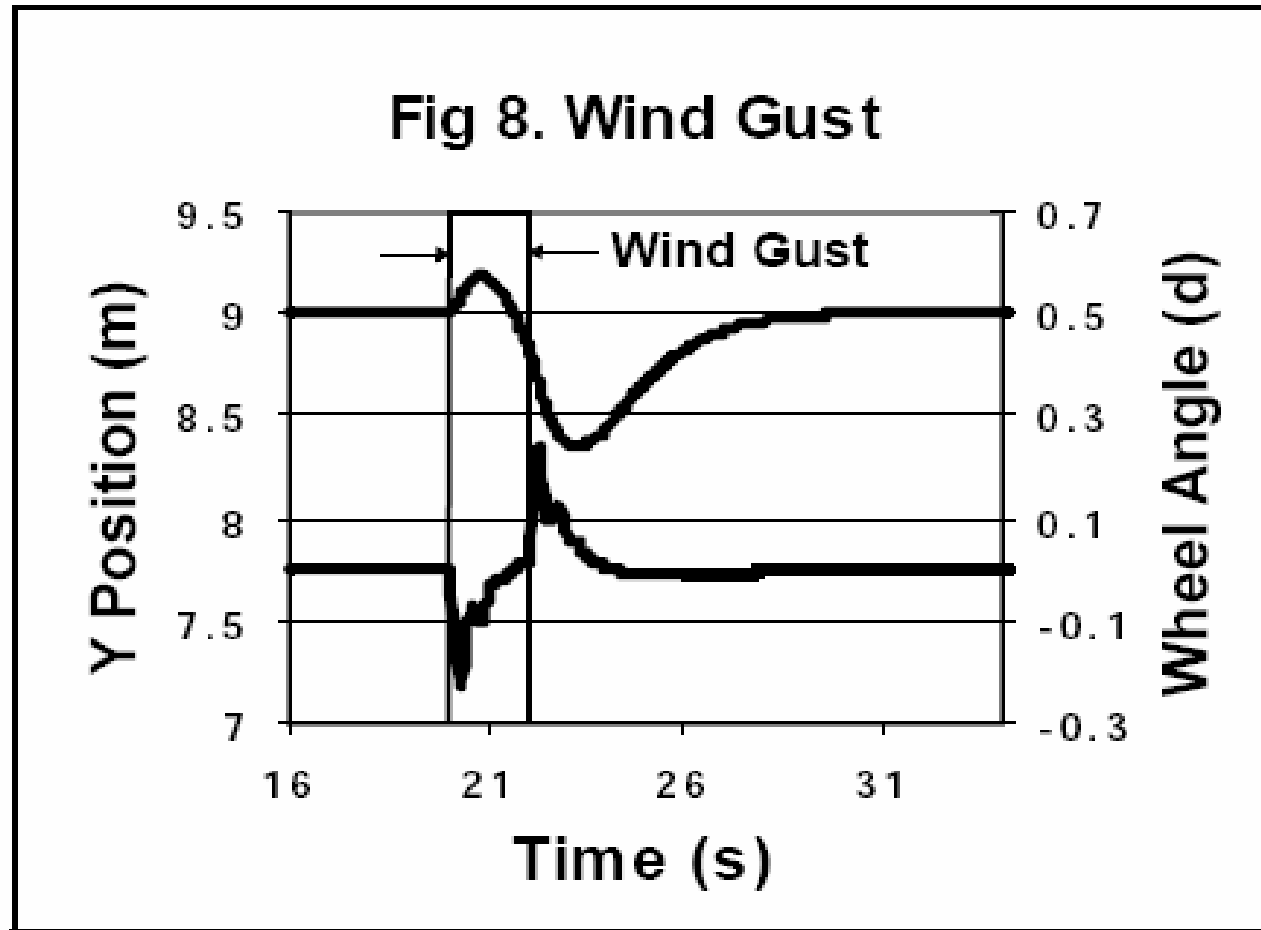
Conclusions from Utility Function Expts., cont.

“Of particular interest to the present researchers is the ability of the DHP method to accept design criteria crafted into the Utility function by the human designers, and to then evolve a controller design whose response “looks and feels” like one a human designer might have designed.”

Expanded task for Autonomous Vehicle Controller:

Additional road condition of encountering an ice patch while doing lane change: also, accommodate side-wind load disturbances.

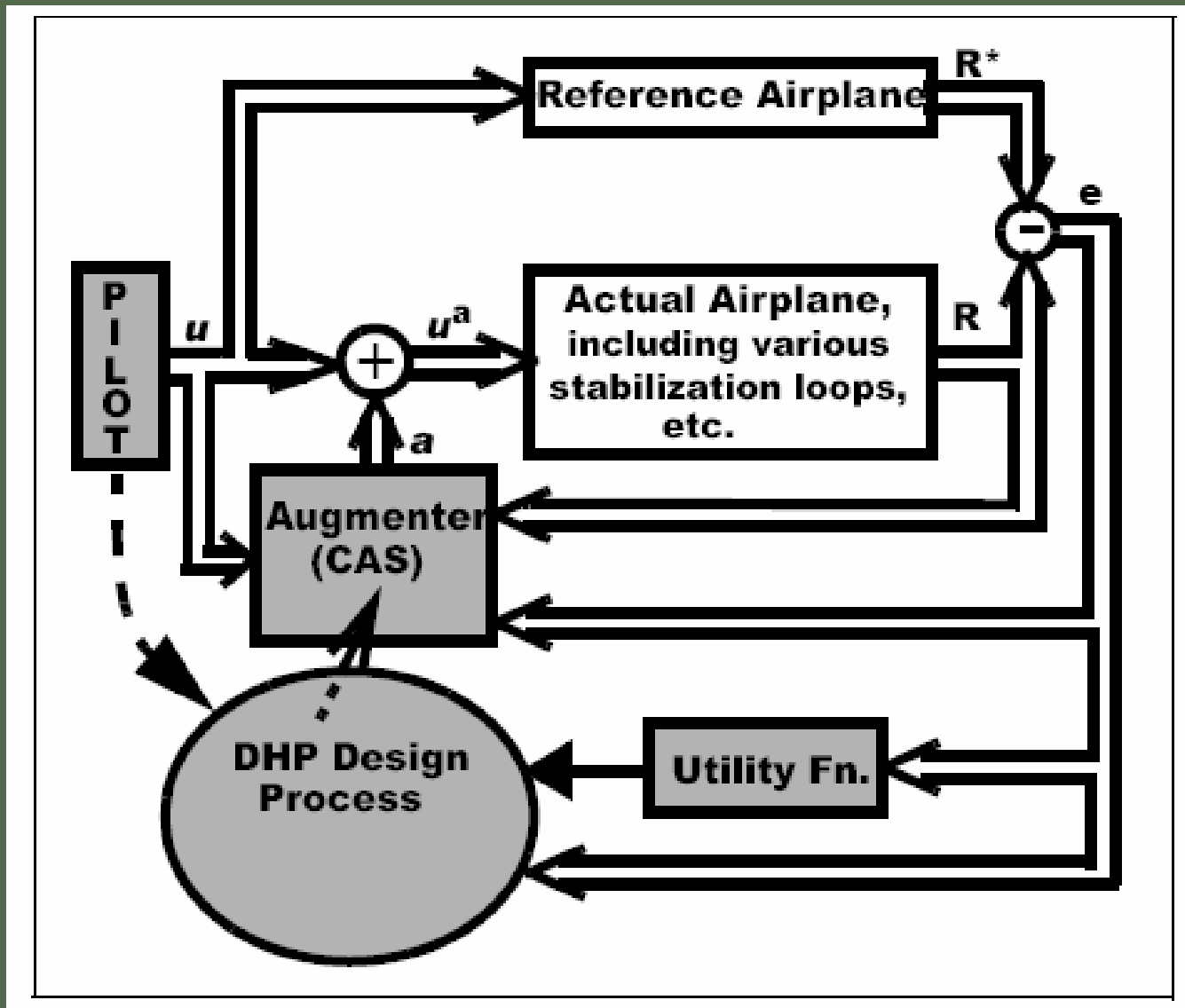
Generalize Test: Wind Gusts (side forces)

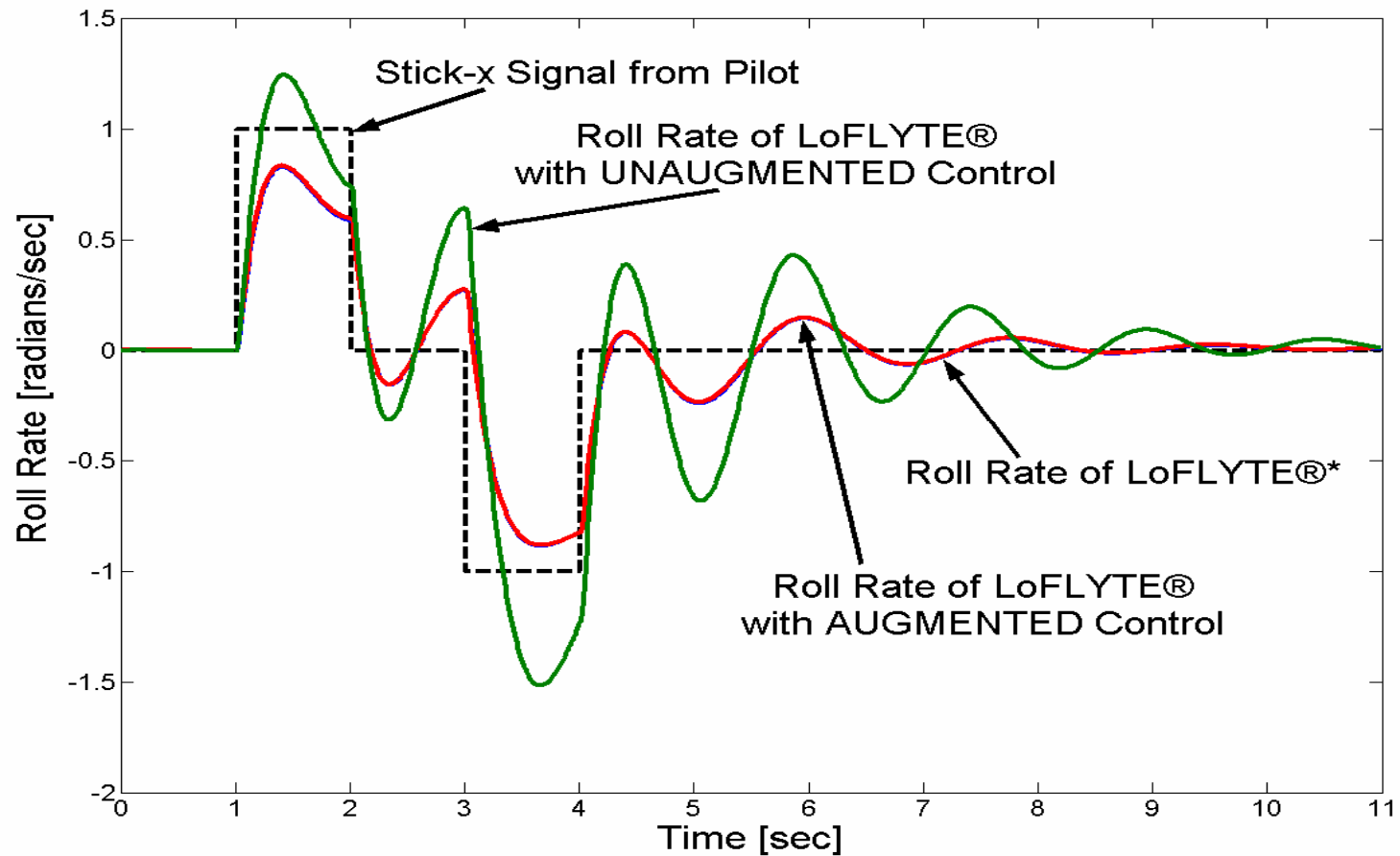


Aircraft Control Augmentation System



System configuration during DHP Design of Controller Augmentation System

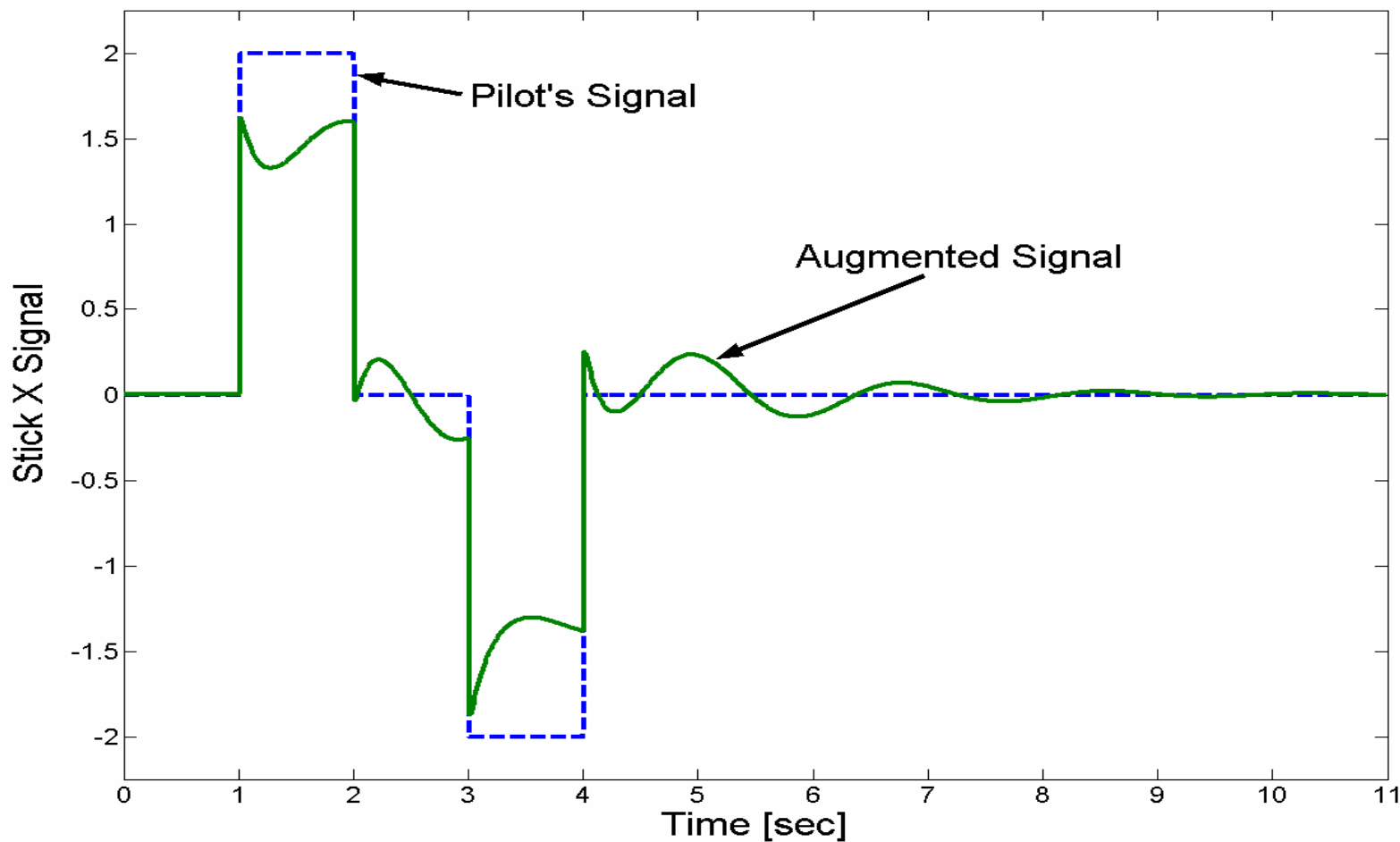




Pilot stick-x doublet signal (arbitrary scale in the Figure), and roll-rate responses of 3 aircraft:

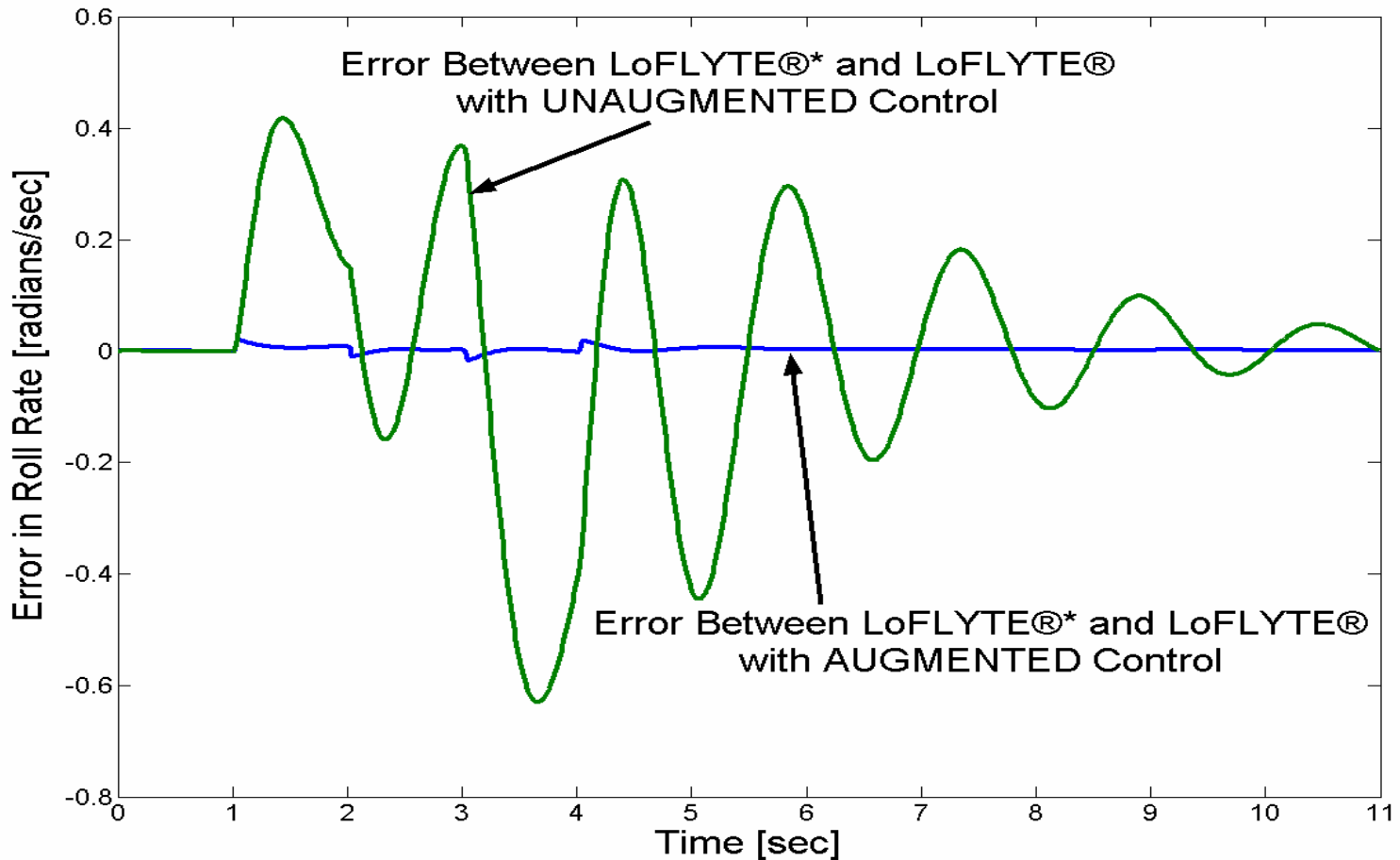
LoFLYTE® w/Unaugmented control,
 LoFLYTE® w/Augmented Control, and LoFLYTE®*.

(Note: Responses of latter two essentially coincide.)



Stick-x doublet: pilot's stick signal vs. augmented signal (the latter is sent to aircraft actuators)





**Roll-rate error (for above stick-x signal)
 between LoFLYTE®* and LoFLYTE® w/Unaugmented Control, and
 between LoFLYTE®* and LoFLYTE® w/Augmented Control signals**



- **Blue:** LoFLYTE® w/ **Unaugmented** control
- **Red:** LoFLYTE® w/**Augmented** Control
- **Black:** LoFLYTE®*

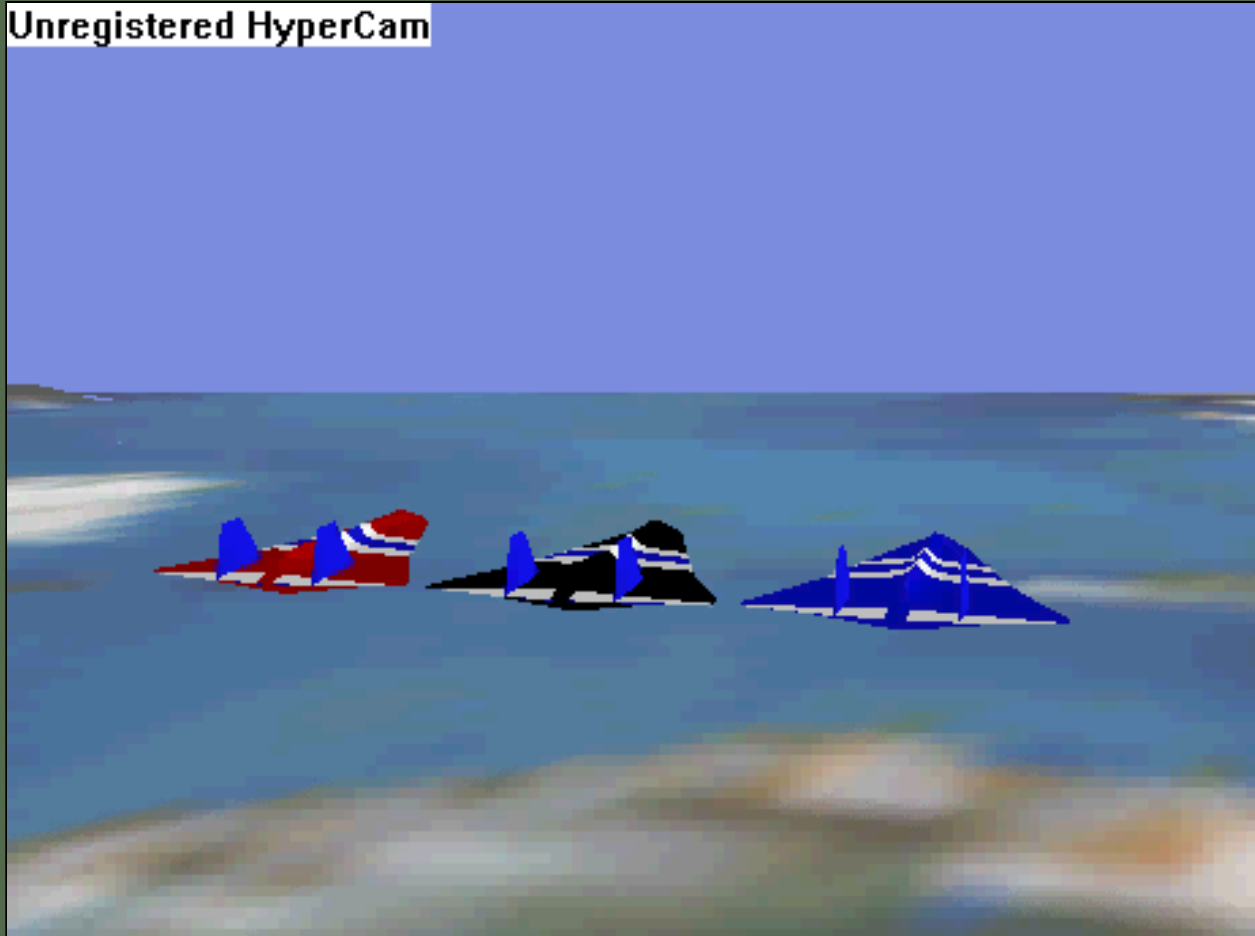
Unregistered HyperCam

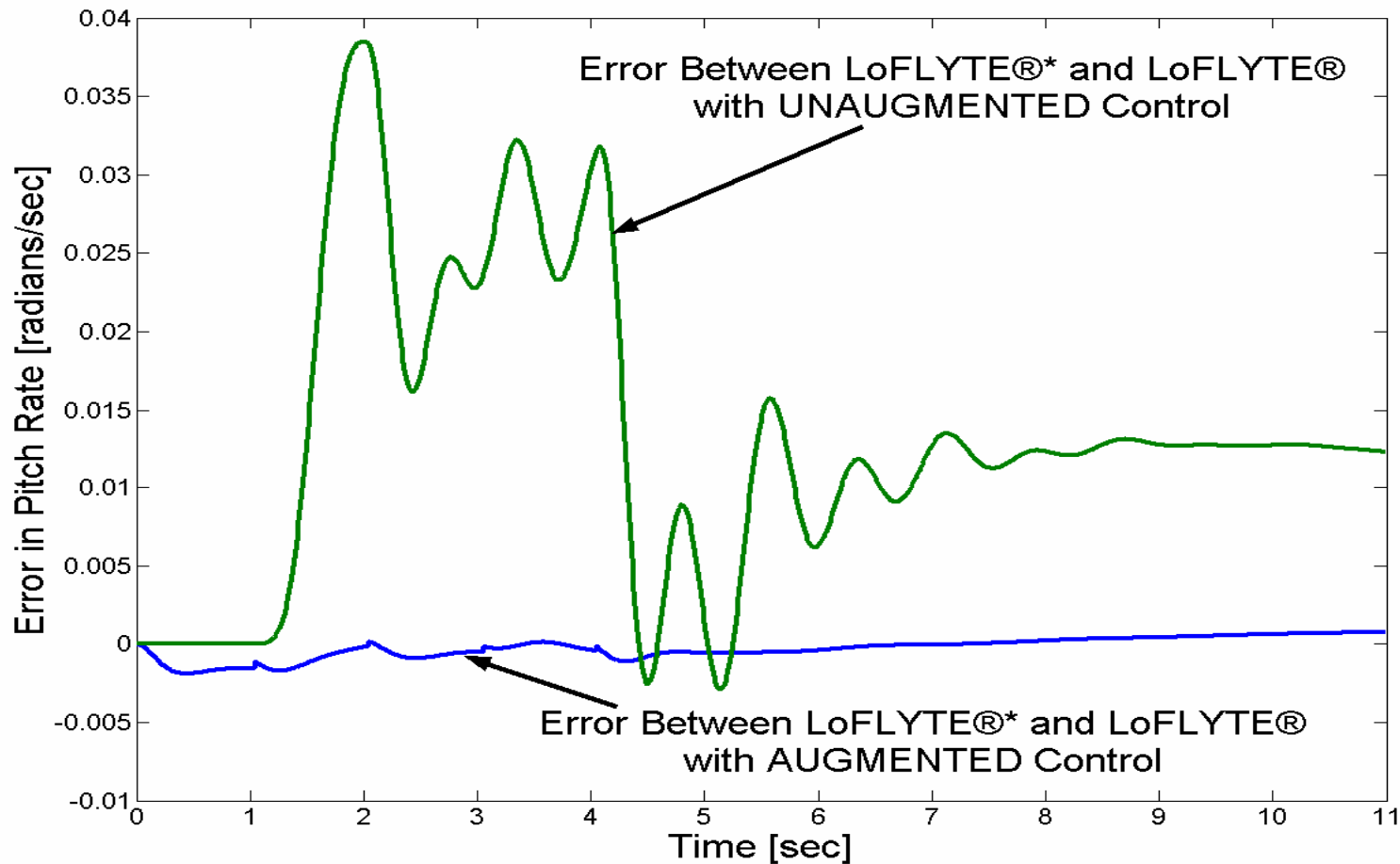
Roll
1



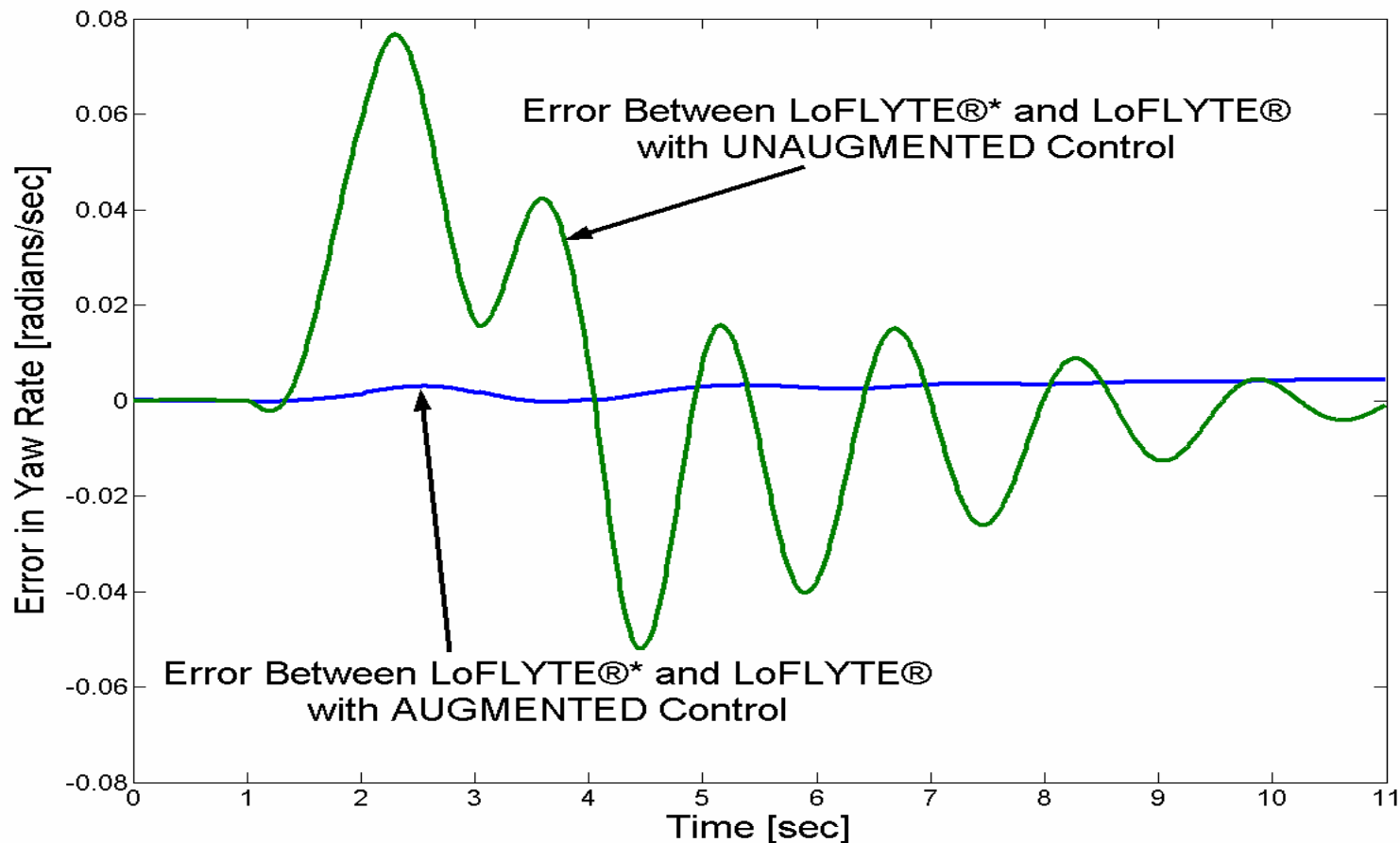
- **Blue:** LoFLYTE® w/ **Unaugmented** control
- **Red:** LoFLYTE® w/Augmented Control
- **Black:** LoFLYTE®*

**Roll
2**

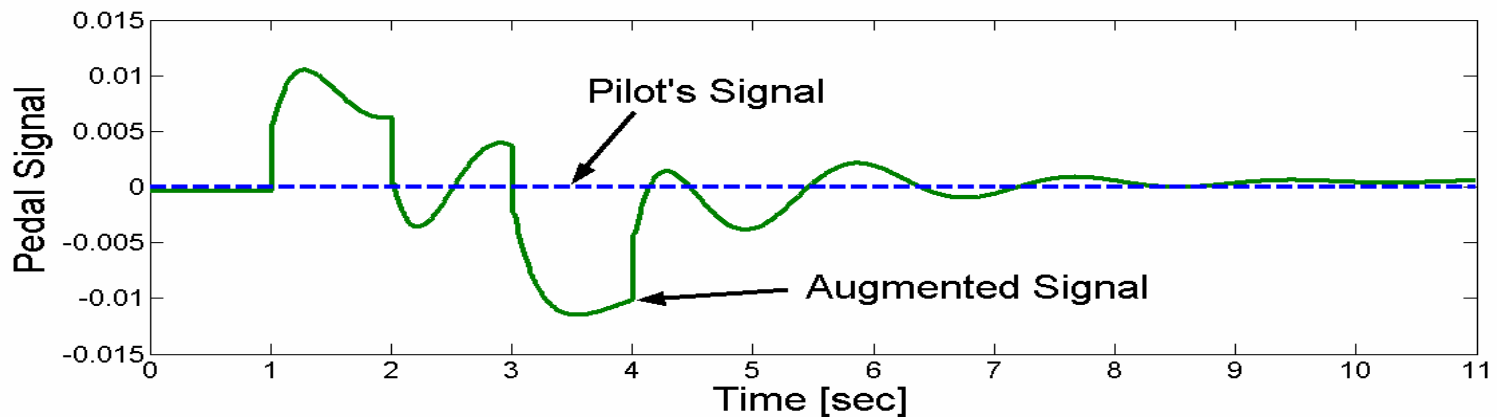
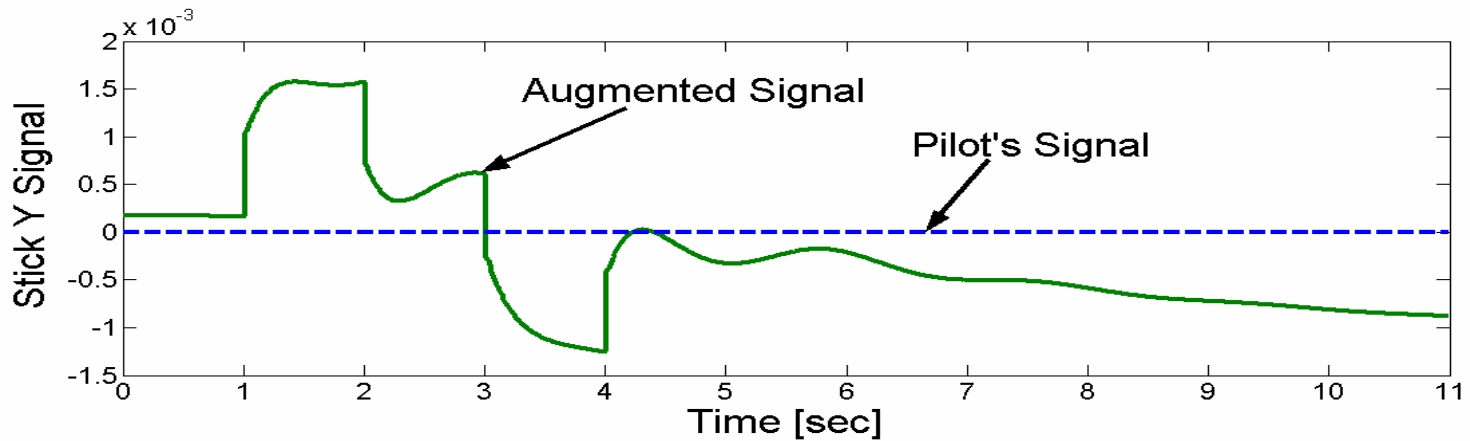




Pitch-rate error (for above stick-x signal) between LoFLYTE®* and LoFLYTE® w/Unaugmented Control, and between LoFLYTE®* and LoFLYTE® w/Augmented Control signals.



Yaw-rate error (for above stick-x signal) between LoFLYTE®* and LoFLYTE® w/Unaugmented Control, and between LoFLYTE®* and LoFLYTE® w/Augmented Control signals.



Augmentation commands for stick-y and pedal that the controller learned to provide to make the induced a) pitch (stick-y) and b) yaw (pedal) responses of LoFLYTE® match those of LoFLYTE®*

- **Blue:** LoFLYTE® w/ **Unaugmented** control
- **Red:** LoFLYTE® w/**Augmented** Control
- **Black:** LoFLYTE®*

**Pitch
w/
cg
Shift**



Conclusions re. Control Augmentation Example:

Basic motivation behind this work is the desire to ultimately generate a non-linear controller that has control capabilities equivalent to that of an “experienced pilot.”

Successful base demonstration was presented here.





Perform search over Criterion Function (J function) Surface in Weight (State) Space

