Proceedings of the 34th
Conference on Decision & Control
New Orleans, LA - December 1995

Hybrid Control System Design Based on Natural Invariants

James A. Stiver, Panos J. Antsaklis, and Michael D. Lemmon
Department of Electrical Engineering
University of Notre Dame, Notre Dame, IN 46556

Abstract

The hybrid control systems considered here consist of a continuous plant under the control of a discrete event system. Communication between the plant and controller is provided by an interface that can convert signals from the continuous domain to the symbolic domain, and vise-versa. Hybrid control system design generally involves designing a controller, possibly designing some or all of the interface, and in some cases, designing a continuous controller that will become part of the continuous-time plant.

This paper examines the case where the interface is partially given, along with the plant and a set of control goals. A method is presented which designs the symbol generating portion of the interface and also yields a controller for the system. The method is based on the natural invariants of the plant. This technique is illustrated by an example in which the design process is simulated via a computer program for an unmanned underwater vehicle.

1 Hybrid Control System Modeling

A hybrid control system, can be divided into three parts, the plant, interface, and controller as shown in Figure 1.

The plant is a nonlinear, time-invariant system represented by a set of ordinary differential equations,

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{r}(t)), \qquad (1)$$

where $\mathbf{x}(t) \in \mathbf{X}$ and $\mathbf{r}(t) \in \mathbf{R}$ are the state and input vectors respectively, and $\mathbf{X} \subset \Re^n, \mathbf{R} \subset \Re^m$, with $t \in (a, b)$ some time interval. For any $\mathbf{r}(t) \in \mathbf{R}$, the function $f : \mathbf{X} \times \mathbf{R} \to \mathbf{X}$ is continuous in $\mathbf{x}$ and meets the conditions for existence and uniqueness of solutions for initial states, $\mathbf{x}_0 \in \mathbf{X}$. Note that the plant input and state are continuous-time vector valued signals. Boldface letters are used here to denote vectors and vector valued signals.

The controller is a discrete event system which is modeled as a deterministic automaton, $(\tilde{S}, \tilde{X}, \tilde{R}, \delta, \phi)$, where $\tilde{S}$ is the set of states, $\tilde{X}$ is the

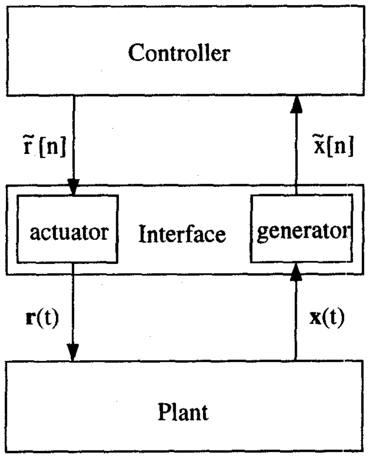

Figure 1: Hybrid Control System

set of *plant symbols*, $\tilde{R}$ is the set of *controller symbols*, $\delta : \tilde{S} \times \tilde{X} \to \tilde{S}$ is the state transition function, and $\phi : \tilde{S} \to \tilde{R}$ is the output function. The symbols in set $\tilde{R}$ are called controller symbols because they are generated by the controller. Likewise, the symbols in set $\tilde{X}$ are called plant symbols and are generated based on events in the plant. The action of the controller is described by the equations

$$\begin{aligned} \tilde{s}[n] &= \delta(\tilde{s}[n-1], \tilde{x}[n]) \\ \tilde{r}[n] &= \phi(\tilde{s}[n]) \end{aligned}$$

where $\tilde{s}[n] \in \tilde{S}, \tilde{x}[n] \in \tilde{X}$, and $\tilde{r}[\tilde{n}] \in \tilde{R}$. The index n is analogous to a time index in that it specifies the order of the symbols in the sequence. The input and output signals associated with the controller are sequences of symbols.

The interface consists of two simple subsystems, the *generator* and the *actuator*. The generator converts the continuous-time output (state) of the plant to an asynchronous, symbolic input for the controller. The set of plant events recognized by the generator is determined by a set of smooth functionals, $\{h_i : \Re^n \to \Re, i \in I\}$, whose null spaces form

$n-1$ dimensional smooth hypersurfaces in the plant state space. Whenever the plant state crosses one of these hypersurfaces, a plant symbol is generated according to

$$\tilde{x}[n] = \alpha_i(\mathrm{x}(\tau_e[n])) \tag{2}$$

where i identifies the hypersurface which was crossed and $\tau_e[n]$ is the time of the crossing.

The actuator converts the sequence of controller symbols to a plant input signal, using the function $\gamma : \tilde{R} \to \mathbf{R}$, as follows.

$$\mathrm{r}(t) = \sum_{n=0}^{\infty} \gamma(\tilde{r}[n]) I(t, \tau_c[n], \tau_c[n+1]) \tag{3}$$

where $I(t, \tau_1, \tau_2)$ is a characteristic function taking on the value of unity over the time interval $[\tau_1, \tau_2)$ and zero elsewhere. $\tau_c[n]$ is the time of the nth control symbol which is based on the sequence of plant symbol instants,

$$\tau_c[n] = \tau_e[n] + \tau_d \tag{4}$$

where τ_d is the total delay associated with the interface and controller.

2 Invariant Based Approach

Here a methodology is presented to design the controller and the interface together based on the natural invariants of the plant. In particular, this section discusses the design of the generator, which is part of the interface, and also the design of the controller. We assume that the plant is given, the set of available control policies is given, and the control goals are specified as follows. Each control goal for the system is given as a starting set and a target set, each of which is an open subset of the plant state space. To realize the goal, the controller must be able to drive the plant state from anywhere in the starting set to somewhere in the target set using the available control policies. Generally, a system will have multiple control goals.

We propose the following solution to this design problem. For a given target region, identify the states which can be driven to that region by the application of a single control policy. If the starting region is contained within this set of states, the control goal is achievable via a single control policy. If not, then this new set of states can be used as a target region and the process can be repeated. This will result in a set of states which can be driven to the original target region with no more than two control policies applied in sequence. This process can be repeated until the set of states, for which a sequence of control policies exists to drive them to the target region, includes the entire starting region (provided the set of control policies is adequate).

When the regions have been identified, the generator is designed to tell the controller, via plant symbols, which region the plant state is currently in. The controller will then call for the control policy which drives the states in that region to the target region.

2.1 Generator Design

A *common flow region* (CFR), for given target region, is a set of states which can be driven to the target region with the same control policy. The following definition is used.

Definition 1: For a plant given by Equation (1) and a target region, $\mathbf{T}$, the set $\mathbf{B}$ is a common flow region for $\mathbf{T}$ if

$$\forall \mathbf{x}(0) \in \mathbf{B}, \exists \tilde{r} \in \tilde{R}, t_1, t_2, t_1 < t_2$$

such that

$$x(t) \in \mathbf{B}, t \leq t_1$$

and

$$x(t) \in \mathbf{T}, t_1 < t < t_2$$

subject to

$$\dot{\mathrm{x}}(t) = f(\mathrm{x}(t), \gamma(\tilde{r}))$$

To design the generator, it is necessary to select the set of hypersurfaces, $\{h_i : \mathbf{X} \to \Re \mid i \in I\}$ and the associated functions, $\{\alpha_i : \mathcal{N}(h_i) \to \tilde{R} \mid i \in I\}$, described above.

We present two propositions which can be used to determine whether a given set of hypersurfaces identifies a common flow region. In different situations, one of the propositions may be easier to apply than the other. The following propositions give sufficient conditions for the hypersurfaces bounding $\mathbf{B}$ and $\mathbf{T}$ to ensure that all state trajectories in $\mathbf{B}$ will reach $\mathbf{T}$.

Proposition 1 *Given the following:*

1. *A flow generated by a smooth vector field, f*
2. *A target region, $\mathbf{T} \subset \mathbf{X}$*
3. *A set of smooth hypersurfaces, $h_i, i \in I_B \subset 2^I$*
4. *A smooth hypersurface (exit boundary), h_e*

such that $\mathbf{B} = \{\boldsymbol{\xi} \in \mathbf{X} : h_i(\boldsymbol{\xi}) < 0, h_e(\boldsymbol{\xi}) > 0, \forall i \in I_B\} \neq \emptyset$. For all $\boldsymbol{\xi} \in \mathbf{B}$ there is a finite time, t, such that $\mathrm{x}(0) = \boldsymbol{\xi}, \mathrm{x}(t) \in \mathbf{T}$, if the following conditions are satisfied:

1. $\nabla_{\xi} h_i(\boldsymbol{\xi}) \cdot f(\boldsymbol{\xi}) = 0, \forall i \in I_B$
2. $\exists \epsilon > 0, \nabla_{\xi} h_e(\boldsymbol{\xi}) \cdot f(\boldsymbol{\xi}) < -\epsilon, \forall \boldsymbol{\xi} \in \mathbf{B}$

3. $\mathbf{B} \cap \mathcal{N}(h_e) \subset T$

Proof: See [2,3]. □

The second proposition uses a slightly different way of specifying a common flow region. In addition to the invariant hypersurfaces and the exit boundary, there is also a cap boundary. The cap boundary is used to obtain a common flow region which is bounded. So for this case

$$\mathbf{B} = \{\boldsymbol{\xi} \in \mathbf{X} : h_i(\boldsymbol{\xi}) < 0, h_e(\boldsymbol{\xi}) > 0, h_c(\boldsymbol{\xi}) < 0, \forall i \in I_B\}. \tag{5}$$

Proposition 2 *Given the following:*

1. *A flow generated by a smooth vector field, f*
2. *A target region, $\mathbf{T} \subset \mathbf{X}$*
3. *A set of smooth hypersurfaces, $h_i, i \in I_B \subset 2^I$*
4. *A smooth hypersurface (exit boundary), h_e*
5. *A smooth hypersurface (cap boundary), h_c*

such that $\mathbf{B} = \{\boldsymbol{\xi} \in \mathbf{X} : h_i(\boldsymbol{\xi}) < 0, h_e(\boldsymbol{\xi}) > 0, h_c(\boldsymbol{\xi}) < 0, \forall i \in I_B\} \neq \emptyset$ *and* $\overline{\mathbf{B}}$ *(closure of* $\mathbf{B}$*) is compact. For all* $\boldsymbol{\xi} \in \mathbf{B}$ *there is a finite time, t, such that* $\mathbf{x}(0) = \boldsymbol{\xi}, \mathbf{x}(t) \in \mathbf{T}$, *if the following conditions are satisfied:*

1. $\nabla_{\xi} h_i(\boldsymbol{\xi}) \cdot f(\boldsymbol{\xi}) = 0, \forall i \in I_B$
2. $\nabla_{\xi} h_c(\boldsymbol{\xi}) \cdot f(\boldsymbol{\xi}) < 0, \forall \boldsymbol{\xi} \in \mathbf{B} \cap \mathcal{N}(h_c)$
3. $\mathbf{B} \cap \mathcal{N}(h_e) \subset T$
4. *There are no limit sets in* $\overline{\mathbf{B}}$

Proof: See [2,3]. □

Remark: Each of the two propositions gives sufficient conditions for a set of hypersurfaces to form a common flow region. They can be used to check a design.

Remark: For a discussion of how to find the invariant hypersurfaces see [2,3].

3 Optimizing the Invariant Based Approach

The invariant based approach to interface design produces a set of hypersurfaces in the state space of the plant. These hypersufaces form the boundaries of CFRs, which are regions of the state space where all the trajectories flow to the same subsequent CFR under a particular control policy. By driving the state through a sequence of CFRs the controller can eventually drive it to the target region. Since each CFR is associated with a unique control policy, the controller has only determine the CFR in which the current state resides and apply the associated policy.

In general, however, the CFRs will intersect each other, and therefore the plant state may well lie in multiple CFRs simultaneously. In such cases the controller must choose one of the CFRs in order to decide which control policy to apply. The controller design procedure described here bases this choice on the optimization of some measure of performance. The particulars of the measure used are not important to the design procedure provided they can be expressed as a cost associated with each CFR. The controller will choose to drive the plant through the sequence of CFRs which reaches the target region with the lowest total cost.

3.1 Problem Formulation

Assume the invariant based design scheme has been completed. The result is a set of hypersurfaces which identify a set of CFRs. The set of CFRs is denoted by $\mathcal{B}$ where $\mathcal{B} \subset \mathbf{P}(\mathbf{X})$. For each CFR, $\mathbf{B} \in \mathcal{B}$, four functions are defined.

$depth(\mathbf{B}) \in \{0..D\}$	depth of CFR **B**
$cost(\mathbf{B}) \in \Re$	cost of traversing **B**
$CP(\mathbf{B}) \in \{0..K\}$	control policy for **B**
$next(\mathbf{B}) \in \mathcal{B}$	CFR reached via CFR **B**

These functions are determined by the interface design algorithm. The depth of a CFR refers to the number of CFRs traversed before reaching the target. The target has a depth of 0. The cost of traversing a CFR will depend on the particular control problem, it can reflect the control effort required, the performance of the plant when its state lies in the CFR, etc. The the function, *next*, indicates which CFR is reached from the present CFR. This function can be extended to in the following way.

$$\begin{aligned} next^0(\mathbf{B}) &= \mathbf{B} \\ next^1(\mathbf{B}) &= next(\mathbf{B}) \\ next^2(\mathbf{B}) &= next(next(\mathbf{B})) \\ &\vdots \end{aligned}$$

The set of hypersurfaces which form the boundaries for the various CFRs will form a set of open regions, $\mathcal{S} \subset \mathbf{P}(\mathbf{X})$, in the plant state space. Each of these open regions may belong to a number of CFRs. Through observation of the sequence of plant events the controller knows which of these open regions contains the plant state and must decide which control policy to apply. The following function identifies the CFRs which contain a given open region, S.

$$CFR(\mathrm{S}) \subset \mathcal{B} \quad \text{CFRs containing region S}$$

The relation $\mathbf{B} \in CFR(\mathrm{S})$ is equivalent to $S \subset \mathbf{B}$.

3.2 Optimization

When the plant state is in the open region, S, the controller will select from among the CFRs in the set *CFR*(S), the CFR which provides the lowest total cost to reach the target. The total cost to reach the target from CFR **B** is given by the following summation.

$$\sum_{i=0}^{depth(\mathbf{B})} cost(next^i(\mathbf{B})) \tag{6}$$

Therefore when the plant state is in the open region S, the controller will apply the control policy

$$CP(\mathbf{B})$$

where **B** is determined by the following optimization,

$$\min_{\mathbf{B}\in CFR(S)} \sum_{i=0}^{depth(\mathbf{B})} cost(next^i(\mathbf{B})). \tag{7}$$

3.3 Designing the Optimal Controller

The most straightforward way to implement the controller is to create an automaton with one state for each of the open regions in the set, $\mathcal{S}$. Then as the plant evolves and produces plant symbols, the controller can track the plant state and provide the optimal control policy as defined by (7). Therefore the output function for the controller is given by

$$\phi(S) = CP(\mathbf{B})$$

where **B** is given by (7).

Remark: Using this controller design scheme, it is possible for the controller to change its mind after selecting a CFR as optimal. If, while traversing the selected CFR, the plant state passes through another CFR the controller may find a lower cost in the new CFR and abandon the previous one.

4 An Implementation of the Invariant Based Approach

The design procedure presented in this paper is straightforward, but not computationally easy. For these reasons it is possible as well as desirable to automate the procedure on a computer. Here a simple automation scheme is presented along with results for an example.

The computerized procedure does not find the hypersurfaces which bound the common flow regions, but rather seeks to identify the regions directly. This is achieved by simply back-calculating the state trajectories from the target region and recording the states which are encountered. It requires quantizing and bounding the state space so that the computer will have a finite number of state values to deal with.

To use the program the designer must choose the quantization levels for each state, Δx_i, and the range of each state, $x_{i,\min}$ and $x_{i,\max}$. As can be seen, the order of the computational complexity is q^n, where q is the number of quantization levels and n is the number of states. This limits the size of systems which can be handled in a reasonable amount of time. The limit depends on the computational power available and the on the designer's idea of what is "reasonable". The following example had $q^n \approx 10^5$ and it required 23 minutes of cpu time on a SPARC station 10.

After the designer has quantized and bounded the state space, the procedure requires two additional pieces of information from the designer. The set of available plant inputs (control policies) must be provided and the target region must be specified in terms of the cells. That is, the cells which lie in the target region must be identified as such by the designer.

Once the program has the requisite information, it proceeds according to the algorithm outlined by the flowchart shown in Figure 2. The algorithm shown in the chart will locate all cells from which it is possible to reach the target region via the application of any one control policy. The program resets the target region to include all these cells and then repeats the algorithm. In this way, all cells from which it is possible to reach the original target via the application of two control policies in sequence are identified. The program will repeat the algorithm as many times as the designer has specified. When the program finishes, each cell is marked with the control policy which should be used to reach the target, either directly or as the first in a sequence of control policies.

4.1 Example - Unmanned Underwater Vehicle

Unmanned underwater vehicles (UUV) have a wide variety of practical usages in missions where it is dangerous or impossible to send a manned underwater vehicle. Exploration, search and rescue, salvage, mine disposal, and demolition are some examples of the uses of UUV's.

This example uses a simplified model of a six-degree-of-freedom UUV depicted in Figure 3. The three types of linear displacement are *surge, sway,* and *heave*, which represent translation in the x, y, and z, directions respectively. The three types of angular displacement are *roll, pitch,* and *yaw.* The model employed here has six states which are the time derivatives of the three linear displacements and the magnitudes of the three angular displacements. By expanding to a nine state model, the magnitudes

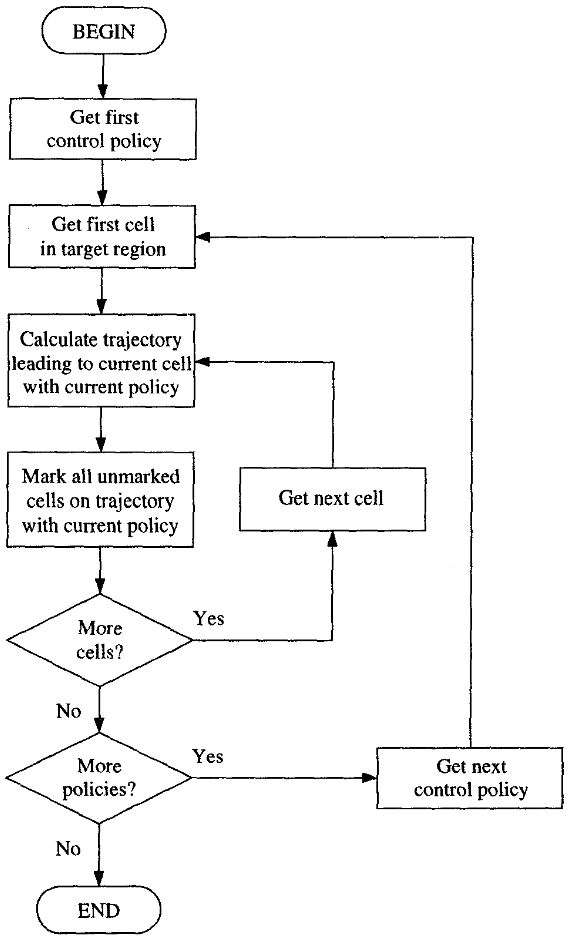
BEGIN
Get first control policy
Get first cell in target region
Calculate trajectory leading to current cell with current policy
Mark all unmarked cells on trajectory with current policy
Get next cell
More cells?
Yes
No
More policies?
Yes
Get next control policy
No
END

Figure 2: Algorithm

of the linear displacements, could also be included. They are omitted here to simplify the control problem and because they do not affect the dynamics of the UUV.

The UUV model has three inputs which control the rudder, stern plane, and screw. The following table summarizes the variables of the model.

x	surge rate (forward speed)
y	sway rate (lateral speed)
z	heave rate (vertical speed)
θ_x	roll angle in radians
θ_y	pitch angle in radians
θ_z	yaw angle in radians
u_x	screw
u_y	stern plane angle
u_z	rudder angle

The simplified model is as follows.

$$\begin{aligned}\dot{x} &= -x + u_x \\ \dot{y} &= -y + 0.01xu_y\end{aligned}$$

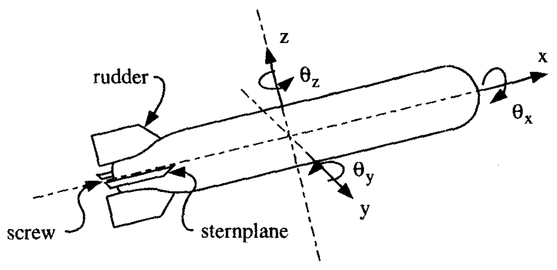

Figure 3: Unmanned Underwater Vehicle

$$\begin{aligned}\dot{z} &= -z + 0.01xu_z \\ \dot{\theta}_y &= 0.15xu_y \\ \dot{\theta}_z &= 0.15xu_z\end{aligned} \quad (8)$$

Notice that the roll angle is not included in the model. It is assumed that the center of mass of the UUV is sufficiently far beneath the center of bouyancy so that the roll angle is always zero.

The algorithm is used to design a controller for the UUV and then the design is evaluated through simulation. The state space of the UUV plant is quantized and bounded as follows.

$$\begin{array}{rcccll} 0 & \le & x_1 & \le & 1, & \Delta x_1 = 0.1 \\ -0.5 & \le & x_2 & \le & 0.5, & \Delta x_2 = 0.2 \\ -0.5 & \le & x_3 & \le & 0.5, & \Delta x_3 = 0.2 \\ -\pi/2 & \le & x_4 & \le & \pi/2, & \Delta x_4 = \pi/20 \\ -\pi/2 & \le & x_5 & \le & \pi/2, & \Delta x_5 = \pi/20 \end{array}$$

The controller has ten control policies to choose from. These policies are obtained by combining two propeller speeds $u_x \in \{0, 1\}$, three stern plane angles $u_y \in \{-10, 0, 10\}$, and three rudder angles $u_z \in \{-10, 0, 10\}$. Of the control policies with $r_1 = 0$, only the one with $r_2 = r_3 = 0$ is kept, reducing the total number of control policies from eighteen to ten.

The target region consists of the following interval.

$$\begin{bmatrix} 0.1 \\ -0.1 \\ -0.1 \\ -\pi/40 \\ -\pi/40 \end{bmatrix} < \mathbf{x} < \begin{bmatrix} 0.2 \\ 0.1 \\ 0.1 \\ \pi/40 \\ \pi/40 \end{bmatrix} \quad (9)$$

The results of a simulation is presented here. The trial has the following initial conditions.

$$\mathbf{x} = \begin{bmatrix} \text{surge} \\ \text{sway} \\ \text{heave} \\ \text{pitch} \\ \text{yaw} \end{bmatrix}, \mathbf{x}_1 = \begin{bmatrix} 0.8 \\ 0 \\ 0 \\ -1.17 \\ 1.00 \end{bmatrix} \quad (10)$$

Results of the trial are shown in Figure 4, which consists of four graphs. The graph in the upper left

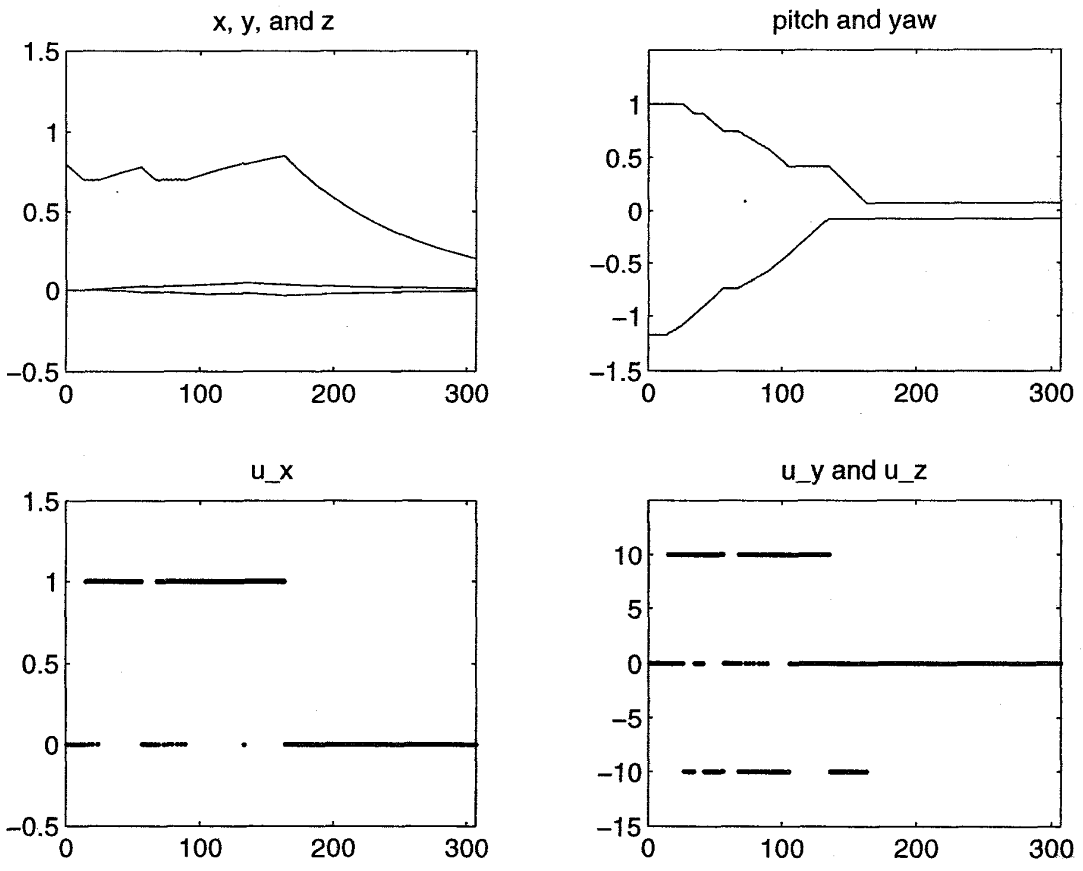

Figure 4: UUV Simulation #1

shows the surge, sway, and heave over time. The graph in the upper right shows the pitch and yaw over time. The trajectories on these two graphs can be distinguished by noting the initial conditions. The two lower graphs show the control signal. On the lower left is the propeller speed which is either 0 or 1. The segments which appear to overlap reveal the presence of chattering (due to quantization). The final graph shows the stern plane angle and the rudder angle. In Figure 4, the stern plane switches between 0 and 10 and the rudder angle switches between −10 and 0. As can be seen, the basic control strategy which developed is simply to accelerate and turn until the pitch and yaw are within the bounds of the target region, and then coast until the forward speed is also in the target.

Acknowledgement - The financial support of NSF/EPRI (grants MSS92-16559 and RP8030-06) is gratefully acknowledged.

References

[1] P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors, *Hybrid Systems II*, Lecture Notes in Computer Science. Springer-Verlag, 1995, To appear.

[2] R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors, *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, Springer-Verlag, 1993.

[3] J. A. Stiver, P. J. Antsaklis, and M. D. Lemmon, "Digital control from a hybrid perspective", In *Proceedings of the 33rd Conference on Decision and Control*, pp. 4241–4246, Lake Buena Vista, FL, December 1994.

[4] J. A. Stiver, P. J. Antsaklis, and M. D. Lemmon, "Interface Design for Hybrid Control Systems", Technical Report of the ISIS Group (Interdisciplinary Studies of Intelligent Systems) ISIS-95-001, University of Notre Dame, January 1995.