Proceedings of the International Joint Conference on Neural Networks, pp. 1036-41, May 1998

Optimal Control Using a Neural/Evolutionary Hybrid System

Dan Ventura and Tony Martinez
Neural Network and Machine Learning Laboratory (http://axon.cs.byu.edu)
Department of Computer Science, Brigham Young University, Provo, UT 84602 USA

One of the biggest hurdles to developing neurocontrollers is the difficulty in establishing good training data for the neural network. We propose a hybrid approach to the development of neurocontrollers that employs both evolutionary computation (EC) and neural networks (NN). EC is used to discover appropriate control actions for specific plant states. The survivors of the evolutionary process are used to construct a training set for the NN. The NN learns the training set, is able to generalize to new plant states, and is then used for neurocontrol. Thus the EC/NN approach combines the broad, parallel search of EC with the rapid execution and generalization of NN to produce a viable solution to the control problem. This paper presents the EC/NN hybrid and demonstrates its utility in developing a neurocontroller that demonstrates stability, generalization, and optimality.

1. Introduction

Any system developed for control of an external plant should possess certain characteristics. For instance, the system should prevent the plant state from oscillating or "pinging", maintaining instead a smooth, stable trajectory for all plant state variables. Also, the control system will hopefully generalize not only to novel plant states, but also to reasonable changes in plant parameters. Finally, the controller should faithfully track the desired trajectory of plant output variables, maintaining plant operation at or near optimal levels according to some metric for performance level. To summarize, any controller should possess the following characteristics:

- Stability,
- Generality,
- Optimality.

Although neural networks (NN) possess great potential for the control of complex systems, the field of neurocontrol faces difficult problems as well. One of the most difficult involves proper training of NN for control of complex systems, which is a complicated endeavor when the system to be controlled is open-loop unstable. This paper extends and improves a method first proposed in [12] for combining evolutionary computation (EC) [3] with NN to control such a system. The broad, parallel search capabilities of the EC are employed to find regions of the system state space that are stable, and the survivors of the evolution constitute the training set used to train the NN for use as a neurocontroller. The goal of this paper is to show that controllers developed using this method do exhibit the desired characteristics listed above.

Combination of NN and EC technology is becoming more prevalent and usually focuses on using EC to develop the architecture (the weights, the topology, or both) of the NN. For example see [2], [4], [9], and [13]. On the other hand, the work presented here presupposes some NN architecture and focuses on using EC to develop a training set suitable for training the NN. It extends previous work done by the authors in which the combination of EC and NN was used for optimization. Though much less common, some work similar in flavor to this approach does exist including [5], [6], and [8].

Section 2 describes the problem of training an NN for system control and section 3 then describes the hybrid EC/NN approach that is the main contribution of this paper. Section 4 discusses applying this approach to the well-known pole balancing problem and demonstrates empirically that the controller developed possesses the desired characteristics of stability, generality, and optimality. Finally, section 5 provides conclusions and directions for future work.

2. Problem description

Given a plant, Θ, the state of Θ may be described at time t by a vector of status variables, $\boldsymbol{s}^t$. Control of the plant is effected by Γ which applies a control vector, $\boldsymbol{c}$, to Θ. That is, given a plant state at time t described by vector $\boldsymbol{s}^t$, the setting of the values of the vector $\boldsymbol{c}$ will result in a different plant state at time $t+\delta$ described by the vector $\boldsymbol{s}^{t+\delta}$. The problem is how should Γ be constructed so that given a status vector, $\boldsymbol{s}^t$, Γ outputs a control vector $\boldsymbol{c}$ such that $\boldsymbol{s}^{t+\delta}$ describes a better plant state, if possible, than $\boldsymbol{s}^t$? We assume the existence of some evaluation function, f, that will determine whether or not one plant state is better than another. The operation of Θ

may be either continuous or discrete and Γ has no information about the internal dynamics of Θ. The only information about Θ available to Γ is the value of $\boldsymbol{s}^t$. Given $\boldsymbol{s}^t$, Γ is expected to output values for $\boldsymbol{c}$, the goal being to maximize f for any given instance (state) of Θ. In earlier papers [10] [11], single iteration, open-loop optimization type problems have been considered.

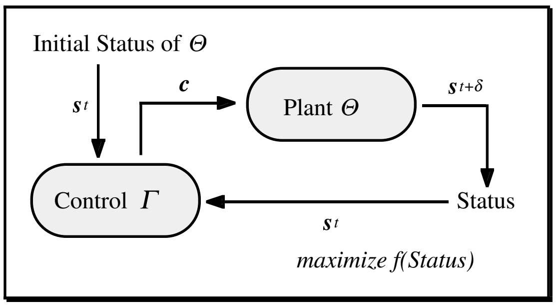

Figure 1. Control system with feedback

Here, we address the problem of continuous, closed loop control using feedback (see figure 1). We assume that the plant Θ to be controlled is open-loop unstable, and the goal is to develop a neurocontroller for the Γ component of the system that will optimize (in some sense) the operation of the plant Θ. Since it is assumed that the internal dynamics of the plant are unknown (if they are known, a good controller may be developed using conventional control theory), using a neural network that learns to control the plant using only the externally available information (plant state $\boldsymbol{s}^t$) is a promising option. However, we are still faced with a difficult problem. Namely, neural networks usually learn inductively by repeated exposure to preclassified examples. What examples should be used for training the neurocontroller? The following section details the use of evolutionary computation and "black box" access to the plant (that is, only the plant state $\boldsymbol{s}^t$ is available) to develop a training set for the neurocontroller.

3. Using evolutionary computation to produce a training set

Assuming that the status and control variables are defined over even a modest range, it is obvious that the input (status) and output (control) spaces will be extremely large. Evolutionary computation lends itself well to the exploration of large spaces. However, such evolutionary exploration is often slow. If we assume that the mapping $\boldsymbol{s} \rightarrow \boldsymbol{c}$ is nonrandom and in some sense generalizeable, then a representative set of points may be discovered via EC, and those points can then be used to train a NN which may then generalize over the rest of the function. The goal of the EC/NN synergism is to obtain the accuracy of evolutionary search and the speed of neural execution.

From the space spanned by $\boldsymbol{s}$ that describes Θ we choose a representative set of plant states by choosing n initial status vectors. We denote these $\boldsymbol{s}_i^{t=0}$, $0<i\leq n$ and refer to the plant described by state $\boldsymbol{s}_i^{t=0}$ as $\Theta_i^{t=0}$, $0<i\leq n$. These choices could be random or could be biased by any *a priori* heuristics as to what constitutes a realistic or desirable plant state (for example, see section 4). The goal is to know, given one of these $\boldsymbol{s}_i^{t=0}$, what a "good" $\boldsymbol{c}$ vector would be. For each of the $\boldsymbol{s}_i^{t=0}$, EC is used to discover such a $\boldsymbol{c}$ in the following way.

Assume a fitness function f that takes as input a status vector $\boldsymbol{s}$ and returns a real-valued fitness measure. Now for each $\boldsymbol{s}_i^{t=0}$, randomly initialize a population of m control vectors, denoted $\boldsymbol{c}_k$, $0<k\leq m$. Evaluate the initial population by simulating the workings of $\Theta_i^{t=0}$ for δ time steps for each $\boldsymbol{c}_k$ and then applying fitness function f to $\boldsymbol{s}_i^{t=\delta}$. Next, until some stopping criterion is reached (a maximum number of generations or a threshold fitness value is achieved, for example), choose parents and use genetic operators (crossover, mutation, etc.) to produce m offspring, evaluate the children and select m survivors from amongst the parents and children. Finally, for each of the n populations, choose the individual, $\boldsymbol{c}_{max_i}$, (the individual with the highest fitness) and build a set of n training examples of the form $\boldsymbol{s}_i^{t=0} \rightarrow \boldsymbol{c}_{max_i}$. The algorithm is summarized in figure 2.

evolve ()
  generate $\boldsymbol{s}_i$
  for each $\boldsymbol{s}_i^{t=0}$
    initialize population $\boldsymbol{c}_k$, $0<k\leq m$.
    evaluate ($\boldsymbol{c}_k$,Θ_i)
    until(done)
      select parents from $\boldsymbol{c}_k$
      apply genetic operators to parents
      evaluate (children,Θ_i)
      $\boldsymbol{c}_k \leftarrow$ choose m survivors

evaluate ($\boldsymbol{c}$,Θ_i)
  run $\Theta_i^{t=0}$ for δ time steps with control $\boldsymbol{c}$
  return $f(\boldsymbol{s}_i^{t=\delta})$

Figure 2. Algorithm for evolving training set

The EC has now found "good" approximate solutions for n points from the input (status) space but can say nothing about any other points, many of which are likely to be encountered during normal execution of Θ.

Obviously, one solution to the problem defined in section 2 would be to employ the evolutionary scheme discussed above as the control Γ. However, this would likely be unacceptable in terms of execution speed. Therefore, the NN is employed to generalize over the entire space defined by $\boldsymbol{s}$ using the relatively small set of approximate solutions as a training set. While the initial training of the network maybe somewhat time consuming, depending on the network and training algorithm employed, the generalization during execution will be extremely fast. The synergistic combination of EC and NN is then employed as the control Γ as in figure 3.

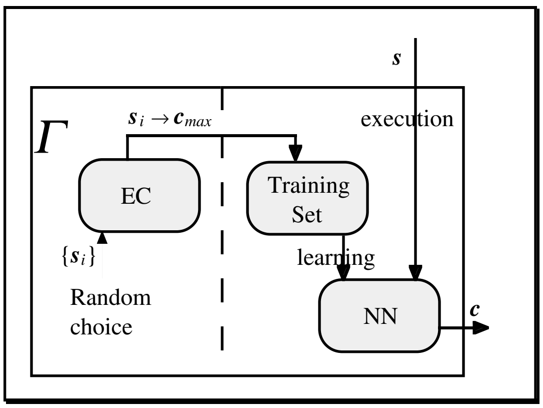

Figure 3. The control Γ

It should be noted that since many (if not most) interesting control problems involve plants that are open-loop unstable, care must be taken in the choice of the parameter δ. On the one hand, if δ is too short, the effect of applying the control to the system will not be readily apparent. On the other hand, if δ is too long, the instability of the plant will have destroyed any useful measure of how good the control vector was.

The power of this EC/NN hybrid approach is its combination of the thoroughness of evolutionary search with the speed and accuracy of neural generalization. Further, it is generally applicable to any control problem for which a fitness function can be found and for which "blackbox" access to the system to be learned (or to a reasonable simulation) is possible. In order to provide proof-of-concept, the next section discusses using the EC/NN approach to solve one such control problem.

4. The pole balancing problem -- an example

The pole balancing problem is a well known, textbook example of a complex control system. The problem exists in many variations, but perhaps the most common consists of a pole attached by a hinge to a wheeled cart that sits in a short track. The challenge is to apply force to the cart in order to keep the pole upright and at the same time not run the cart into the side of the track (since that would of course cause the pole to fall). Figure 4 gives a simple diagrammatic representation.

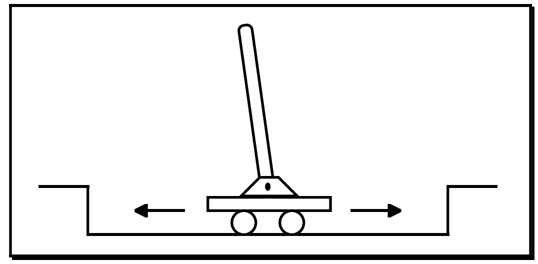

Figure 4. The pole balancing problem

The system is inherently unstable and the solution is, of course, a gentle oscillatory application of force in first one direction and then the other. Other variations on this theme include infinite length tracks, multi-jointed poles, and multiple poles on the cart.

This system was simulated using both Euler and Runge-Kutta approximations (with the integration time step equal to .005 seconds) for solving the second order differential equations that describe the pole/cart system. For this simple problem, the physics is well understood and the equations represent an extremely accurate simulation of the cart and pole motion, including accounting for friction between the pole and the cart and between the cart and the track (see Appendix A for details). A training set of 200 instances of the form $\left(\theta, \dot{\theta}, \ddot{\theta}\right) \rightarrow F$ was generated using the evolutionary method described in section 3. The length of time, δ, allowed for system stabilization during the evolutionary evaluation was set to 3 time steps (.015 seconds). As noted earlier, during training set generation, any knowledge of which plant states are more likely or more useful (stable) should be incorporated into the evolutionary process in order to concentrate on exploring those parts of the state-space that will be most helpful for neural network generalization. In the case of the pole balancer, we are most interested in the those states in which the pole angle and its derivatives are relatively small (since once any of these variables become too large the pole will fall for sure due to the system limitations on available force). Once the training set was generated, it was used to train a simple backpropagation [7] network with a single hidden node for 1000 epochs. The learning rate was set to .5, and a momentum term with a coefficient of .95 was also employed. To test the controller, the pole was given several initial non-zero angles and the system was expected to balance the pole. Tests were performed over varying time increments from 30 seconds up to 15 minutes. In all cases, once the controller stabilized the plant, there was no significant deviation off optimal, deadbeat control. Figures 5 and 6 show the first 10 seconds of the trajectory of the pole's angular vertical displacement, θ, for two such tests (initially $\theta = .4$ radians ≈ 23° and $\theta = -.78$ radians ≈ 45° respectively).

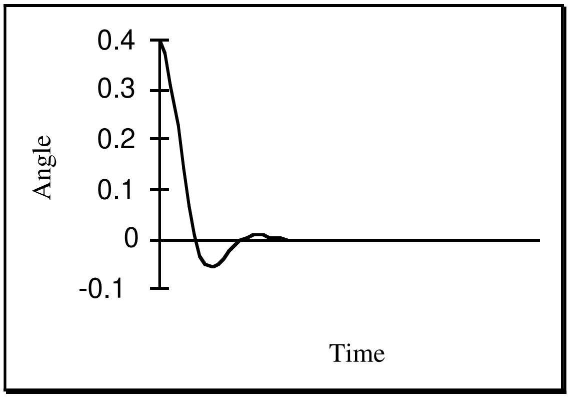

Figure 5. θ-trajectory over 10 seconds for initial pole angle of $\theta = .4$ radians ≈ 23°

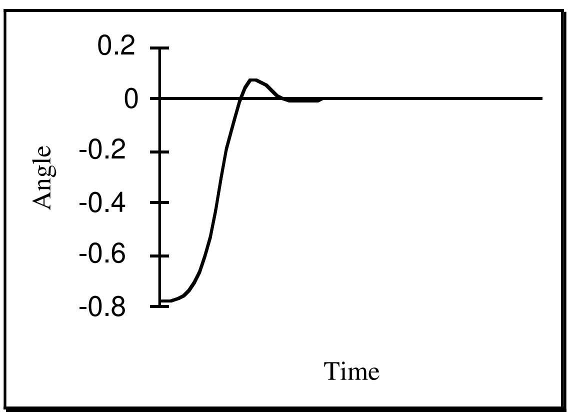

Figure 6. θ-trajectory over 10 seconds for initial pole angle of $\theta = -.78$ radians ≈ -45°

Both tests demonstrate nicely the controller's ability to stabilize the plant. Also, not only is the plant stable, but also it can in several senses be considered to be optimal. First, the number of initial oscillations required to stabilize is just three, the minimum for stabilizing a second order system. Second, the amplitude of those oscillations is very small (maximum overshoot of 4.3 degrees for the -45 degree case, for example). Third, once the plant is stable, it maintains "deadbeat" control right on the desired trajectory of $\theta = 0°$. Finally, since neither of these initial plant states was encountered during training, the control system demonstrates an ability to generalize to previously unseen plant states. To further test the system's generalization ability, the plant parameter corresponding to length of pole (see Appendix A) was changed *after* training. In other words, the control was trained with a pole of one length (1 meter) and tested with poles of other lengths. Thus the control is required to generalize not just to novel plant states but to a plant that is different from the one it was trained to control. Figure 7 shows the first 20 seconds of the θ-trajectory for a 4 meter pole with an initial angle of $\theta = -.78$ radians ≈ 45°.

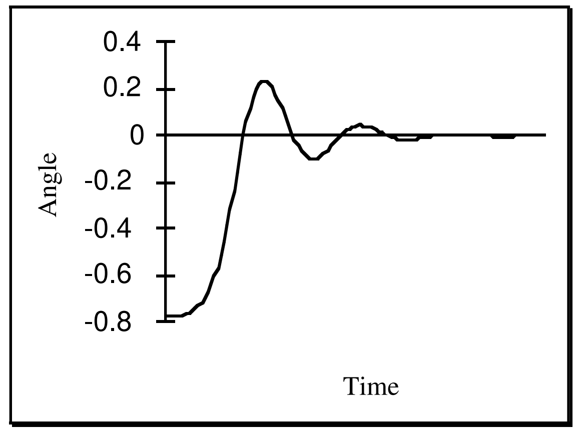

Figure 7. θ-trajectory for 4 meter pole with initial pole angle of $\theta = -.78$ radians ≈ -45°

Notice that the controller is still stable, though the response is not as good. This is, of course, to be expected, but the important point is that a controller trained on one plant is able to control another. Other experiments were conducted altering pole mass, both pole mass and pole length together, and the gravitational constant, etc. (see Appendix B). In the case of altering the gravitational constant, the controller performed very well from .5g up to 1.25g, with little drop-off in response. In other cases, some performance degradation is observed (as in figure 7) as expected. Finally it should be mentioned that because of the architecture of the EC/NN hybrid, it is possible to continue generating training instances in the background and to then improve the controller by on-line learning. In fact, because of the existence of the fitness function, the controller could monitor itself and note the plant states for which its performance could be improved. The controller could then request the EC to generate training instances from regions of the state space near these plant states.

5. Conclusions

A new evolutionary/neural hybrid approach for the development of neurocontrol has been demonstrated. In particular, it has been shown how evolutionary computation and "black box" access to the plant can be used to generate a suitable training set for training a neural network to act as a controller of an inherently unstable system. Further, the resulting controller has been demonstrated to possess the desirable characteristics of

- Stability,
- Generalization, and
- Optimality.

Because many interesting control problems exist for which the dynamics of the plant are unknown, neural networks offer a viable approach to approximating desired control of a plant. However, because many plants are open-loop unstable, training the neural network for control is difficult. The use of evolutionary computation, along with a judicious choice of the stabilization parameter δ facilitates the development of neurocontrol for unstable systems. Currently, we are applying the methods described here to the real world problem of controlling the combination of engine and airframe for a high performance military aircraft. Future work includes quantifying the choice of δ, theoretical analysis and quantification of stability limits, generalization capabilities and optimality results, and combining these techniques with those that evolve the network architecture.

Acknowledgments

The authors would like to thank George Lendaris and Tad Shannon of Portland State University for many insightful comments and helpful suggestions, as well as for providing the code for the pole- cart simulation.

6. References

[1] Barto, Andrew G., Richard S. Sutton and Charles W. Anderson, "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. smc-13, no. 5, September/October, 1983.
[2] Caudill, Maureen, "Evolutionary Neural Networks", *AI Expert*, vol. 6, no. 3, pp. 28-33, March 1991.
[3] Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Publishing, 1989.
[4] Harp, S. A., T. Samad and A. Guha, "Designing Application-Specific Neural Networks Using the Genetic Algorithm", *NIPS-89 Proceedings*, 1990.
[5] Miagkikh, V. V., A. P. Topchy and O. A. Lebedko, "Fast Learning in Multilayered Networks by means of Hybrid Evolutionary and Gradient Algorithms", *Proceedings of the International Conference on Evolutionary Computation and it Applications*, pp. 390-8, June 1996.
[6] Montana, D. J. and L. Davis, "Training Feedforward Neural Networks Using Genetic Algorithms", *Proceedings of the Third International Conference on Genetic Algorithms*, 1989.
[7] Rumelhart, David E., James L. McClelland and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, MIT Press, 1986.
[8] Romaniuk, Steve G., "Evolutionary Growth Perceptrons", *Genetic Algorithms: 5th International Conference (ICGA-93)*, S. Forrest (ed.), Morgan Kaufmann, 1993.
[9] Saunders, Gregory M., Peter J. Angeline and Jordan B. Pollack "Structural and Behavioral Evolution of Recurrent Neural Networks", *Advances in Neural Information Processing Systems*, vol. 6, pp. 88-95, Morgan Kaufmann Publishers Inc., 1994.
[10] Ventura, Dan and Tony Martinez, "Robust Optimization Using Training Set Evolution", *Proceedings of the International Conference on Neural Networks*, pp. 524-8, 1996.
[11] Ventura, Dan and Tony Martinez, "A General Evolutionary/Neural Hybrid Approach to Learning Optimization Problems", *Proceedings of the World Congress on Neural Networks*, pp. 1091-5, 1996.
[12] Ventura, Dan and Tony Martinez, "Using Evolutionary Computation to Generate Training Set Data for Neural Networks", *Proceedings of the International Conference on Artifical Neural Networks and Genetic Algorithms*, pp. 468-71, 1995.
[13] Wieland, Alexis P., "Evolving Controls for Unstable Systems", *Proceedings of the 1990 Connectionist Models Summer School*, pp. 91-102, 1990.

Appendix A: dynamics of the pole balancing problem

The nonlinear differential equations describing the pole balancing system used in the simulations are taken from [1] and are as follows.

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

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

Here x is the horizontal position of the cart, θ is the angle of the pole off of vertical, F is the force applied to the cart, m_c and m are the masses of the cart and pole respectively, l is the half length of the pole, μ_c and μ_p are the friction coefficients for the cart and pole respectively, and g is the gravitational constant. Table 1 gives the values used for the variables during the simulations.

Table 1. Values for the pole balancing simulation

Variable	Meaning	Value
x	horizontal cart position	±10 m
θ	angle of pole off vertical	±1.57 rad
F	force applied to cart	±50 N
m_c	mass of cart	1 kg
m	mass of pole	.1 kg
l	half length of pole	.5 m
g	gravitational constant	-9.8 m/s^2
μ_c	friction of cart on track	.0005
μ_p	friction of pole on cart	.000002

Appendix B: further experimental results

All results shown are for 10 seconds. Figure 8 shows results of using only the angle and its velocity (and not the angular acceleration) as inputs to the controller. Figure 9 shows the result of changing the gravitational constant to .5g, and Figure 10 shows the result for a pole with 4 times more massive than the pole used during training (.4 kg for testing but .1 kg for training).

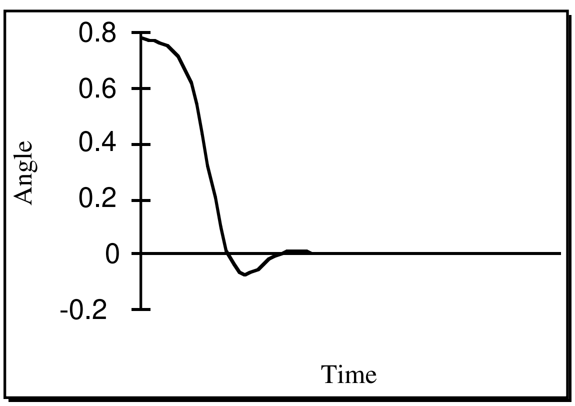

Figure 8. θ-trajectory using only 2 inputs with initial pole angle of θ = .78 radians ≈ 45°

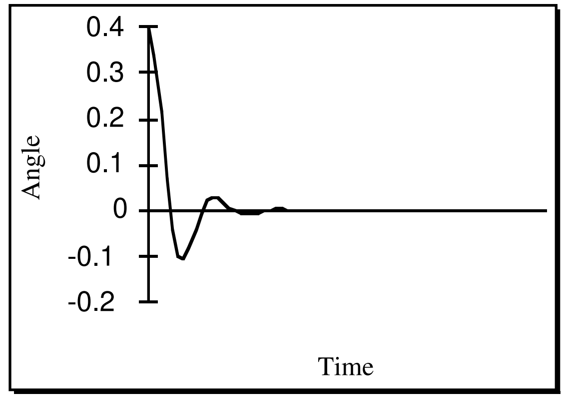

Figure 9. θ-trajectory for .5g with initial pole angle of θ = .4 radians ≈ 23°

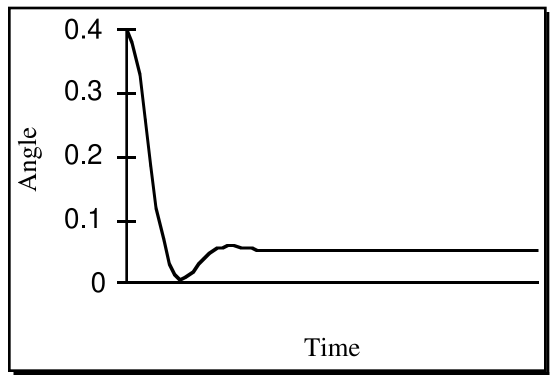

Figure 10. θ-trajectory for .4 kg pole with initial pole angle of θ = .4 radians ≈ 23°