Computer Science and Artificial Intelligence Laboratory
Technical Report

MIT-CSAIL-TR-2009-043 September 12, 2009

Finding aircraft collision-avoidance strategies using policy search methods

Leslie Pack Kaelbling and Tomas Lozano-Perez

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

Finding aircraft collision-avoidance strategies using policy search methods

Leslie Pack Kaelbling and Tomás Lozano-Pérez

September 11, 2009

The current aircraft collision-avoidance system was hand-crafted for a particular sensor configuration and set of assumptions about aircraft dynamics and the behavior of pilots. In this work, we seek to develop collision avoidance strategies automatically, given models of the response of the aircraft and pilots to alerts generated by the system.

Assuming very high-accuracy sensing of the relative position and velocities of both aircraft, as will likely be available in the next-generation of TCAS, we can model the problem of acting to avoid collision as a Markov decision process, where the uncertainty is in the delay and degree with which the pilots will respond to advisories issued by the system.

There are several strategies for solving MDPs. Most typically, methods based on *value iteration* apply Bellman's equation in a form of dynamic programming to compute an estimate, at every state in the space, of the future value of taking each of the actions; then, the action with the highest estimated future value is executed. This method is directly applicable only to discrete state spaces. The collision-avoidance problem takes place in a moderately high-dimensional continuous space. The space is very large and cannot be effectively discretized for the full version of the problem. There are versions of the value iteration algorithm that interpolate over a continuous space that may be more effective in this situation.

In this report, we articulate two alternative strategies, which search more directly in the space of control policies. These methods increase in complexity with the complexity of the space of policies, but have a weaker dependence on the size of the state space. For each strategy, we describe a very simple first implementation and some preliminary results that may be indicative of the appropriateness of pursuing them in more depth.

1 Policy gradient method

In the policy gradient method [8, 2, 7], we articulate the agent's policy in terms of a set of parameters, which are adjusted in order to minimize the cumulative cost of the agent's behavior. Let $\pi(a \mid x; \theta)$ be the probability of taking an action a in state x when the parameters of the policy have the value θ. The policy can have any form as long as $\pi(a \mid x; \theta)$ is a differentiable function of θ for all a and x.

Our goal will be to minimize the quantity

$$E = \sum_{T=0}^{\infty} \sum_{\tilde{s} \in \tilde{S}_T} \Pr(\tilde{s}) \epsilon(\tilde{s}) \quad ,$$

where $\tilde{S}_T$ is the set of all possible experience sequences that terminate at time T. That is,

$$\tilde{s} = \langle x_0, u_0, r_0, \ldots, x_T, u_T, r_T \rangle \quad ,$$

where x_t, u_t, and r_t are the observation, action, and reward at step t of the sequence. The cost incurred by a sequence $\tilde{s}$ is

$$\epsilon(\tilde{s}) = \sum_{t=0}^{T} e(s_t) = \sum_{t=0}^{T} -\gamma^t r_t \quad ,$$

where γ is a discount factor and s_t is the length-t prefix of s.

Baird and Moore [2] showed that the gradient of the global error, E, with respect to a parameter θ_k is

$$\frac{\partial}{\partial \theta_k} E = \sum_{t=0}^{\infty} \sum_{s \in S_t} \Pr(s) e(s) \sum_{j=1}^{t} \frac{\partial}{\partial \theta_k} \ln \pi(u_{j-1} \mid x_{j-1}; \theta) \quad .$$

It is possible to perform stochastic gradient descent of this error by repeating several trials of interaction with this process, using stochastic approximation to estimate the expectation over S in the above equation. During each trial, the parameters are kept constant, and the approximate gradients of the error at each time t,

$$e(s) \sum_{j=1}^{t} \frac{\partial}{\partial \theta_k} \ln \pi(u_{j-1} \mid x_{j-1}; \theta) \quad ,$$

are accumulated.

In an incremental version of this algorithm, parameters are updated at every step t using the following rules:

$$\begin{aligned} \Delta T_k &= \frac{\partial}{\partial \theta_k} \ln \pi(u_{t-1} \mid x_{t-1}; \theta) \\ \Delta \theta_k &= \alpha \gamma^t r_t T_k \end{aligned}$$

where $0 < \alpha < 1$ is a learning rate. The quantities T_k are traces, one for each parameter θ_k. They are initialized to 0, and represent the accumulated contribution of parameter k to the action choices made by the policy in this trajectory. Each reward is used to adjust the parameters in proportion to their current trace values.

1.1 Feature-based policy and gradient

We have experimented with a feature-based policy representation. Let Φ be a function that takes an input x into a vector of k real-valued features. Then we will represent our policy as follows:

$$\pi(u|x; \theta) = \frac{1}{z(x)} Q(u, x; \theta) \quad ,$$

where

$$Q(u, x; \theta) = e^{\theta_u \cdot \Phi(x)/T} \quad ,$$

θ_u is a vector of k parameters for the discrete action u, and T is a temperature parameter. The normalizer is

$$z(x) = \sum_u Q(u, x; \theta) \ .$$

This is a Boltzmann action-selection mechanism. The closer T is to 0, the more likely we are to select the u for which $Q(x, u; \theta)$ is maximized.

In order to implement the algorithm updates, we need an analytic form of the gradient of log of the policy with respect to the parameters. For this form of policy, we have parameters $\theta_{u,k}$, one for each action u and feature k.

We must consider two cases.

Case 1: A parameter for this action:

$$\frac{\partial \pi(u \mid x; \theta)}{\partial \theta_{u,k}} = \frac{1}{T} \Phi_k(x)(1 - \pi(u \mid x; \theta))$$

Case 2: A parameter for another action ($u' \neq u$):

$$\frac{\partial \pi(u \mid x; \theta)}{\partial \theta_{u',k}} = -\frac{1}{T} \Phi_k(x) \pi(u' \mid x; \theta)$$

1.2 Features for two-dimensional problem

The point of closest approach between the two aircraft is an important quantity: we computed both τ, the time until closest approach, and d, the signed distance to the closest approach, where the sign indicates whether the point is above or below the aircraft we are controlling.

For our experiments we have used the following features.

- $\phi_1(s) = d * c_1$
- $\phi_2(s) = c_2/d$
- $\phi_3(s) = \tau * c_3$
- $\phi_4(s) = c_4/\tau$
- $\phi_5(s) = \begin{cases} 1 & \text{if an alert has been generated already} \\ -1 & \text{otherwise} \end{cases}$
- $\phi_6(s) = 1$

Adding reciprocals of the relevant features allows the policy to be sensitive to the values when they are near zero, if that is appropriate. We need to know if an alert has already been generated, in order to avoid generating extraneous alerts. The constant 1 features allows the log-linear model to have a constant term.

Gradient methods only work well when the gradient is not too small; if the feature values are too large, the gradient of the policy with respect to the weights becomes flat and the policy cannot move. Thus, we selected constants $c_1, \ldots, c_4$, automatically, to scale the feature values to be within the interval $(-1, 1)$ as much as possible.

2 Experiments

We have carried out some initial experiments in a simple two-dimensional version of this problem, in which the intruder is on a fixed course and an alert must be selected only for one of the aircraft. To test the approach, we focused on a section of the state space where collisions are very likely.

- The range of initial vertical distances (h) is $[-100, 100]$ feet
- The range of initial horizontal distances (r) is $[2000, 5000]$ feet
- The range of initial vertical velocities is $[-40, 40]$ feet per second

The other parameters in the simulation are as follows:

- Acceleration due to alert: $8 ft/s/s$
- Near mid-air collision (NMAC) zone: $500 ft$ horizontally and $\pm 100 ft$ vertically
- Horizontal closure rate: $-500 ft/s$
- Verical rate limit: $41.67 ft/s$
- Near Mid Air Encounter (NMAC) zone (in which the policy is in effect): $10000 ft$ horizontally and $\pm 500 ft$ vertically
- Cost of NMAC: 1.0
- Cost of alert: 0.01
- Per-second standard deviation in intruder's vertical velocity: $1.0 ft/s/s$

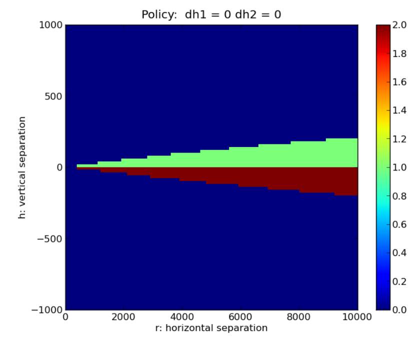
We used 3 actions: do nothing, climb > 1500, descend > 1500. The climb and descend actions accelerate until a vertical rate of $1500 ft/s$ up or down is reached.

We found that random initialization of the policy parameters led to a poor exploration of the space, with the search often converging to a local minimum. To counteract this, we performed policy gradient optimization starting from 1000 "spanning" values of the policy parameters, each of which starts with a vector of parameters for each action that has a single non-zero entry, either positive or negative. This focuses each action at the start on only one of the features. We then chose the resulting policy with the best performance. We found that this reliably led us to find policies with good scores.

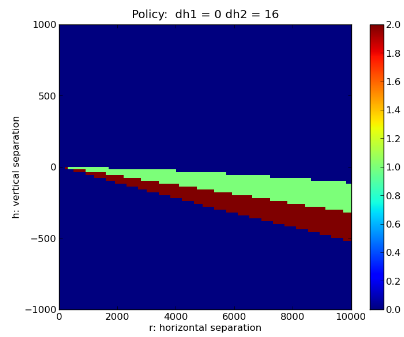
The best policy parameters from one of our runs were:

	d	$1/d$	τ	$1/\tau$	alerted	1
do nothing	-0.024	0.234	0.301	3.313	1.333	3.364
climb	0.147	13.171	-0.150	-1.346	-0.720	-0.229
descend	-0.123	-13.405	0.848	-1.966	-0.612	-0.134

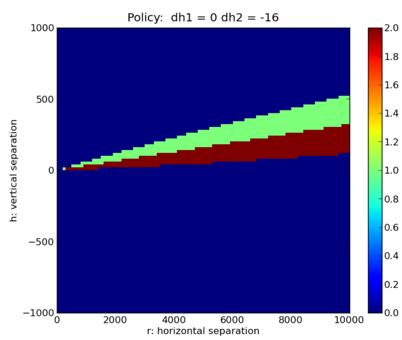
We can visualize the policy by looking at the actions chosen in various cross-sections of the state space (Figure 1). The x axis in the diagrams is horizontal distance, the y axis is vertical distance. Blue represents the *do nothing* action, green is *climb* and red is *descend.*



(a) Cross-section with $\mathrm{dh}_1 = \mathrm{dh}_2 = 0$

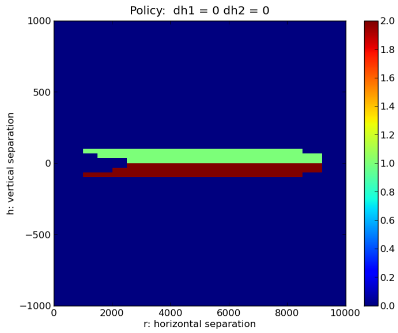


(b) Cross-section with $\mathrm{dh}_1 = 0, \mathrm{dh}_2 = 16$

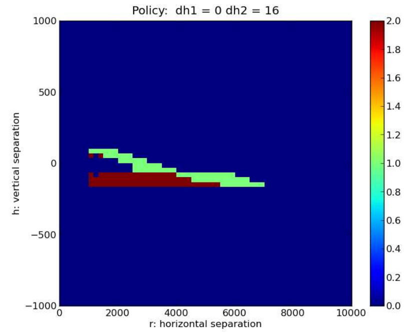


(c) Cross-section with $\mathrm{dh}_1 = 0, \mathrm{dh}_2 = -16$

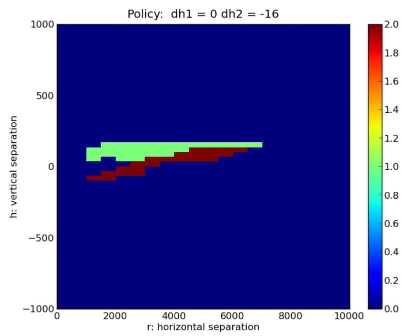
Figure 1: Policy gradient policy: dark blue = do nothing; green = descend; red = climb.



(a) Cross-section with $dh_1 = dh_2 = 0$



(b) Cross-section with $dh_1 = 0, dh_2 = 16$



(c) Cross-section with $dh_1 = 0, dh_2 = -16$

Figure 2: Value iteration policy: dark blue = do nothing; green = descend; red = climb.

We solved a discretized version of this problem (with no noise) with approximately 300,000 states using value iteration (VI); we used this as a baseline for comparing the effectiveness of the policies found via policy gradient. It was discretized into four-dimensional cells with the following boundaries:

- Horizontal distance: $[-1000, -875, -750, -625, -500, -475, -450, -425, -400, -375, -350, -325, -300, -275, -250, -225, -200, -175, -150, -125, -100, -75, -50, -40, -30, -20, -10, 0, 10, 20, 30, 40, 50, 75, 100, 125, 150, 175, 200, 225, 250, 275, 300, 325, 350, 375, 400, 425, 450, 475, 500, 625, 750, 875, 1000]$
- Vertical distance: $[0, 125, 250, 375, 500, 625, 750, 875, 1000, 1125, 1250, 1375, 1500, 1625, 1750, 1875, 2000, 2125, 2250, 2500, 2625, 2750, 3000, 3250, 3500, 3750, 4000, 4250, 4500, 4750, 5000, 5500, 6000, 6500, 7000, 7750, 8500, 9250, 10000]$
- Intruder vertical velocity: $[-40, -20, -8, -4, 4, 8, 20, 40]$
- Own vertical velocity: $[-40, -20, -8, -4, 4, 8, 20, 40]$

In addition, we stored in the state whether or not we had already alerted on this trajectory. We can visualize the policy by looking at the actions chosen in various cross-sections of the state space (Figure 2).

We found that in 10,000 random draws of initial state, the VI policy had an average discounted reward (cost) of 0.002 (for the case with no noise in intruder velocity). This policy was also robust to noise in intruder velocity, producing a nearly identical discounted reward. The policy discovered by policy gradient had an average discounted reward of 0.007 (for the case with no noise in intruder velocity). In the case with noise on the intruder velocity, the discounted reward is 0.012. Recall that the cost of one action is 0.01 the cost of a collision is 1.0. Given such a simple and compact policy representation (18 numbers for PG, $900,000$ Q values for VI), however, these results are promising.

Ultimately, we need to scale up to three-dimension representations of positions and velocities and to select actions for both aircraft; we anticipate that a relatively small policy space will still suffice in such cases, but that the state and action space will become much too large for value-based approaches.

3 Policy Search by Dynamic Programming

An alternative strategy, which is particularly appropriate for finite-horizon problems, is to search for a policy, but to do it in stages, working from horizon 0 (only a single action left to go) back to whatever horizon is desired. The learning problem at each stage is reduced to a supervised multi-class classification problem, which can be addressed by any existing classifier that is suited to the domain.

This method, called *non-stationary approximate policy iteration* (NAPI) [4] and *policy search by dynamic programming* (PSDP) [1], has been independently developed by several researchers, following a line of building reinforcement learning on classification:[6, 3]. We have explored a variation of the method that is sensitive to the fact that the aircraft collision-avoidance problem is not finite-horizon, in the strict sense that episodes last for a fixed length of time. The algorithm described below is appropriate for *episodic* tasks, in which each episode is guaranteed to terminate and where, in addition, there is some reliable indication of the expected number of steps until termination. In this collision-avoidance domain, we can use the expected time to contact as an index into a non-stationary policy. This is related to the idea of space-indexed PSDP[5].

The algorithm proceeds as follows:

1. Generate n trajectories through the space until they reach a terminal state (in this case, NMAC or the aircraft are 'behind' one another, or sufficiently vertically separated).

2. Collect all of the terminal states of the trajectories into a data set D_0, all of the penultimate states into data set D_1, etc.

3. For each state s in D_1, and each action a in the set of possible actions, simulate the outcome several times and compute an expected reward. Pair each s with the a that resulted in the highest expected reward. This set of s, a pairs constitutes a training set that can be fed into a multi-class classification algorithm, resulting in a classifier Π_1, which maps states at horizon 1 into optimal actions.

4. For horizons i from 2 through n:

    (a) For each state s in D_i and each action a in the set of possible actions, generate a simulated trajectory that starts by taking action a and then continues following policy

Π_{i-1} at the resulting state, policy Π_{i-2} at the next state, etc. until the end of the trajectory, and summing the resulting rewards. This an estimate of the value of taking action a in state s at horizon i.

(b) Pair each s with the a that maximizes the expected value, to a training set. Feed the resulting training set into a classification algorithm, obtaining Π_i.

5. Return $\Pi_1, \ldots, \Pi_N$, which constitute a non-stationary finite-horizon policy.

3.1 Pilot Implementation

To test the feasibility of PSDP in the collision-avoidance domain, we constructed a very simple implementation in Python. Using a dynamics simulation with only a small amount of noise, we found, unsurprisingly, that in many situations, there was more than one action with the same, optimal value. Arbitrarily selecting an action to put into the training set for the supervised learning causes the space to be highly fragmented, and makes it very difficult for a standard supervised-learning algorithm to get traction on the problem.

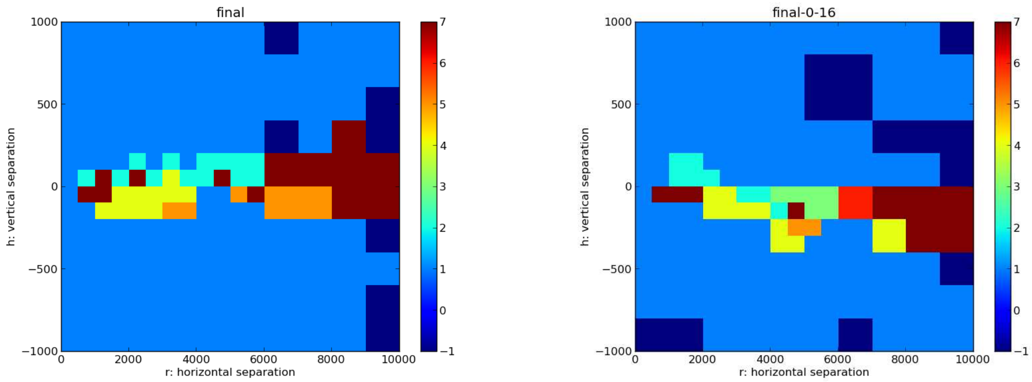
To alleviate this problem, we implemented our own classifier which took as input a set of pairs of input x_i values and *sets* of possible labels $\{y_i^1, \ldots, y_i^n\}$. The classifier was given the freedom to map each x_i to some y_i^j in the set of allowable labels, and to choose the y_i^j to simplify the complexity of the overall classifier, with the aim of improving generalization performance. The classifier we implemented was divisive, adaptively partitioning the continuous input space into regions of X space in which there was a single y value that was in the label sets of each x in the region. The pilot version of this algorithm was not noise-tolerant, but it could be extended to handle noise using methods from decision-tree learning.

Because our training data from each stage had very little overlap in feature space or in raw space, we consolidated the non-stationary policy into a single stationary policy by taking the union all of the training data used for constructing the individual supervised classifiers and used it to train a final policy. It is this final policy that we run to generate the final predictions.

We tried two different feature sets. The first was $\langle \phi_1, \phi_3, \phi_5 \rangle$, that is, the distance and time to closest approach. This worked moderately well, but we discovered that, in fact, the optimal policy is not expressible in this feature space: there are points in state space that have different optimal actions, but map to the same point in feature space. This feature space is not rich enough to express the consequences of the aircraft's current vertical velocities, and makes decisions largely on relative position. This is usually appropriate, but not in cases of extreme initial vertical velocities. Thus, we also experimented with the PSDP method in the raw state space, with features r, h, dh_1, dh_2, *alerted*, encoding relative range, height, velocities of both planes, and whether an alert had already been generated for this trajectory.

3.2 Results

Using raw features, and 1000 training trajectories, 500 of which resulted in NMAC and 500 of which did not, and testing on our standard testing distribution, the learned policy had an average cost of 0.0074, making approximately 5 collisions per 1000 trials. Two slices of the policy are shown in figure 3. Each block is labeled according to the actions that are optimal: in some blocks there is more than one optimal action.



(a) Cross-section with $dh_1 = dh_2 = 0$ (b) Cross-section with $dh_1 = 0, dh_2 = 16$

Figure 3: PSDP policy with raw features, trained on 1000 trajectories: dark blue = no data; medium blue = no alert; cyan = descend; green = no alert or descend; yellow = climb; orange = climb or no alert; red = climb or descend; brown = do any action.

Increasing to 10000 training trajectories, 5000 of which resulted in NMAC and 5000 of which did not, and testing on our standard testing distribution, the learned policy had an average cost of 0.0028, making approximately 1 collisions per 1000 trials. Two slices of the policy are shown in figure 4. This is a considerable improvement over the previous results.

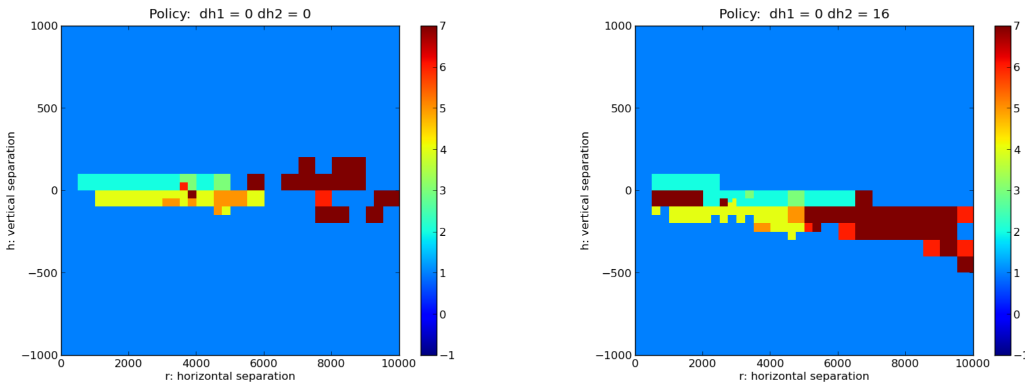
Using features 1 and 3, and 1000 training trajectories, 500 of which resulted in NMAC and 500 of which did not, and testing on our standard testing distribution, the learned policy had an average cost of 0.0027, making approximately 1 collision per 1000 trials. Two slices of the policy are shown in figure 5. Each block is labeled according to the actions that are optimal: in some blocks there is more than one optimal action. This policy is much more fine-grained, because the blocks in the classifier are in a two-dimensional, rather than a four-dimensional space, and even with just 1000 trajectories, the space is well covered.

Obvious improvements to the algorithm are:

- Use importance sampling to appropriately weight training data (currently we use rejection sampling to get half collisions and half non-collisions, but the trajectories are not appropriately weighted).
- Use importance sampling in roll-outs to estimate action values.
- Use a classifier that handles noisy data and weighted training examples.

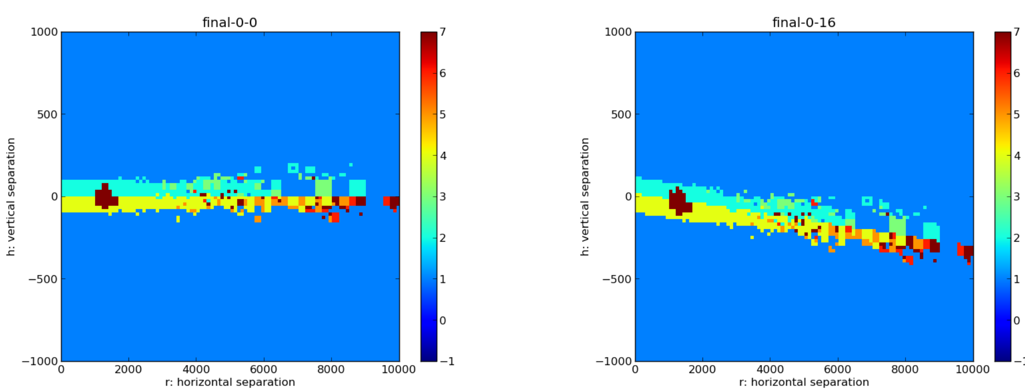
4 Conclusions

We believe that policy search by dynamic programming may be an effective technique on this problem, due to the fact that the problem is episodic, has a continuous state space, and has a discrete action space. It warrants further investigation, with a different classifier and possibly with state-indexed execution.



(a) Cross-section with $\mathtt{dh}_1 = \mathtt{dh}_2 = 0$ (b) Cross-section with $\mathtt{dh}_1 = 0, \mathtt{dh}_2 = 16$

Figure 4: PSDP policy with raw features, trained on 10,000 trajectories: dark blue = no data; medium blue = no alert; cyan = descend; green = no alert or descend; yellow = climb; orange = climb or no alert; red = climb or descend; brown = do any action.



(a) Cross-section with $\mathtt{dh}_1 = \mathtt{dh}_2 = 0$ (b) Cross-section with $\mathtt{dh}_1 = 0, \mathtt{dh}_2 = 16$

Figure 5: PSDP policy with features based on point of closest approach: dark blue = no data; medium blue = no alert; cyan = descend; green = no alert or descend; yellow = climb; orange = climb or no alert; red = climb or descend; brown = do any action.

Policy gradient methods are so highly sensitive to initial conditions and the scales of the features that they are very difficult to make work even in simple situations. It seems relatively unlikely that they will perform effectively in larger problems.

Acknowledgments

This report is the result of research sponsored by the TCAS Program Office at the Federal Aviation Administration. The authors wish to acknowledge the support of the TCAS Program Manager, Neal Suchy, and to thank collaborator Mykel Kochenderfer of Lincoln Laboratories for framing the problem and inspiring us to work on it.

References

[1] J. Andrew Bagnell, Sham Kakade, Andrew Y. Ng, and Jeff Schneider. Policy search by dynamic programming. In *Advances in Neural Information Processing*, 2003.

[2] Leemon C. Baird and Andrew W. Moore. Gradient descent for general reinforcement learning. In *Advances in Neural Information Processing Systems 11*. The MIT Press, 1999.

[3] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, 2002.

[4] Sham Machandranath Kakade. *On the Sample Complexity of Reinforcement Learning*. PhD thesis, Gatsby Computational Neuroscience Unit, University College London, 2003.

[5] J. Zico Kolter, Adam Coates, Andrew Y. Ng, Yi Gu, and Charles DuHadway. Space-indexed dynamic programming: Learning to follow trajectories. In *Proceedings of the International Conference on Machine Learning*, 2008.

[6] Michail Lagoudakis and Ronald Parr. Reinforcement learning as classification: Leveraging modern classifiers. In *Proceedings of the Twentieth International Conference on Machine Learning*, 2003.

[7] N. Meuleau, L. Peshkin, K.E. Kim, and L.P. Kaelbling. Learning finite-state controllers for partially observable environments. In *Fifteenth Conference on Uncertainty in Artificial Intelligence*, 1999.

[8] Ronald J. Williams. A class of gradient-estimating algorithms for reinforcement learning in neural networks. In *Proceedings of the IEEE First International Conference on Neural Networks*, San Diego, California, 1987.