Learning Executable Models of Multiagent Behavior from Live Animal Observation

Brian Hrolenok BHROLENOK3@GATECH.EDU
Georgia Institute of Technology, 225 North Ave NW Atlanta, GA 30332

Tucker Balch TUCKER@CC.GATECH.EDU
Georgia Institute of Technology, 225 North Ave NW Atlanta, GA 30332

Abstract

Agent-based simulation is a valuable tool for biologists studying animal behavior, however constructing models for simulation is often a time-consuming manual task. Predictive probabilistic graphical models have shown success in classifying behavior, but are not easily translated into models that can be executed in a multi-agent simulation. We propose a novel behavior representation and method for learning executable models suitable for execution in simulation. Results on synthetic and real video from lab experiments are given.

1. Introduction

One concrete motivation for this work has been to enable the work of biologists that study collective behavior through agent-based models. Agent-based models have been successful in analyzing the behavior of social insects such as ants and bees (Pratt et al., 2005; List et al., 2009), although currently such models are constructed after manual processing of video of the collective behavior of the animal. This manual process usually consists of frame-by-frame annotation of video of the animals in question and statistical analysis of the resulting data. Our approach aims to automate these two tasks; the first by using multi-target tracking to obtain tracks of individual animals from video, and the second by learning an executable model from those tracks. This workflow is outlined in Figure 1. This paper focuses on the second of these in developing a novel behavior representation and an algorithm for learning behaviors from tracking data.

Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28.

We define our problem as:

- **Given**:
  1. Tracks of animals interacting in their environment.
  2. A sensor model for the animals.
  3. A set of boolean triggers which are important to switching behavioral states.
- **Assume**: The animal's behavior can be modeled as a Markov Process.
- **Construct**: An executable model of behavior suitable for running in a multi-agent simulation.

It's important to note that our goal is the construction of an executable behavior, and as such we will focus more on the structure of the learned behaviors and aggregate measures of their performance in simulation than on the predictive capabilities of our model.

2. Related Work

Learning models of behavior from video is a broad and well studied area of research, even in the relatively specific domain of social insects. Fruit fly behavior has been studied using techniques from machine vision and machine learning to automatically create ethograms of observed fly behavior (Branson et al., 2009; Dankert et al., 2009). Researchers have successfully used switching linear dynamic systems (SLDS) to model and predict the distinct states in the "honey bee waggle dance", using MCMC sampling for approximate inference (Oh et al., 2005). Egerstedt et al. present a method for tracking ants and reconstructing minimal control programs of their movements from video (2005).

Much of this previous work has focused on behavior *recognition*, however, and it is not necessarily straightforward to construct an executable behavior directly from learned parameters of the model. Some

researchers from the robotics community have used HMMs for *action reproduction*, but the resulting trajectories must be post-processed to ensure smoothness and continuity constraints (Sugiura & Iwahashi, 2007; 2008; Kulić et al., 2008; Inamura et al., 2004). Meila & Jordan introduce an early framework for learning fine motion control for a robot arm using Expectation Maximization (EM) over a Markov Mixture of Experts, but only present results in a recognition task (1996).

Our approach, much like the behavior recognition focused work above, models behavior as a type of Markov Process, but it applies a more specific probabilistic graphical model so that the learned parameters can be used directly to construct an executable behavior. The model we use is based on the *Input/Output HMM* (IOHMM) (Bengio & Frasconi, 1995), which introduces a new input variable to the standard HMM model. Balch et al. present a framework for using IOHMMs to learn executable behavior, but their work separates the learning task in to two parts where the low level behaviors are learned separately from the transitions between them (2006). Our method allows for simultaneous learning of both the behaviors and the transition structure.

3. Modeling behavior

We take the approach outlined by Yang et al. for representing agent behavior (2012). The behavior of an agent can be described by a graph in which each node represents a state the agent can be in. Each edge represents a transition from one state to another, and the label for that edge corresponds to a boolean clause that must be true for the agent to make that transition. For a given state, the agent's output is a direct function of its sensors: the agent has no "memory" outside of the state structure. This graphical representation of behavior is quite similar to the ethograms that biologists use to describe observed behavior.

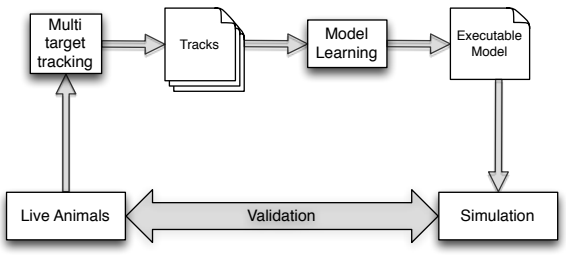

Figure 1. Workflow for automatically constructing executable behaviors from observation.

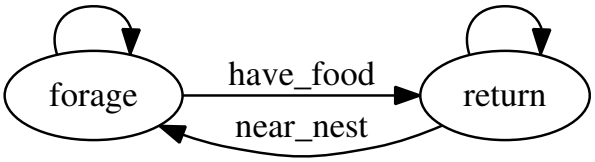

Figure 2. A simple foraging behavior. An agent will repeatedly forage until it acquires food, then return to the nest.

Figure 2 gives an example of how one might graphically represent a simple foraging behavior for an ant. The ant remains in the foraging and returning states until it finds food or is near the nest respectively. While in the foraging behavior, the ant might head towards the nearest food item within it's sensor range, and while returning it might head in the direction of the nest. In both of these states the ant might try to avoid walls and other ants. This can be naturally extended to include a probabilistic transition function rather than a deterministic one. We can formally define a behavior under this model as a triple $(\pi, A_{i,j}(\tau), B_i(\mathbf{y}))$, where π_i is the probability of starting in state i, $A_{i,j}(\tau)$ is the probability of transitioning from state i to j given the logical clause τ, and $B_i(\mathbf{y})$ is the output function which gives an agent's response to a given input vector $\mathbf{y}$ for state i. A specific behavior uses a finite number of boolean variables, k, in the clauses that control transitions between the N states, so we can represent A as a $N \times N \times 2^k$ table, and τ is the index of a specific configuration of the boolean variables. The input $\mathbf{y}$ is a real-valued vector in the feature space computed from the agent's sensors, and as mentioned earlier $B_i(\mathbf{y})$ is a function strictly of $\mathbf{y}$.

If we consider sequences of triples consisting of inputs, binary switches, and outputs $s_t = \langle \mathbf{y}_t, \tau_t, \mathbf{z}_t \rangle, t = 1 \ldots T$, of an agent following behaviors of this form, we can construct a graphical model which describes the probability of any particular sequence along with the unobserved state variables, as shown in Figure 3. Specifically, the action $\mathbf{z}_t$ ("turn θ radians left", "go forward") that the agent takes at time t depends on it's sensors $\mathbf{y}_t$ (vectors in the direction of the nest/food), and its current state x_t ("forage", "return"). The state at the next time step x_{t+1} is dependent on both the previous state x_t, and state of the boolean variables τ ("have_food" and "near_nest"). This model allows us to recast the problem of learning a behavior into finding the set of parameters which maximizes the likelihood of the data under those parameters, which in standard HMMs can be solved using Baum-Welch (Baum et al., 1970). Section 4 introduces an iterative parameter updating procedure similar to the Baum-Welch update equations.

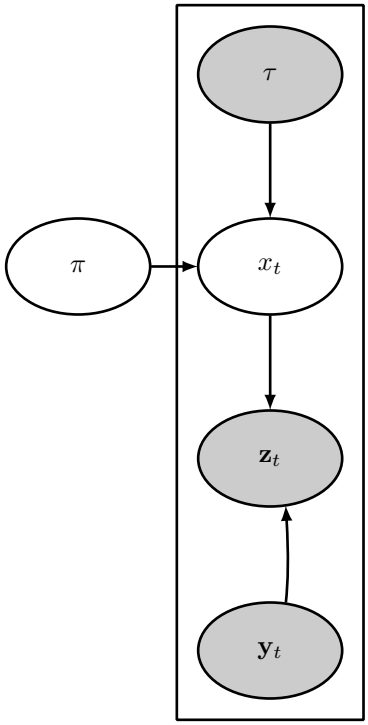

Figure 3. A Binary-switched IOHMM (BIOHMM). Observed variables are shaded

This is a natural extension of the classic HMM model which makes explicit the dependance relationships specific to this domain which derive from the fact that an agent is *reacting* to its environment. The IOHMM presented in (Bengio & Frasconi, 1995) also exploited this relationship, but that model made no distinction between the parts of the input that influenced state transitions and the parts that effected the state output (Figure 4). Our modification splits the input variable in two, and separates the variables that are important to state switching from those that are important to how the agent responds in a given state. It is important to note that our model is not any more expressive than an IOHMM. Once trained, however, our representation allows us to more easily construct the types of executable models described earlier.

4. Learning

Learning a behavior $\Theta = (\pi, A_{i,j}(\tau), B_i(\mathbf{y}))$ given a sequence of observations $S = \{s_t = \langle \mathbf{y}_t, \tau_t, \mathbf{z}_t \rangle | t = 1 \dots T\}$ can be accomplished through Expectation Maximization in a way similar to how Baum-Welch solves the analogous problem for HMMs.

First, the likelihood for a given sequence in the complete data case can be written as follows:

$$P(S, X \mid \Theta) = \pi_{x_0} \cdot \prod_t A_{x_t, x_{t+1}}(\tau_t) \cdot b_{x_{t+1}}(\mathbf{z}_t, \mathbf{y}_{t+1})$$

where $b_i(\mathbf{z}, \mathbf{y}) = P(\mathbf{z} \mid B, \mathbf{y})$ is the probability that the output function for state i produced output $\mathbf{z}$ for an input $\mathbf{y}$. Details on estimating $b_i(\mathbf{z}, \mathbf{y})$ will be discussed later.

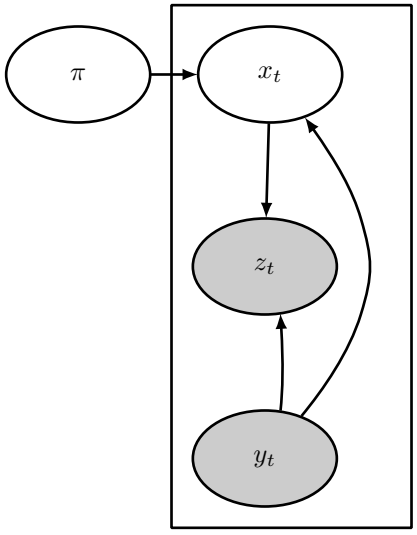

Figure 4. An IOHMM. Observed variables are shaded

The likelihood of a specific (complete data) sequence is just the product of the probability of the initial state, the probability of each state transition, and the probability of each observed output. The differences between a BIOHMM and a standard HMM only come in to play with variables that are observed in the training data, so the changes in forward-backward and the Baum-Welch updates are intuitive, and involve conditioning on the input and switching variables[1]. The forward variable, α, is computed by:

$$\alpha_t(i) = P(x_t = i \mid s_{1:t})$$
$$\alpha_1(i) = \pi_i \cdot b_i(\mathbf{z}_0, \mathbf{y}_0)$$
$$\alpha_{t+1}(j) = \left[\sum_i \alpha_t(i) A_{i,j}(\tau_t) \right] b_j(\mathbf{z}_{t+1}, \mathbf{y}_{t+1})$$

The backward variable, β:

$$\beta_t(i) = P(x_t = i \mid s_{t:T})$$
$$\beta_T(i) = 1$$
$$\beta_t(i) = \sum_j A_{i,j}(\tau_t) b_j(\mathbf{z}_{t+1}, \mathbf{y}_{t+1}) \beta_{t+1}(j)$$

[1] In this section we use the standard notation for EM on HMMs. For definitions and details on the original Baum-Welch we refer the reader to (Rabiner, 1989)

The smoothed posterior marginal, γ:

$$\begin{aligned}\gamma_t(i) &= P(x_t = i \mid s_{1:T}) \\ &= \frac{\alpha_t(i)\beta_t(i)}{\sum_j \alpha_t(j)\beta_t(j)}\end{aligned} \tag{1}$$

For convenience in computing the update to the transition function, we also define the ξ variable:

$$\begin{aligned}\xi_t(i,j) &= P(x_t = i, x_{t+1} = j \mid s_{1:T}) \\ &= \frac{\alpha_t(i)A_{i,j}(\tau_t)b_j(\mathbf{z}_{t+1},\mathbf{y}_{t+1})\beta_{t+1}(j)}{\sum_i \sum_j \alpha_t(i)A_{i,j}(\tau_t)b_j(\mathbf{z}_{t+1},\mathbf{y}_{t+1})\beta_{t+1}(j)}\end{aligned} \tag{2}$$

There is no difference between BIOHMMs and HMMs that effects the initial state distribution π, which leaves its update equation unchanged:

$$\pi'_i = \gamma_0(i) \tag{3}$$

Since the transition matrix is now conditioned on the switching variable, the update must be modified to only account for transitions under a specific τ:

$$A'_{i,j}(\tau) = \frac{\sum_t \xi_t(i,j)\delta_t(\tau)}{\sum_t \gamma_t(i)\delta_t(\tau)} \tag{4}$$

where $\delta_t(\tau)$ is the indicator function for switching variable τ at time t.

4.1. Output function

One of the key distinctions made between this work and the initial IOHMM model is how the output function is modeled. In the original IOHMM a neural network was used to model the output function, and back propagation was used to calculate the update as part of the EM step. In contrast, we have so far not restricted the form of the output function. In order to compute the updates and variables described above, some estimate of the probability that the output function for a given state produced a given output for a given input is necessary. In order to avoid making assumptions about the form of the output function we chose to use weighted kernel density estimation to provide this estimate.

Given n samples $\mathbf{x}_i$ of a random variable with an unknown probability density function, a KDE estimates the density at a query point $\mathbf{x}$ by taking the weighted average of the kernel applied to the difference between each sample and the query point:

$$\hat{f}(\mathbf{x}) = \frac{1}{h\sum_i w_i}\sum_i w_i K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right) \tag{5}$$

where K is a kernel function $K : \mathcal{R}^d \to \mathcal{R}$ and h is the bandwidth parameter.

By using the observed input/output $\langle \mathbf{y}_t, \mathbf{z}_t \rangle$ pairs from the training sequences as samples, we can construct a KDE estimate of the output function for each state, and updating these estimates in the learning procedure becomes a matter of assigning appropriate weights to each sample:

$$w_t^i = \frac{\gamma_t(i)}{\sum_{t'} \gamma_{t'}(i)} \tag{6}$$

This updates the weight of the t^{th} sample point for output density estimate b_i for state i to be proportional to the probability of being in that state when that output was generated. In the special case where $\sum'_t \gamma_{t'}(i) = 0$ we set $w_t^i = 0$ in equation 6 since, by construction, this means that $\gamma_t(i) = 0$. Similarly, because all weights are non-negative, if $\sum_i w_i = 0$ in equation 5, then we set $\hat{f}(\mathbf{x}) = 0$. This ensures that the output probability density is bounded.

Given equations 3, 4, and 6, we can iteratively compute an improved Θ. Given a specific Θ, the transition function A and initial state distribution π can be used directly in an executable behavior, and the weighted samples from each state's estimated output density function can be used as input to a non-parametric function approximation technique to produce an output function. We use a modified form of k-NN for this, where the output $\hat{B}_i$ for a given input $\mathbf{y}$ is sampled randomly from the set of outputs of the k nearest neighbors of $\mathbf{y}$ according to their weight w_t^i.

5. Experiments

Since our motivation is to provide executable models, and the capabilities of hidden Markov models in behavior recognition are well documented in previous literature, our experiments focus on the performance of the generated behavior.

For our work, it is important that an agent acting under a learned behavior actually *behaves* like the animal that generated the training data. To test this we constructed a simulation of foraging ants, trained an executable model from traces of the simulation, ran simulations using the learned behavior, and compared several aggregate measures of behavior between the original simulation, the learned behavior, and a random wandering behavior for validation.

The foraging simulation consisted of 10 ants and 10 stationary food items in a rectangular arena. The ants were programmed to follow the behavior shown in Figure 2. While in the foraging state, the ant would head

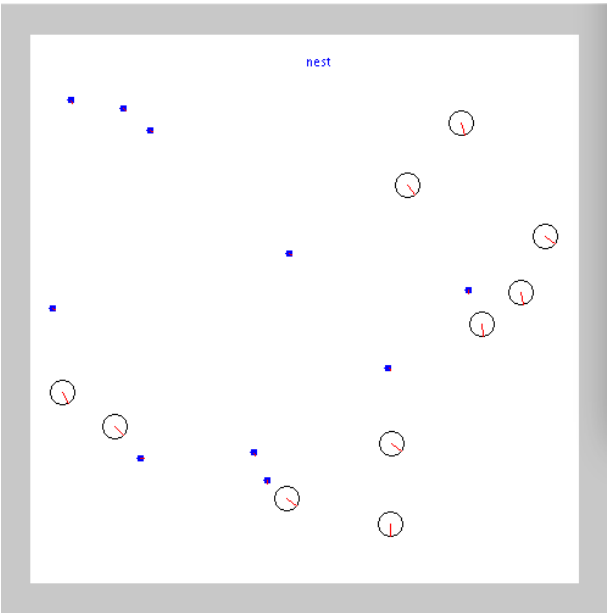

Figure 5. The simulation environment. Ants are the larger transparent circles, food items are the smaller filled circles

towards the nearest food item within range, and wander randomly until it came within range of a food item. While returning, the ant would head in the direction of the nest. If the ant got close enough to touch a food item in the foraging state, it would pick it up and transition to the return state. Similarly, when the ant returned to the nest while in the return state, it would drop any food it was carrying and transition back to the foraging state. In both states, the ant would try to avoid running into walls and other ants. Transitions in each of these cases were deterministic, and the ants always started in the foraging state. Sensor ranges were set to three body-lengths (25.8mm). Figure 5 is a screenshot of the simulation at initialization.

To collect training data, we recorded traces from the hand-coded behavior of each ant's output velocity ($\mathbf{z}$), the 4 sensor vectors ($\mathbf{y}$) — obstacle sensor, ant sensor, food sensor and direction-to-home sensor — and the two binary switching sensors "have_food" and "near_nest" (τ) at each simulation step. A model with two states was then trained using these traces as the observation sequences as described in section 4. The initial values for π and A were chosen randomly and each $\langle \mathbf{y}_t, \mathbf{z}_t \rangle$ was randomly assigned to one of the output functions. For the kernel density estimator we used a bandwidth $h = 1$ and a Gaussian kernel $\Sigma = I$. After converging, the learned parameters were used to construct an executable behavior, and a new simulation was run with 10 food items and 10 ants using the learned behavior.

Table 1. Foraging transition function. HF and NN stand for the boolean variables have_food and near_nest.

	¬HF ¬NN		¬HF NN	
	RETURN	FORAGE	RETURN	FORAGE
RETURN	—	—	—	—
FORAGE	0.0	1.0	0.0	1.0

	HF ¬NN		HF NN	
	RETURN	FORAGE	RETURN	FORAGE
RETURN	1.0	0.0	0.0	1.0
FORAGE	—	—	—	—

As a baseline comparison, we also implemented a random behavior, where the ants simply moved in a random direction at each time step. Whenever an ant moved over a food item it automatically picked it up unless the ant was already carrying food, and whenever it moved over the nest while carrying a food item it dropped the item off.

6. Results

The synthetic foraging behavior is simple enough that the learned parameters can be compared directly with the model that produced the training data. Figure 6 shows how the transition function converges to the correct values given in Table 1. Figure 7 provides a visualization of a slice of the output function at iterations 1 and 40. The slice illustrated in this graph fixes the value of the direction-to-home sensor to directly in front of the ant, with the other sensors set to zero, and plots the probability of the output angular velocity for the return state. Initially, the probability of any particular turn is fairly uniform, but after several iterations, the ant has learned to preferentially move in the direction of the nest (zero angular velocity) when returning food.

Table 2 lists the aggregate data collected in the three simulations. Simulations were run for 120 simulated seconds ($\approx$ 4000 steps), and were repeated 20 times; mean values are reported with 95% confidence intervals. RETURN TIME is the average time (in seconds) between when a food item is first picked up, and when it is dropped off near the nest. DISTANCE FORM NEAREST ANT is the average distance (in millimeters) between each ant and the nearest other ant within sensing range. DISTANCE FROM NEAREST WALL is the average distance between each ant and the nearest wall within sensing range. FOOD COLLECTED PER RUN is the average number of food items returned to the nest by all ants throughout the course of a single run. These particular statistics were chosen to highlight the as-

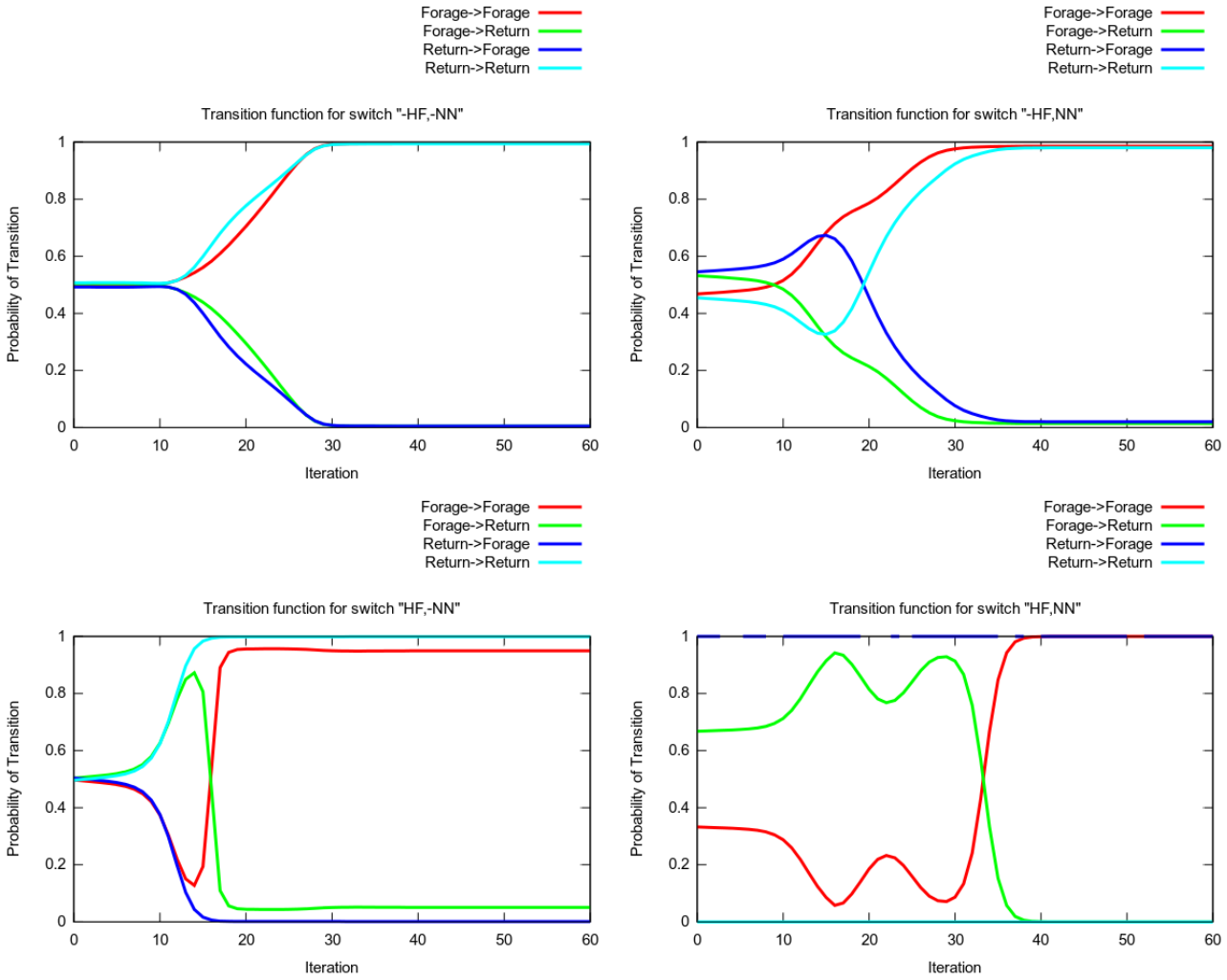

Figure 6. Evolution of the transition function parameters over time. Transitions that occur in the hand-coded model converge to their correct values given in Table 1.

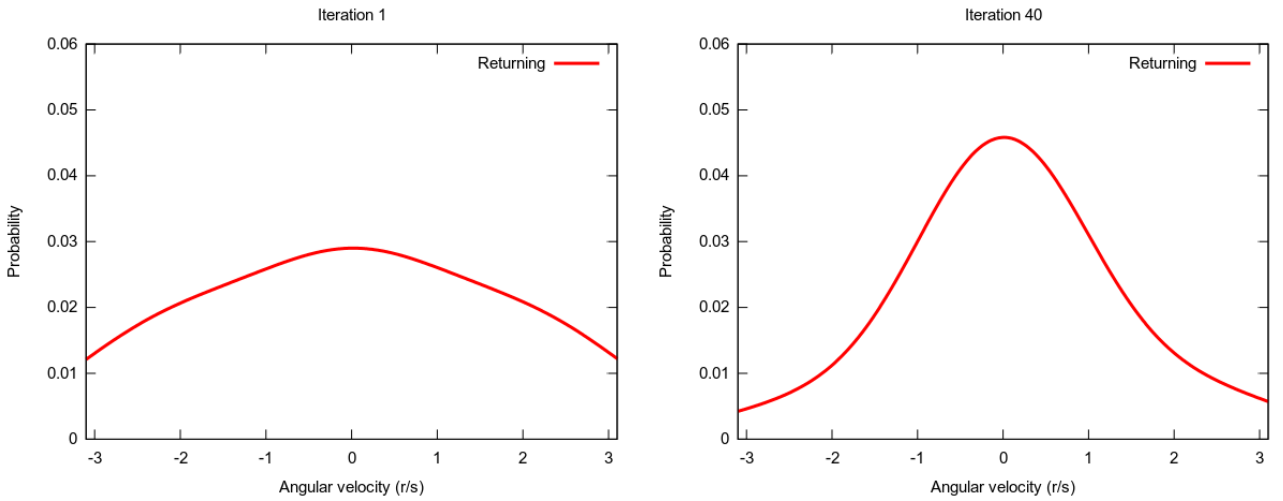

Figure 7. Evolution of the output function parameters over time. At successive iterations, the probability of producing an output in the direction of the nest increases.

Table 2. Aggregate statistics for hand-coded foraging.

	Hand-coded	Learned	Random
Return time	15.5 ± 1.1(s)	34.5 ± 3.9(s)	80.7 ± 8.5(s)
Distance from nearest ant	19.6 ± 0.02(mm)	14.36 ± 0.03(mm)	12.3 ± 0.03(mm)
Distance from nearest wall	17.6 ± 0.04(mm)	7.24 ± 0.02(mm)	5.5 ± 0.01(mm)
Food collected per run	9.45 ± 0.26	6.25 ± 0.6	0.9 ± 0.35

Figure 8. One frame of a video of *A. cockerelli* in an empty arena

pects of the foraging behavior that a learned behavior should capture such as preferentially heading toward the nest after acquiring food and avoiding walls and other ants.

These results illustrate that the executable behavior constructed from the learned parameters produces similar aggregate statistics to the original behavior. While the return time is longer in the learned behavior, it is still much shorter than the time it takes a randomly wandering ant, which suggests the learned behavior is trying to head towards the nest after picking up a food item. Similarly, the fact that the average distance between nearby ants and obstacles is larger for the learned behavior than the random behavior suggests that the ant is attempting to avoid collisions. The food collected per run suggests that the learned behavior produces aggregate behavior that is very similar to the hand-coded foraging ants.

6.1. Preliminary results for live animals

We captured video in the lab of a group of *Aphaenogaster cockerelli* in an empty arena. Figure 8 shows a frame from this video. The ants in the video were tracked using Multi-Iterative Closest Point tracking (Feldman et al., 2012) which provides the x, y, and θ of each ant in each frame. These tracks are then processed to create egocentric ant sensor values (including binary switches) paired with changes in (x, y, θ), and combined into sequences.

Table 3. Aggregate statistics for wandering ants.

	Real	Learned
Distance from nearest ant	15.11 ± 0.73(mm)	17.5 ± 0.03(mm)
Time spent near wall	3.1 ± 1.3(s)	2.9 ± 0.3(s)
Time spent near ants	1.6 ± 0.5(s)	1.3 ± 0.08(s)
Time spent near nest	2.2 ± 1.5(s)	3.8 ± 0.7(s)

Using this training data, we were able to learn a two-state model of ant wandering behavior and compare the learned behavior to the real ants. While there is no "ground truth" model to compare to, Table 3 lists several aggregate statistics of both the real and learned ants, and illustrates how similar the learned and actual behaviors are. These kinds of interaction statistics are useful in answering important questions about how the animals in question behave socially, such as whether ants have a preferred encounter rate (Gordon et al., 1993).

7. Conclusion

We have described a new method for representing and learning executable models of animal behavior. Results on learning a synthetic foraging behavior have shown that this approach can successfully construct executable behaviors which produce similar aggregate statistics to the original behavior. We have illustrated how the entire workflow of creating executable models of behavior for agent-based simulation studies of animal behavior can be automated.

There are many avenues for extending this work in the future. As is the case with EM in HMMs, our learning algorithm is only locally optimal, and techniques for avoiding sub-optima are crucial for the high-dimensional domain which we work within. These in-

clude pre-processing in order to find better initial parameter selections, modifications to the iterative updates to detect and escape local optima, as well as incorporating features of global optimization techniques such as evolutionary computation.

References

Balch, Tucker, Dellaert, Frank, Feldman, Adam, Guillory, Andrew, Isbell, Charles L, Khan, Zia, Pratt, Stephen C, Stein, Andrew N, and Wilde, Hank. How multirobot systems research will accelerate our understanding of social animal behavior. *Proceedings of the IEEE*, 94(7):1445–1463, 2006.

Baum, Leonard E, Petrie, Ted, Soules, George, and Weiss, Norman. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The annals of mathematical statistics*, 41(1):164–171, 1970.

Bengio, Yoshua and Frasconi, Paolo. An input output hmm architecture. In *Advances in neural information processing systems*, pp. 427–434. Morgan Kaufmann Publishers, 1995.

Branson, Kristin, Robie, Alice A, Bender, John, Perona, Pietro, and Dickinson, Michael H. High-throughput ethomics in large groups of drosophila. *Nature methods*, 6(6):451–457, 2009.

Dankert, Heiko, Wang, Liming, Hoopfer, Eric D, Anderson, David J, and Perona, Pietro. Automated monitoring and analysis of social behavior in drosophila. *Nature methods*, 6(4):297–303, 2009.

Egerstedt, Magnus, Balch, Tucker, Dellaert, Frank, Delmotte, Florent, and Khan, Zia. What are the ants doing? vision-based tracking and reconstruction of control programs. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2005)*, pp. 18–22, 2005.

Feldman, Adam, Hybinette, Maria, and Balch, Tucker. The multi-iterative closest point tracker: An online algorithm for tracking multiple interacting targets. *Journal of Field Robotics*, 29(2):258–276, 2012.

Gordon, Deborah M, Paul, Richard E, and Thorpe, Karen. What is the function of encounter patterns in ant colonies? *Animal Behaviour*, 45(6):1083–1100, 1993.

Inamura, Tetsunari, Toshima, Iwaki, Tanie, Hiroaki, and Nakamura, Yoshihiko. Embodied symbol emergence based on mimesis theory. *The International Journal of Robotics Research*, 23(4-5):363–377, 2004.

Kulić, Dana, Takano, Wataru, and Nakamura, Yoshihiko. Incremental learning, clustering and hierarchy formation of whole body motion patterns using adaptive hidden markov chains. *The International Journal of Robotics Research*, 27(7):761–784, 2008.

List, Christian, Elsholtz, Christian, and Seeley, Thomas D. Independence and interdependence in collective decision making: an agent-based model of nest-site choice by honeybee swarms. *Philosophical transactions of The Royal Society B: biological sciences*, 364(1518):755–762, 2009.

Meila, Marina and Jordan, Michael I. Learning fine motion by markov mixtures of experts. In *Advances in Neural Information Processing Systems 8*, pp. 1003–1009. MIT Press, 1996.

Oh, Sang Min, Rehg, James M., Balch, Tucker, and Dellaert, Frank. Data-driven mcmc for learning and inference in switching linear dynamic systems. In *Proceedings of the 20th national conference on Artificial intelligence - Volume 2*, pp. 944–949. AAAI Press, 2005.

Pratt, Stephen C, Sumpter, David JT, Mallon, Eamonn B, and Franks, Nigel R. An agent-based model of collective nest choice by the ant *Temnothorax albipennis*. *Animal Behaviour*, 70(5):1023–1036, 2005.

Rabiner, Lawrence R. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

Sugiura, Komei and Iwahashi, Naoto. Learning object-manipulation verbs for human-robot communication. In *Proceedings of the 2007 workshop on Multimodal interfaces in semantic interaction*, pp. 32–38. ACM, 2007.

Sugiura, Komei and Iwahashi, Naoto. Motion recognition and generation by combining reference-point-dependent probabilistic models. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pp. 852–857. IEEE, 2008.

Yang, Yu-Ting, Quitmeyer, Andrew, Hrolenok, Brian, Shang, Harry, Nguyen, Dinh Bao, Balch, Tucker, Medina, Terrance, Sherer, Cole, Hybinette, Maria, and Athens, GA. Ant hunt: Towards a validated model of live ant hunting behavior. In *Twenty-Fifth International FLAIRS Conference*, 2012.