Learning Human Motion Models

Bulent Tastan
Department of EECS
University of Central Florida
Orlando, FL 32816, U.S.A
bulent@cs.ucf.edu

Abstract

My research is focused on using human navigation data in games and simulation to learn motion models from trajectory data. These motion models can be used to: 1) track the opponent's movement during periods of network occlusion; 2) learn combat tactics by demonstration; 3) guide the planning process when the goal is to intercept the opponent. A training set of example motion trajectories is used to learn two types of parameterized models: 1) a second order dynamical steering model or 2) the reward vector for a Markov Decision Process. Candidate paths from the model serve as the motion model in a set of particle filters for predicting the opponent's location at different time horizons. Incorporating the proposed motion models into game bots allows them to customizes their tactics for specific human players and function as more capable teammates and adversaries.

Introduction

In this summary, I provide an overview of my dissertation work on learning human motion models from trajectory data. A *motion model* can be used to characterize the distribution of a specific human's movement actions, given the current environment. I describe two methods for predicting future trajectories: 1) fitting parameters to a realistic human steering model (HSM) 2) using inverse reinforcement learning to learn rewards for different map regions. Here, I provide an overview of three applications for motion models within games and simulations: 1) human tracking 2) the creation of human-like bots 3) pursuing opponents.

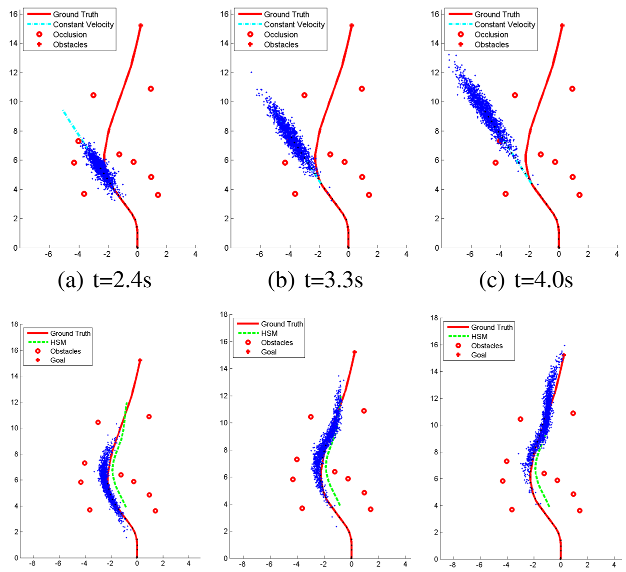
Human Tracking

One of the most powerful constraints governing many activity recognition problems are those imposed by the human actor. It is well-known that humans have a large set of physical and cognitive limitations that constrain their execution of various tasks. In human tracking, prior knowledge of perception and locomotion can be exploited to enhance path prediction and tracking in indoor environments for pervasive computing applications.

To track humans with sensor networks, detect behavior anomalies, and offer effective navigational assistance, we need to be able to predict the trajectory that a human will follow in an environment. Although human paths can be approximated by a minimal distance metric, humans often exhibit counter-intuitive behaviors; for instance, human paths can be non-symmetric and depend on the direction of path traversal (e.g., humans walking one route and returning via a different one). Obviously tracking and goal prediction algorithms that assume distance-minimizing behavior will generate errors in environments where the humans' behavior diverges from this model.

To address this problem, a psychologically-grounded model of human steering and obstacle behavior are incorporated into the tracking and goal prediction system. The selected model, originally proposed by (Fajen et al. 2003) incorporates environmental features that are accessible to the human vision system into a second-order dynamical model; all calculations are based on local perceptual information and do not require global knowledge of the environment. In this steering model, human locomotion is assumed to have a constant translational speed (s); goals and obstacles affect the subject's angular acceleration ($\ddot{\phi}$) according to a set of differential equations. Objects in the environment (goals and obstacles) are represented in an egocentric coordinate frame using features that are easily extracted by the human visual system. Data for this study was collected from subjects steering avatars in Second Life. Second Life is a user-constructed virtual environment that allows the users to construct their own 3D world.

To apply the model to the problem of predicting virtual avatar movement, we retrained the parameters with a set of short (56 second) movement traces collected from one subject in Second Life. To track the movement of the human subjects, an adapted particle filter tracking system using the human steering model is implemented to robustly track the human subjects during sensor blackout periods. At



(a) t=2.4s (b) t=3.3s (c) t=4.0s

(d) t=2.4s (e) t=3.3s (f) t=4.0s

Figure 1: Particle filter using constant velocity model (top) and human steering model (bottom)

each time step, for each particle, the closest segment on the path generated by the motion model is found, and the particle's position is updated so that it moves parallel to this segment at the predicted speed. Figure 1 shows sample traces from the particle filter using the constant velocity and HSM motion models. Clearly, the popular linear model is poorly suited for predicting the user's position during sensor blackout when the motion is affected by the presence of obstacles. As a result, the predicted distribution of particles drifts significantly from the ground truth during these periods. In contrast, the path generated by the proposed method (HSM) captures the motion of the user, enabling the particle filter to robustly track the user even through extended periods of sensor blackout.

The HSM provides a good general model for steering behavior in cases where the human is viewing the world through a first-person viewpoint. However, this is not always the case in games since the player can be viewing the scene through an overhead view. In the next two sections, I describe a method, inverse reinforcement learning, for learning environment specific movement patterns.

Human-like Bot Design

The creation of effective autonomous agents (bots) for combat scenarios has long been a goal of the gaming industry. However, a secondary consideration is whether the autonomous bots behave like human players; this is especially important for simulation/training applications which aim to instruct participants in real-world tasks. Bots often compensate for a lack of combat acumen with advantages such as accurate targeting, predefined navigational networks, and perfect world knowledge, which makes them challenging but often predictable opponents. In this section, we examine the problem of teaching a bot to play like a human in first-person shooter game combat scenarios. Our bot learns attack, exploration and targeting policies from data collected from expert human player demonstrations in Unreal Tournament.

Creating human-like adversarial intelligence poses significantly different challenges from the related problems associated with creating non-player characters with realistic dialog, emotion, and facial animations (Cassell et al. 2000). Variability in execution has been listed as an important desiderata for adversaries (Wray and Laird 2003); the ideal opponent should not be repetitive and predictable in its action selections (Schaeffer, Bulitko, and Buro 2008). However, over-reliance on randomized action selection can sabotage effective gameplay by making autonomous opponents seem like distracted amnesiacs (Hingston 2010).

The problem with intelligent bots is that their valuation of actions is significantly different from a human player's. For instance, a human player will recklessly chase an opponent even when their health status is dangerously low just to experience the thrill of the chase. Many commonly-used game algorithms such as A* path planning optimize subtly different metrics than human players; for instance, A* implementations often use an exclusively distance-based heuristic that does not express player movement preferences to avoid certain types of terrain.

The proposed approach resolves this problem of mismatched valuations by having bot directly learn reward functions from expert human player demonstrations. The experimental testbed, constructed using the Pogamut toolkit (Gemrot et al. 2009), can be used to acquire data from expert human Unreal Tournament players playing deathmatch games and to learn policies for **attack**, **exploration**, and **targeting** modes from human data. Rewards for attack and exploration actions are learned using inverse reinforcement learning (IRL) (Ng and Russell 2000).

IRL attempts to solve the opposite problem as reinforcement learning; instead of learning policies from rewards, IRL recovers rewards from policies by computing a candidate reward function that could have resulted in the demonstrated policy. During the demonstrations, the most frequent actions executed by human players in a set of designated game states are recorded. A reward model is learned offline using the CPLEX solver based on a set of constraints extracted from the expert demonstration. Using value iteration (Sutton and Barto 1998), a policy is created from the reward vector and then executed by the bot. For the targeting mode, we simply fit a parameterized model to the expert data. Rather than having the bot rely on the automated targeting provided by the game, our bot samples from the learned distribution. The bot switches between these three modes (attack, exploration, and targeting) based on game cues such as opponent proximity and availability of resources.

Overall	Human-like	Bot-like
UT Bot	3	8
New Bot	11	2
Left p value = **0.007**		

Table 1: Questionnaire responses relating to overall performance (statistically significant)

The combat performance (attack and targeting) of the bot was evaluated by having a set of human subjects compare their play experiences with the new bot vs. a standard bot. Players had the opportunity to play both bots directly in deathmatch competition multiple times, as well as to view movies of the two bots. The findings for this experiment indicate that human players rate the new bot as more human-like in gameplay as seen in Table 1.

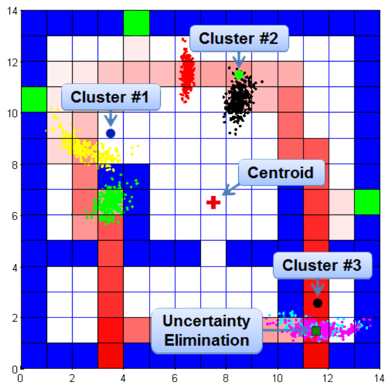


Figure 2: The output of different planning heuristics shown for the same particles. The bot chooses the riskiest cluster which is Cluster #3 in this case.

Pursuit Problem

Anticipating and countering the movement of adversaries is an important element of many combat games, both squad-based (Darken, McCue, and Guerrero 2010) and individual (Hingston 2009). In cases where the terrain hides movement, the simplest way to simulate the sense of an anticipatory intelligence driving the bot is to permit the automated bot to make decisions with an omniscient view of the current game state. This approach has the following weaknesses: 1) it reduces the role of different maps in shaping the flow of play, and 2) it trains the players to ignore cover, which is problematic for serious game applications such as military training.

Furthermore there are some cases where it is useful to be able to predict future movement of the player even when the current state is wholly observable, such as the archetypical predator-prey game (Panait and Luke 2005), football pass interceptions (Laviers et al. 2009), or in first person shooters when the player with superior armament is motivated to chase down a more lightly armed character. Hence, one measure of computational intelligence in games is that combat bots exhibit refined interception strategies.

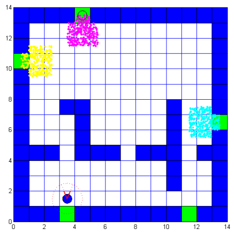
This section presents a framework for learning how to intercept opponents in a partially occluded Unreal Tournament map during an iterated game scenario where the bot plays repeated games against the same group of opponents and has the opportunity to learn their evasion strategies. Inverse reinforcement learning is used to learn a feature-based reward model from sets of example traces. Since the learned model is feature-based, it can generalize to areas that the player has not yet visited and to other maps. The model, combined with the map navigational network, is then used to estimate probability distributions for transitioning between different map regions and to generate a set of likely opponent paths. This path set is used to seed the motion model of a set of particle filters that track the opponent's state over multiple future time horizons, and our bot creates an interception plan based on the output of the particle filters.

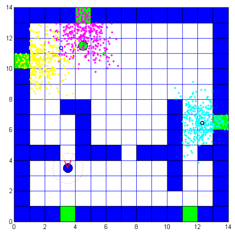
The planning strategy to move the bot around the map and eliminate possible hiding spots while attempting to intercept the target is based on three heuristics:

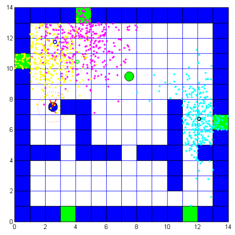
- **Centroid:** The bot heads to the center of the entire particle set.
- **Uncertainty Elimination:** The bot goes to the cell that has the maximum number of particles within its sensor radius.
- **Cluster:** The particles are clustered into a number of gates, and the cluster centroid with smallest total distance to the bot and the exit is picked as the target.

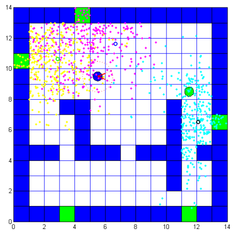
A snapshot of target points selected by these three heuristics is shown in Figure 2. Note that in spite of the fact that all the techniques use the same point cloud, the target point selections can be quite different. The centroid technique works best when the probability distribution is unimodal. On the other hand, uncertainty elimination attempts to bring the bot's "sensors" in contact with the maximum number of particles and aims to eliminate the potential opponent hiding spots.

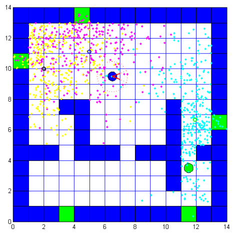
The clustering heuristic uses k-means clustering to cluster particles and maintains multiple target points. The planner chooses the cluster that is closest to an exit and nearest to the bot. When the bot reaches a cluster center and does not see the target, the particles corresponding to this cluster (even those outside the sensor area) are eliminated. This forces the bot to change its focus to other clusters. This can be seen in Figure 3 on bottom row between timesteps 8 and 13. At timestep 8 the bot goes towards the cluster center (shown in red) and once it realizes the target is not there, it eliminates the cluster and particles belonging to that cluster.

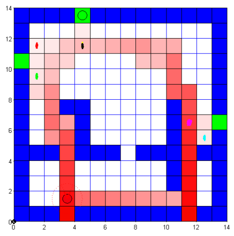

(a) t=1

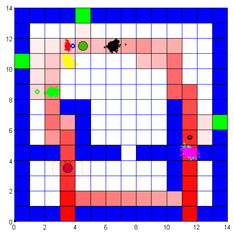

(b) t=3

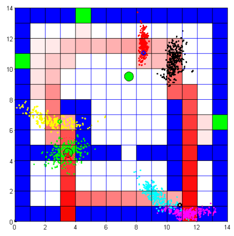

(c) t=8

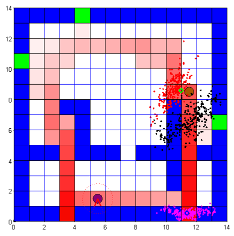

(d) t=13


(e) t=18


(f) t=1


(g) t=3


(h) t=8


(i) t=13

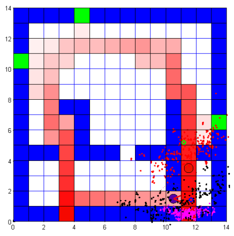

(j) t=18

Figure 3: Particle filter using Brownian motion model (top) and IRL motion model (bottom). The opponent is marked in green and the interception bot is marked in blue. Note that both filters have some particles near the opponent but including learned paths in the candidate hypotheses IRL model produces more cohesive motion.

Overall the learned motion model has a higher tracking accuracy and results in more consistent interceptions than simpler motion models and prediction methods. Since the model is learned from previous data, the best way to foil the system is to intelligently use cover to avoid observation while deviating from previously employed strategies. This creates more variability in gameplay, forces the player to find new winning ambush strategies, and is potentially useful in serious game applications where the goal is to teach the user tactics that will generalize to real scenarios outside the training simulation. My current work is embedding this model into a complete Unreal Tournament bot and conducting a user study evaluating the model performance.

Acknowledgements

This research was funded by AFOSR YIP FA9550-09-1-0525 and DARPA award N10AP20027.

References

Cassell, J.; Sullivan, J.; Prevost, S.; and Churchill, E. F., eds. 2000. *Embodied Conversational Agents*. Cambridge, MA, USA: MIT Press.

Darken, C. J.; McCue, D.; and Guerrero, M. 2010. Realistic fireteam movement in urban environments. In *Proceedings of Artificial Intelligence for Interactive Digital Entertainment Conference (AIIDE)*.

Fajen, B. R.; Warren, W. H.; Temizer, S.; and Kaelbling, L. P. 2003. A Dynamical Model of Visually-Guided Steering, Obstacle Avoidance, and Route Selection. *International Journal of Computer Vision* 54(1–3):13–34.

Gemrot, J.; Kadlec, R.; Bida, M.; Burkert, O.; Pibil, R.; Havlcek, J.; Zemck, L.; Lovic, J.; Vansa, R.; Tolba, M.; Plch, T.; and Brom, C. 2009. Pogamut 3 can assist developers in building AI (not only) for their videogame agents. In *Agents for Games and Simulations*, Lecture Notes in Computer Science. Springer.

Hingston, P. 2009. A Turing Test for Computer Game Bots. *IEEE Transactions on Computational Intelligence and AI in Games* 1(3):169–186.

Hingston, P. 2010. A New Design for a Turing Test for Bots. In *IEEE Computational Intelligence and Games (CIG)*.

Laviers, K.; Sukthankar, G.; Molineaux, M.; and Aha, D. 2009. Improving offensive performance through opponent modeling. In *Proceedings of Artificial Intelligence for Interactive Digital Entertainment Conference (AIIDE)*, 58–63.

Ng, A. Y., and Russell, S. 2000. Algorithms for Inverse Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 663–670.

Panait, L., and Luke, S. 2005. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-agent Systems* 11(3):387–434.

Schaeffer, J.; Bulitko, V.; and Buro, M. 2008. Bots get smart. *IEEE Spectrum*.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press.

Wray, R., and Laird, J. 2003. Variability in Human Behavior Modeling for Military Simulations. In *Proceedings of Behavior Representation in Modeling and Simulation Conference (BRIMS)*.