

Learning Outdoor Mobile Robot Behaviors

Richard Roberts and Tucker Balch
Center for Robotics and Intelligent Machines
Georgia Tech
Atlanta, GA 30332

Email: richard@cc.gatech.edu, tucker@cc.gatech.edu

Abstract—With mobile robots, most current methods of controlling the speed and direction of the robot require complex logic and a great deal of parameter tuning. Instead, we are working on a system in which a human demonstrates how and where to drive, by driving the robot with a remote control, and the robot learns from demonstration how to navigate to its goal. At this point in this study, we have implemented such a system, which is currently capable of avoiding obstacles and following a path. Qualitative testing has shown that acceptable performance in such behaviors is attainable after training for one hour. At this point, issues are present with what appears to be dynamics and latency effects not being completely learned, and also with control commands oscillating between two modes in ambiguous situations. To counter these issues, we will investigate incorporating hysteresis into a sum-of-Gaussians model for voting on command direction. Later in this study, we will also incorporate learning of behavior changes, automatic dimensionality reduction, and automatic tuning of obstacle detection parameters.

I. INTRODUCTION

With mobile robots, most current methods of controlling the speed and direction of the robot require complex logic and a great deal of parameter tuning, especially when they involve obstacle avoidance. Georgia Tech’s LAGR motion control software is comprised of several low-level behaviors, such as obstacle hugging for moving around large obstacles and a potential-field behavior for traversing mostly-clear terrain. A state machine switches between behaviors, according to triggers that are also constructed by hand. Behaviors and triggers take into account, among other factors, the positions of the obstacles around the robot and the heading to the robot’s goal. Constructing and tuning these behaviors and triggers by hand has turned out to be error-prone and time-consuming. Instead, we would like a human to demonstrate how and where to drive, by driving the robot with a remote control, and the robot to learn from demonstration how to navigate to its goal. Implementing these “learned behaviors” would replace unintuitive hand-tuning with driving.

II. RELATED WORKS

A. Learning by imitation

Hayes and Demiris used learning by imitation to allow one robot to learn how to traverse a maze by following another robot [1]. The input to their learner was the wall configuration, for example, “wall to the right and in front → turn left”. Walls were detected by IR sensors, and the other robot was detected by a set of LEDs on top of it. The learner builds a list of rules, where actions are associated with wall configurations. It

is not clear how the robot manages lower-level control, such as staying parallel to walls, turning corners, etc.

Atkeson and Schaal focus on the task of swinging up and balancing an inverted pendulum, with observation of a human performing the task [2]. The robot learns a physical model by watching itself perform the task, and learns a reward function from the human demonstration. Both a parametric (based on a dynamic pendulum model) and a non-parametric physical model were tried. For the non-parametric method, the authors cite Atkeson et. al, 1997 [3]. To learn how to swing the pendulum up, the learner used a reward function that penalized deviation from the trajectory demonstrated by the human. Atkeson and Schaal [2] demonstrate that the robot can learn a “better” control schema by trying to achieve the same pendulum trajectory, instead of executing the same hand motions. Similar techniques could be used to learn longer trajectories with mobile robots, instead of learning single-time-step motor commands.

B. Behavior-based reinforcement learning

In Matarić, 1997, Matarić cites many examples of robot systems utilizing behaviour-based, as opposed to deliberative, control [4]. In particular, he cites Matarić, 1994 [5], which describes the design of reward functions for reinforcement learning of control based on behaviors. Another paper by the same author discusses learning behaviors and learning behavior selection, using reinforcement learning, and presents a set of experiments with this method [6].

C. Nearest-neighbor methods

K-nearest-neighbors is very easy to implement, and is very efficient when implemented with a spacial index, such as a KD-tree. KD-trees were first proposed by Bentley, and later applied and studied with the nearest-neighbor problem by Friedman, Bentley, and Finkel [7, 8]. While lookups take place in logarithmic time with respect to the number of data points, they increase exponentially with feature dimensionality. Several methods exist, however, for computing approximate nearest-neighbors, and these operate much more efficiently than exact nearest-neighbor in high dimensions. Among these approximate nearest-neighbor methods are one first proposed by Arya and Mount [9], Locality-Sensitive Hashing [10], and a collection of methods demonstrated by Liu, Moore, *et al.* [11]. We are currently using the method developed by Arya and Mount [12], which is available in a free library.

D. Dimensionality Reduction

The raw sensor input for mobile robots, whether images, cost maps, stereo hits, etc, are very high-dimensional. By intelligently mapping this high-dimensional input to lower dimensionality, we can reduce complexity of nearest-neighbor lookups, and reduce the amount of training data needed. The ray parametrization described in Section III is one way of reducing dimensionality, but a multitude of methods exist for creating parametrization automatically. It will be beneficial to investigate principal component analysis, which represents low-dimensional vectors as linear combinations of the components of high-dimensional vectors. Restricted Boltzmann machines have also been shown to be capable of learning feature extraction from high-dimensional data [13, 14]. Another concept that may be useful in this project is mixtures of local experts, which can be used to train learners when data comprises several “cases”, which in my case could be separate behaviors. Jacobs and Jordan [15, 16] describe systems that can learn these cases and learner decompositions.

III. SYSTEM DESCRIPTION

The currently implemented system learns control behaviors using a nearest-neighbor method. A switch on the remote control signals the robot to either learn from a human driving, or drive autonomously. When learning from the human, training pairs are recorded that associate the current state of the world with the motion performed by the human. The “feature vectors” representing the state of the world are fixed-length vectors of distances from the robot to the nearest obstacle along rays extending out from the robot, as seen in Figure 1, plus the robot’s heading to the goal. The motion performed by the human is represented as a vector describing the commanded linear velocity and angular velocity. When running autonomously, the robot builds a feature vector from it’s sensor readings, finds the closest-matching example from the recorded training pairs, and commands the same motion taken by the human in that example. Currently, separate sets of examples are recorded for each low-level behavior, and behavior changes are still commanded by the hand-constructed triggers. In the future, we will implement learning to control behavior changes.

A. Feature vectors

The rays of the feature vector indicate the closest obstacle to the robot in a “wedge” subtending the angle between adjacent rays. Experimentation has shown that 25 rays provides a good balance between resolution and dimensionality. In this case, each wedge subtends an angle of 7.2° , which means that this ray representation allows the robot to perceive gaps of 1.25 m at a range of 5 m. This minimum gap size is sufficient, given that the robot is approximately 80 cm in width, and that the minimum perceivable gap size decreases as the robot gets closer to the gap.

In addition to these rays, one additional component of the feature vector is the heading to the goal from the robot. In practice, we desire this heading to have a very high weight, so



Fig. 1. The feature vector used for learning the state of the environment is a vector of distances to the closest obstacle, in a plane extending out in front of the robot, as depicted by the green rays. Red areas indicate obstacles. The rays always extend from the same point on the robot, which is located at the point of convergence of the rays.

that closest feature vector matches are only found whose goal heading matches very closely. To implement this high weight, headings are discretized into seven “bins”, and a separate example database is maintained for each bin.

When stored and queried, feature vectors undergo both a weight transformation, to alter the relative importance of each component of a vector, and a distance transformation, to alter the importance of obstacle range measurements depending on their distance from the robot.

Specifically, the weight transformation makes the range measurements to obstacles in front of the robot more important than the range measurements to obstacles at the sides of the robot. Qualitatively, this weight transformation improves the responsiveness of the robot to curvature in the path ahead of it, and to obstacles in its path.

The distance transformation exaggerates small changes in distance for obstacles that are close to the robot, while diminishing changes in distance for obstacles that are far from the robot. Qualitatively, this distance transformation improves the ability of the robot to smoothly navigate around obstacles as the robot approaches them, and to smoothly maintain constant distance from a wall when hugging it.

B. Motion commands

The recorded motion commands are the actual forward and turning velocities sent to the low-level motor controller. A software remote-control mode has been implemented in the same process as the behavior learner, which reads the remote-control joystick positions, and maps them to forward and turning velocities. These velocities are then sent to the low-level motor controller and also recorded with the training examples. The purpose of recording the motion commands this way is to attempt to capture all properties of the robot motor system, including latency and dynamics, and to have the same control schema learned by the human be usable by the robot as it replays the same motor control inputs.

C. Nearest-neighbor learning

Learning is accomplished by finding the neighbor with approximately the smallest sum-of-squares difference of each feature vector component. This is simply the Euclidean distance between feature vectors. We are currently using the ANN library by Arya and Mount [12]. Currently, only the closest match is being used to control the robot, instead of, for example, a mixture of several of the closest matches. The reason for only using one match is that there are certain situations in which motion commands are ambiguous, and averaging them would result in a worse motor command than any individual example. For example, if the robot is travelling directly towards an obstacle, a human trainer may sometimes go around the left side of the obstacle, and sometimes go around the right side. Naïvely averaging these commands, obviously, would result in the robot continuing to travel forwards. Instead of naïvely averaging, for example, each closest match could contribute a Gaussian distribution of votes, weighted by the closeness of the match to the query, to a sum-of-Gaussians model, with the command receiving the largest vote ultimately being used.

Later in this study, we will investigate combining several close matches. In addition, there is currently an issue where if a bimodal distribution of commands matching the current distribution exists, the chosen commands will oscillate between modes while the low-level motor controller and the robot dynamics effectively average the oscillating commands, and the robot appears to wait a long time before changing its direction. To counter this, we will investigate incorporating hysteresis, perhaps by weighting commands in the sum-of-Gaussians model described above in favor of the last-issued commands.

IV. RESULTS AND ADDITIONAL WORK

Results thus far are primarily qualitative. It took approximately one hour of training before the robot was able to follow a path bounded by obstacles on both sides. Throughout training, and between autonomous runs, the shape of the path was changed by moving the obstacles, to investigate how well the learning generalized.

During path-following tests, a frequent problem was that although the robot turned in the right direction, it sometimes clipped obstacles, getting too close to one side of the path or another in bends. While this behavior improved with further training, it suggests that maybe learning the dynamics of controlling the robot should be separated from learning the correct trajectory. To do this, the robot trajectory could be recorded, instead of velocity commands, and a separate controller (either learned or hand-coded), could control the robot as it followed that trajectory.

Another problem, with outdoor mobile robots in general, is the difficulty in perceiving the difference between obstacles and traversable terrain. Often, including in our system, this perception relies on a set of thresholds. Each ray in the current feature representation encodes the distance to the closest obstacle, so it is possible for small, close, false obstacles obscure a

more distant true obstacle, if the obstacle detection thresholds are too sensitive. This behavior-learning system can be used to address the issue of selecting obstacle detection thresholds if the training examples are either stored with feature vectors built at various thresholds, or if the state of the world is stored in such a way that feature vectors at various thresholds can be reconstructed later. Able to build training example databases at various thresholds, the threshold can be selected that allows the learning system to best predict the correct motion commands, as evaluated with test feature-motion pairs picked from the database.

V. CONCLUSIONS

At this point in this study, we have implemented a system capable of learning mobile robot behaviors from human example, which currently can avoid obstacles and follow a path. Qualitative testing has shown that acceptable performance in such behaviors is attainable after training for one hour. At this point, issues are present with what appears to be dynamics and latency effects not being completely learned, and also with control commands oscillating between two modes in ambiguous situations. To counter these issues, we will investigate incorporating hysteresis into a sum-of-Gaussians model for voting on command direction. Later in this study, we will also incorporate learning of behavior changes, automatic dimensionality reduction, and automatic tuning of obstacle detection parameters.

REFERENCES

- [1] G. Hayes and J. Demiris, "A robot controller using learning by imitation," 1994.
- [2] C. G. Atkeson and S. Schaal, "Robot learning from demonstration," in *International Conference on Machine Learning*, 1997.
- [3] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning," *Artificial Intelligence Review*, vol. 11, no. 1-5, pp. 11-73, 1997.
- [4] M. J. Matarić, "Behaviour-based control: Examples from navigation, learning, and group behaviour," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 9, pp. 323-336, 1997.
- [5] M. J. Matarić, "Reward functions for accelerated learning," in *Proceedings of the Eleventh International Conference on Machine Learning (ML-94)* (W. W. Cohen and H. Hirsh, eds.), (New Brunswick, NJ), pp. 181-189, International Conference on Machine Learning, Morgan Kaufman, 1994.
- [6] M. J. Matarić, "Interaction and intelligent behavior," *MIT Artificial Intelligence Lab*, no. Technical Report AI-TR-1495, 1994.
- [7] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509-517, 1975.
- [8] J. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," in *ACM Transactions on Mathematical Software*, vol. 3, pp. 209-226, September 1977.
- [9] S. Arya and D. M. Mount, "Approximate nearest neighbor queries in fixed dimensions," in *Proc. 4th Ann. ACM-SIAM Symposium on Discrete Algorithms*, pp. 271-280, 1993.
- [10] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *STOC*, pp. 604-613, 1998.
- [11] T. Liu, A. W. Moore, A. Gray, and K. Yang, "An investigation of practical approximate nearest neighbor algorithms," in *NIPS*, 2004.
- [12] D. M. Mount, "Ann version 1.1.1," 2006.
- [13] G. E. Hinton and T. J. Sejnowski, "Optimal perceptual inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 448-453, 1983.
- [14] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, pp. 504-507, July 2006.
- [15] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural Computation*, vol. 3, pp. 79-87, 1991.

- [16] R. A. Jacobs and M. I. Jordan, "Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks," *Cognitive Science*, vol. 15, no. 2, pp. 219–250, 1991.