Experiments in Programming by Demonstration: Training a Neural Network for Navigation Behaviors

George Chronis and Marjorie Skubic
Dept. of Computer Engineering and Computer Science,
University of Missouri-Columbia,
Columbia, MO 65211
email: gchronis@mac.com
skubic@cecs.missouri.edu

Abstract

To address the difficulties in programming robots, we have been investigating the programming by demonstration (PbD) paradigm, i.e., extracting robot behaviors from demonstrated control actions. Our goal is to develop robot programming methods that allow domain experts to define their own task use of robots as semi-autonomous tools. For this reason, we want to inject, into the acquired behavior, the biases of the human trainer, and thus have chosen PbD instead of autonomous learning methods. In this study, we test the feasibility of training a neural network from demonstrated navigation actions, and compare the results of using three different methods to collect the training data -- a mouse-driven software joystick, programmed control, and a novel interface using a PDA (e.g., PalmPilot). Results of the experiments show that the PDA-generated training sets produce the most robust behaviors.

1. Introduction

To address the difficulties in programming robots, we have been investigating the programming by demonstration (PbD) paradigm, i.e., extracting robot behaviors from demonstrated control actions. Our motivation is to facilitate the use of robots as semi-autonomous tools. In particular, the goal is to develop robot programming methods that allow domain experts to define their own task use of robots as intelligent tools. For this reason, we want to inject, into the acquired behavior, the biases of the human trainer, and thus have chosen PbD instead of autonomous learning methods.

From our previous work in robot programming by demonstration (e.g., [1]), we have concluded that the following challenges exist:

1. Inconsistent demonstrator responses: The demonstrator often makes unintentional mistakes while demonstrating a task to a robot. This results in inconsistent training data sets.
2. Response time delays: There are variable response times among different people and even during the same demonstration session by a single person, also resulting in inconsistent training sets.
3. Inadequate interfaces: Related to the previous problems is the inadequacy of user interfaces and demonstration platforms for PbD tasks. A good interface should be straightforward for a human to use and should produce consistent data.
4. Uncertainties in measurement: Noise in the sensory input and the demonstrator input is another source of uncertainties in the training data sets. This noise is even more evident in real-time on-line training sessions, as opposed to simulated ones.

These problems aggravate the skill learning process because training data often have inconsistent mappings of sensory input to actuator commands. In addition, past PbD approaches have put an exceptional pressure on the demonstrator to produce numerous and/or perfect demonstrations, resulting in an impractical skill acquisition method. In order for this method to be feasible, we must consider the human trainer and provide an interface that facilitates quick and easy skill transfer.

In this study, we test the feasibility of training a neural network from demonstrated navigation actions and compare the results of using different demonstration interfaces. A corridor-following behavior was used as a test case because it is simple enough for comparison with other methods yet complex enough to provide a challenging test. Three methods have been investigated to collect the training data, including a mouse-driven software

joystick, programmed control, and a Personal Data Assistant (PDA) (e.g., PalmPilot).

The artificial neural network (ANN) approach has been suggested as promising for training mobile robots, based on the fact that ANNs imitate real biological systems with adaptive capabilities [2]. Rylatt et al offer a comprehensive survey on learning mobile robot behaviors using ANN's. Gaussier and Zrehen [3] also discuss the use of ANNs for mobile robot control but dismiss using supervised learning for autonomous robots. In our case, however, we want to include the expert's knowledge in a semi-autonomous robot tool. From the supervised learning point of view, Gaussier and Zrehen further argue that there has to be an explicit evaluation function, for the system to be able to adjust the weights of the ANN. We use PbD as a tool to provide the evaluation function required in ANNs trained with supervised learning. Interesting results using ANNs in supervised learning methods have also been presented in [5], [6], and [7].

The results and conclusions of our experiments are discussed in the paper. In Section 2 we describe the neural network controller, and in Section 3 the methods used for generating the training data. We discuss the experiments and results in Section 4. In Section 5, we analyze these results and conclude.

2. Neural Network Control System

To accomplish the corridor-following behavior, a Neural Network Control System (NNCS) was designed to map sensory input to the robot actuator commands, and then implemented for a Nomad 200 mobile robot. A variety of neural network structures and learning techniques have been proposed in recent years. In this study, however, we were primarily investigating the user interface issues. Thus, a simple feedforward network was selected, a multi-layer perceptron (MLP) with one hidden layer. The sensory input was provided by the 16 sonar range sensors, which are distributed equally around the circumference of the robot. Note that the learned behavior is based only on the sonar sensor readings, not on position information.

The Nomad robot has a three-wheel synchro drive system and an individually turning turret. Control of the robot requires issuing commands to two sets of actuators--the steering actuators and the translation actuators. The steering actuators are responsible for rotating the wheels of the robot and the translation actuators are responsible for moving the robot to a different location in a two-dimensional world. The actual commands are in the form of velocities. A steering velocity (SV) determines the speed and direction at which the steering actuators will move, and a translation velocity (TV) determines the speed and direction at which the translation actuators will move. Note that there is no temporal parameter, so the robot will constantly move at the given velocities until a stop or a new velocity command is issued.

A variety of network configurations and parameters were prototyped, using a MLP both as a classifier and as a controller. For the classifier case, we used a single network, kept the TV constant, and made three classes for the SV control-- one for a left turn, one for a right turn and one for moving forward. For the controller, we tried different variations, using the MLP as a function approximator. In this case, the network outputs may be any values within specified ranges. We tried using a single network with one output neuron for the TV command and one for the SV command. We also tried two networks with one output neuron each, representing the two velocity commands. Finally, we tried a single network with the SV command in the output neuron, keeping the TV constant. The number of layers and neurons per layer were also explored in the experiments, as well as the activation function.

For the experiments presented here, we used a simple and efficient network configuration which was a single MLP with 16 neurons in the input layer (one for each sonar reading), 20 neurons in the hidden layer and 1 output neuron for the SV command. A constant TV was used. For activation functions, a sigmoid function was used in the hidden layer, and a linear function (in [-1,1]) was used in the output layer. The output was then scaled to produce steering velocities up to 30 degrees per second.

3. Generation of the Training Data

The training data for the NNCS was generated by using the simulator environment developed by Nomadic Technologies for their Nomad robots. Our experience has shown that the simulator is generally adequate for learning behaviors that are also robust in the real world [8]. A simple environment map was first defined to allow for demonstrated turns in a

corridor. The sonar readings were simulated for each robot position within the environment. Figure 1(a) shows the environment map used for demonstration. The actual demonstration process was accomplished by a user "driving" the robot through the turns of the environment, using one of the tested user interfaces.

As a means of investigating and isolating the user interface issues from the neural network learning issues, we also generated one training set with programmed control using position information to guide the robot through the turns of the environment, as illustrated in Figure 1(b). Note, however, that, in all training cases, the neural network did not use position information as inputs, but only sonar data, so it learned how to navigate based on sonar inputs and not Cartesian coordinates. The intent was to acquire a behavior that can navigate successfully in any environment regardless of the environment configuration used for training.

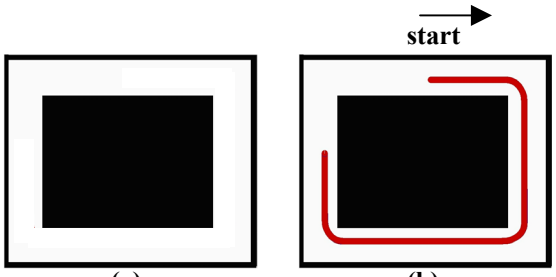

Figure 1. The environment map used for demonstrating the corridor-following behavior. (a) The blank map (b) The map showing the programmed trajectory used for capturing training data.

Two user interfaces were tested for training data generation. Because the Nomad robot control commands are TV and SV, all of the interfaces had to output commands in this format.

Both the hardware joystick and the software joystick developed by Nomadic Technologies have the vertical axis of the joystick representing TV and the horizontal axis representing SV. Figure 2(a) shows the software joystick that is included with the Nomadic simulator, which is controlled using the mouse. Users found it to be very awkward for navigating the robot on a desired path. The hardware joystick has a similar implementation, but is somewhat more user-friendly than the software joystick. While it is easier for a user to control the robot using the hardware joystick, the synchro drive train in conjunction with the Nomad software control architecture make plotting of a consistent route a very tedious process. Figure 2(b) shows a typical demonstration run produced by an inexperienced "driver", which was used as the Joystick training set #1. Although the robot was directed very close to the inner wall, the bump sensors showed that the robot never actually touched the wall in this run. Figure 2(c) shows the Joystick training set #2, which was produced by an experienced user after numerous practice trials.

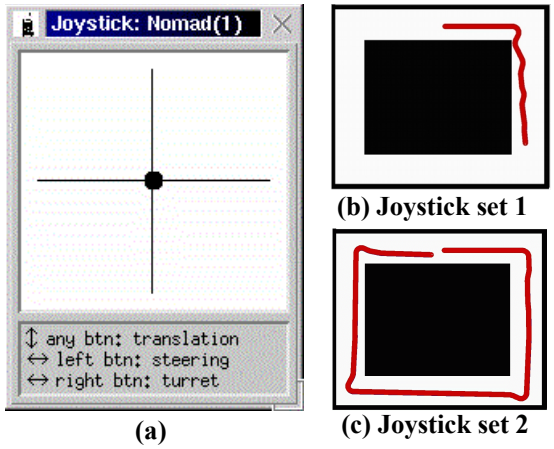

Figure 2. The software joystick used for demonstration. (a) The joystick interface window. (b) The training data set #1, captured using the joystick. (c) The training data set #2, also captured using the joystick (after intensive practice).

The difficulties in using the joystick interface have motivated the development of a novel user interface using a PDA (a PalmPilot). We hypothesize that the key to producing a user-friendly interface for demonstrating training data is to successfully map the robot control commands from Cartesian space to actuator space. The PDA allows the user to control the robot in Cartesian space by sketching a two-dimensional route for the robot to follow, as illustrated in Figure 3(a). The user-drawn route is then converted into commands that control the robot actuators. With such an interface, the user can easily generate accurate and complete training sets by demonstration. A typical demonstration run is shown in Figure 3(b). The experiments presented in the next section test the hypothesis.

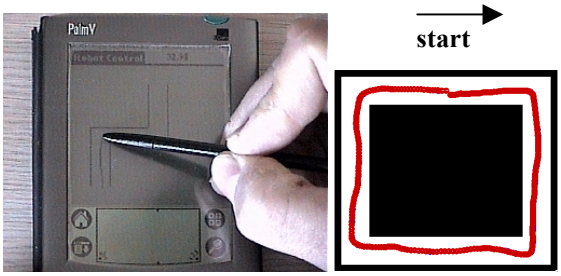

Figure 3. The PDA used for demonstration. (a) The Palm Pilot interface. (b) A typical data run (no practice required).

For the PDA interface, an environment map is pre-defined on the PDA screen to match the map defined in the simulator environment. As the user draws the trajectory on the PDA screen, each (x,y) coordinate is logged. Because of the screen pixels, the resolution is limited, and the drawn trajectory can appear quite jagged. The trajectory is smoothed by calculating an average (x,y) value for 5 adjacent points. Figure 4 shows a portion of a sampled trajectory and illustrates the averaging process. The 5-point averages are then used as endpoints of a segment to determine a new heading. The inverse kinematics of the robot are used to calculate a turning and translational velocity at each step, which are input as robot commands to the simulator. To collect the training data, the velocities are logged with the corresponding sonar readings.

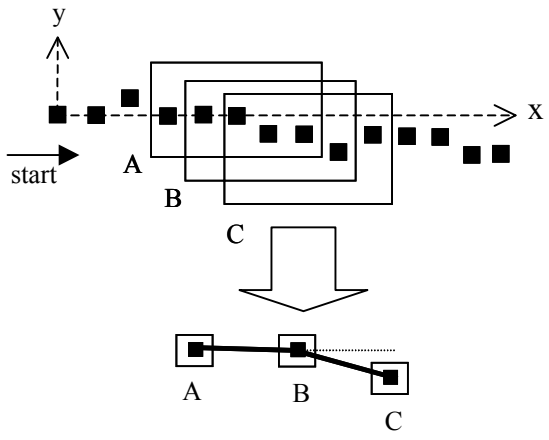

Figure 4. Deriving a change in heading using the PDA-drawn coordinates. The endpoints of the segments are (x,y) coordinates calculated by averaging the (x,y) values for 5 adjacent points.

4. Experimental Results

To test the effectiveness of the demonstration interfaces, the NNCS was trained using data generated by the three methods described above, and the trained networks were then used for autonomous robot control. The neural network configuration was the same in all cases. As described in Section 2, 16 sonar readings provided input, and one output was generated for the steering velocity (SV), while assuming a constant translational velocity (TV). The training was accomplished using the Matlab Neural Network Toolbox [9]. Fast training was achieved by using the Levenberg-Marquardt algorithm for numerical optimization.

The training results are summarized in Table 1 for 5 training sets, representing the 3 methods described above for generating training data. For each set, Table 1 shows the total number of points in the training set and the final mean square error (MSE) at the end of training, which was arbitrarily set for 25 epochs. The table also shows the epoch number and the corresponding MSE at which the network converged (Conv. Epoch and Conv. MSE), i.e., where the error curve leveled off.

Table 1. Training Results

Training Data Set	Points	Conv. MSE	Conv. Epoch	Final MSE
Programmed	1532	.00013	15	.00011
Joystick Set 1	650	.040	10	.036
Joystick Set 2	2073	.011	10	.009
PDA Set 1 (right turns only)	659	.028	5	.023
PDA Set 2 (right & left turns)	1277	.032	10	.030

The Programmed training set, shown in Figure 1(b), contains 3 right turns. The corresponding results after training with the programmed set is shown in Figure 5 for two representative test runs. As shown, the learned behavior controls the robot around several right turns but eventually hits the wall. The training results show that the network converges nicely; however, the training set is perhaps too "clean" and does not contain enough sensory variation to learn a robust behavior.

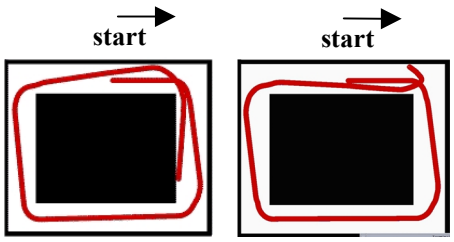

Figure 5. The results after training with the programmed training data set.

Two training sets used the software joystick for data generation. The demonstration runs are show in Figure 2; the results after training are shown in Figure 6. It is not surprising that the training results using the Joystick Set #1 would produce a hit on the first curve, as this is quite close to the demonstrated run. The results using Joystick Set #2 show that the joystick interface can produce a reasonable behavior, although the robot still hit the inside wall on the fourth turn. However, the test run took several dozens of trials and several hours for an experienced human demonstrator to generate.

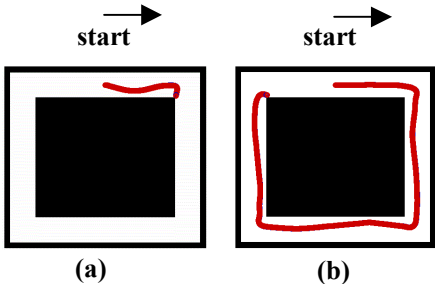

Figure 6. The results after training with the joystick-generated training data. (a) Using Joystick Set #1. (b) Using Joystick Set #2.

Finally, we present the results using the PDA-generated training data. Two sets were used. PDA Set #1 includes 2 demonstration runs around the environment square, thus producing 8 right turns. PDA Set #2 includes the data from Set #1 and also includes 2 demonstration runs around the square, traveling in the opposite direction. Thus, Set #2 includes 8 right turns and 8 left turns. The results after training are shown in Figure 7. In both cases, the learned behavior was quite robust, and the robot could travel indefinitely without hitting the walls.

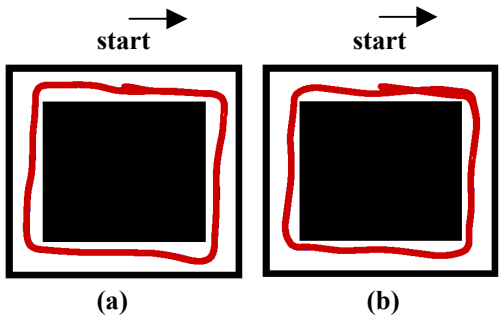

Figure 7. The results after training with the PDA-generated training data. (a) PDA Set #1 - Right turns only (b) PDA Set #2 – Right & left turns

To further test the robustness, we also tested the learned behavior in a different environment, as shown in Figure 8. The new environment contains corridors of varying width and length, and the learned behavior is robust enough to navigate through the environment without hitting any walls.

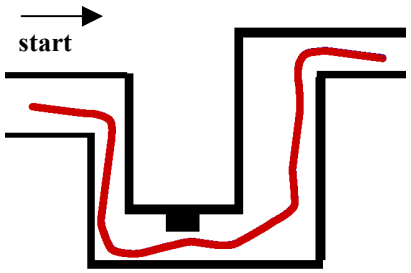

Figure 8. The corridor-following behavior in a new map, after training with the PDA-generated training data (PDA Set #2).

5. Conclusions

In this paper, we have presented a neural network configuration that may be used for training a mobile robot for a corridor-following behavior. Although we have not investigated optimal learning techniques, we can make conclusions about the training data sets and, thus, the user interfaces designed to collect the training data.

For good convergence of a feedforward MLP, the training data set must provide a consistent mapping between the inputs and the outputs of the network. To provide robust control under a variety of conditions, the training data should also include the complete range of possible sensor variations.

For mobile robot control, it is important that the training data sets be both consistent and complete. A consistent data set should not have very different actions for similar sonar readings. A complete data set should account for as many as possible of the different situations that the robot will encounter in the real world. The defect of generating a data set using the software joystick is that it is not consistent. The lack of a straightforward interface and the sensitivity of the joystick make it quite difficult for a user to generate a consistent training set as shown in Figure 2. The defect of generating a data set using the programmed control is that it is not complete. Although a program will generate the most consistent data set, it cannot account for several different situations that the robot will encounter. Since the neural network outputs are not precise and certainly not the same as the programmed outputs used for training, the robot often gets in a position that was not included in the training set. When such a case occurs, the neural network does not know how to handle it and the output is unpredictable. This explains why the robot bumps into an obstacle after navigating in the corridor for a certain amount of time, as shown in Figure 5.

With the PDA user interface, it was easier to capture a training data set that was both complete and consistent. Our assessment is that the PDA was a straightforward interface for users to demonstrate navigation behaviors. Clearly, the best results were achieved using the PDA-generated data. This serves to validate the use of the PDA as a training interface.

We succeeded in generating a reasonable behavior using training data from the PDA; however, we did not really test whether we could learn the bias of a user, e.g., stay close to the right wall or take a wide turn vs. a sharp turn. This capability was not really tested in the experiments here, although there is some evidence that it is possible. For example, the PDA-generated turns tended to be wider and the Joystick-generated turns tended to be sharper. The resulting behaviors tend to correspond to their respective training sets. Part of the challenge in demonstrating biases (which are based on sensory conditions) is that the user must be able to observe the corresponding sensory condition during the demonstration, and this will require sensory feedback to the user, in some form.

Finally, there are limitations in the neural network configuration. The translational velocity and the steering velocity are in fact coupled; ideally, the output should yield both. There will always be limitations if the translation velocity is held constant. In the future, we may explore different network configuration (such as recurrent networks) and also different learning methods.

References

[1] M. Skubic and R.A. Volz, "Acquiring Robust, Force-Based Assembly Skills from Human Demonstration", *IEEE Transactions on Robotics and Automation*, to appear.

[2] M. Rylatt, C. Czarnecki and T. Routen, "Connectionist Learning in Behaviour-Based Mobile Robots: A Survey", *Artificial Intelligence Review*, 12: 445-468, 1998.

[3] P. Gausier and S. Zrehen, "A Topological Neural Map for On-Line Learning", Animals to Animats 3: Proceedings of the Third International Conference on Adaptive Behavior, pp. 282-290, 1994.

[4] T. Kohonen, "Self-Organized Formation of Topologically Correct Feature Maps", Biological Cybernetics, 43: 59-69, 1982.

[5] G. M. Saunders, J. F. Kolen and J. B. Pollack, "The Importance of Leaky Levels for Behaviour-Based AI", Animals to Animats 3: Proceedings of the Third International Conference on Adaptive Behavior, pp. 275-281, 1994.

[6] U. Nehmzow, "Flexible Control of Mobile Robots Through Autonomous Competence Acquisition", Measurement and Control, 28: 48-54, 1995.

[7] J. Tani and N. Fukumura, "Learning Goal-Directed Sensory-Based Navigation of a Mobile Robot", Neural Networks, 7(3): 553-563, 1994.

[8] G. Chronis, J. Keller and M. Skubic, "Learning Fuzzy Rules by Evolution for Mobile Agent Control", in *Proceedings of the 1999 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, Monterey, CA, Nov., 1999.

[9] H. Demuth and M. Beale, *Neural Network Toolbox User's Guide*, The Math Works, Inc., 1998.