Fido: A Universal Robot Control System Using Reinforcement Learning with Limited Feedback

Joshua Gruenstein Michael Truell

Control System Objectives

Fido was created to fulfill the following goals:

- **Trainability**: Allow both human and autonomous training and retraining rather than programming
- **Universality**: Run on any robot and perform any task, even without prior knowledge of the host
- **Performance**: Require few learning iterations and low latency for quick and efficient training

To achieve this we developed a novel machine learning algorithm utilizing wire-fitted Q-Learning with a neural network, a probabilistic action selection model, and history sampling. The control system could be trained at a rate **3 times** the industry standard, allowing practicality over traditional pre-programmed control systems.

System Overview

Fido Control System Algorithm

Initial empty replay memory M
Initial neural network with random weights
Uncertainty value $u \leftarrow \infty$
while $true$ **do**
    Observe state s_t
    Feed model s and generate continuous function $r(a)$ of action versus reward
    Select action a_t with expected reward r_t^* using a Softmax action selection policy with temperature $T \propto u$ and the generated $r(a)$ function.
    Execute action a_t and observe reward r_t and new state s_{t+1}
    Store (s_t, a_t, r_t, s_{t+1}) in M
    $u \propto (r_t - r_t^*)^2$
    Sample $n \propto 1/u$ experiences e giving weight to newer experiences.
    Perform SPSA search on model architectures with cost function as error in predicting e when trained on e using SGD and Adadelta
    Update current model with result of SPSA search

From a macro perspective, Fido can be viewed as a "black box" system, where a set of inputs (such as sensors) go in and a set of outputs (actions) go out. Fido's goal is to maximize given reward by altering its behavior. The system operates through the following steps:

1. Sensor values are fed to a neural network.
2. The neural network outputs data points, each is an action and its expected reward.
3. A wire-fitted least squares interpolator creates a continuous function of action to expected reward using these data points.
4. An action is chosen using an Softmax selection policy that dynamically adjusts Fido's exploration level as its confidence level changes.
5. After receiving reward, the neural network is trained to output a new set of data points using Adadelta for gradient descent. Dynamic sizing of the neural network and history sampling are employed to improve neural network performance.

Reinforcement Learning

- **Q-Learning:** A popular reinforcement learning algorithm that develops a function that intakes a state-action pair and outputs expected utility
  - However, the Q-function is ordinarily modeled by storing state-action pairs in a table, but this is impractical for large state spaces
  - A function approximator such as **Artificial Neural Networks** can be used to model the Q-function instead
- Additionally, with regular Q-Learning no relation is made between similar states or actions
  - Can be optimized by coupling a wire-fitted interpolator with the neural network (**Wire-Fitted Q-Learning**)
- Complete solution: neural network is given sensor values and outputs data points on a graph of possible action vs expected utility, which are interpolated and used to generate a continuous function of action to expected reward.

Action Selection

- Cannot just perform the action with the greatest expected utility: must "explore" to be trainable and re-trainable
  - Use a Softmax Probability Distribution to select an action, temperature constant governs degree of exploration
- However cannot just hardcode the exploration level, as Fido should stop exploring if confident in its actions and explore more if it is being retrained
  - Can measure confidence using changes in the neural network weight matrix and adjust exploration accordingly
  - Increases universality by eliminating the need for an expert to determine the optimal exploration level for a given task

Training

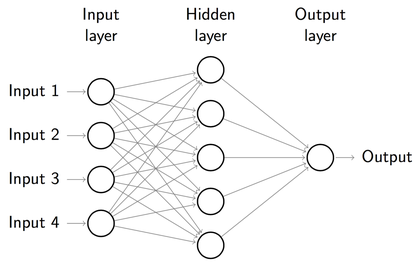

Figure: Single Output Feed-forward Neural Network

- After receiving reward, the Fido control system must update its model
  - Use Stochastic Gradient Descent to generate a new action to expected reward function for the past state, then train the neural network to output the data points that define this function
- However, many neural network training algorithms require a learning rate that depends on the specifics of the task: must be set by an expert in advance
  - Can use the new Adadelta training method which dynamically adjusts learning rate, decreases latency, and results in better neural network accuracy
- The control system uses **experience replay** to increase performance, randomly training the neural network on some of Fido's past actions every reward iteration while giving greater weight to newer actions.
  - Drastically decreases learning iterations
- Additionally, different tasks have different optimal neural network architectures. For example, larger networks are more suited for complicated tasks
  - Fido efficiently grows and prunes its neural network using neuron sensitivity approximations gathered from fluctuations in the network's weights.

Implementation

The Fido control system was programmed in C++, with no external dependencies. Two hardware implementations and a simulator were constructed to test Fido's performance. The first robot utilized a differential drive system and was powered by an Intel Edison embedded GNU/Linux computing platorm, while the second used a holonomic 3x swedish 90 degree wheel arrangment and ran on a $5 Raspberry Pi Zero.

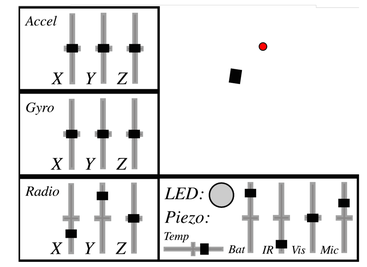

Figure: Fido Simulator Graphical User Interface

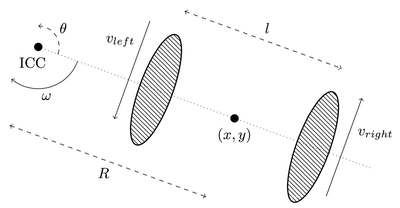

Figure: Differential Drive Kinematics

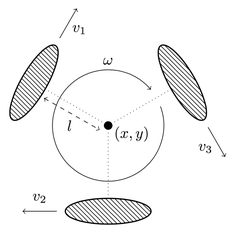

Figure: Holonomic Kiwi Drive Kinematics

Results

Results were gathered both from simulation and hardware for a variety of tasks. Gathered data for each task included the number of learning iterations, or how many pieces of reward it took for Fido to master the task, action selection time, and training time, or latency in updating Fido's model.

Table: Fido Results in Simulation (400 trials per task)

Task	Learning Iterations	Action Selection (ms)	Training Time (ms)
Flash	6	0.	6
Float to Point	14	1	6
Drive to Point	17	1	11
Line Following	18	0.	2
Noisy Line Following	21	0.	105

Table: Fido Results on Thing One (20 trials per task)

Task	Learning Iterations	Action Selection (ms)	Training Time (ms)
Stay Still	3	1	43.5
Drive to Point	18	4	65

Table: Fido Results on Thing Two (20 trials per task)

Task	Learning Iterations	Action Selection (ms)	Training Time (ms)
Drive Straight	13	2	30
Line Following	15	21	95
Fetch	8	1	70
Limping Line Following	6	20	37

Applications

Fido provides numerous advantages over traditional procedural-programmed control systems, making it practical for a number of real world applications. As Fido can be trained and retrained without expert knowledge, it could be useful in making robotics more accessible to consumers and small-buisness owners. Fido can also be retrained to adapt to new and unexpected circumstances such as malfunctions in the field, an advantageous feature for militiary applications.

Selected References

[1] S. Dini and M. Serrano, "Combining q-learning with artificial neural networks in an adaptive light seeking robot," 2012.

[2] C. Gaskett, D. Wettergreen, and A. Zelinsky, "Q-learning in continuous state and action spaces," pp. 417–428, 1999.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[4] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[5] R. Reed, "Pruning algorithms-a survey," *Neural Networks, IEEE Transactions on*, vol. 4, no. 5, pp. 740–747, 1993.