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 dynamically resized neural network, a confidence-based probabilistic action selection model, and history sampling. The control system could be trained at a rate **4 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:

❶ Sensor values are fed to a neural network.

❷ The neural network outputs data points, each is an action and its expected reward.

❸ A wire-fitted least squares interpolator creates a continuous function of action to expected reward using these data points.

❹ An action is chosen using an Softmax selection policy that dynamically adjusts Fido’s exploration level as its confidence level changes.

❺ 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

- Fido is a **reinforcement learning system**. Its goal is to accurately estimate the reward for any action it can perform.
- The system uses **Q-Learning**, a popular reinforcement learning algorithm that develops a function, the “Q-function”, that intakes a state-action pair and outputs expected utility.
- Fido utilizes a **neural network** coupled with a **wire-fitted moving least squares interpolator** instead of the traditional table to estimate its Q-function such that relations can be made between similar actions and similar states. Our neural network accepts the state and outputs a few actions along with their expected rewards. Fido interpolates these action-reward pairs to create a continuous function of action to reward.
- Once Fido performs an action and receives a reward (a single **reward iteration**), the control system calculates its **uncertainty value** as the difference between its expected reward and the real reward that it has received. Higher uncertainty means that Fido is still learning or is undergoing retraining.
- Fido selects actions to perform using a probabilistic Softmax policy which gives greater weight to actions with greater expected utility, but allows for exploration.
  - Fido dynamically adjusts its exploration level to be proportional to its uncertainty value.
  - This allows Fido to continuously execute the best actions possible once it has been trained and is sure of its actions, but also allows it to explore when being retrained. This allows for lower training times and higher average reward.

Training

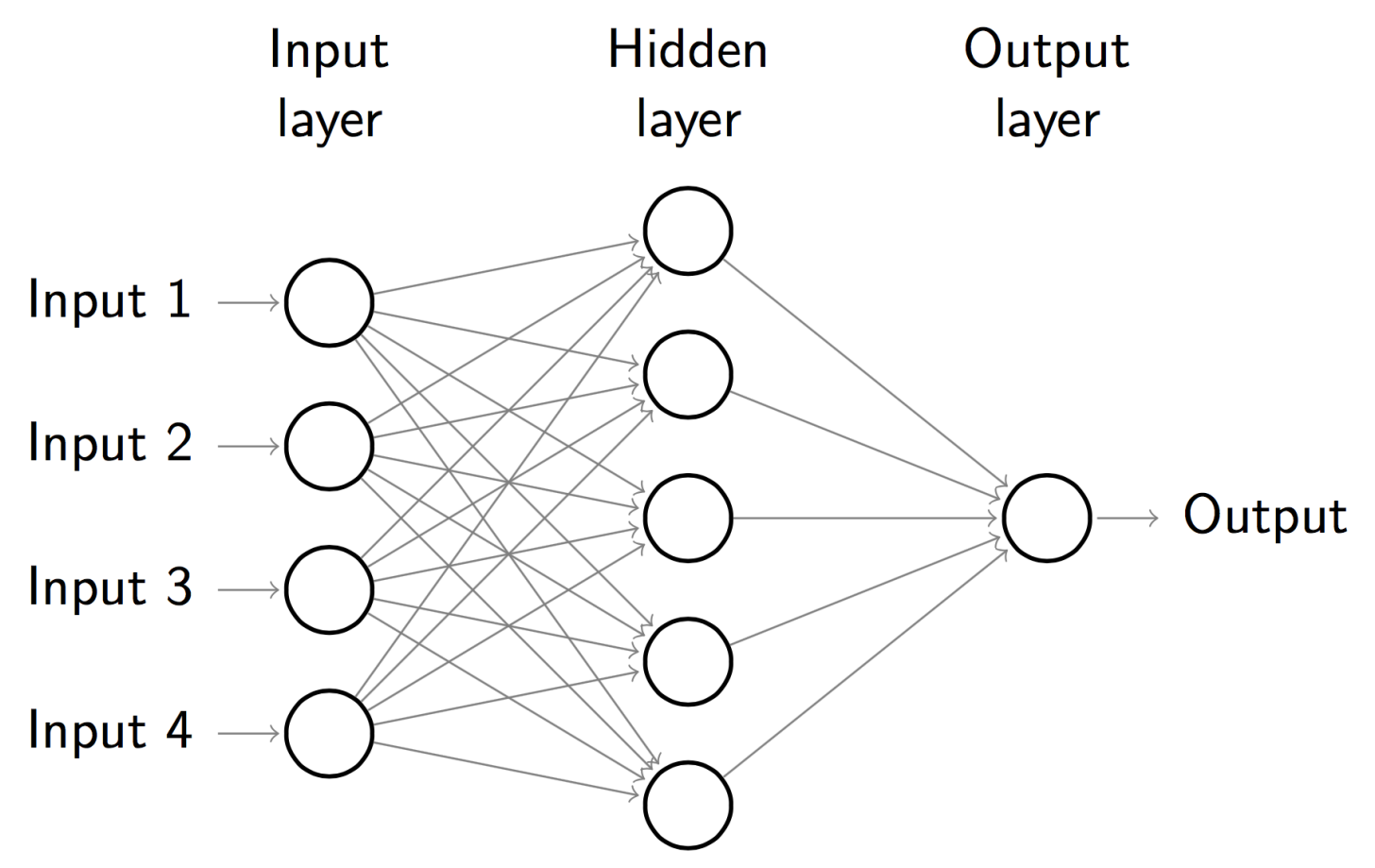

Figure: Single Output Feed-forward Neural Network

- After every reward iteration, Fido adjusts its model to correct the disparity between its expected reward and the reward it just recieved using **Stochastic Gradient Descent** to modify the action-reward pairs that are outputted by its neural network.
- Fido trains its neural network using the Adadelta training method, eliminating the need to specify a hard-coded learning rate.
- Fido uses **experience replay**, the practice of training a system on past states, actions, and rewards to decrease training time. Fido gives weight to newer experiences. If uncertainty value of the model begins to rise, Fido samples fewer experiences, because past rewards will be incorrect if Fido is being trained on a new task.
- After each reward iteration, Fido efficiently grows and prunes its neural network to the optimal size and architecture for the task at hand, using neuron sensitivity approximations.

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

Performance vs. Discrete Q-Learning

Hola

Implementation

The Fido control system was programmed in C++, with no external dependencies. Three 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, the second used a holonomic 3x swedish 90 degree wheel arrangment and ran on a $5 Raspberry Pi Zero, while the third was a 4-axis jointed arm running of a Raspberry Pi Zero.

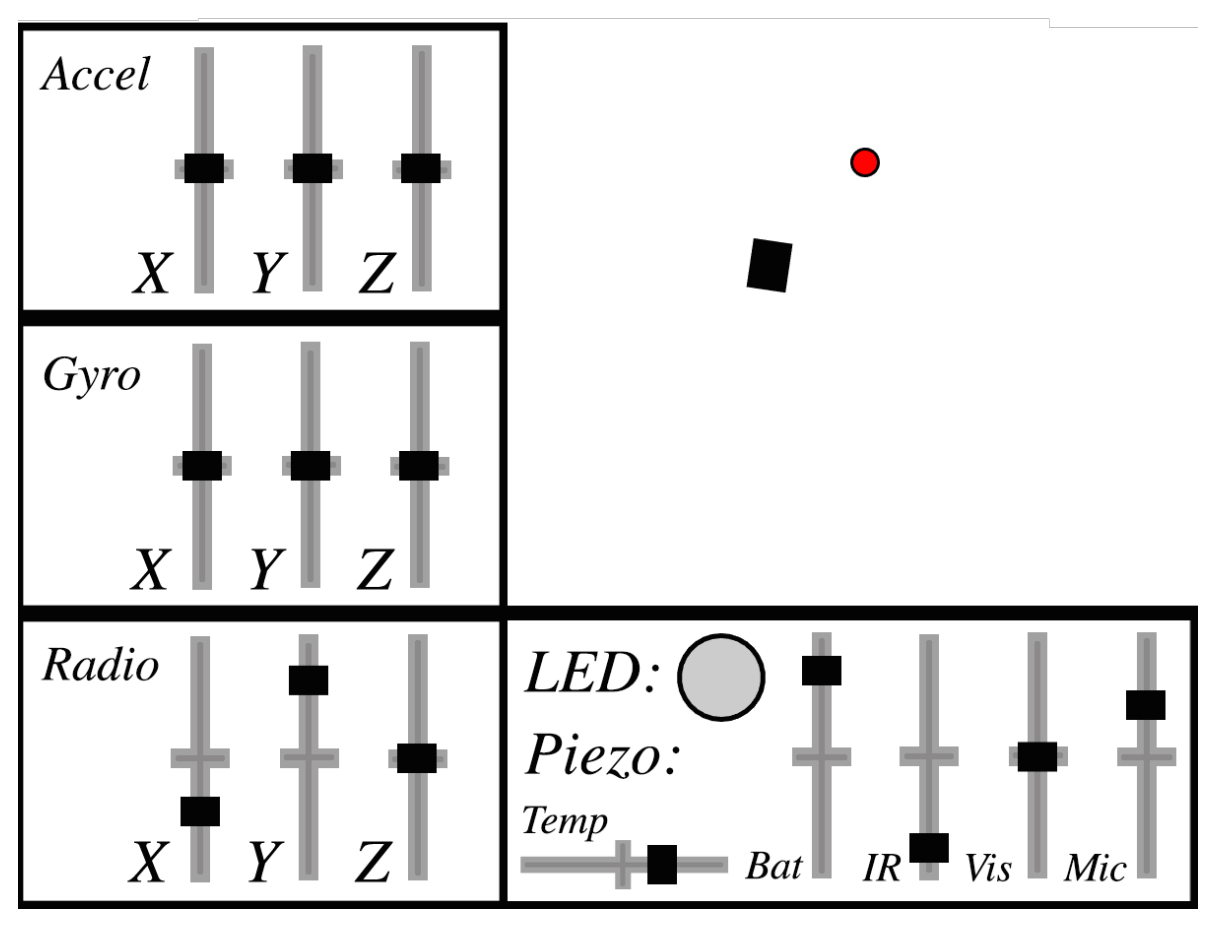

Figure: Fido Simulator Graphical User Interface

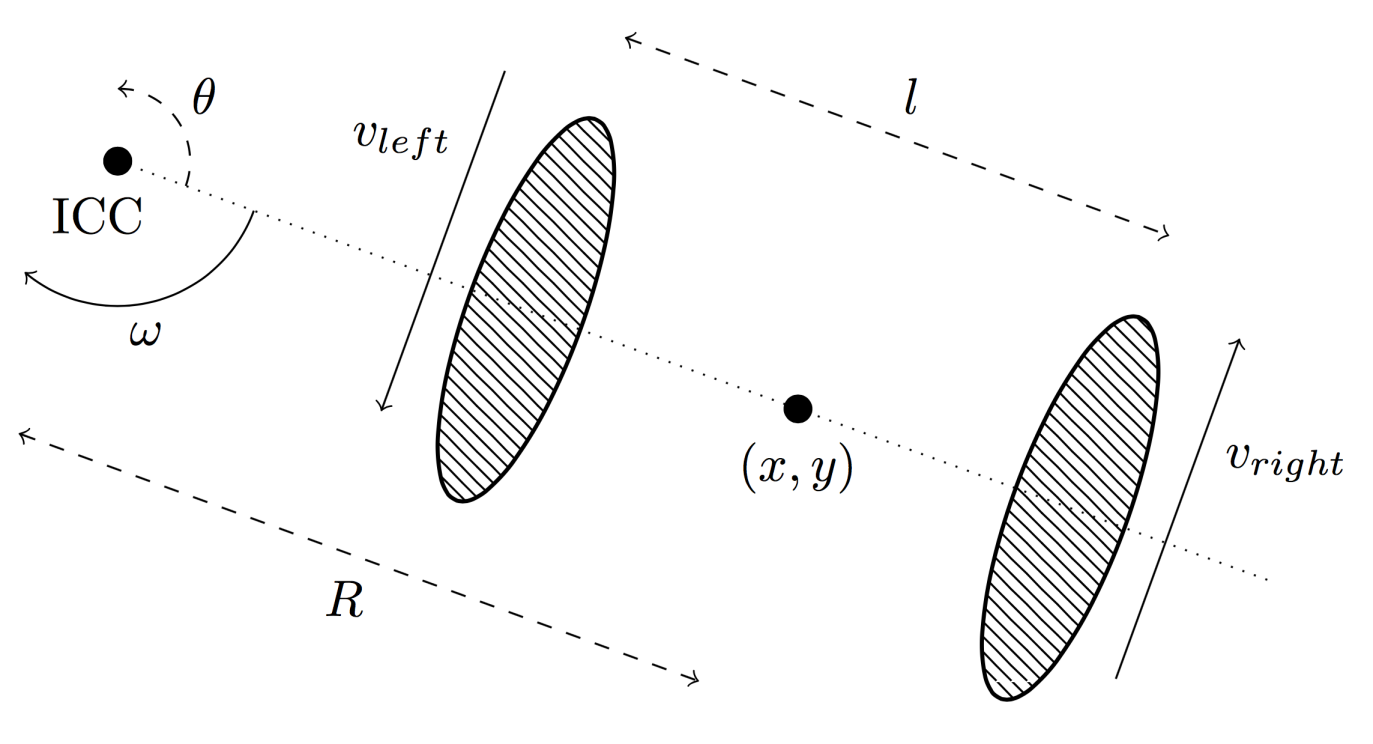

Figure: Differential Drive Kinematics

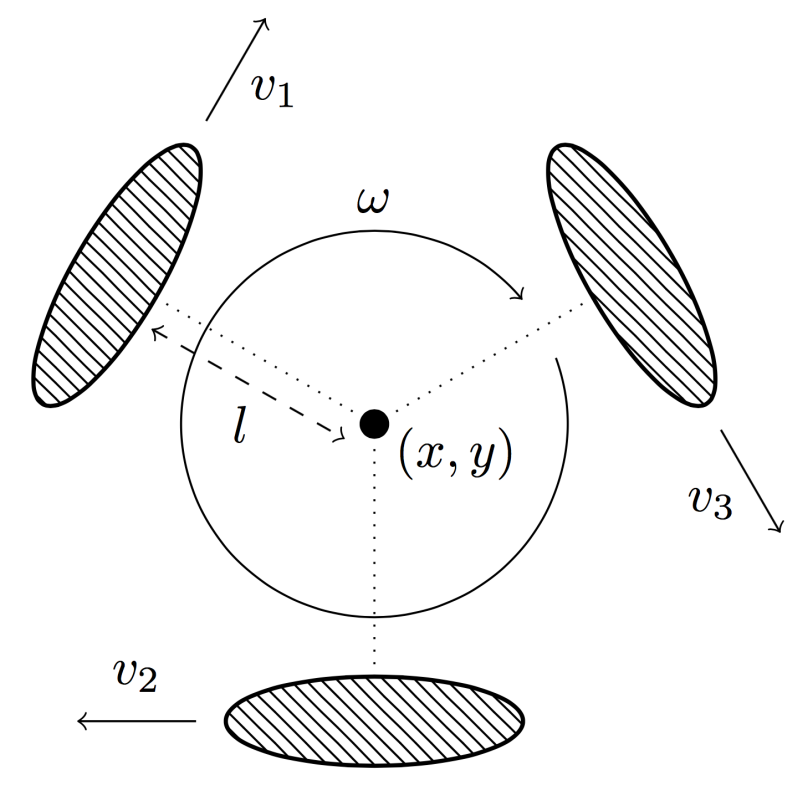

Figure: Holonomic Kiwi Drive Kinematics

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.