BHV Input: Implementing Learned Behaviors

Spring 2018

Arjun Gupta, argupta@mit.edu
Department of Computer Science
MIT, Cambridge MA 02139

1 BHV Input: Implementing Learned Behaviors

The goal of the `BHV Input` is to allow for automatic, intelligent behavior creation to aid in the Aquaticus project of capture the flag. This behavior allows the user to load in arbitrary reinforcement learning models using a python script that is embedded in the behavior. The behavior can use these models to approximate the best action for any given state the robot is in.

2 BHV Input Dependencies

Prior to using the `BHV Input` Learning platform, the following dependencies must be installed.

2.1 MAC OS X and Ubuntu

This platform requires Python2.7 (comes pre-installed on recent MacOS) as well as a number of Python packages. If you are not running on MacOS and do not already have Python2.7 installed, you can download it here:

https://www.python.org/downloads/release/python-2714/

After Python has been successfully installed, run the following commands in order to install the python dependencies:

```
$ pip install numpy
$ pip install matplotlib
$ pip install tensorflow
$ pip install keras
```

3 Configuration Parameters

Unlike traditional behaviors, the parameters for BHV Input are primarily set in the configuration file, table.csv, which is output by the machine learning platform. All important information for the behavior is contained in the configuration file and taken from local node reports.

4 Variables Published

Publishes INP_STAT which contains infomation about the current state and action taken at that state which can be parsed and used as training data for the learning algorithms. Also publishes GRAB_REQUEST when within range of the flag to automatically capture the flag.

5 Functionality

The behavior initializes by reading in the table.csv, which needs to be in the same directory as the .bhv file that is executing BHV Input, to get information about state setup parameters, heading mode, whether it is optimal or not, and the absolute path to the model. It initializes the embedded python script using the python utility and then loads the models from the given directory into a python object. That object is then used for the duration of the behavior to find the optimal action. On each iteration, the behavior reads in information from the environment and crafts a state vector containing the necessary, ordered, state information and passes it to the embedded python script to predict which action it should take. The Behavior finally takes the action, which consists of a speed and a heading, and sets those values as the peak of the IvP function that it outputs.