Documentation Foraging and odNEAT

Cees Schouten and Tjarco Kerssens

|  |  |
| --- | --- |
| **Date** | **Change** |
| 20-06-2016 | Initial document |
|  |  |
|  |  |

# Introduction

We continued building on the existing codebase of the project which implemented the Obstacle avoidance and Foraging task. We have improved the existing foraging task and Augmented the Obstacle avoidance with Social learning by implementing the odNEAT algorithm.

# Foraging

We improved on the foraging task by implementing the principle of asynchronous programming, which improved performance because threads where not waiting on each other. Furthermore, we included the input from the proximity sensors of the robots to improve the behaviour while searching for the puck (i.e. by not driving into the walls)

# odNEAT

odNEAT is an extension on the normal NEAT algorithm where multiple robots communicate their best controller to the other robots. The algorithm is implemented in such a way that it can be used in the same way as NEAT. That is, one only needs to replace the import from neat to odneat in the task class file of the desired task.

The implementation logs the results of of the sharing of controllers to a file in logs/TIME\_STAMP\_odNEAT, where TIME\_STAMP is the time of creation of the log.

# Tasks

A task can be implemented by subclassing the TaskEveluator class. This TaskEvaluator instance can than be given as an argument to NEATpopulation object, which will run the NEAT algorithm on the task

## Structure of a task

There are three important methods to implement in your task subclass:

* **evaluate**This function is used to give a fitness to the current controller.
* **\_step**A single step in the controller. Here the neural network gets it input, depending on the task, and gives an output which is generally the motorspeed for the left and right motor.
* **getFitness**Implements the fitness function for the given task.

## Running a task

A task can be run by giving it as an argument to the NEATpopulation object. There are many parameters for these objects, which will be explained in the paper about neat. It is best to look at the implementation of obstacle\_avoidance and look for the main execution point (if \_\_name\_\_ == '\_\_main\_\_').

A task can then be run on one robot by running the start\_one.sh script on the robot itself, with as arguments the task, the IP and the git version (obsolete, one can just provide a random integer here)

Example: ./start\_one.sh obstacle\_avoidance.py 192.168.1.42 1

Multiple tasks can be started at the same time, by having all the IP-addresses of the robots you want to use in the bots.txt file, each on a separate line. One can than start the task by calling ./start\_all.sh.

Example: ./start\_all.sh obstacle\_avoidance\_distributed.py