Project 4:
Artificial Swarm Behavior

Mikael Brevik, Pål Moen Møst and Øyvind Selmer

April 20, 2012

Contents

Chapter 1

Introduction

1.1 About the project

The main task of this project was to create a webots controller based on brook's architecture for behavior-based systems. The robots' task was to cooperate to find and transport a box(artificial food) to the one of the world's edges. The strategy is inspired by ants' swarm behavior, and should be performed without any form of centralized organisation or communication between the individuals.

The e-puck robot used in this project is equipped with proximity and IR sensors, and left and right wheels. The box' density is set so that it'll take more than one e-puck to accomplish the task of retrieving the artificial food.

In addition to programming the controller we had to build our world in webots. This world was supposed to simulate an ant's world, with the box as artificial food and four surrounding walls acting as the hive.

1.2 Deliverables

1. Describe briefly the overall system in your words with diagrams and text
2. Describe the limitations of e-puck to implement such behavior based intelligent swarm behaviors.
3. Mention the changes if you made in any of the behavior modules to achieve the box-pushing more effectively and/or efficiently.
4. All design codes used with the project should be delivered.
5. A working demo of box-pushing task with eight robots (in simulation).

Chapter 2

Deliverables

2.1 Description of system overview

2.1.1 Behavior-based control

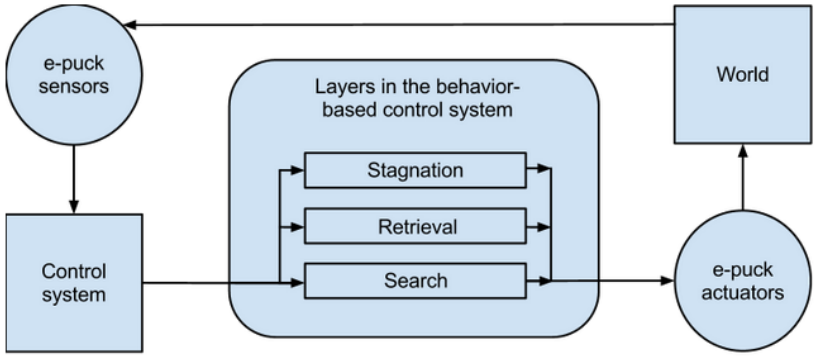

Figure 2.1: Structure of our behavior-based system

In behavior-based systems, the robot's movement is determined via interaction between behaviors. All the information that the robot has about the environment comes through sensors. In this project we've implemented Brook's subsumption architecture. As shown in Figure 1, this architecture consists of several layers, where the top layer has the highest priority. The lowest layer is usually considered what's needed to survive, while the other layers are used to trigger certain behavior. Higher priority layers can supress lower ones, and shouldn't be triggered if there's a risk of not surviving.

2.1.2 Code overview

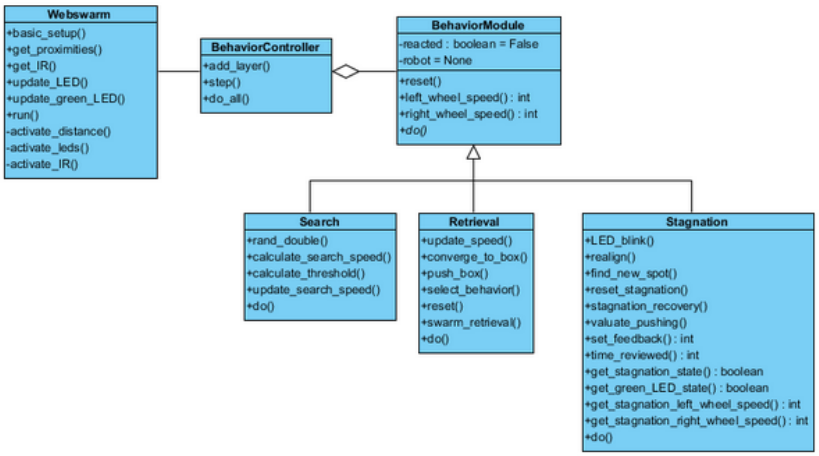

Figure 2.2: Class diagram

2.2 E-puck limitations

The e-puck has certain limitations when it comes to implementing intelligent swarm behaviours.The e-puck has no way of simulating stigmergy, indirect communication between agents using the environment. If we take example an ant colony. The ants leaves a pheromone trail to the food source and back to the nest. The pheromone trail functions as a shared external memory. The trails are also dynamic and fades when a food source is empty. Direct communication can also be a challenge. The e-puck is equipped with IR emitter and receiver so agents can do direct communications with each other it also have bluetooth adapter, but none of the solutions scale well in a swarm. In the master thesis[Berg and Karud(2009)] they have tested out multiple bluetooth connection to epucks they experienced interference problems and delays. Sound signals could have been used to replicate direct communication in a swarm. The e-puck is equipped with three sound sensors and one speaker.

Navigation is also a limitation, has no GPS sensor. Like ants and bees that could navigate by using the sun. What the e-puck do have is an accelerometer that could have been used to track the distance travelled by the e-puck. Like for example ants are claimed to have internal pedometers [Khamsi(2006)] that they use to keep track of steps so they know the distance they have travelled.

The epuck have some physical limitations as well, it have no way of grabbing things that leads to issues when retrieving resources. In the master thesis[Berg and Karud(2009)] they experienced that e-pucks slipped when in trying to move the artificial food source, it have limited pushing power from the master thesis[Berg and Karud(2009)] one e-puck is able to push an object 3/4 of its weight, in comparison of the ant that is known have a dragging capacity of several times its body weight.

2.3 Changes to the behaviour module

Beyond the obvious changes to the code base, like translating to the python programming language, and with that change some of the architecture, some minor changes has been made the to behaviour modules. We found that the original code worked rather well, and there was no clear weakness that could be enhanced.

2.3.1 Search module

Fixed inconsistencies in the case scripts

The case scripts, used to handle avoidance of objects, worked mostly very well, but there were some errors that could be fixed. E.g. The case $[1, 1, 0, 1]$ (block on both the left sensor and the one far to the right) was set to turn left. This doesn't make that much sense, as it should turn to the side with less resistance/obstacles. This can also be reflected to the case $[1, 0, 1, 1]$, which was also set to turn left. The $[1, 1, 0, 1]$ case was then set to turn right, instead of turning left.

Reset random factor for turn after a wall crash

When searching for the food source, the search module uses a random factor to be able to search in different places. This, sometimes, results in illogical decisions. One could argue that ant aren't that logical to start with, but its seen that ants remembers the path they've taken. So if the e-puck crashed in the wall, it shouldn't try to crash into that same wall again.

Lets say the random value for left wheel is set to 0.2 and the right wheel at 0.8. When meeting a wall and turning right, the e-puck will again turn into the wall (if it's a while until the random factors resets). In stead of doing this, we've expanded the code to do the following: After a turn, reset the random factors to favour turn in the same direction that the e-puck just turned – but only do so for 10 epochs (half of the original iteration count). The reductions of epochs is due to not over compensate and turning into the wall again the other direction.

2.3.2 Retrieval module

The threshold values were altered to suit the simulator, instead of the real world example as with the code given with this projects. There were some differences both in the IR light source from the food source, as well as with the other sensory inputs.

Minor enhancement in checking for convergence

There were implemented some minor changes to the check if the robot is converting. The robot will now set the converge flag to false if it gets out of range during converging. As the stagnation module is using this module to check for state, it needs to be responsive and stable.

2.3.3 Stagnation module

As with the retrieval module, the threshold had to be adjusted. The set thresholds were not working with their original values. Some work went into trying to set these thresholds to be optimal.

Overall algorithm for stagnation

We implemented stagnation according to the description in chapter 6 of the master thesis[Berg and Karud(2009)] whereas you start at 150 epochs when a robot registers a food source. After iterating through

all epochs, the push state is evaluated and results in either a positive or negative feedback. A negative feedback (where either reposition or realignment is performed), will result in the epochs of action is decreased, positive feedback will keep the e-puck pushing, i.e. the controller will turn to the next module (retrieval) to handle behaviour. The controller will also go to the next module if no push or converge flags are detected.

To evaluating if stagnation occurs, we first read out the proximity sensor values, wait for 10 time steps (in our case wait $(10 * 64)$), and then measure the distance again.

Repositioning

We set the reposition to have a high precedence. When it's started it won't halt until it's done. This might not sound very efficient for solving the overall task, but in practice it seems to work really well. This way it does not take part of the epoch runs mentioned in the previous section. While in the reposition mode, the search module is used to avoid obstacles if necessary.

As partly mentioned earlier in this report, we contemplated about finding a way to measure positive or negative distance covered by the robot, but couldn't find a way of doing that, only using the IR and distance sensors. If we had access to this information, it would improve the overall system a great deal. The e-pucks could have reposition straight away, when a negative distance covered was measured. It could also have worked the other way, create a more stable push evaluating, i.e. when positive distance covered measured and e-puck in push mode, we know it's working properly.

List of Figures

Bibliography

[Berg and Karud(2009)] Jannik Berg and Camilla Haukenes Karud. Swarm intelligence in bio-inspired robotics. Master's thesis, NTNU, 2009.

[Khamsi(2006)] Roxanne Khamsi, 2006. URL http://www.newscientist.com/article/dn9436-ants-use-pedometers-to-find-home.html. Ants use pedometers to find home.