A new Approach to Control a Population of Mobile Robots using Genetic Programming

Anderson Luiz Fernandes Perez
Department of Automation and Systems
Federal University of Santa Catarina - UFSC
Florianópolis, SC, Brazil
anderson@das.ufsc.br

Guilherme Bittencourt
Department of Automation and Systems
Federal University of Santa Catarina - UFSC
Florianópolis, SC, Brazil
gb@das.ufsc.br

Mauro Roisenberg
Department of Computer Science
Federal University of Santa Catarina - UFSC
Florianópolis, SC, Brazil
mauro@inf.ufsc.br

ABSTRACT

This paper describes a new Evolutionary Control System (ECS) able to control a population of mobile robots. The system has two main modules: the first one, called EMSS (Execution, Management and Supervision System), is the system responsible for managing all the evolutionary process that takes place in an embedded fashion in each robot. The second module, called DGP (Distributed Genetic Programming), is an extension of classical Genetic Programming algorithm to support the control system evolution for the robots that are part of the mobile robots population. Simulation experiments of the DGP algorithm are presented and their results are compared with the classical GP algorithm.

Categories and Subject Descriptors

I.2.2 [**Artificial Intelligence**]: Automatic programming, program synthesis; I.2.9 [**Robotics**]: Autonomous vehicles

General Terms

Mobile Robots, Embedded, Experimentation

Keywords

Evolutionary Robotic, Genetic Programming, Embedded Evolutionary Control System

1. INTRODUCTION

Control systems for mobile robots are considered as hierarchical structures of basic behavior. For this reason, using Genetic Programming (GP) to develop control systems for mobile robots represents an advantage. In GP the handled structures, functions and terminals, are high level and the generated programs are hierarchical associations between them.

GP is an Evolutionary Computation (EC) [5] technique that aims at automatically generating computer programs. The main goal of GP is to allow computers to program themselves, i.e., from specified primary behaviors, the computer must be able to generate a program that satisfies some conditions that aim at the solution of some task or problem [15].

Evolutionary Robotics (ER) [9] aims at the development of adaptable control systems, based on the EC techniques. One of the research areas in ER is Embodied Evolution (EE) [3] where the evolutionary process takes place among the robots, that is, the reproduction takes place among the individuals that are part of the robot population.

In this paper we describe a new Evolutionary Control System (ECS) to a population of mobile robots. The ECS have two main modules. The first one, called EMSS (Execution, Management and Supervision System), is responsible for executing and managing the second module, called DGP (Distributed Genetic Programming). The DGP is an extension of the classical GP algorithm [7].

To validate the DGP we accomplished one experiment with a population of mobile robots using the Khepera Simulator. The objective of each robot was to navigate through an unknown environment looking for food and avoiding obstacles. The experiment was accomplished with both, traditional GP and DGP, algorithms.

This paper is organized in the following way: Section 2 presents the description of the Evolutionary Control System and the Distributed Genetic Algorithm, respectively. Section 3 describes the experiment setup. Results comparing DGP's performance and GP's performance are presented in Section 4. Section 5 describe conclusion and future works.

2. EVOLUTIONARY CONTROL SYSTEM

The Evolutionary Control System (ECS) has two main modules. The first one, called EMSS (Execution, Management and Supervision System) is responsible for executing and managing the algorithm responsible for all evolutionary process. The second module, called DGP (Distributed Genetic Programming) is an extension of classical GP algorithm to support the control system evolution for the robots.

Figure 1 illustrates, generically, the functioning of the ECS.

SAC'08 March 16-20, 2008, Fortaleza, Ceará, Brazil

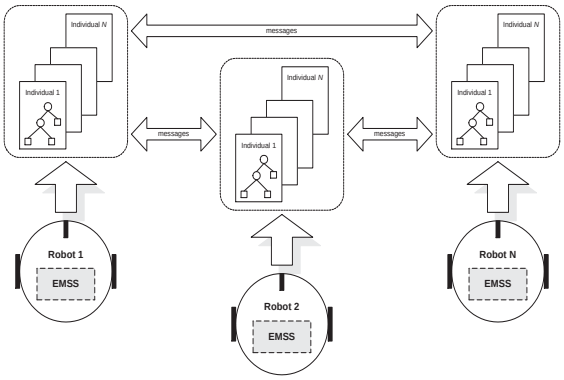

Figure 1: General view of ECS.

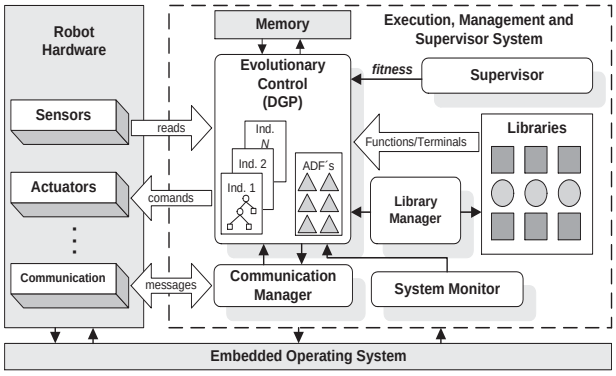

Figure 2: Basic Structure of EMSS.

In each robot, the EMSS is executed. Each robot has a local population, a set of programs that may solve a problem (a task that the robot should execute), that interacts with the local populations of the other robots. The goal is to build, by using only GP, a control system for a population of mobile robots in which the robots interact among them to execute some task.

Sections 2.1 and 2.2 describe in details how EMSS and DGP work, respectively.

2.1 Execution, Management and Supervision System - EMSS

The EMSS (Execution, Management and Supervision System) is the system responsible for managing the evolutionary process that takes place in an embedded fashion in each robot. Figure 2 illustrates the EMSS components and the relationships among them.

Below is the complete description of each component of the EMSS and the connection among them (when there is one).

Evolutionary Control (EC): it is the main component in the EMSS. The EC is responsible for creating, randomly, a local population of individual programs using the functions and terminals contained in the library. In each new generation, the generated individuals are tested and executed by the EC.

Memory: the memory stores the representation of the most adapted individuals in each generation. These individuals can be used either in a reinitialization process, in the case the EC fails, or in the optimization of tasks that involve more than one competence, such as avoiding obstacles and moving an object from one place to another. In both cases, the memory works as a kind of system snapshot, which can be reused to speed up the learning process through experiences taken place in the past.

Library Manager (LM): this component manages the sets of terminals and functions in each robot. In this component all the functions and terminals are identified through an unique identifier, an *even* number for the functions and an *odd* number for the terminals, so they can be sent to other robots through the Communication Manager. At any time, it is possible to add or remove functions or terminals without the need of redefining the control system.

Communication Manager (CM): it is the component responsible for sending and receiving the messages exchanged by the robots. In the message sending process, the EC sends to CM the identification number of the sequence of functions and terminals and the fitness value associated with the best local individual. The CM creates a message containing this information and sends it to the other robots. In receiving, the CM receives the messages sent by the other robots and sends their contents to the EC.

Supervisor: it is responsible for evaluating and attributing a fitness value to each individual, using a *punishment* and *reward* method. For this method to work properly, for each task is assigned to the robot, one must define how and when reward and punishment take place.

Monitor: it is the component responsible for evaluating the system execution. In the case the system is inactive, i.e., the robot stays still for a long period of time, the monitor activates an EC reinitialization process. When the EC is reinitialized, the images of the best individuals, which are stored in memory, are retrieved and the evolutionary process initiates from these individuals, that is, the local population is completed through the use of parts of these individual programs, instead of starting from a random population, as it happens in the original GP algorithm.

2.2 Distributed Genetic Programming - DGP

The DGP is the algorithm responsible for all the evolutionary process of the robots control system. The DGP is an extension of the classical GP algorithm to support the control system evolution for the robots that are part of the mobile robots population.

The DGP is based on Microbial GA [6], a GA variation. Its functioning is similar to the genetic combination (infection) that takes place in bacteria, where DNA segments are transferred between two members of the population.

In DGP two population sets are considered. The first, called local set or $P_{local_{R_i}}$, refers to the local population of each R_i robot, i.e., the set of individuals or solutions that are embedded on the robot. Each $x \in P_{local_{R_i}}$ represents a candidate solution to a problem. The second set, called total set or P_{total}, is made of the union of all the local populations in each robot, where: $P_{total} = P_{local_{R_1}} \cup P_{local_{R_2}} \cup \cdots \cup P_{local_{R_n}}$. The evolutionary process takes place always considering the total population, that is, parts of a local individual of a certain robot may be considered in another robot's local population evolutionary process.

Differently from other approaches in EE, such as [11], [12], [13] and [14], in DGP the evolutionary process is asyn-

chronous, that is, it is not necessary that two robots are synchronized to reproduce.

In DGP, parts of a more adapted individual are sent to all the other robots. The step sequence in DGP is the following:

1. Randomly create a program population;
2. Iteratively execute the following steps until some halt criterion is satisfied:
   (a) Evaluate each population program through a heuristic function (fitness) that expresses its fitness, that is, how well the program solves the robot task;
   (b) Receive[1] parts of a remote[2] individual sent by another robot;
   (c) Select the t best individuals in the local population, using the tournament selection method;
   (d) Randomly select a part of the best (more adapted) local individual and broadcast it to the other robots;
   (e) Compare whether the worst locally selected individual fitness is lower than the remote individual fitness. If so, execute the mutation operator, replacing parts of the individual by the parts received from a remote individual;
   (f) Execute the crossover and mutation operators;
3. Return the best program found.

The selection method used in the DGP is tournament selection with the parent preservation after the crossing. The remotely received parts are added to the worst individual structure, from the t selected best, following equation 1:

$$M(A) = \left\{ \begin{array}{ll} Muta(A) & \text{if } FitnessR > FitnessL \\ A & \text{if } FitnessR \leq FitnessL \end{array} \right\} \quad (1)$$

where $FitnessR$ is the value of the remote individual's fitness, which sent a part of its structure. $Muta(A)$ is the mutation function, where a part of the A structure in the local individual is randomly chosen to be replaced by the received part of the remote individual program.

The messages passing between the robots must contain the fitness and a part of the best local population individual program. For such, all the functions and terminals receive an unique numerical identification, an *even* number for each function and an *odd* number to each terminal.

It is important to emphasize that for a correct functioning of DGP all robots must contain the same sets of functions and terminals, or at least they must use the same functions and terminals for a particular problem.

Differently from other approaches, such as, [4], [10], [16], DGP assures the continuity of the control system of the evolutionary process, even when there is a problem with the other robots that are part of the robot population. This is possible because each robot has a local population of programs, which guarantee the continuity of the evolutionary process.

[1]In each execution cycle the DGP considers only one received message. Each new message, containing parts of a remote individual, is stored in the local buffer. The buffer is overwritten every time a new message is received.

[2]A remote individual is a program that is part of the local population of another robot.

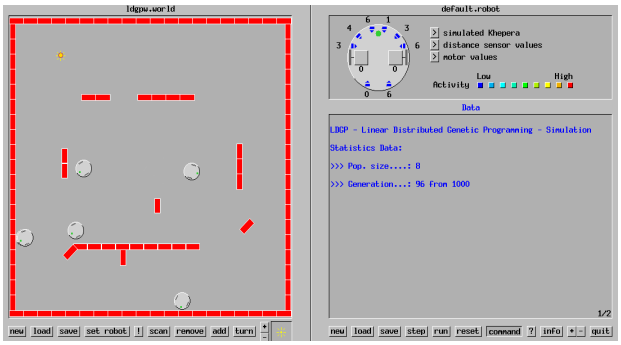

Figure 3: Khepera Simulator screenshot.

3. EXPERIMENTAL SETUP

With the purpose of evaluating the DGP algorithm we defined a simulation environment in which a set of mobile robots must navigate looking for food (foraging task). The simulation was accomplished in the Khepera Simulator version 2.0 [8]. This version of the Khepera Simulator can operate with a group of mobile robots instead of a single robot.

Figure 3 illustrates the main interface of Khepera Simulator. In the figure there are five robots dispersed in a random position along the environment, a set of rectangles that represents walls and one lamp that represents food.

Each Khepera Robot has eight (8) infrared sensors for proximity measurements, eight (8) light sensors for ambient light measurement and two motor-drive wheels for movement. Each infrared sensor returns a value ranging [0 - 1023]. Where 0 means that no object is perceived and 1023 means that an object is very close to the sensor. The light sensors can measure the level of ambient light around the sensor and return a value ranging [0 - 525]. Where 0 means a maximum brightness and 525 a maximum darkness.

The goal of each robot is to navigate along the environment looking for food (lamp) and avoiding obstacles (walls and other robots). Table 1 describes the function and terminal sets used in the experiment.

Table 1: Functions and Terminals sets

Functions			
Name	**Arity**	**Id.**	**Definition**
FoodAhead	2	2	If found food, perform left node; otherwise, perform right node.
Prog2	2	4	Perform two branches of the tree.
Prog3	3	6	Perform three branches of the tree.
Terminals			
TurnRight	0	1	Turn right (15 degrees).
TurnLeft	0	3	Turn left (15 degrees).
GoForward	0	5	Go forward (300 ms).
Return	0	7	Return (300ms).

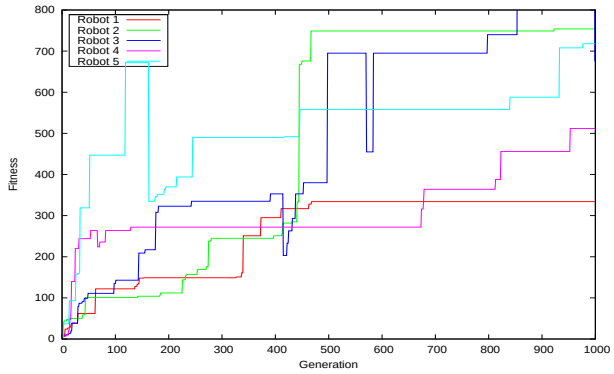

Figure 4: Fitness of the best individual at each generation with DGP.

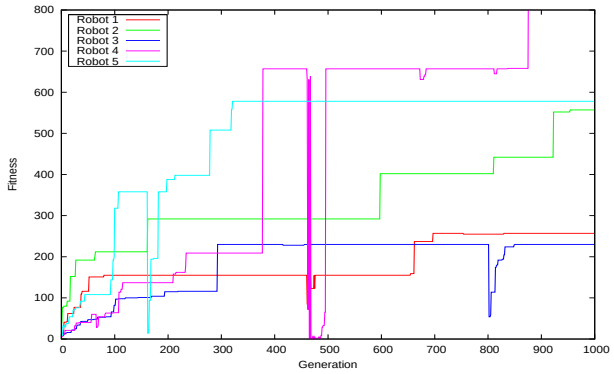

Figure 5: Fitness of the best individual at each generation with GP.

In Table 1 the *Id.* column represents the identification of functions and terminals according to identification rule of DGP (see Section 2.2), an *even* number for each function and an *odd* number for each terminal. The parameters used in both, DGP and GP algorithms, are described below:

- **Number of robots**: 5
- **Generations number**: 1000
- **Population size**: 8 individuals
- **Vector size**: 50 positions
- **Crossover probability**: 50%
- **Mutation probability**: 20%
- **Selection method**: tournament selection
- **Execution times**: 10
- **Fitness function**: punishment and reward method. `If FOODAHEAD; fitness += 10; if BUMPED; fitness -= 1; else fitness += 1`

For this experiment no limit was imposed to fitness value. That is, the fitness could be any negative or positive value. The highest value corresponds to the best individual in the population and the lowest value to the worst one.

The DGP algorithm was developed in C with a linear representation method [1]. Each population individual (program) is represented by one fixed size vector. Each vector position only contains the identification number of a function or a terminal, according to *Id.* column in Table 1.

4. RESULTS AND COMMENTS

The simulation was executed using both, DGP and GP, algorithms. Figures 4 and 5 illustrate the fitness value evolution graphs of the best individual at each generation for each robot, during 1000 generations.

In DGP (Figure 4) the fitness value of the best individual increases faster because a genetic variation occurs any time a message from other robots is received and accepted to mutate one individual (from the t selected best) in the local population, according to DGP rules.

However, in simulation with GP algorithm (Figure 5), some generations have the same fitness value to the best

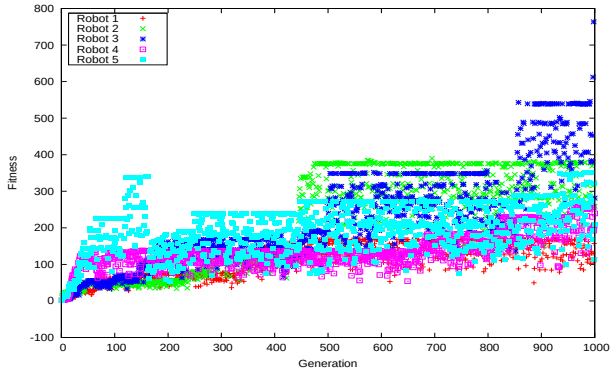

Figure 6: Average population fitness at each generation with DGP.

individual and there is a great disparity among the best individuals fitness. To try to simulate the same behavior of DGP, always the second parent in GP after crossing, was mutated by the mutation operator.

In both algorithms, occasionally, the fitness of the best individual does not change during some generations. This happens because a local maximum solution was found and the selection method is tournament with parent preservation after the crossing.

The average population fitness graphs at each generation are illustrated in Figures 6 and 7 for both, DGP and GP algorithms, respectively.

In GP (Figure 7) there are generations where the average population fitness has a negative value. This situation doesn't happen with DGP (Figure 6) because of the genetic variation caused by remote mutation, when one robot receives a part of the best individual program from another robot.

The main difference between classical GP algorithm and DGP is the size of population. Classical GP algorithm needs a population with a great number of individuals to converge for a solution. In DGP the population in each robot is small and the evolutionary process is faster than GP.

A disadvantage of GP as to other EC [15] techniques is that the set of functions and terminals developed for a particular problem may not apply to other kinds of problems. Is important that the set of functions and terminals must be

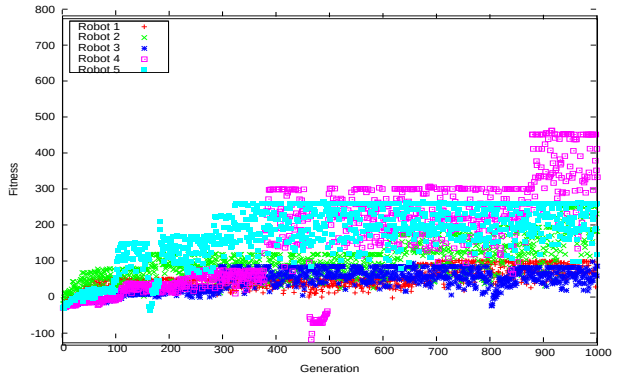

Figure 7: Average population fitness at each generation with GP.

designed to be as comprehensive as possible, that is, with few modifications or even none, in such a way that it is possible to use the same functions and terminals in many kinds of problems, only redefining the evaluation function.

5. CONCLUSIONS

We described an Evolutionary Control System (ECS) able to control a population of mobile robots. The ECS has two main modules: the first one, called EMSS (Execution, Management and Supervision System), is the system responsible for managing all the evolutionary process that takes place in an embedded fashion in each robot. The second module, called DGP (Distributed Genetic Programming), is an extension of classical Genetic Programming algorithm to support the control system evolution for the robots that are part of the mobile robots population.

To validate DGP algorithm one experiment in Khepera Simulator was accomplished. The experiment was a foraging task, in which a group of five mobile robots must navigate through an unknown environment avoiding obstacles and looking for food. The results obtained with DGP was compared with results obtained with classical GP algorithm.

The results show that DGP performance is better than that of classical GP. Remote mutation, caused by local individual program transfer from robots to robots, helps to increase the variability of the local population at each robot. For this reason, the average fitness value of each individual at each robot with DGP algorithm is higher than that of the classical GP algorithm.

Future works include experiments with ECS in other problems in mobile robotics, such that, box pushing, and predator and prey. With these experiments we intend to evaluate how easy it is to change the task of robots without reprogramming the whole control system. These experiments will be accomplished using a simulator and real robots. The Eyebot [2] robots will be used for the experiments. Eyebot is a kind of mobile robot developed to be used in robot soccer.

6. ACKNOWLEDGMENTS

The author, Anderson Luiz Fernandes Perez, acknowledge the CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico), Brazil, for the scholarship.

7. REFERENCES

[1] Broodier, Markus and Banzhaf, Wolfgang. Linear Genetic Programming. Springer (2006).

[2] Brunl, Thomas: *EyeBot Online Documentation*. http://robotics.ee.uwa.edu.au/eyebot/ (2006).

[3] Ficici, S. and Watson, R. and Pollack, J. Embodied Evolution: A Response to Challenges in Evolutionary Robotics. J. L. Wyatt and J. Demiris, *Eighth European Workshop on Learning Robots* (1999) 14–22.

[4] Floreano, Dario and Nolfi, Stefano. Adaptive Behavior in Competing Co-Evolving Species. *Fourth European Conference on Artificial Life*. The MIT Press, Cambridge, MA, Phil Husbands and Inman Harvey (1997) 378–387.

[5] Fogel, D.B.: What is Evolutionary Computation? *IEEE Spectrum* 37:2 (2000) 28–32.

[6] Harvey, Inman. Artificial Evolution: A Continuing SAGA In Evolutionary Robotics: From Intelligent Robots to Artificial Life. *8th International Symposium on Evolutionary Robotics (ER2001)*. Springer-Verlag Lecture Notes in Computer Science LNCS 2217 (2001).

[7] Koza, John R. Genetic Programming: A Paradigm for Genetically Breeding Computer Population of Computer Programs to Solve Problems. MIT Press (1992).

[8] Michel, O., *Khepera Simulator Package version 2.0: Freeware Mobile Robot Simulator written by Oliver Michel.* http://www.epfl.ch/lami/team/michel/khep-sim/ (1996).

[9] Nolfi, Stefano and Floreano, Dario: Evolutionary Robotics. The Biology, Intelligence, and Technology of Self-Organizing Machines. MIT Press (2001).

[10] Nolfi, S. and Floreano, D. Synthesis of autonomous robots through evolution. *Trends in Cognitive Science* 6:1 (2002) 31–36.

[11] Nordin, Peter Nordin and Banzhaf, Wolfgang. An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming. *Adaptive Behaviour* 5:2 (1997) 107–140.

[12] Pollack, Jordan B. and Lipson, Hod and Ficci, Sevan and Funes, Pablo and Hornby, Greg and Watson, Richard A. Evolutionary Techniques in Physical Robotics. *ICES* (2000) 175–186.

[13] Simões, Eduardo D. V. and Dimond, Keith R. Embedding a Distributed Evolutionary System into Population of Autonomous Mobile Robots. *Proceeding of The 2001 IEEE System, Man, and Cybernetics Conference* (2001).

[14] Watson, R. and Ficici, S. and Pollack, J. Embodied Evolution: Distributing an Evolutionary Algorithm in a Population of Robots. *Robotics and Autonomous Systems* 39:1 (2002) 1–18.

[15] Whitley, Darrell. An overview of evolutionary algorithms: practical issues and common pitfalls. *Information and Software Technology* 43:14 (2001) 817–831.

[16] Zhang, Byoung-Tak and Cho, Dong-Yeon. Evolving Complex Group Behaviors Using Genetic Programming with Fitness Switching. *Artificial Life and Robotics* 4:2 (2000) 103–108.