2nd International Conference on Autonomous Robots and Agents
December 13-15, 2004 Palmerston North, New Zealand

# Mapping of unknown environments using genetically evolved behaviour for a team of robots

Vivek Srikumar, Prashanth Ellina
S.R.M. Engineering College, Anna University
Chennai
{viveksrikumar, prashanthellina} @gmail.com

## Abstract

A method for mapping of unknown terrains using a group of robots is proposed. The behaviour of an individual robot is optimized using genetic algorithms in a simulated environment. Specific behaviour patterns are evolved for known terrain types. When the team is employed to map a real world environment, a terrain template is chosen and the corresponding behaviour is adopted. As the system learns more about the environment, the behaviour might be changed to a more optimum one.

**Keywords:** Evolutionary Robotics, Genetic Algorithms, Distributed Robotics, Mapping

## 1 Introduction

Mapping of unstructured environment has potential uses in varied fields like surveillance and coalmine exploration. This problem can be solved using a single robot that traverses the unknown environment. However, distributed robotics has proven to be an effective solution for varied known reasons like improved fault tolerance and robustness

The limitation of such an implementation is that the rules have to be explicitly coded by the designer. This leaves no scope for adaptation to changing environments. One way to approach this is to allow the robots to learn their behaviour based on the environment using genetic algorithms. The behaviour might be learnt during simulation and the learnt behaviour is used when the robots are used in real terrains.

In this paper, terrains are classified based on their statistical properties. For example, a terrain that has the obstructions at right angles to each other is different from the one that has them randomly and the same behaviour need not be optimum for both the terrains. It is proposed that simulation be used to evolve the optimum behaviour for each type of terrain. Such behaviour patterns could be generated for different terrain types. When the team maps a real environment, it is initially loaded with the behaviour of a generic terrain and as more information about the terrain is obtained, pattern matching is performed and specific behaviour is loaded into the robots.

The technique proposed is based on the assumption that the lower level implementations of several components are available. This is especially true of the hardware of the robots; many robot systems have been designed with the aim of optimizing the hardware functioning. This paper mainly deals with the behaviour of such machines rather than the actual working of the hardware components.

## 2 Related work

Several groups are working on the use of multiple agents to perform mapping at various levels of detail. Many implementations of distributed robotics have been successful. Millibots [1] developed at the Robotics Institute of Carnegie Mellon University is a notable example. This implementation provides a method of localization of distributed robot teams.

Distributed Robotics is also seen as a subset of swarm robotics. Agents are seen as part of a robot swarm, with individual capacities being very limited. This approach is motivated by observation of swarms in real life (ants, bees, etc). There are three advantages that can be gained by using swarms of robots – the increased number of agents, which is generally seen in swarms, leads to better efficiency, the design of each individual agent is simplified and the direct communication between agents is not necessary. A system of swarm robots has been designed at the University of Wales [2].

Some advances have been made in the field of Genetic learning for robotics. Use of machine learning for developing the control systems for autonomous robots is generally done in simulation [3]. Instead of assuming ideal conditions during simulation, artificial evolution using noise to model dissipative forces has been done [4]. Artificial machine learning is also discussed in [5] [6] [7].

## 3 System Components

The system to be described here can be divided into two primary parts – the robots and the central computer.

### 3.1 The Robot

The system design is largely kept independent of the design of the individual robot. This allows for further extensions, if necessary. However, it is necessary all the robots that form part of the system satisfy certain criteria, namely

1. Each robot must be mobile, the motion controlled by a microcontroller
2. The robots must have some way of sensing their environment
3. There must be network for communication of between the elements of the system
4. There must be some system that allows the positions of the robot to be ascertained to reasonable accuracy as in [1]

In addition to these criteria, in the following discussion, a few more assumptions are made for ease of characterization. The robot is assumed to be a wheeled differential drive robot. The microcontroller only supplies two signals for each driving motor- 1 indicates forward direction and 0 indicates reverse. It is assumed that the robots have two tactile sensors – one each in left and right. The output of the sensor is a two bit word – the left sensor information and the right sensor information. The description of a simple robot is provided by K.A.Hawick, et al [2].

When multiple agents are at work concurrently, the issue of communication plays an important role in determining the efficiency of the system. Instead of specifying a model for communication for this system, it is suggested that an existing network protocol stack be used. For example, Bluetooth technology can be used for establishing communication between the robot and the central computer. Systems that use IEEE802.11b wireless Ethernet have also been designed [2].

### 3.2 The Robot Code

The motion of the robot is controlled by the "robot code". The robot code is a finite automaton whose states are Front, Back, Left and Right. The states of the robot code reflect the most recent action that was performed by the robot. While Front and Back indicate translation of the robot, Right and Left indicate that the robot has rotated itself about a predefined axis of rotation.

The inputs to the robot code are the inputs from the sensors. For each input the robot code specifies the motion of the robot by listing out a sequence of actions to be performed. This sequence is defined as the "action sequence". For example, the robot may be in state Front, indicating that the most recent action it performed was to move forward. In this state, if the sensors indicate that there is an obstacle on the left and none on the right, the input is 10. For this input, the robot code may have the action sequence BRFL. This indicates that the robot attempts to move back, then right, front and left – in that order. The action sequence is loaded into the robot's buffers and the actions are executed in succession. The buffers shall be referred to as the robot's "action queue". As the actions are performed, the states also correspondingly changed. If, however, the sensors indicate an obstacle in while an action sequence is being processed, then the entire action sequence is flushed and a new action sequence is loaded for the current state and input.

It must be noted that though the above discussion uses input from a tactile sensor, other sensors can be used with equal or more efficiency by making their outputs take discrete values.

The robot code defines the behaviour of the system of robots. If the behaviour needs to be changed, then it is enough to change the robot code.

### 3.3 The Central Computer

The central computer is primarily used as a repository for storing the map of the environment as it is obtained. Each robot must possess a method for communication with the central computer. The robot specifies its position with respect to some fixed origin.

When the computer receives the position of the robot, it first checks to see if the position has already been mapped. If so, the robot is in an area that was previously visited by a robot. In this case, the computer generates a path from the robots current position to a point that is nearest to it and is just beyond the boundary of the map. The generation of path can be done using Djikistra's Algorithm. This path is given the form of an action sequence and is sent to the robot. The robot performs the actions specified in the action sequence and reaches a point that is yet unmapped. This maneuver ensures that the robots do not map regions that have already been explored.

If the position of the robot has not been mapped already, the computer notes this position and the map expands.

Whenever the map expands, the computer also tries to match the current map with one of the terrain types. The matching is done using a Counter propagation Neural Network (CPN) [8]. Since the CPN is ideally suited for classifying information, it can be used to classify the map among the terrain types, even if insufficient information is available. If the terrain type that is identified is different from the one in use, then its robot code is downloaded to all the robots.

Each terrain type is identified by two aspects – the statistical properties that define the terrain and the robot code that should be used when it has been detected. When the robot system is launched, the

robot code for a terrain must be optimal for that terrain type. This means that the optimality of the entire mapping will be improved.

## 4 Generation of Robot Code

Before the system can be used, the optimal robot code for each terrain type has to be generated. This is done by simulating the system and use of evolutionary learning to obtain the optimal robot code for each terrain type.

### 4.1 Terrain Classification

A terrain type is just a set of constraints upon the distribution, location and orientation of obstacles in the environment. Instead of naming terrain types, they can be specified by representing the above constraints.

For example, consider a terrain class that has all obstacles being straight walls of varying sizes. Then the following criteria can be chosen for classification of terrains

1. Number of obstacles
2. Size of obstacles
3. Distance between obstacles
4. Angle between obstacles

Each criterion could be divided into three sets as shown in Table 1.

**Table 1:** Example for classification criteria

| Criteria | Range | | |
|---|---|---|---|
| | **Set 1** | **Set 2** | **Set 3** |
| Number of obstacles | <10 | <50 | >50 |
| Size of obstacles | <10 units | <40 units | >40 units |
| Distance between obstacles | <50 units | <150 units | >150 units |
| Angle between obstacles | <90 degrees | 90 degrees | >90 degrees |

These numbers have been chosen arbitrarily. For a real system, they must be chosen with the actual expected terrains in mind. With these sets, all terrains can be classified among $3^4$ different types based on the features. It must be noted, however, that all features of the terrain need not conform all the rules of a terrain type. If a majority of features are part of one type, then the terrain itself is part of that type.

### 4.2 Simulation for genetic optimization

Simulation of the system is done with a few assumptions to simplify the process. It is assumed that the terrain is ideally smooth and dissipative effects like friction are not modeled. The communication between the robots and the computer is also assumed to be ideal, i.e. there is no loss of information and delays. In addition, all movements are discrete, that is, robots can be present only at discrete co-ordinates.

In order to compensate for the inefficiencies that may result due to these assumptions, noise is introduced while performing simulation [3]. The simulation process is done on the basis of the algorithm shown in Figure 1.

The gene for each system is its robot code. The aforementioned algorithm finds the best robot code for each terrain type. For each terrain type, it generates a set of robot codes that form the population. These are initialized randomly. Then the genetic optimization of that population commences and continues till a sufficient level of fitness is obtained. For each generation, a new terrain that satisfies the constraints of the current terrain type is generated. Each robot code of the population is allowed to map the terrain independently. The fittest robot code is chosen and mitosis is used to generate the next generation.

```
for each terrain type do
{
        generate a new population of Robot Codes
        while fitness of population < Epsilon do
        {
                generate a new terrain of this type
                allow each robot code to map the terrain independently
                decide the best robot code
                mutate the fittest robot code to generate the new population
        }
        set the fittest robot code as the robot code for this terrain type
}
```

**Figure 1.** Algorithm for simulation

### 4.3 Selection Benchmarks for Fitness

The fitness of the robot codes is decided by comparing the generated terrain type with the map generated by the corresponding system. Fitness can be described as a function of the area mapped, accuracy of mapping and the time taken to perform the mapping.

Since early generations can get stuck in infinite loops, the time to perform mapping is fixed and the robot system is allowed to map the terrain within the fixed time. In such a case, the area mapped and the accuracy are the benchmarks to determine fitness. The area covered can be conveniently expressed as a ratio of the area of the map formed to the terrain area, which is fixed.

*Area ratio = Area mapped / Area of the terrain*

(1)

Accuracy can be expressed as follows

*Accuracy = 1 – Number of error locations/Area of terrain*

(2)

where a location is said to be an error location when the map shows an obstacle where there is none in the terrain or vice versa.

For a robot code to be considered fit, it must have a high value of both area ratio and accuracy. Thus the fitness can be defined as

*Fitness = Area ratio * Accuracy* (3)

The evolutionary mechanism attempts to maximize this fitness within a specified period of time to obtain the ideal robot code. The fitness of the population is the fitness of the fittest individual in it. The genetic optimization continues till the fitness of the population is less than some predetermined value, denoted by Epsilon.

## 5 Putting it all together

This section consolidates all the abovementioned components of the system into one unit. To start off, the types of terrains that the robots might encounter have to be identified. This need not be a completely deterministic characterization because the environment that the robots will map is unknown.

Once classification is done, the optimal robot code for each terrain type has to be learnt. This is done by simulating the system and genetic learning. Another step that must be performed after the terrains have been classified is the training of the CPN. The CPN must be trained using its training algorithms till it can identify the terrain type if it is given a terrain as input.

After sufficient learning, the robots can be used in real terrains. Initially, the robots are loaded with the robot code corresponding to a generic terrain type. For example, the robot code for an empty terrain can be loaded. As the robots learn more about the environment, the map expands. The central computer performs pattern matching and tries to ascertain the closest matching terrain type to the current map. If the match is different from the one that is in use, then the new robot code must be downloaded into all the robots and the action sequence buffers in the robots must be flushed. When the robot gets a new robot code, it loads a new action sequence into its memory based on the current state and the current input.

When the robot code that is obtained by simulation is transferred to the real robots, there is bound to be a drop in performance with respect to the one expected by simulation. This can be attributed to the simplifications that were used during simulation. The loss of performance can be compensated by two ways - by the introduction of “noise” during simulation, and by continuing evolution in the real environment for some generations. This has been dealt with by Miglino et al [3].

### 5.1 Proof of optimality

Essentially, the genetic algorithm tries to optimize the following quantity, defined as efficiency of mapping

$$\eta = \delta A/T \quad (4)$$

where $\delta$ is the accuracy, A the area ratio and T the time taken for mapping.

To prove that the method of classifying terrains generates more optimal robot codes on an average than using a generic robot code, consider the efficiency of mapping for the generic robot code

$$\eta_g = \delta_g A/T_g \quad (5)$$

If the terrains are numbered 1 to n, then the corresponding efficiencies, when the robots are in that terrain, are defined similarly to be $\eta_1$ through $\eta n$. Let the probability that the current terrain is of type i be $p_i$.

Since the quantity $\eta$ is the dependent on the fitness of the corresponding robot code,

$$\eta_i \geq \eta_g \quad (6)$$

$$\Sigma p_i \eta_i \geq \eta_g \quad (7)$$

The left side of the inequality is the average efficiency for the system when terrain classification is performed while the right side is the efficiency without terrain classification. This means that the average performance of the system, when terrain classification is performed is better than the performance of the system without classification of terrains.

## 6 Further improvements

The system described above can be improved to be a fully distributed network of robots. In its current form, the system experiences a bottleneck due to the presence of a central computer to store the map. Furthermore, the use of a central computer implies that any failure of the central computer will lead to failure of the entire system. The presence of a centralized command can be replaced by distributing the map across the robots themselves. Any decisions involving the map will also be taken by the robots. However, the complexity of such a system is very high as the pattern recognition to determine the terrain type has to be done by a distributed system.

Another improvement in the system can be brought about by the use of heterogeneous robots. Different robot codes can be used instead of having all robots use the same robot code. Another type of heterogeneity can be implemented by using different hardware configurations in the same group of robots. This could lead to a situation where certain robots can perform tasks specialized to their hardware configuration.

## 7 Conclusion

Genetic optimization is a powerful technique to converge upon efficient behaviour. Instead of explicit programming by humans, the group of robots learns its behaviour. Learning is done at two levels; first during the training by simulation and then in real environments when the terrain type is identified. This entails that the job of the designer is considerably reduced as every possible sequence of actions need not be considered.

The system described above has many applications, most of which deal with unknown terrain navigation and mapping. For example, a group of such robots can be used to navigate across a landmine field to detect the mines. Loss of some robots will not affect the group in general and mapping can proceed even if a robot is destroyed. Other applications include surveillance and planetary exploration.

## 8 References

[1] Navarro-Serment, L.E., Paredis, C.J.J., Khosla, P.K., “A Beacon System for the Localization of Distributed Robotic Teams”, *Proceedings of the International Conference on Field and Service Robotics, Pittsburgh, PA,* (1999).

[2] Hawick,K.A., James, H.A., Story, J.E. and Shepherd, R.G., "An Architecture for Swarm Robots", *Technical Note DHPC-121*, (2002).

[3] Miglino, O., Lund, H. H., and Nolfi S., “Evolving Mobile Robots in Simulated and Real Environments” *Artificial Life,* Pp 417—434 (1996).

[4] Jakobi, N., Husbands, P., and Harvey, I., “Noise and the reality gap: The use of simulation in evolutionary robotics”, *Advances in Artificial Life: Proc., 3rd European Conference on Artificial Life,* pp 704—720, (1995).

[5] Koza, J.R. and Rice, J.P. “Automatic Programming of Robots using Genetic Programming” *AAAI-92*, pp 194-207 (1992).

[6] Dorigo, M, Schnepf, U, “Genetics-based Machine Learning and Behaviour Based Robotics: A New Synthesis”, *IEEE Transactions on Systems, Man, and Cybernetics,* Pp 141–154 (1993).

[7] Ram, A., Arkin, R., Boone, G., and Pearce, M., “Using Genetic Algorithms to Learn Reactive Control Parameters for Autonomous Robotic Navigation”, *Adaptive Behaviour, volume 2, issue 3*, pp 227-304, (1994).

[8] Freeman, J.A. and Skapura, D.M. *Neural Networks Algorithms, Applications and Programming Techniques*, Pearson Education, (Chapter 6, pp 213-262, 2003)

[9] Feng, L., Borenstein, J., Wehe, D., “A Completely Wireless Development System for Mobile Robots”, *ISRAM Conference*, Montpellier, France, Pp 571-576 (1996).

[10] Borenstein, J., Everett, H. R., Feng, L., and Wehe, D., “Mobile Robot Positioning & Sensors and Techniques”, *Journal of Robotic Systems, Special Issue on Mobile Robots, Vol. 14 No. 4*, pp 231 – 249 (1997).

[11] Tangamchit, P., Dolan, J., and Khosla, P., “Crucial Factors Affecting Cooperative Multirobot Learning”, *IEEE/RSJ International Conference on Intelligent Robots and Systems 2003* (2003)

[12] Ye, C. and Borenstein, J., "A new terrain mapping method for mobile robots obstacle negotiation." *Proceedings of the UGV Technology Conference at the 2003 SPIE AeroSense Symposium, Orlando, FL* (2003).

[13] Kim, J. Pearce,R.A. and Amato, N.M. "Multiple Robot Navigation and Localization Using Sonar Sensors in an Indoor Environment", *Technical Report, TR01-004, Texas A&M University* (2001).