Evolving a Multiagent System for Landmark-based Robot Navigation

Madhur Ambastha[†], **Dídac Busquets**[§], **Ramon López de Màntaras**[§], **Carles Sierra**[§]

[†]Department of Computer Science
University of Rochester
Rochester, NY 14627, USA
ambastha@cs.rochester.edu

[§]Artificial Intelligence Research Institute (IIIA)
Spanish Council for Scientific Research (CSIC)
Campus UAB, 08193 Bellaterra, Spain
{didac,mantaras,sierra}@iiia.csic.es

Abstract In this paper, we build upon a multiagent architecture for landmark based navigation in unknown environments. In this architecture, each of the agents in the navigation system has a bidding function that is controlled by a set of parameters. We show here the good results obtained by an evolutionary approach that tunes the parameter set values for two navigation tasks.

Keywords: Multiagent systems, Robotics, Evolultive computation

1 Introduction

In landmark-based navigation, the robot must be able to start in an unknown location and navigate to a desired target using visually-acquired landmarks. The specific scenario that we are studying assumes that there is a target landmark that the robot is able to recognize visually. The target is visible from the robot's initial location, but it may subsequently be occluded by intervening objects. The challenge for the robot is to acquire enough information about the environment (locations of landmarks and obstacles) so that it can move along a path from the starting location to the target position. The robot should do this quickly but safely.

We have proposed a bidding coordination architecture to accomplish this objective [6]. This architecture is composed of three systems: the *Pilot* system, the *Vision* system and the *Navigation* system. Each system competes for the two available resources: motion control (direction of movement) and camera control (direction of gaze). The three systems have the following responsibilities. The Pilot is responsible for all motions of the robot. It selects these motions in order to carry out commands from the Navigation system and, independently, to avoid obstacles. The Vision system is responsible for identifying and tracking landmarks (including the target landmark). Finally, the Navigation system is responsible for choosing higher-level decisions in order to move the robot to a specified target. This requires requesting the Vision system to identify and track landmarks (in order to build a map of the environment) and requesting the Pilot

to move the robot in various directions in order to reach the goal position or some intermediate target.

From the brief description of the robot architecture given above, it can be observed that the three systems must *cooperate* and *compete*. They must cooperate because they need one another in order to achieve the overall task of reaching the target position. But at the same time they are competing for motion and camera control.

The Navigation system is implemented as a multiagent system, where each agent is competent in a specific task. Depending on its responsibilities and the information received from other agents, each agent proposes which action the Navigation system should take. Again, we find that the agents must cooperate, since an isolated agent is not capable of moving the robot to the target, but they also compete, because different agents want to perform different actions.

For both the overall robot system and the Navigation system, we have proposed the use of a new competitive coordination system based on a *bidding mechanism*. In the overall robot system, the Navigation and the Pilot systems generate bids for the services offered by the Pilot and Vision systems. These services are to move the robot toward a given direction, and to move the camera and identify the landmarks found on its view-field, respectively. The service actually executed by each system depends on the winning bid at each point in time. Similarly, in the Navigation system, each agent bids for the action it wants the robot to perform. These bids are sent to a special agent that gathers all bids and determines the winning action. The selected action is then sent as the Navigation system's bid for the services of the Vision and Pilot systems.

The bidding functions of each of the agents in the Navigation system are controlled by a set of parameters. These parameters need to be tuned in order to achieve the best performance of the Navigation system and of the overall system. Adjusting these parameters manually can be very difficult, particularly because of the tradeoffs confronting the top-level agents. An alternative to manual tuning is to employ an evolutionary approach to tune them. This paper describes this approach.

The paper is organized as follows. Section 2 is devoted to relevant related work. The multiagent architecture of the Navigation system is described in Section 3. Section 4 describes each one of the agents and their bidding parametric functions. Section 5 describes the evolutionary approach to tune these functions. Finally, the experimental results are discussed in Section 6.

2 Related work

In the last years it has been mainly focused on Behavior-based architectures [2]. The most representative of such architectures are Brook's subsumption architecture [5], Maes' action selection [13] and Arkin's motor schema [3]. Since then, many other architectures have been proposed. Liscano et al [10] use an activity-based blackboard consisting of two hierarchical layers for strategic and reactive reasoning. A blackboard database keeps track of the state of the world and a set of activities to perform the navigation. Arbitration between competing activities is accomplished by a set of rules that decides which activity takes control of the robot and resolves conflicts. Other hierarchical centralized architectures similar to that of Liscano et al are those of Stentz [18] to drive CMU's Navlab and Isik [11] among others. Our approach is com-

pletely decentralized which means that the broadcast of information is not hierarchical. This approach is easier to program and is more flexible and extensible than centralized approaches. Arkin [3] also emphasized the importance of a non-hierarchical broadcast of information. Furthermore, we propose a model for cooperation and competition between activities based on a simple bidding mechanism. A similar model was proposed by Rosenblatt [16] in the CMU's DAMN project. A set of modules cooperated to control a robot's path by voting for various possible actions, and an arbiter decided which was the action to be performed. However, the set of actions was pre-defined, while in our system each agent can bid for any action it wants to perform. Moreover, in the experiments carried out with this system (DAMN), the navigation system used a grid-based map and did not use at all landmark based navigation. Also at CMU, the FIRE project [7] uses a market-oriented approach to model the co-operation of a team of robots. Sun and Sessions [19] have also proposed an approach for developing a multiagent reinforcement learning system that uses a bidding mechanism to learn complex tasks. The bidding is used to decide which agent gets the control of the learning process. The agents bid according to the expected reward that would receive if they were given the control. Thus, although they are competing for the control, they also cooperate, since they seek to maximize the overall system reward.

The map building approach we use is based on the work by Prescott [15], who proposed a network model that stores the spatial relationships among landmarks for robot navigation. By matching a perceived landmark with the network, the robot can find its way to a target, provided it is represented in the network. While Prescott's approach is quantitative, ours uses a fuzzy extension of his model to work with fuzzy qualitative information about distances and directions. Levitt and Lawton [12] also proposed a qualitative approach to the navigation problem, but assume unrealistically accurate distance and direction information between the robot and the landmarks. Another qualitative method for robot navigation was proposed by Escrig and Toledo [9], using constraint logic. However, they assume the robot has some a priori knowledge of the spatial relationship of the landmarks, whereas we build these relationships whilst exploring the environment.

There is a vast literature on evolutionary approaches to parameter optimization. For this reason we will not single out any particular work. Nonetheless, an application of genetic algorithms to a similar problem on path planning was done in [17] where the low-level parameters tuned correspond to an insect-inspired pheromone based model defining a potential field over the space, whereas our approach is based on a group of deliberative agents. Also in [1] an evolutionary approach to the generation of an optimal colony of robots is presented.

3 The multiagent architecture

The architecture is composed of three systems (see Figure 1). Each system competes for two available resources: motion and vision. The Pilot is responsible for all motions of the robot. It selects these motions to carry out commands from the Navigation system and (independently) to avoid obstacles. The Vision system is responsible for identifying and tracking landmarks (including the goal). Finally, the Navigation system is responsible for choosing higher-level robot motions to move the robot to a specified goal. This requires requesting the Vision system

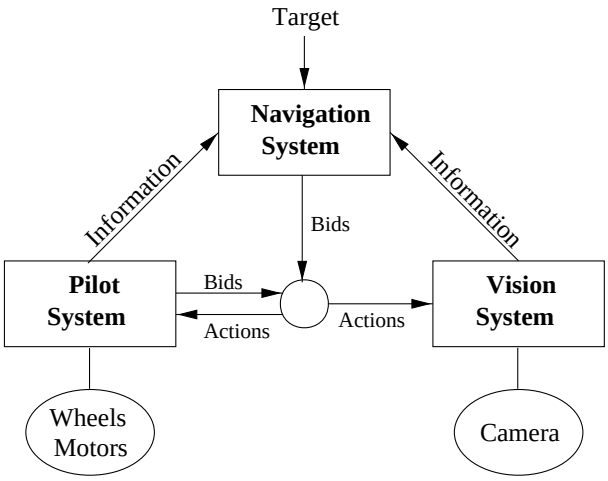

Figure 1: *Left:* Robot architecture

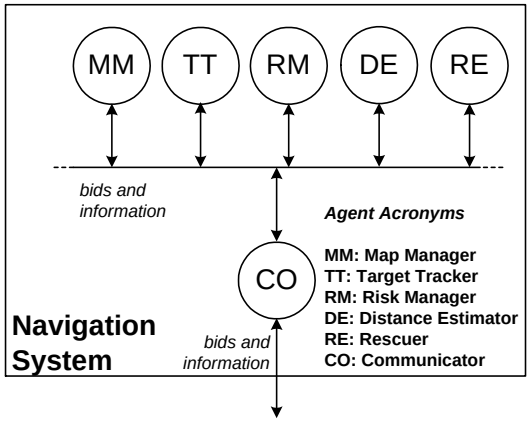

Figure 2: *Multiagent Navigation System*

to identify and track landmarks, to build a map of the environment, and requesting the Pilot to move the robot toward the goal position or toward some intermediate target position.

From this brief description, two observations can be made. First, these three systems must cooperate to achieve the overall task of reaching the goal landmark position. For instance, the Pilot needs the Vision system to identify obstacles, and it needs the Navigation system to select a path to the goal. Second, the systems are also competing —there are some tradeoffs between them. For example, both the Pilot and the Navigation system compete for the Vision system. The Pilot needs vision for obstacle avoidance, while the Navigation system needs vision for landmark detection and tracking.

To manage this cooperation and competition, we use a bidding mechanism. Each system generates bids for the services offered by the Pilot and Vision systems. The service actually executed by each system depends on the winning bid at each point in time.

The Navigation system itself is implemented as a multiagent system (see Figure 2). This system is composed of six agents with the following responsibilities:

- keep the target located with maximum precision and reach it (*Target Tracker*),
- keep the risk of losing the target low (*Risk Manager*),
- recover from blocked situations (*Rescuer*),
- keep the error in the distance to landmarks low (*Distance Estimator*),
- and keep the information on the map consistent and up-to-date (*Map Manager*)

There is an additional agent, *Communicator*, which manages the communication between the Navigation system with the other systems. As with the overall system, the Navigation system employs a bidding mechanism to coordinate these agents. Each agent bids for the action it wants the robot to perform. These bids are sent to the *Communicator* agent, which determines the winning action. The selected action is then sent as the Navigation system's bid for the services

of the Vision and Pilot systems. Each action can involve a combination of requests to the Vision and the Pilot systems. The resulting bids coming from the agents depend on bidding functions associated to each agent. These functions depend on the values of different sets of parameters, which affect the overall performance of the Navigation system. Since a manual adjustment is extremely difficult we propose to employ a genetic algorithm to find optimal sets of values. Next section describes the bidding functions in detail and the rest of the paper is devoted to describe this evolutionary approach and the results of our experiments.

For map representation and wayfinding, we have extended Prescott's beta-coefficients system [15]. Prescott's model stores the relationships among the landmarks in the environment to build a map. The location of a landmark is encoded based on the relative locations (headings and distances) of three other landmarks. This relationship is unique and invariant to viewpoint. Once this relationship has been stored, the location of each landmark can be computed from the locations of the three landmarks encoding it, no matter where the robot is located as long as the robot can compute the heading and distance to each of the three landmarks.

As the robot explores the environment, it stores the relationships among the landmarks it sees. This creates a network of relationships among the landmarks in the environment. If this network is sufficiently-richly connected, it provides a computational map of the environment. Given the headings and distances to a subset of currently-visible landmarks, the network allows to compute the locations of all landmarks, even if they are currently not visible.

Prescott's model assumes that the robot is able to measure the exact location of the landmarks. But this is not the case in our robot: the vision system gives only imprecise information about the location of the landmarks, and we cannot rely on the odometry of the robot, as it is also imprecise. To deal with this imprecision, our extended model represents all the network coordinates as fuzzy numbers and carries out all map computations using fuzzy arithmetic [4]. The focus of this paper is on the evolutionary approach to tune the agents' bidding behaviour. For this reason, from now on, we will skip the details of the map representation (see [6] for details).

4 The Agents

The Navigation system is decomposed into six different agents that are responsible for different tasks, which when coordinated by the *Communicator* provide the desired effect of leading the robot to a desired target. As mentioned before, each agent has certain parameters which affect its bidding behaviour. The agents and their parameters are described next.

*4.1 Map Manager (**Parameters** : none)*

This agent is responsible for maintaining the information of the explored environment as a map. Since the Map Manager does not bid, there are no parameters to tune and therefore it is not on the focus of this paper. The details of the map management algorithmics are also not given here due to space limitations (see [6] for details).

4.2 *Target Tracker (**Parameters**: $\alpha, \beta, \kappa_1, \kappa_2$)*

The goal of this agent is to keep the target located at any time. The imprecision I_a associated with the location of the target is computed as a function on the size of the angle arc, ϵ_α calculated from the robot's current position to where the target is thought to be located, and the agent acts to keep the imprecision as low as possible. The bids for moving towards the target start at the value κ_1 and decrease polinomically to 0, depending on the parameter α. The rationale of this is that when the imprecision about the target location is low, this agent is confident about the target position and therefore bids high to move towards the target. As the imprecision increases, this confidence decreases and so does the bid. Bids for looking at the target increase from 0 to a maximum of κ_2 and then decrease again to 0. The rationale behind this is that when the imprecision is low there is no urgency in looking to the target as its location is known with high precision. This urgency starts to increase as the imprecision increases. When the imprecision reaches a level in which the agent has no confidence on the target location it starts decreasing the bid so as to give the opportunity to better informed agents to win the bid. The equations involved are :

$$I_a = (\epsilon_\alpha / 2\pi)^\beta$$
$$bid(move(\epsilon_\alpha)) = \kappa_1(1 - I_a^{1/\alpha})$$
$$bid(look(\epsilon_\alpha)) = \kappa_2 \sin(\pi I_a)$$

where β controls the shape of the imprecision function.

4.3 *Distance Estimator (**Parameters** : κ, ϕ, δ)*

The goal of this agent is to keep the distance error to the target landmark as low as possible. This agent plays a very important role at the beginning of the navigation. When analysing the first viewframe to obtain the initial landmarks, the error in distance is maximal, there is no reference view to obtain an initial estimation of the distance to the target. This agent generates high bids to move orthogonally with respect to the line connecting the robot and the target in order to get another view on it and establish an initial estimation of the distances to the target. Similarly, when a target switch is produced (by the intervention of the Rescuer) this agent may become relevant again if the distance value to the new selected target is very imprecise. Again, the same process will have as consequence a decrease in the new target distance error.

We model distance imprecision as the size of the support of the fuzzy number modeling distance. We note ϵ_t the imprecision error to the current target. Thus, the imprecision in distance to the target can be modeled as $I_d = 1 - 1/e^{\kappa\epsilon_t}$ where κ is a parameter that changes the shape of I_d; high values of κ gives faster increasing shapes. At the beginning of a run the $distance$ is the fuzzy number $[0, \infty]$, $\epsilon_t = +\infty$ and hence $I_d = 1$.

This agent is relevant when the imprecision is very high. Its action is to bid to move the robot in an orthogonal direction using as bid the value of I_d, that is:

$$bid\left(move\left(\epsilon_\alpha + \frac{\pi}{2}\right)\right) = I_d$$

This agent is also responsible for deciding (up to a certainty degree ϕ) whether the robot is *at target*. It considers that the robot has reached the target if the upper bound of the α-cut of level ϕ of the fuzzy number modeling the distance to the target is less than δ times the body size of the robot.

*4.4 Risk Manager (**Parameters** : $\gamma_A, \gamma_B, \gamma_r$)*

The goal of this agent is to keep the risk of losing the target as low as possible. To do so, it tries to keep a reasonable amount of landmarks, as non collinear as possible, in the surroundings of the robot. The less landmarks around, the more risky is the current situation and the higher the probability of losing the target. Also, the more collinear the landmarks the higher the error in the location of the target and thus the higher the imprecision on its location.

We model the risk as a function that combines: 1) the number of landmarks ahead (elements in set A), 2) the number of landmarks around (elements in set B), and 3) their "quality"(q_A and q_B). These qualities are computed by the *Map Manager*. A minimum risk of 0 is assessed when there are at least four visible landmarks in the direction of the movement and minimally collinear. A maximum risk of 1 is assessed when there are no landmarks ahead nor around:

$$R = 1 - \min\left(1, q_A\left(\frac{|A|}{4}\right)^{\gamma_A} + q_B\left(\frac{|B|}{4}\right)^{\gamma_B}\right)$$

The values γ_A and γ_B determine the relative importance of the position of landmarks (ahead or around).

Given that the robot cannot decrease the collinearity of the landmarks, the only way to decrease the risk level is by increasing the number of landmarks. We privilege the fact of having landmarks ahead by bidding

$$bid\left(look\left(random\left(\left[-\frac{\pi}{4}, +\frac{\pi}{4}\right]\right)\right)\right) = \gamma_r \cdot R$$

for the action of looking at a random direction in front of the robot and tyring to identify the landmarks in that area, and

$$bid\left(look\left(random\left(\left[+\frac{\pi}{4}, +\frac{7\pi}{4}\right]\right)\right)\right) = \gamma_r \cdot R^2$$

(which is obviously smaller than $\gamma_r \cdot R$) for the action of looking at a random direction around the robot and trying to identify landmarks, where γ_r is a parameter to control the maximum value of the bidding function.

*4.5 Rescuer (**Parameters** : $\overline{I}_a, \overline{R}$)*

The goal of the Rescuer agent is to rescue the robot from problematic situations. These situations may happen due to three reasons. First, the pilot can lead the robot to a position with an obstacle ahead. Second, the imprecision of the location of the target (see Section 4.2) is over the threshold

$\overline{I}_a$. Finally, the robot can be at a very risky place, that is a place where the risk to get lost (see Section 4.4) is over a threshold $\overline{R}$. If any of these situations happen, the rescuer agent asks the Map Manager for a diverting target and communicates it to the other agents. The algorithm uses a stack where the different diverting targets are stacked, to avoid repeating them.

*4.6 Communicator (**Parameters** : none)*

The *Communicator* agent is responsible for managing the communication between the Navigation system and the Pilot and Vision systems. It is also responsible for gathering the bids of the other agents, and decide which are the actual Navigation system's bids. It has no parameters to tune.

5 Evolving the Multiagent system

As we have already mentioned, trying to manually find the best values for the parameters of the bidding functions is an extremelly difficult task. In this section we follow an evolutionary approach to do this optimization.

5.1 Representation

We seek to optimize the Navigation system with respect to its 10 parameters: Target tracker (α, β, κ_1, κ_2), Distance Estimator (κ), Risk manager (γ_A, γ_B, γ_r), and Rescuer ($\overline{I}_a$, $\overline{R}$). ϕ and δ are fixed to 0.7 and 2 respectively since they do not affect the efficiency of the system. We use a real valued chromosome, each chromosome being a vector in 10 dimensions. The initial population is generated randomly.

5.2 Navigation Tasks

For a given environment we consider two different navigation tasks. Each one of them with a different level of complexity. The best parameter set may change depending on the complexity of the task. We conjecture that the parameters found depend mainly on the complexity of the navigation task and not so much on the structure of the overall environment. This complexity is dependent, though not equal, to the cartographic complexity of the world in which the agent moves, and is based on the following factors:

- number of visible landmarks at any time,
- density of obstacles in the region of navigation, and
- visibility of the target at any time.

Using this notion of navigational complexity, the total space of all navigation tasks can be split into two representative classes: going towards the target free of obstacles, and reaching targets located behind obstacles. In our experiments we use clusters C_1 and C_2 (encircled targets

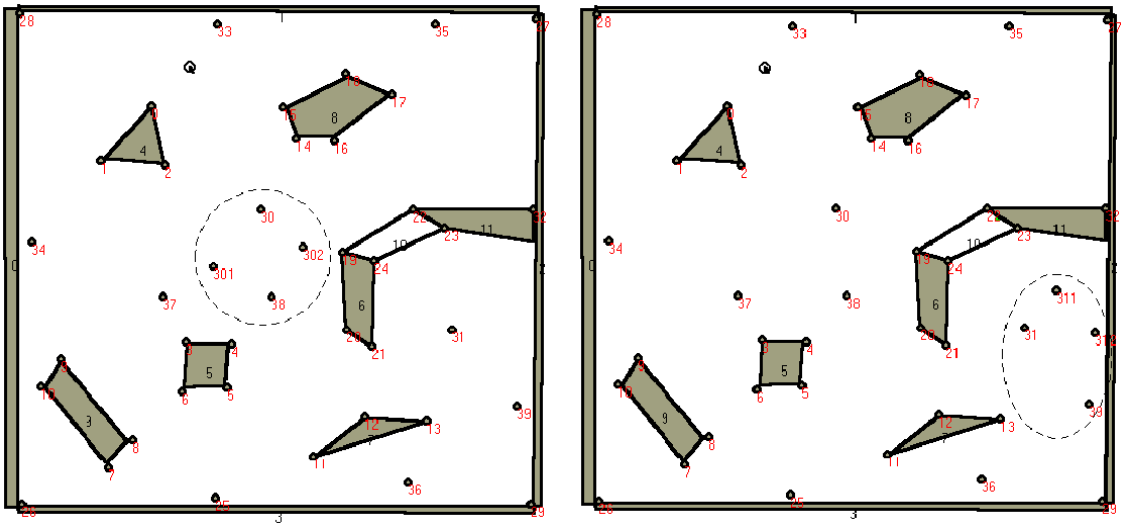

Figure 3: Cluster C_1 (left) and C_2 (right). White polygons are non occluding obstacles.

in Figure 3) as representatives of the two task complexity classes. The best parameter set is determined for both these classes. The aim of the experiments is to endow the Navigation system of the robot with the capability to switch between these two parameter sets according to the actual task complexity it is facing.

5.3 Evaluation

Each individual in the population specifies a particular parameter set for the system, and is evaluated by running a simulation with the specified parameters in a given environment. Consider that the agent navigates from an initial position p_0 to the target cluster C containing the n target positions (t_1, t_2, ..., t_n) and that it takes d_i steps to reach the target t_i from p_0 with a success value s_i. A threshold is defined for the number of steps that are taken to reach the target, above which the agent is said to have failed in its attempt to navigate to the target i.e. its success value is 0, otherwise it is 1.

This formalization gives the clues to define the fitness function, f, that permits the selection of the best parameter sets. It is clear that the average cost, $\overline{c}$, of reaching a target from the initial position p_0 is defined as the summation of the steps required to reach each target divided by the number of targets. Similarly, we can naturally define the average success, $\overline{s}$. The best behaviour for a Navigation system is the one that has a high success rate with a low average cost and with a low standard deviation σ_c for this average cost:

$$\overline{c} = \frac{\sum_{i=1}^{n} d_i}{n} \qquad \overline{s} = \frac{\sum_{i=1}^{n} s_i}{n} \qquad f = \frac{\overline{s}}{\overline{c}+\sigma_c}$$

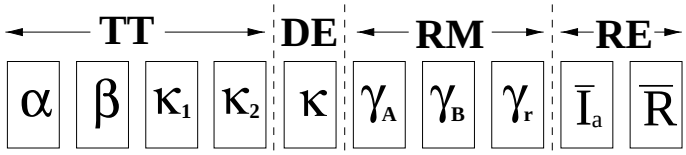

Figure 4: Chromosome with the set of parameters

5.4 Evolution

We follow an elitist approach. That is, from a population of individuals, the fittest individual is passed to the next generation. The remaining individuals form the pool from which the new generation offspring are created. We randmonly select two individuals from the mating pool whose fitness is over a randomly determined value. Then we apply crossover and mutation on them to generate new individuals.

5.5 Crossover

A simple two point crossover is used with the two parents exchanging their genetic material between two randomly generated breakpoints in the gene string. Chromosomes are broken only at agent boundaries (see Figure 4). The idea is that one of the parents may have good genes for a particular agent while the other parent may have good genes for another agent. This way the crossover could result in an offspring having a higher fitness value than both its parents.

5.6 Mutation

The mutation operator for the genetic algorithm has been adopted from the Breeder Genetic Algorithm [14]. Given any set of parameters as a chromosome, we can view it as a point *x* within a 10 dimensional space. Using our mutation operator, we seek to search for optimality within a "small" hypercube centered at *x*. How small this hypercube is, depends on the ranges in each parametric dimension within which we allow the chromosome to mutate. The parametric dimensions are not homogeneous, hence mutation ranges differ for each dimension, being directly proportional to the variance allowed in that parameter. Another feature of this mutation operator is that while it searches within the hypercube centered at *x*, it tests more often in the very close neighbourhood of *x*, the idea being that, while we want to conduct a global search for optimum using our recombination, mutation is used for a more restricted local search. Having understood the broad features which the mutation operator should demonstrate, we formally define the mutation as follows:

Given a chromosome *x*, each parameter x_i is mutated with probability 0.1. The number of parameters being 10 implies that at least one parameter will be probably mutated. Further, given the mutation range for the parameter x_i as $range_i$, the parameter x_i is mutated to the value $x_i{}^*$ given by

$$x_i{}^* = x_i \pm range_i \cdot \rho$$

As previously discussed, ρ should be such that it lies between 0 and 1 (to generate the hypercube centered at x) and also it should probabilistically take on small values so as to test more often in the close neighbourhood of x. This is realized by computing ρ from the distribution

$$\rho = \sum_j \alpha_j 2^{-j}$$

where each α_j is probabilistically either 0 or 1.

5.7 Diversity

The convergence of the genetic algorithm is estimated through its population diversity. Initially, the population has a high diversity since all the individuals are randomly selected. As the algorithm converges, the individuals in the population converge towards the best solution, thus decreasing the diversity. In our case, the individuals are points in a heterogeneous dimension space, with α, β, γ_A and $\gamma_B \in \Re^+$ while the other parameters ranging between 0 and 1. Hence we use the Mahalanobis distance measure to determine the diversity of a population [8].

The Mahalanobis distance takes into account the heterogeneity in dimensions and correspondingly scales each dimension while estimating the distance between two points. Given a set of data points $\{z_i\}$ with each data point z_i being an n-tuple $\langle z_{ij} | 1 \leq j \leq n \rangle$, the Mahalanobis distance d_m between two points z_k and z_l is given as

$$d_m(z_k, z_l) = (z_k - z_l)^T \Sigma^{-1} (z_k - z_l)$$

Here Σ is the $n \times n$ variance-covariance matrix for the given data points. To compare the diversity of populations across generations, the covariance matrix is computed taking into account all the chromosomes over all generations. The diversity of a population is then calculated as the average Mahalanobis distance of each chromosome from the mean chromosome.

6 Results

The genetic algorithm was run on the two task complexity classes represented by the target clusters C_1 and C_2 in our simulator. The population size was of 20 individuals, and we ran the genetic algorithm for 100 generations. The initial position was the same for both with the crossover and the mutation rates being 0.8 and 0.1 respectively. In the algorithm, four of the parameters — α, β, γ_A and γ_B lie on the positive real axis and hence we have to choose an upper limit on the real line. This upper limit is important since a low upper limit value implies that we implicitly restrict our real valued parameters to that limit, while a high upper limit value may increase the number of generations for which the genetic algorithm may have to be run since the initial random generation will be very disperse. α and β are exponents of numbers less than 1 and hence their large values will not be useful. Keeping these factors in consideration, the upper limit value has been fixed to 5 in our simulations.

The genetic algorithm converges to an optimal solution for each cluster as can be seen in Figures 5-10. The optimal values for some of the parameters differ significantly for the two

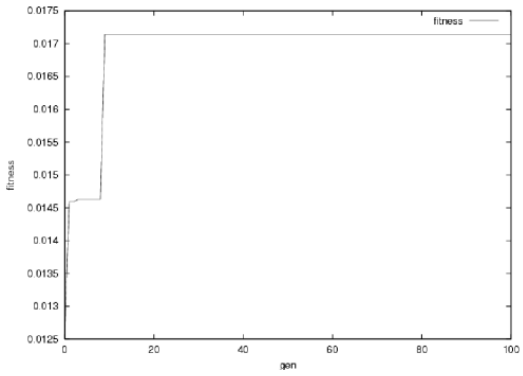

Figure 5: Fitness of the fittest individual along generations (cluster C_1)

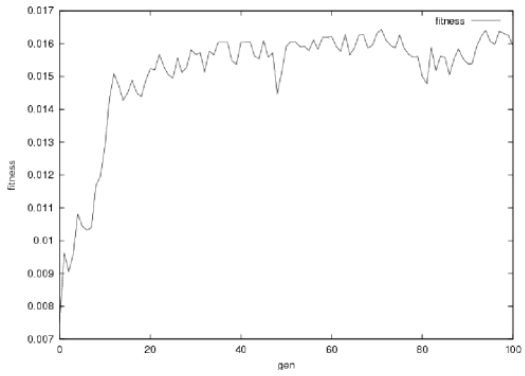

Figure 6: Average fitness of the population along generations (cluster C_1)

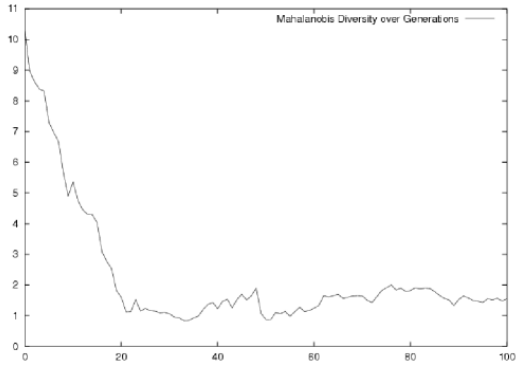

Figure 7: Mahalanobis diversity (cluster C_1)

Cluster	α	β	κ_1	κ_2	κ	γ_A	γ_B	γ_R	$\overline{I}_a$	$\overline{R}$
Cluster1	1.731	2.03	0.314	0.493	0.355	0.240	0.521	0.054	0.386	0.215
Cluster2	1.231	2.12	1.0	0.564	0.178	1.377	4.39	0.707	0.871	0.906

Table 1: Optimal parameter values for each of the clusters for one execution of the GA over 100 generations

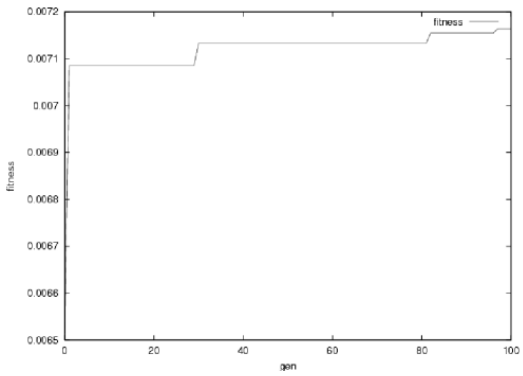

Figure 8: Fitness of the fittest individual along generations (cluster C_2)

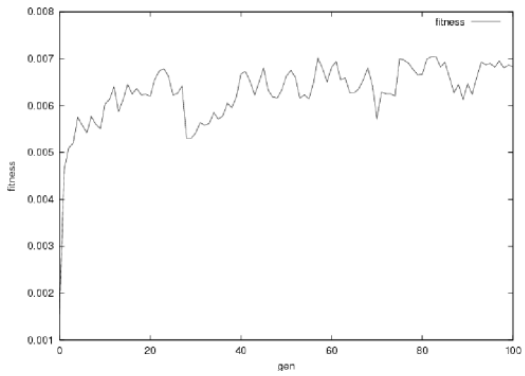

Figure 9: Average fitness of the population along generations (cluster C_2)

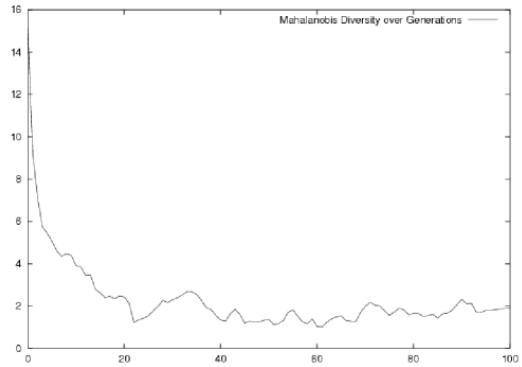

Figure 10: Mahalanobis diversity (cluster C_2)

clusters as shown in Table 1. The parameters associated to the bidding function of the *Risk Manager* agent differ the most between the two clusters. This is so because the Risk Manager is very sensitive to the complexity of the task. The more obstacles, the higher the risk of losing sight of landmarks.

In order to check the results obtained for each of the clusters, we have tested the two parameter sets found by the genetic algorithm on the two different navigation tasks (going to cluster C_1 and going to cluster C_2). We have also tested our original parameter set, which we set by hand, on the same two navigation tasks. The results obtained by each set on each of the tasks are shown in Table 2. For each task, the mean average success value ($\overline{s}$), average cost ($\overline{c}$) and the fitness value (f) is computed. As expected, the parameter set found for cluster C_1 performs perfectly when going to cluster C_1 and it only reaches the targets of cluster C_2 50% of the time. On the other hand, the parameter set found for cluster C_2 reaches the targets of cluster C_2 all the time, while it only reaches the targets of cluster C_1 50% of the time. Finally, the hand-tuned parameter set reaches 50% of the time for targets in cluster C_1, and never reaches the targets of cluster C_2. Therefore, the evolutionary approach has improved the global navigation behaviour.

In Figures 11 and 12 we can see some paths followed by the robot using each of the parameter set on each of the tasks. Succesful paths are only shown for those parameter set with a

	Going to C_1			Going to C_2		
	$\overline{s}$	$\overline{c}$	f	$\overline{s}$	$\overline{c}$	f
C_1 set	1	50.5	0.017	0.5	127.5	0.003
C_2 set	0.5	42.5	0.011	1	122	0.007
HT set	0.5	69	0.005	0	–	0

Table 2: Results obtained by the different parameter sets

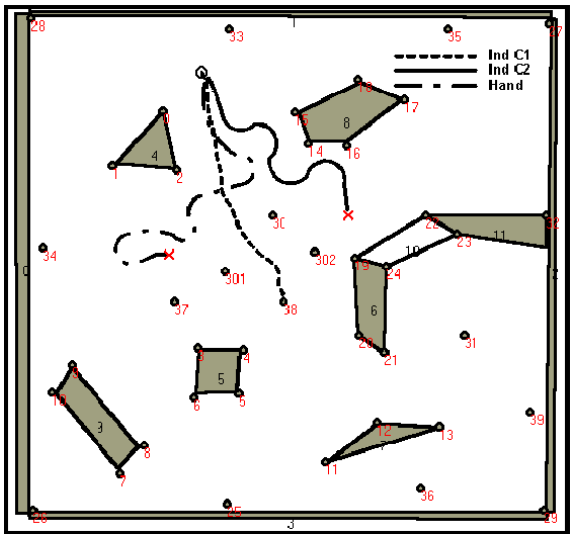

Figure 11: Going to C_1

success value of 1. Otherwise, an example of a failing path (marked with a cross at its end) is shown.

We are currently testing the parameter sets on a real robot, and we will analyse the generality, in terms of different environments and starting point, of the parameters obtained by the genetic algorithm. Further work should also focus on designing an agent capable of identifying the complexity of the task being performed, so that the parameters can be switched from one set to another. We will explore the use of Case Base Reasoning techniques on this "situation identifier" agent.

7 Acknowledgements

This work has been partially supported by the Spanish MCYT project ARGOS (DPI2000-1352-C02-02). Dídac Busquets enjoys a CIRIT doctoral scholarship 2000FI-00191.

References

[1] Arvin Agah and George A. Bekey. A genetic algorithm-based controller for decentralized multi-agent robotic systems. In *IEEE International Conference on Evolutionary Computation*. IEEE, 1996.

[2] R. C. Arkin. *Behaviour-based Robotics*. MIT Press, 1998.

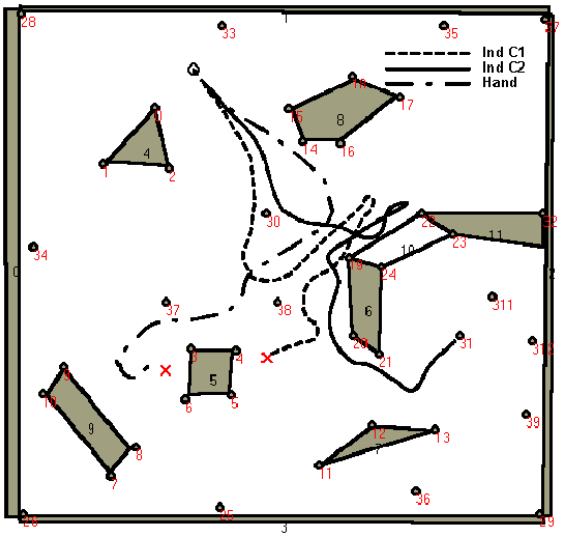

Figure 12: Going to C_2

[3] R.C. Arkin. Motor schema-based mobile robot navigation. *International Journal of Robotics research*, 8(4):92–112, 1989.

[4] G. Bojadziev and M. Bojadziev. *Fuzzy sets, fuzzy logic, applications*, volume 5 of *Advances in Fuzzy Systems*. World Scientific, 1995.

[5] R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, 1986.

[6] Dídac Busquets, Carles Sierra, and Ramon López de Mantaras. A Multiagent Approach to Qualitative Navigation in Robotics. *Autonomous Robots*, 15:129–154, 2003.

[7] M.B. Dias and A. Stentz. A market approach to multirobot coordination. Technical report, Robotics Institute, CMU, 2001.

[8] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley & Sons, Inc., 2001.

[9] M. Teresa Escrig and F. Toledo. Autonomous robot navigation using human spatial concepts. *International Journal of Intelligent Systems*, 15:165–196, 2000.

[10] R. Liscano et al. Using a blackboard to integrate multiple activities and achieve strategic reasoning for mobile-robot navigation. *IEEE Expert*, 10(2):24–36, 1995.

[11] C. Isik and A.M. Meystel. Pilot level of a hierarchical controller for an unmanned mobile robot. *IEEE J. Robotics and Automation*, 4(3):241–255, 1988.

[12] T.S. Levitt and D.T. Lawton. Qualitative navigation for mobile robots. *Artificial Intelligence Journal*, 44:305–360, 1990.

[13] P. Maes. The dynamics of action selection. In *Proceedings of IJCAI'89*, pages 991–997, 1989.

[14] H. Muhlenbein and D. Schlierkamp-Voosen. The science of breeding and its application to the breeder genetic algorithm (BGA). *Evolutionary Computation*, 1(1):335–360, 1993.

[15] T.J. Prescott. Spatial representation for navigation in animats. *Adaptive Behavior*, 4(2):85–125, 1996.

[16] J. Rosenblatt. Damn: A distributed architecture for mobile navigation. In *Proceedings of the 1995 AAAI Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents*. AAAI Press, March 1995.

[17] John Sauter, Robert Matthews, H. van Dyke Parunak, and Sven Brueckner. Evolving adaptive pheromone path planning mechanisms. In *Proceedings of AAMAS'02*, pages 434–440, Bologna, July 2002. ACM Press.

[18] A. Stentz. The codger system for mobile robot navigation. In C.E. Thorpe, editor, *Vision and Navigation, the Carnegie Mellon Navlab*, pages 187–201, Boston, 1990. Kluwer Academic Pub.

[19] R. Sun and C. Sessions. Bidding in reinforcement learning: A paradigm for multi-agent systems. In O. Etzioni, J.P. Müller and J.M. Bradshaw, editor, *Proceedings of 3rd Annual Conference on Autonomous Agents*, pages 344–345, Seattle, 1999.