Evolution of visual controllers for obstacle avoidance in mobile robotics

Renaud Barate, Antoine Manzanera

Evolutionary Intelligence manuscript No.
(will be inserted by the editor)

Evolution of visual controllers for obstacle avoidance in mobile robotics

Renaud Barate · Antoine Manzanera

Received: date / Accepted: date

Abstract The purpose of this work is to automatically design vision algorithms for a mobile robot, adapted to its current visual context. In this paper we address the particular task of obstacle avoidance using monocular vision. Starting from a set of primitives composed of the different techniques found in the literature, we propose a generic structure to represent the algorithms, using standard resolution video sequences as an input, and velocity commands to control a wheel robot as an output. Grammar rules are then used to construct correct instances of algorithms, that are then evaluated using different protocols: evaluation of trajectories performed in a goal reaching task, or imitation of a hand-guided trajectory. A genetic program is applied to evolve populations of algorithms in order to optimize the performances of the controllers. The first results obtained in a simulated environment show that the evolution produces algorithms that can be easily interpreted and which are clearly adapted to the visual context. However the resulting trajectories are often erratic, and the generalization capacities are poor. To improve the results, we propose to use a two-phase evolution combining imitation and goal reaching evaluations, and to add some constraints in the grammar rules to enforce a more generic behavior. The results obtained in simulation show that the evolved algorithms are more efficient and more generic. Finally, we apply the imitation based evolution on real sequences and test the evolved algorithms on a real robot. Though simplified by dropping the goal reaching constraint, the resulting algorithms behave well in a corridor centering task, and show certain generalization capacities.

Keywords Genetic programming · Computer vision · Mobile robotics · Obstacle avoidance

R. Barate and A. Manzanera
ENSTA - 32 Bd Victor
75739 PARIS CEDEX 15
Tel.: +33 1-45-52-44-42
Fax: +33 1-45-52-83-27
E-mail: Renaud.Barate@ensta.fr, Antoine.Manzanera@ensta.fr

1 Introduction

The autonomy of a mobile robot refers to its capacity of interacting with a - possibly unknown - environment, with the lowest level of human supervision. This capacity obviously requires perception abilities. Amongst these, vision is certainly one of the cheapest, most informative, but most difficult to use.

Aside from perception, the other condition for acquiring autonomy is cognition: the robot makes the most of experiences gathered through learning or adaptation. An abundance of work can be found in the literature combining machine learning and vision in mobile robotics.

But it is notable that, generally, the cognitive dimension of vision itself is ignored: the visual processing is not the fruit of learning but some benchmarked, validated, and often sophisticated algorithm which produces features, and only some descriptors attached to these features take part in the learning process.

However, from a biological point of view, even the earliest stages of visual perception are the result of an adaptation process. For a mobile robot, using some simple and *ad hoc* visual procedure results in a loss of genericity, whereas using more sophisticated and robust visual feature induces a high computational load, both solutions being at the detriment of autonomy.

It is true that there are good combinatorial arguments for not integrating image processing in the adaptation loop: the size and variability of data and algorithms lead to very high computational costs. Nevertheless, the present development level of High Performance Computing [17] allows us to envisage massive concurrent computation of large number of visual primitives on the same machine.

Our objective is to automatically design vision algorithms that are well suited to a given environment, and to a given task of a mobile robot. The task that we address in this paper is the obstacle avoidance problem, as it is one of the basics for the robot to acquire autonomy. Considering a large family of visual primitives, we propose a generic structure of visual obstacle avoidance algorithm, used to construct any instance of valid algorithms from those primitives. Evolutionary computation is then used, together with an evaluation protocol to automatically select the algorithms that are best suited in the current environment. Experiments are shown in a simulated environment, then on a real robot.

Section 2 recalls the related works, i.e. learning visual obstacle avoidance techniques, or evolutionary computer vision. Section 3 presents some state of the art visual obstacle avoidance techniques (Sec. 3.1) used to define the set of algorithmic primitives (vocabulary) chosen to construct the algorithms (Sec. 3.2). The rules (grammar) that construct valid obstacle avoidance algorithms are presented in (Sec. 3.3). Section 4 presents the different evaluation methods used to select the algorithms, first in the context of reaching a given target (Sec. 4.1), and then imitating an existing trajectory (Sec. 4.2). We then present the evolution process, which is implemented through genetic programming (Sec. 4.3). In Section 5, we discuss the experimental results obtained in simulation. The global evolution, and the behavior of the best evolved algorithms are analyzed (Sec. 5.1). We then perform a comparison of different strategies used to improve the evolution (Sec. 5.2). Finally we evaluate the generalization abilities of the evolved controllers (Sec. 5.3), i.e. their capacity to keep on avoiding obstacles when their positions change (the environmental visual appearance remaining the same). Section 6 presents the results on a real Pioneer 3 DX Robot. We detail the evolution principles in the real world and show the evolved controllers, first in their evolution environment

(Sec. 6.1), and next in an unseen environment (Sec. 6.2). Finally, Section 7 draws the conclusion and discusses the perspective of this research.

2 Related works

Different machine learning methods have been proposed in the literature for obstacle avoidance using vision. Michels and Saxena used reinforcement learning to associate depth to texture patches [24,30]. Low and Wyeth used three-layer neural network to learn the robot heading command from the apparent motion (optical flow) vector field input [20]. In these works, a depth map acquired by laser range sensor is used as ground truth for supervision purposes. In all cases, one visual primitive is used exclusively, neglecting any other source of information. Le Cun *et al* designed an obstacle avoidance learning system using a neural network whose input is a pair of stereo images and output is a steer angle [18]. This system uses video and command sequences recorded by a hand guided robot as supervised learning input.

On the other hand, evolutionary techniques have already been used for robotic navigation and the design of visual obstacle avoidance controllers [29,34] but in general vision is overly simplified. For instance, Marocco used only a 5×5 pixels retina as visual input [22]. Aside from obstacle avoidance, genetic programming has been proved to achieve human-competitive results in image processing systems, e.g. for the detection of interest points [10,32]. Cooperative coevolution methods (e.g. Parisian evolution) have also produced good results for obstacle detection [28] and 3D reconstruction, the latter used either for computing the 3D coordinates from a pair of images [27], or for optimizing the placement of the different cameras [8]. A recent tutorial on evolutionary computer vision was given by Cagnoni in [5].

Ebner and Zell have used genetic programming to automatically design interest point detectors. From a set of basic image operations, they first tried to retrieve an existing operator (the Moravec detector), by minimizing the sum of squared differences between the desired and actual output images [9]. As a second approach, they used a task dependent fitness based on the quality of matches between two sets of interest points in a training sequence [10]. Furthermore, they were also interested in the obstacle avoidance problem and developed a monocular vision system to center a robot in a hallway that performed quite well [11]. However this system did not used an automatically evolved algorithm, instead it used the original Moravec detector.

To our knowledge, only Martin tried evolutionary techniques with monocular images for obstacle avoidance [23]. The structure of his algorithm is based on the floor segmentation technique and the evaluation is done with a database of hand labeled real world images. The advantage of such an approach is that the evolved algorithms are more likely to work well with real images than those evolved with computer rendered images. Nevertheless, it introduces an important bias since the algorithms are only selected on their ability to label images in the database correctly and not on their ability to avoid obstacles.

3 Generic structure of visual controllers

3.1 Visual obstacle avoidance algorithms

In the literature, the obstacle avoidance algorithms using monocular vision can be clearly divided in two categories: motion-based and appearance-based strategies.

The first category, sometimes referred to as "monocular stereovision" uses the parallax principle, according to which, when the robot moves in purely translational motion, the apparent velocity of closer objects is greater. This geometric property can be used through the estimation - by image processing - of different types of measure:

- local flow divergence, which is based on the local apparent velocity, and which can provide the depth or the time before impact of the projected point, depending on the available odometry information [26, 6].
- global flow balance, which is a measure of the symmetry between the left and the right sides of the apparent motion field. This technique is inspired by the strategy of flying insects, which use it to center themselves when flying in a narrow environment. The insects naturally take advantage of their pair of lateral sensors, however the technique has been used with success on an autonomous helicopter using a wide field single camera [25].

Within this category, the necessary algorithmic primitives are: computation of the apparent motion field (optical flow), spatio-temporal smoothing filters, and regional integral computations.

In the second category of visual obstacle avoidance algorithms, a local visual feature is associated with contextual information, which can be related to depth, orientation, or even to a segmentation of the projected scene in principal surfaces (floor, walls typically) [16, 19, 33]. The algorithmic primitives from this category are: multi-scale and multi-orientation derivative measures, image threshold and binarization functions, horizontal and vertical projection measures, and regional integral computations.

Our objective is now to construct a generic description model of a visual obstacle avoidance controller which can be used to automatically design a formally valid algorithm, in such a way that any of the approaches presented above could be derived as an instance of a valid algorithm.

3.2 Choice of the primitives

Generally speaking, a vision algorithm can be divided in three main parts: First, the algorithm will process the input image with a number of filters to highlight some features. These features are then extracted, i.e. represented by a small set of scalar values. Finally these values are used for a domain dependent task, in our case to generate motor commands to avoid obstacles. We designed the structure of our algorithms according to this general scheme. First, the filter chain consists of spatial and temporal filters, optical flow calculation and projection that will produce an image highlighting the desired features. We then compute the mean of the pixel values on several windows of this transformed image (feature extraction step). Finally those means are used to compute a single scalar value by a linear combination. We will use this scalar value to determine the presence of an obstacle and to generate a motor command to avoid it.

An algorithm is represented as a tree, the leaves being input data (video image or scalar constant value), the root being output command (generating linear and yaw speed), and the internal nodes being primitives (transformation steps). The program can use different types of data internally, i.e. scalar values, images, optical flow vector fields or motor commands. For each primitive, the input and output data types are fixed. Some primitives can internally store information from previous states, thus allowing temporal computations like the calculation of the optical flow.

Most of those primitives also use parameters along with the input data (for example, the standard deviation value for the Gaussian filter). The parameters are specific to each algorithm; they are randomly generated when the corresponding primitive is created by the genetic programming system described in Sec. 4.3. For each parameter, we define the distribution used to generate it (uniform or normal) along with the range of valid values. We also define the mean and standard deviation of the normal distribution where applicable. The parameters with their distributions are detailed in Table 1. Here is the list of all the primitives that can be used in the programs and the data types they manipulate:

- **Spatial filters** (*input: image, output: image*):
  - Gaussian filter: This low-pass filter has one parameter: the standard deviation of the Gaussian function.
  - Laplacian filter: This high-pass filter has no parameter, it is defined by the following 3×3 convolution kernel:

0	1	0
1	-4	1
0	1	0

Convolution kernel defining the Laplacian filter used in our system.

  - Threshold filter: The parameter defining this filter is the threshold value.
  - Gabor filter: This filter is used to detect patterns of given orientation and frequency. The parameters are: orientation, wavelength and bandwidth.
  - Difference of Gaussians: The parameters are the standard deviations of each Gaussian function.
  - Sobel filter: This filter is defined by its orientation. The 3×3 convolution kernels used for each orientation are:

-1	0	1
-2	0	2
-1	0	1

-1	-2	-1
0	0	0
1	2	1

Convolution kernels defining the vertical (left) and horizontal (right) Sobel filters used in our system.

  - Subsampling filter: This filter is defined by the size coefficient.
- **Temporal filters** (*input: image, output: image*):
  - Pixel-to-pixel min, max, sum and difference of the last two frames. These filters have no parameters.

  - Recursive mean operator. This filter is computed with the following formula at each pixel: $M_t = \alpha I_t + (1 - \alpha) M_{t-1}$ where I_t is the current pixel value and M_{t-1} is the filtered value at time $t - 1$. The parameter for this filter is the coefficient α.
- **Optical flow** (*input: image, output: vector field*):
  - Horn and Schunck global regularization method [15].
  - Lucas and Kanade local least squares calculation [21].
  - Simple block matching method.

  The rotation movement is first eliminated by a transformation of the two images in order to facilitate further use of the optical flow.
- **Projection** (*input: vector field, output: image*):
  - Projection on the horizontal or vertical axis.
  - Euclidean norm computation.
  - Manhattan norm computation.
  - Time to contact calculation using the flow divergence.
- **Windows integral computation** (*input: image, output: scalar*): The method used for this transformation is:
  1. A global coefficient α_0 is defined for the primitive.
  2. Several windows are defined on the left half of the image with different positions and sizes. With each window is paired a second window defined by symmetry along the vertical axis. A coefficient α_i and an operator ($+$ or $-$) are defined for each pair.
  3. The resulting scalar value R is a simple linear combination calculated with the following formula:

$$\begin{aligned} &R = \alpha_0 + \textstyle\sum_{i=1}^{n} \alpha_i \mu_i \\ &\mu_i = \mu_{\mathrm{L}i} + \mu_{\mathrm{R}i} \text{ or } \mu_i = \mu_{\mathrm{L}i} - \mu_{\mathrm{R}i} \end{aligned} \tag{1}$$

     where n is the number of windows and $\mu_{\mathrm{L}i}$ and $\mu_{\mathrm{R}i}$ are the means of the pixel values over respectively the left and right window of pair i.

  The number of windows pairs, their positions, sizes, operator and coefficient along with the global coefficient are characteristic parts of the primitive and will be customized by the evolutionary process.
- **Scalar operators** (*input: scalar(s), output: scalar*):
  - Addition, subtraction, multiplication and division operators.
  - Temporal mean calculation. This is simply the mean value computed on the N last time steps.
- **Command generation** (*input: two scalars, output: command*): The motor command is represented by two scalar values: the requested linear and angular speeds.

3.3 Construction grammars

We use context-free grammars to represent and build these vision algorithms. We will describe here the sets of terminal and non-terminal symbols in these grammars, and the two different structures we use to design our obstacle avoidance algorithms. The build process itself will be detailed in Section 4.3.

The base set of non-terminal symbols is simply the list of primitives that we have just described in Section 3.2. The terminal symbols are the current video image and

Table 1: Parameters of the primitives used in our system (units are shown in parenthesis where applicable).

Primitive and parameter	**Distribution**	**Min.**	**Max.**	**Mean**	**Std**
Scalar constant					
Value	Normal	-	-	0,0	50,0
Temporal regularization					
Number of values	Normal	1	100	1	3
Windows integral computation					
Global coefficient α_0	Uniform	-180,0	180,0	-	-
Window					
Position x_1 (rel. to image width)	Uniform	0,0	0,5	-	-
Position x_2 (rel. to image width)	Uniform	0,0	0,5	-	-
Position y_1 (rel. to image height)	Uniform	0,0	1,0	-	-
Position y_2 (rel. to image height)	Uniform	0,0	1,0	-	-
Coefficients α_i	Normal	-	-	0,0	3,0
Operator (+ or -)	Uniform	-	-	-	-
Gaussian filter					
Standard deviation (pixels)	Normal	0,0001	20,0	3,0	2,0
Threshold filter					
Threshold (gray level)	Uniform	0	255	-	-
Gabor filter					
Orientation (degrees)	Uniform	0,0	180,0	-	-
Wavelength (pixels)	Normal	2,1	20,0	5,0	2,0
Bandwidth	Normal	0,5	2,0	1,0	0,3
Difference of Gaussians					
Standard deviation 1 (pixels)	Normal	0,0001	20,0	3,0	2,0
Standard deviation 2 (pixels)	Normal	0,0001	20,0	3,0	2,0
Sobel filter					
Orientation (H or V)	Uniform	-	-	-	-
Subsampling filter					
Size coefficient	Uniform	0,01	1,0	-	-
Recursive mean					
Multiplying coefficient α	Uniform	0,01	1,0	-	-
Horn-Schunck optical flow					
Weight coefficient α^2	Normal	0,0	100,0	2,0	1,0
Lucas-Kanade optical flow					
Size of the window (pixels)	Uniform	1	15	-	-

a scalar constant (the constant value is a random generated parameter, see Table 1). Depending on the kind of structure used to build the algorithm, these sets will be completed with a few other symbols.

The first controllers we have constructed can be represented with a single algorithmic tree. This tree is entirely built by the evolutionary process without any *a priori* in the structure of the algorithm. In this paper, we will call them *structure-free controllers*. In this case, we shall add two terminal symbols to the base set: the distance (in cm) and the direction (in degrees) of the target point. We shall also add one non-terminal symbol: an if-then-else test, which is a scalar operator with four inputs and one output. If the first input value is greater than the second, the output value is equal to the third input value, otherwise it is equal to the fourth input value. This allows the evolutionary process to create more complex and non-linear combinations between new scalar input variables and the scalars issued from the vision part of the algorithm. Fig. 1 shows a controller that can be built with this structure. The algorithm is a single tree and all the primitives between the input data and the resulting motor command are created and parameterized by the evolution.

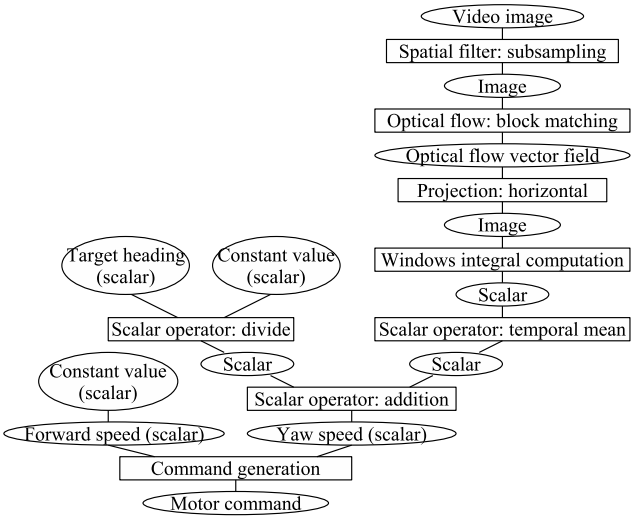


Fig. 1: Example of a structure-free controller for obstacle avoidance based on optical flow. Rectangles represent primitives and ellipses represent data.

The price to pay for the maximal genericity in these structure-free controllers is that there is no guarantee that the evolved controllers will use the input variables (target heading and distance in particular). Thus, it is quite unlikely that such a controller evolved in a given environment could develop a geometric abstraction of the target reaching task.

This is why we also designed a second kind of controllers with a more constrained structure, which facilitates the trade-off between avoiding obstacles and reaching the target point. We will call them *structure-restricted controllers* in the rest of this paper. First, a vision algorithm is used to detect nearby obstacles. If no obstacle is around, the robot will just go straight to the target point. If an obstacle is detected, another vision algorithm will be used to generate a command to avoid this obstacle. The parts that will be customized by the evolution are the two vision algorithms along with a few parameters like the speed of the robot when going straight to the goal. This global structure is represented on Figure 2.

The first vision algorithm (obstacle detection) is in fact always a simple function chain, starting with the video image and computing a Boolean as output. We obtain this Boolean value by comparing the scalar produced by the feature extraction step with a scalar threshold. This result will indicate the presence or absence of a nearby obstacle. Formally, this means that the grammar used to generate this algorithm will be slightly different. We add two non-terminal symbols to the base set: the Boolean function *not* and a threshold function with a scalar value as input, a Boolean value as output, and a scalar parameter (the threshold value). We also remove the terminal symbol *scalar constant*, the binary scalar operators and the command generation primitive.

The second vision algorithm (command generation) can use as an input either the original video image or the image filtered by the obstacle detection algorithm. This allows the reuse of interesting features between the two algorithms. The grammar used to generate this algorithm contains the base set of terminal and non-terminal symbols,

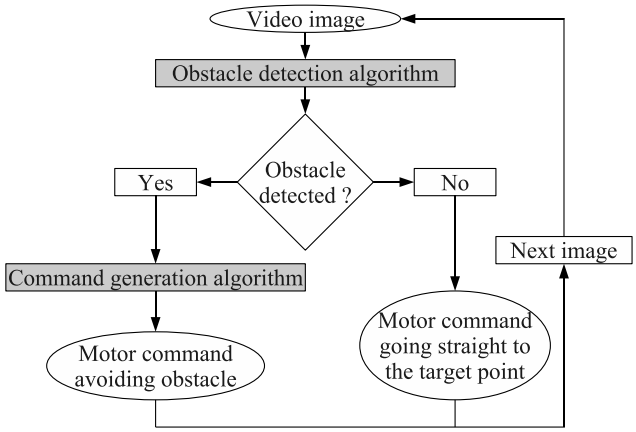


Fig. 2: Overview of the global fixed structure of the structure-restricted controllers. The algorithms in light gray will be customized by the evolution.

completed with this *filtered image* terminal symbol. Fig. 3 illustrates this restricted structure with an example of the two different algorithms.

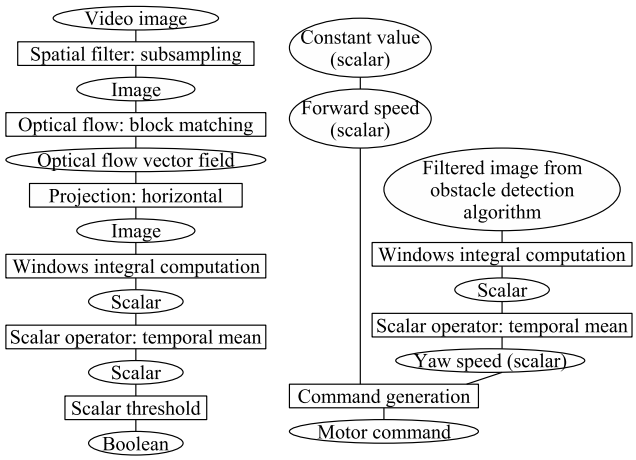


Fig. 3: Structure-restricted controller. Left: an example of obstacle detection algorithm. Right: an example of command generation algorithm. Rectangles represent primitives and ellipses represent data.

4 Evaluation and evolution process

In this section we describe how the grammar presented above is used to automatically construct efficient algorithms. Starting from an initial population of randomly produced

algorithms, we evaluate them to select the ones most adapted to the obstacle avoidance task, we then combine the best individuals to produce a new population, and reiterate the process to obtain a certain number of generations. Let us first present the different evaluation (fitness) functions we have used.

4.1 Evaluation 1: goal reaching *vs* contact

The principle of the first evaluation method is to measure the ability of the controllers to reach a given target whilst avoiding the obstacles. The protocol is as follows: a *course* is defined by the location of a starting point and a goal point in the environment. We place the robot at the starting point and let it move in the environment during 30 or 60 s (depending on the experiments) driven by the obstacle avoidance algorithm. Two scores are attributed to the algorithm depending on its performance: a goal-reaching score G rewards algorithms reaching or approaching the goal location, whereas contact score C rewards the individuals that didn't hit obstacles on their way. Those scores are calculated with the following formulas:

$$\begin{aligned} G &= \begin{cases} t_{\mathrm{G}} \text{ if the goal is reached} \\ t_{\max} + d_{\min}/V \text{ otherwise} \end{cases} \\ C &= t_{\mathrm{C}} \end{aligned} \tag{2}$$

where t_G is the time needed to reach the goal in seconds, $t_{\max}$ is the maximum time in seconds (30 or 60 s), $d_{\min}$ is the minimum distance to the goal achieved in meters, V is a constant of 0.1 m/s and t_C is the time spent near an obstacle (i.e. less than 18 cm, which forces the robot to keep some distance away from obstacles). The goal is hence to minimize those two scores G and C. For better generalization performances, we designed several courses with different starting points and goal locations (2 or 4 courses depending on the experiments). Final scores are the means of the scores obtained on the different courses. The starting and goal points are fixed because we want to evaluate all algorithms on the same problem.

Due to the number of individuals to be evaluated and the necessity of keeping the conditions the same for all of them, this protocol cannot be implemented in a real environment. As such, we use a simulation environment in which the robot moves freely during each experiment. The simulation is based on the open-source robot simulator Gazebo [1]. The simulator uses ODE physics engine [2] for the movement of the robot and collisions detection and OpenGL [3] for the rendering of the camera images. The physics engine update rate is 50 Hz, while the camera update rate is 10 Hz. In all the experiments presented in this paper, the simulated camera produces 8-bits gray-value images of size 320×160 representing a field of view of approximately $100^\circ \times 60^\circ$. This large field of view reduces the dead angles and hence facilitates obstacle detection and avoidance. The simulation environment is a closed room of 36 m^2 area (6 m $\times$ 6 m) containing three bookshelves (Fig. 4). All the obstacles are immovable to prevent the robot from just pushing them instead of avoiding them.

4.2 Evaluation 2: imitation

In the obstacle avoidance problem, it is very difficult to manually design a mediocre controller but it is very easy to manually guide the robot toward the target point while

Fig. 4: Snapshot of the simulation environment, containing three bookshelves in a 6 m × 6 m closed room.

avoiding obstacles. We can obtain a good example of an efficient behavior by recording the video sequence and command parameters while we guide the robot. With this evaluation method, we try to evolve algorithms that imitate this efficient example behavior.

For the evaluation of the algorithms, we replay the recorded sequence, using it as input of the evaluated algorithm, and compare the command issued by this algorithm with the command recorded during the manual control of the robot. The goal is to minimize the difference between these two commands along the recorded sequence. Formally, we try to minimize two variables F and Y defined by the formulas:

$$F = \sqrt{\sum_{i=1}^{n} (f_{\mathrm{R}i} - f_{\mathrm{A}i})^2} \text{ and } Y = \sqrt{\sum_{i=1}^{n} (y_{\mathrm{R}i} - y_{\mathrm{A}i})^2} \tag{3}$$

where $f_{\mathrm{R}i}$ and $y_{\mathrm{R}i}$ are the recorded forward and yaw speed commands for frame i, $f_{\mathrm{A}i}$ and $y_{\mathrm{A}i}$ are the forward and yaw speed commands from the tested algorithm for frame i and n is the number of frames in the video sequence.

In this case, we also perform a set of several courses using different recorded sequences, and compute the average of the scores obtained on the different sequences.

Compared with the previous evaluation, this method is somewhat less generic, as it forces the controller to follow a guided trajectory, which restricts the type of solution that an individual can provide. Nonetheless, the imitation strategy may favor efficient solutions from an energetic point of view, as the guided trajectories are deliberately the smoothest, safest and as direct as possible.

Furthermore, unlike the last method, the imitation strategy can be implemented in a real environment as easily as in a virtual environment: the recorded video and command sequence can be acquired using a real platform, such as the Pioneer 3 DX robot (Fig. 5).

4.3 Evolution through genetic programming

We use genetic programming to evolve vision algorithms with little *a priori* on their structure. As usual with evolutionary algorithms, the population is initially filled with randomly generated individuals. We use the grammar based genetic programming system introduced by Whigham [35] to overcome the data typing problem. It also allows us to bias the search toward more promising primitives and to control the growth of the algorithmic tree.

Fig. 5: The Pioneer 3 DX robot with its Canon VC-C50i camera.

Table 2: Grammar used in the genetic programming system for the creation and transformation of the command generation algorithm for structure-restricted controllers. The grammars used for the obstacle detection algorithm and for the structure-free controllers are very similar to this one.

[1.0] START → COMMAND	[0.15] SPATIAL_FILTER → gaussian
[1.0] COMMAND → directMove(REAL,REAL)	[0.14] SPATIAL_FILTER → laplacian
[0.15] REAL → scalarConstant	[0.14] SPATIAL_FILTER → threshold
[0.075] REAL → add(REAL,REAL)	[0.14] SPATIAL_FILTER → gabor
[0.075] REAL → subtract(REAL,REAL)	[0.14] SPATIAL_FILTER → diffOfGaussians
[0.05] REAL → multiply(REAL,REAL)	[0.14] SPATIAL_FILTER → sobel
[0.05] REAL → divide(REAL,REAL)	[0.15] SPATIAL_FILTER → subsampling
[0.1] REAL → temporalRegularize(REAL)	[0.2] TEMPORAL_FILTER → temporalMin
[0.5] REAL → windowsIntegralCompute(IMAGE)	[0.2] TEMPORAL_FILTER → temporalMax
[0.3] IMAGE → videoImage	[0.2] TEMPORAL_FILTER → temporalSum
[0.3] IMAGE → previouslyFilteredImage	[0.2] TEMPORAL_FILTER → temporalDiff
[0.25] IMAGE → SPATIAL_FILTER(IMAGE)	[0.2] TEMPORAL_FILTER → recursiveMean
[0.1] IMAGE → PROJECTION(OPTICAL_FLOW)	[0.2] PROJECTION → horizontalProjection
[0.05] IMAGE → TEMPORAL_FILTER(IMAGE)	[0.2] PROJECTION → verticalProjection
[0.33] OPTICAL_FLOW → hornSchunck(IMAGE)	[0.2] PROJECTION → euclideanNorm
[0.33] OPTICAL_FLOW → lucasKanade(IMAGE)	[0.2] PROJECTION → manhattanNorm
[0.34] OPTICAL_FLOW → blockMatching(IMAGE)	[0.2] PROJECTION → timeToContact

In the same way that a grammar can be used to generate syntactically correct random sentences, a genetic programming grammar is used to generate valid algorithms. The grammar defines the primitives and data (the bricks of the algorithm) and the rules that describe how to combine them. The generation process consists in successively transforming each non-terminal node of the tree with one of the rules. This grammar is used for the initial generation of the algorithms and for the transformation operators. The crossover consists in swapping two subtrees issued from identical non-terminal nodes in two different individuals. The mutation consists in replacing a subtree by a newly generated one. Table 2 presents the grammar we used in our experiments with the structure-restricted controllers.

The numbers in brackets are the probability of selection for each rule. A major advantage of this system is that we can bias the search toward the usage of more promising primitives by setting a high probability for the rules that generate them. We

can also control the size of the tree by setting small probabilities for the rules that are likely to cause an exponential growth (rules like REAL → add(REAL,REAL) for example).

As described previously, we wish to minimize two criteria (G and C for the first method, F and Y for the second one). There are different ways to use evolutionary algorithms to perform optimization on several and sometimes conflicting criteria. For the experiments described in this paper, we chose the widely used multi-objective evolutionary algorithm called NSGA-II. This algorithm is based on the Pareto dominance principle. Individuals are sorted by non-dominance rank, so that non-dominated individuals get a higher probability of being selected for breeding. This algorithm is elitist and a "crowding distance" is used to promote diversity among the individuals. More details can be found in the paper by K. Deb [7].

In order to prevent problems of premature convergence, we separate the population of algorithms in 4 islands, each containing 100 individuals. Those islands are connected with a ring topology; every 10 generations, 5 individuals selected with binary tournament will migrate to the neighbor island while 5 other individuals are received from the other neighbor island. For the parameters of the evolution, we use a crossover rate of 0.8 and a probability of mutation of 0.01 for each non-terminal node. We use a classical binary tournament selection in all our experiments. Due to the length of the experiments, we did not conduct a thorough statistical analysis of the influence of those parameters, which were determined empirically.

5 Experiments in simulation

In this section we present and discuss the results obtained in simulation using the proposed system. In section 5.1, we focus on the first results, obtained with the structure-free grammar, i.e. with the least *a priori*, and using the goal-reaching evaluation. In section 5.2, we apply a two-phase evolution using the imitation strategy to guide the evolution, and compare the results with other strategies classically used to improve or speed up the evolution. In section 5.3, we discuss the generalization performance and show the interest of using the restricted structure grammar.

5.1 Analysis of the evolved controllers

The objectives of our first experiments were to see what kind of controllers could be automatically designed with the minimal level of *a priori*, and without biasing the evolution with any subjective decision. As such, the evolution process in this section has been made with genetic programming using structure free grammar, and an evaluation based on the objective performances of the algorithm, i.e. the goal-reaching evaluation. The evolution lasts 100 generations, and the population is divided in 4 islands of 100 individuals. Every experiment then represents 40,000 evaluations. Three experiments have been conducted, using three different simulation environment: (1) A simple environment made of non-textured blocks, (2) A simple environment made of textured blocks and walls, and (3) A more realistic environment, made of a room with three bookshelves (analogous to Fig. 4).

The performance analysis of the successive generations can be done by plotting the contact *vs* goal accession scores of the non dominated individuals of every generation, which correspond to the Pareto fronts. In the three types of environment, it can be

observed that the performances increase rapidly during the first generations. The progression is much slower during the second half of the evolution, but is always globally significant, and the best algorithms are always better than the reference controller, that has been designed by hand for this specific environment. As an example, Fig. 6 shows the Pareto fronts for the textured blocks environment. In this case, the reference controller (represented here as the cross), is the algorithm based on balancing the average optical flow horizontal components on the left and right sides of the image. (The constructed algorithm corresponds to the tree shown on Fig. 1).

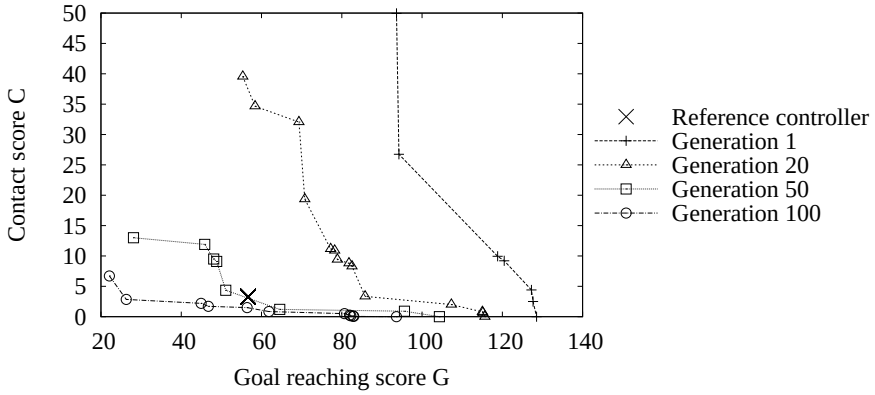


Fig. 6: Pareto fronts during the evolution process with the textured blocks environment.

Another useful analysis to be made is to observe the results of the best individuals of the evolution, to see what type of behavior they have developed to avoid the obstacles. To do this, we first plot the actual trajectory that has been performed by the robot guided by the evolved algorithm, and then display what we can call the genotype of the individual, which corresponds to the constructed algorithm. Fig. 7 shows as an example those data for one individual from the last Pareto front of the evolution realized on the textured blocks environment. More specifically, this individual corresponds to the point with the smallest contact score, i.e. the most careful behavior. If we look at the algorithmic tree, we can see that the forward speed command is based on an optical flow computation, which allows to detect close frontal obstacles, and to generate a negative velocity command (the robot moves backward when a frontal obstacle is detected). The yaw speed command is based on a Gabor filter whose purpose is to move away from the lateral obstacles seen at a certain distance. The resulting trajectories show many backward motions, relatively poor results in goal reaching, but very few collisions.

Fig. 8 displays the same observations for an individual evolved in the bookshelves environment. In this case, the forward speed relies on integral measures made after a Gaussian filter followed by a threshold. This corresponds to the detection of an obstacle, since the floor is globally lighter than the bookshelves or the walls. When the area covered by an obstacle is beyond a certain threshold, it generates a negative forward speed, corresponding to a backward motion. The yaw speed is simply provided by a linear function of the target direction, which allows the robot to maintain the global heading of the trajectory. As seen in the resulting trajectories, this algorithm

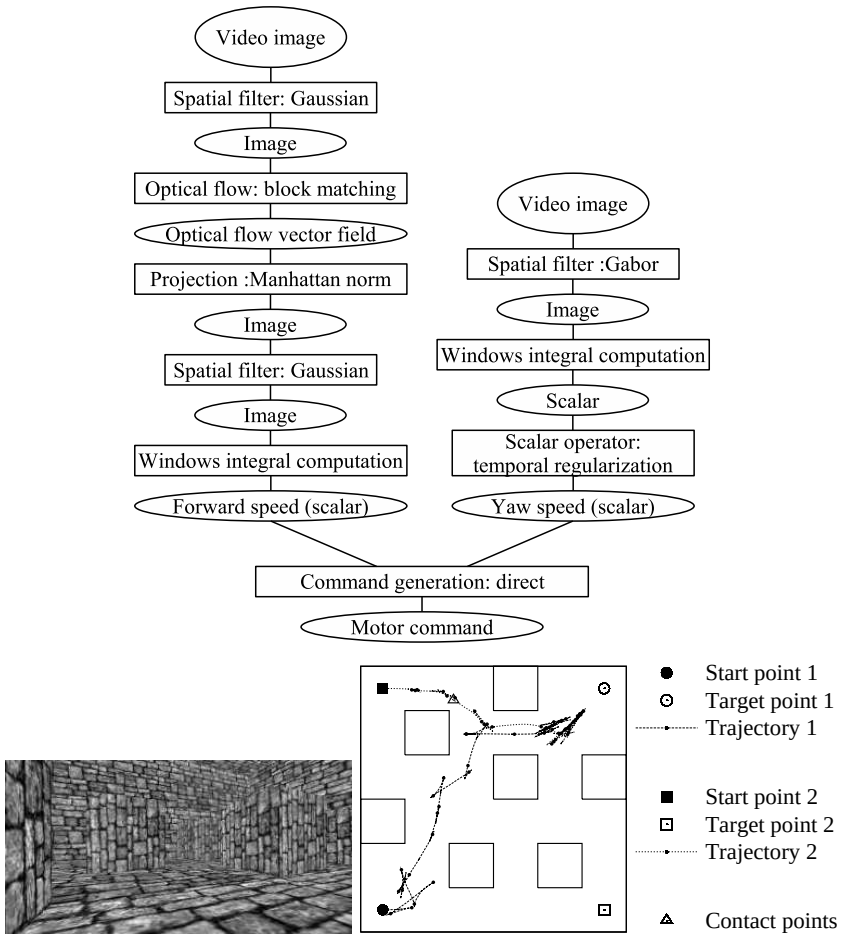


Fig. 7: Example of an algorithm evolved in the textured blocks environment

is very efficient in the evolution environment: the target is always reached, and the trajectories are relatively rapid with very few contacts.

To summarise these first results, it can be said that our initial evolutionary system with objective evaluation measure has shown a certain level of efficiency since relevant adaptation behaviors were observed in the different environments. Furthermore, the progression of the Pareto curves proves that the best individuals of the last generations can favorably compete with hand-designed controllers in the evolution environment. Now, the main problems we have to address at this point are that: (1) the extreme variability of the best individuals from one experience to the other limits the usability and the generality of the evolved controllers, and (2) the trajectories obtained with the best individuals are often chaotic and not very efficient.

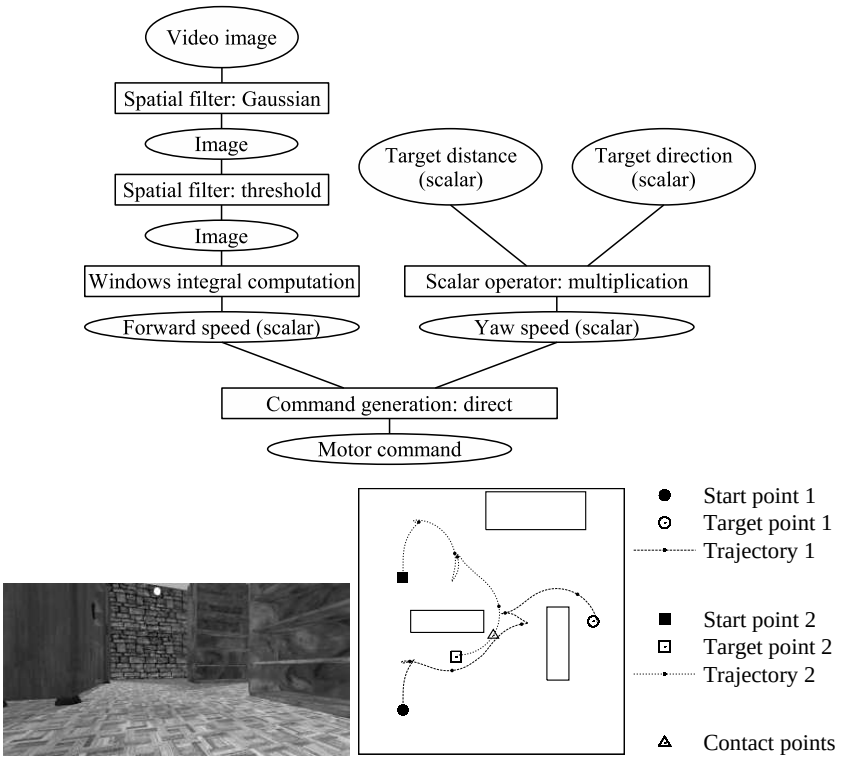


Fig. 8: Example of an algorithm evolved in the bookshelves environment

5.2 Comparison of different strategies

It is obvious that the size of the optimization space, corresponding to all the constructible algorithms is huge, and that only a tiny part of this space can be explored with the 40,000 evaluations. In order to limit the variability of the experiments and to get smoother trajectories, we have decided to use a 2-phase evolution using the imitation based evaluation, with the objective to guide the optimization process toward more promising regions of the controller space. Hence, in this section, the evolution is split into 2 phases: the overall number of evaluations remains the same, but, during the first 50 generations, the fitness functions correspond to formula 3, every candidate algorithm using as input the video sequence and the command sequence recorded by a hand guided robot. For the 50 last generations, we use the same goal accession *vs* contact fitness functions as in Sec. 5.1 (i.e. using formula 2).

In Fig. 9, we show the resulting trajectories using this method, compared with the one-phase evolution presented in the previous section, and with two other classical methods used to guide the evolution, that will be presented in further detail. To get an idea of the variability of the different systems, we have realized several experiments (every experiment corresponding to a set of 40,000 evaluations) for each type of evolution. In Fig. 9, we display for each system one individual from the worst experiment, and one individual from the best experiment. The ranking of two different experiments

can generally be done in an objective way, as long as the Pareto curves of their last generation do not cross each other, which is often the case. The choice of one individual in the Pareto curve is more subjective, and was selected here by using the "visually best" trajectory.

The controllers obtained in one phase (Fig. 9(a)), have been presented in the previous section. The chaotic character of the trajectory is particularly visible in the individual issued from the worst experiment (bottom).

Fig. 9(b) corresponds to incremental evolution, which is a classical method used to improve the evolution. The principle is to divide the evolution into several phases corresponding to different environments with increasing complexity [14]. In our case, we have divided the evolution in 3 phases, using the realistic synthesis environment with 1, 2 then 3 bookshelves. What we observe in this case is that the evolution immediately provides adapted individuals in the simplest environment, which is quite trivial. In the intermediate environment, it manages to improve the algorithms performance a little, but in the final environment it generally fails to improve the controller's behavior. The main problem is that in our case it is very difficult to design an evolving environment with increasing complexity. It seems that better results could be obtained by increasing the difficulty in a more gradual way, but this would require deeper modifications of the simulation protocol.

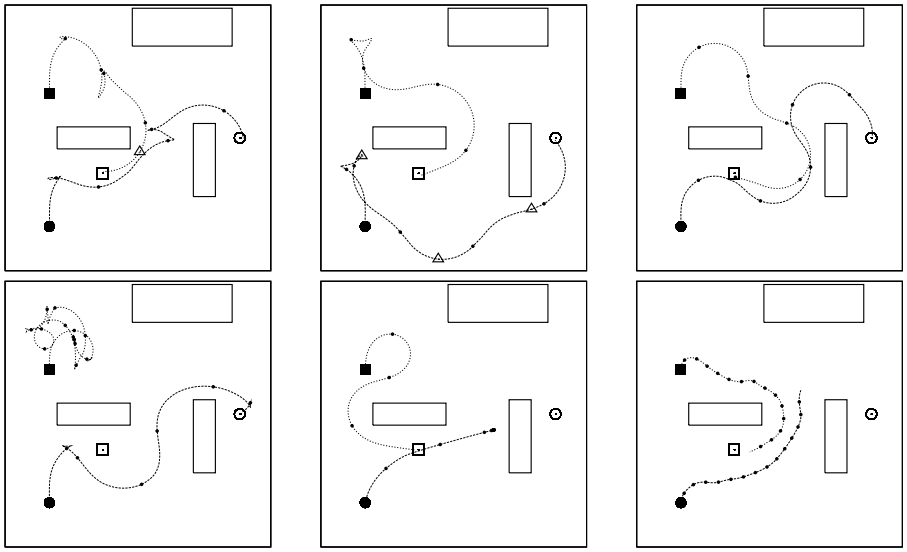
Another classical method to guide the evolution is seeding (Fig. 9(c)). Its principle is to introduce in the evolution individuals with acceptable performances. In our case, we simply added in the initial population the individual corresponding to the hand-design algorithm for the specific environment. What we observe in this case is that the evolution manages to improve the performance with respect to the seed, but that the structure (genotype) of the evolved individuals is always very close to that of the seed, which means that this approach seems to drastically limit the innovation within the algorithms.

Finally, the two-phase evolution (Fig. 9(d)) seems the most stable with respect to the different experiments (Note the little difference between the best and worst experiment). The evolved controllers are efficient and rapid in the evolution environment, and the resulting trajectories are much smoother than in other cases, which is clearly a benefit from an energetic point of view. Another important advantage of two-phase evolution with respect to the other strategies is that it is much easier to implement: recording a video sequence from a hand-guided robot is indeed straightforward, compared to conceiving a gradual complexity increase in the environment, or designing a visual controller by hand.

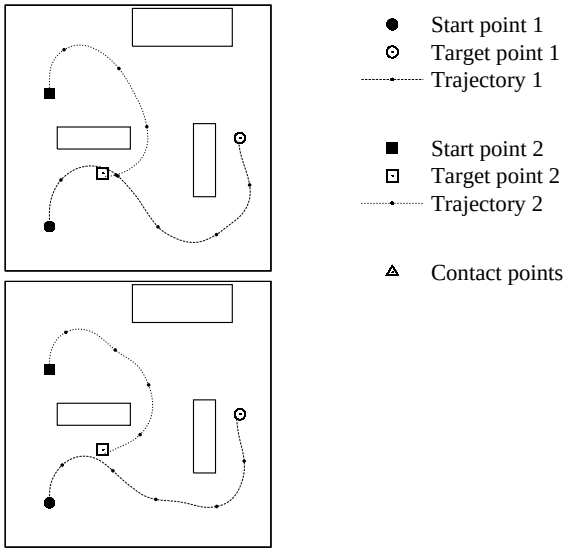
5.3 Generalization performances

In this section, we discuss the ability of the evolved controllers to generalize obstacle avoidance behavior, i.e. we create an environment whose visual appearance is the same as the evolution environment, but the geometry and location of the obstacles have been changed. Fig. 10 shows the results for the different evolution strategies presented in the previous section.

We can see that the generalization performances are poor for all the strategies. The first explanation that can be given for this problem is over-learning: the best evolved individuals have not only learned to avoid obstacles from their appearance or apparent motion, but they also have learned the geometry and location of the different obstacles

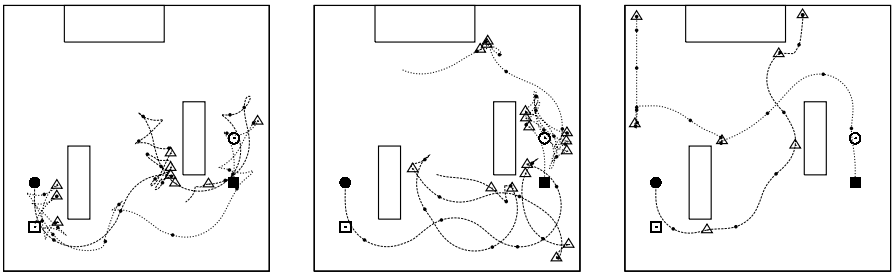


(a) One-phase evolution (b) Incremental evolution (c) Seeded evolution

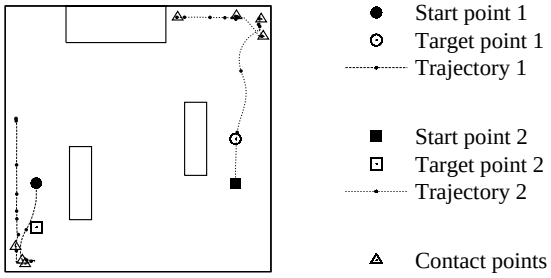


(d) Two-phase evolution

Fig. 9: Comparison of trajectories produced by controllers evolved with different kinds of evolution process. Top: best experiment. Bottom: worst experiment.



(a) One-phase evolution (b) Incremental evolution (c) Seeded evolution



(d) Two-phase evolution

Fig. 10: Performance of the different controllers (designed with a free structure) in a test environment where the obstacles, and start and target points, have been moved.

in the evolution environment. The first idea to develop a more position-independent controller was to increase the number of courses that should be performed by each individual in its evaluation process (Those courses are referred to as “learning courses” from now on). Changing the number of learning courses from 2 to 4 resulted in better generalization performances (see Fig. 11), but consequently the evolution time was multiplied by 2.

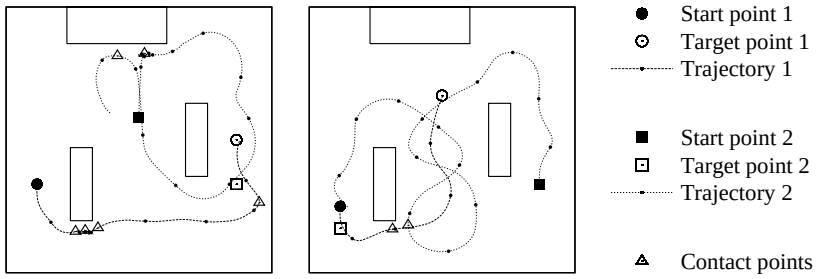


Fig. 11: Generalization performance of a 2-phase evolved controller using a free structure and 4 learning courses. The two figures show 4 trajectories obtained on test courses that are all different from the learning courses.

In fact, a more fundamental problem in the structure of the controllers can explain the difficulty in generalizing the obstacle avoidance behavior. In the structure-free grammar, i.e. without *a priori* in the structure of the controller, it is very difficult for the algorithms to automatically develop a trade-off between obstacle avoidance and target reaching, as the target heading information is just an input of the algorithm, and nothing guarantees a relevant use of this input. This is why we also developed a more restricted grammar, making a more explicit distinction in the controller structure between goal reaching and obstacle avoidance behaviors. The structure restricted grammar has been presented in detail in Section 3.3.

Fig. 12 shows the performances of an individual evolved with structure restricted grammar on 4 courses different from the learning courses. The generalization performance in this case is much better, due to explicit separation between target reaching and obstacle avoidance within the algorithms structure, thus making it easier for the emergence of a real obstacle detection and avoidance behavior. Another positive impact of the structure restricted grammar is that it usually simplifies the structure of the algorithms, thus lowering the evaluation time of an individual on one course. This allows for a greater number of courses to be evaluated, this further improving the generalization capacity.

Regarding the quality of the results, it is noticeable that the individuals such as the one shown Fig. 12, considered to be the "most evolved" algorithms obtained from simulation, may fail the goal reaching task or hit some obstacles. However, it should be pointed out that: (1) We do not consider in this work the goal reaching task as fundamental; it is rather a way to enforce a certain heading and keeping the robot from staying still or turning around. (2) The most generic algorithms are those which globally best perform on different environments, but they also make more errors than the less generic individuals on a specific environment.

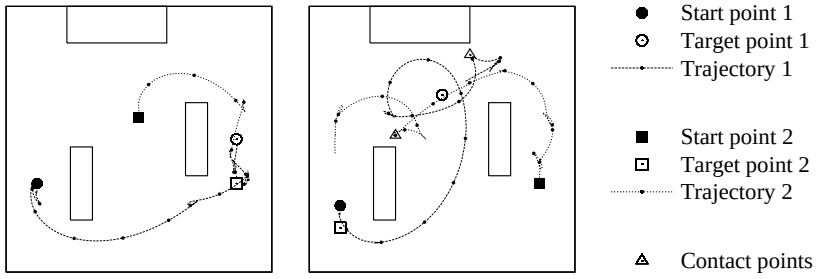


Fig. 12: Generalization performance of a 2-phase evolved controller using a restricted structure and 4 learning courses. The two figures show 4 trajectories obtained on test courses that are all different from the learning courses.

An open problem regarding the generalization capacities of the algorithm, is how to select the best individuals in terms of generalization, and what is the best moment to stop the evolution process to avoid over-learning and favor the generalization abilities. Gagné [13] proposed an interesting solution to address this problem. The idea is to evaluate all the individuals from the Pareto front in a validation environment, which is different from the evolution environment. When the performances on the validation environment decrease, it means that we enter the phase of over-learning and

the evolution is stopped. Naturally, this biases the evolution results with respect to the validation environment. As such, a third distinct environment (test data) should be used to test the generalization performance.

6 Experiments on the real robot

In this section, we show the results obtained on the real robot platform Pioneer 3DX. Using an off-line evolution method based on the imitation strategy, we present and discuss some results obtained for a corridor centering task, first around the evolution environment, and then in another unseen environment.

6.1 Evolving controllers in real environment

When evolving controllers in a real environment, it is clearly not feasible to perform the same evaluation protocol as in simulation, because that would imply repeating a huge number of experiences with exactly the same conditions. However, the imitation strategy presented above (Sec. 4.2) can be applied on synthesis or real sequences alike, therefore we have implemented an off-line evolutionary algorithm based on the imitation only, using video sequences recorded by the robot Pioneer 3DX guided by hand.

Our first experiments have shown that is was difficult to obtain acceptable obstacle avoidance performances using long sequences involving complex trajectories. On the contrary, very promising results have been obtained relatively quickly, using a large set (around 20) of very short (approx. 2 or 3 s) video sequences. In those sequences, the robot was placed in different positions along the corridor, with its optical axis forming an angle of around 30° with the wall. The robot was then guided manually in such a way that it moves away from the closest wall and centers itself in the corridor. The top of Fig. 13 shows 3 examples of learned trajectories (red arrows) in a corridor of our laboratory. At the bottom of the figure, 3 images extracted from these sequences are shown, with the corresponding angular speed command represented as the yellow arrow. The forward speed was approximately constant (30 cm/s) in all the sequences and is not represented in the figure.

The genetic algorithm is then run in one phase, using 100 generations and fitness functions equal to the mean values of Y and F functions of formulas 3 over the 20 sequences. As there is no starting and target points here, the expected behavior when plugging the evolved algorithm in the robot is not a precise displacement, but rather a wandering behavior, allowing to explore the environment with no pre-defined objective.

The progression of the Pareto fronts in the successive generations shows that learning the forward speed is straightforward, which is logical since the forward speed was almost constant in all the command sequences. Consequently, the system acts like mono-objective genetic algorithm, as shown by a significant progression in the yaw speed command error values Y.

Fig. 14 shows an example of one of the best evolved algorithms using this method. The filter chain used to generate the angular speed is not shown completely, as it contains more than 29 operators. This complexity is partially due to bloating, but it is also a solution found by the evolution process to overcome a limitation of our system. This algorithm is mostly based on the use of Sobel filters, to detect the edge

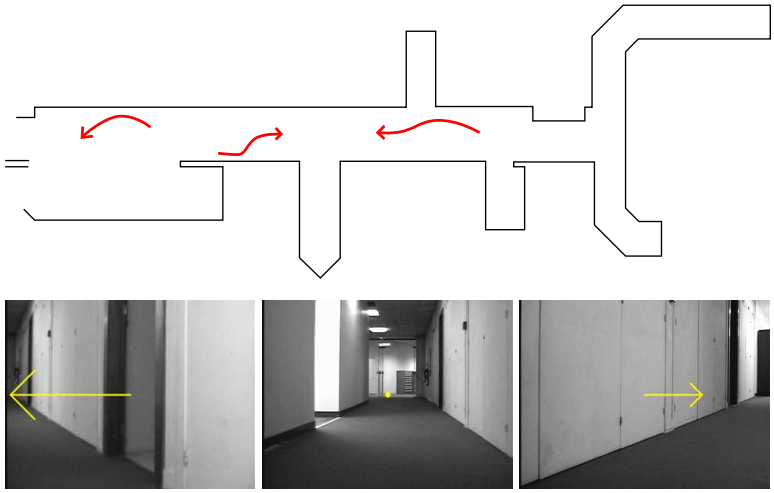


Fig. 13: Top: Examples of trajectories used for the learning base. Bottom: Examples of recorded images and commands.

between the floor and the walls. As the command generation operator needs to extract a scalar from an image, and as the functions that produce scalars from images are based on combination of integral measures on left and right region in the images, the long operator sequence of the filter chain is mainly used to enlarge the edge in order to enhance the detection of the floor border. This could have been done more efficiently using a morphological dilatation filter (not in the available primitives), but the fact that the evolution found a way to compensate a limitation of the system is a good indication of its adaptation capabilities.

Fig. 15 illustrates the method used by this evolved algorithm to compute the motor command. The filter chain highlights and enlarges the boundary between the floor and the wall, as well as the more contrasted zone at the end of the corridor. In the resulting image, the wall appears completely white and the boundary is darker. This difference is used by the windows integral computation operator to produce a command that drives the robot away from the wall: the resulting command depends on the difference between the mean pixel value of each red window (right image). On the left image, we display a red arrow corresponding to the command issued by the algorithm, and a yellow one which is the command that was recorded when the robot was manually guided.

6.2 Generalization performances

In order to test the robustness and generalization performance of these evolved controllers, we placed the robot at different positions in the corridor and allowed it to be driven by the evolved algorithm. The robot should move to the end of the corridor without hitting the walls. We placed the robot so that the direction it faces and the corridor make an angle of approximately 30°. In this position, the problem is possible to solve without being trivial. We made about ten tests with different starting positions. Each time, the robot managed to reach the end of the corridor except once where

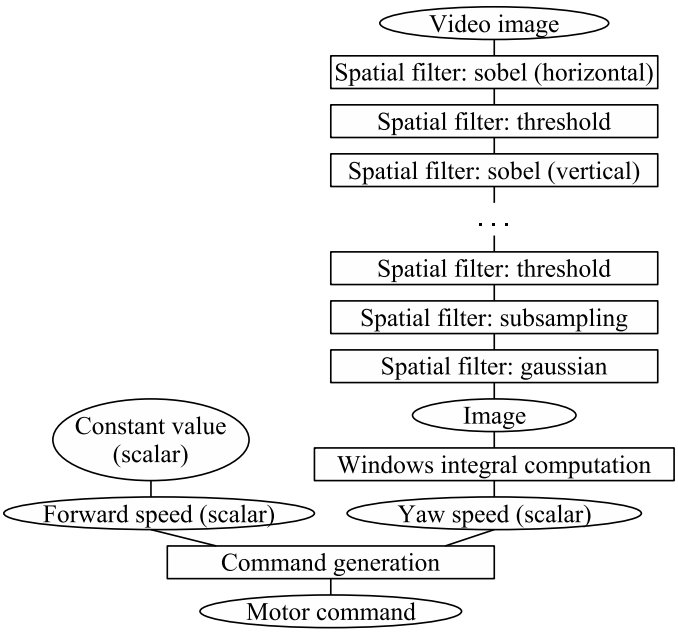


Fig. 14: Example of an evolved algorithm.

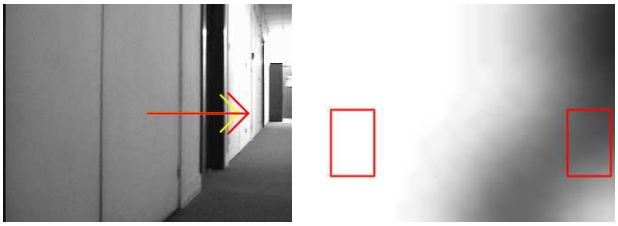


Fig. 15: Left: Resulting command from the evolved algorithm on an image from the learning base. Right: The same image transformed by the filter chain.

it turned into one of the openings in the wall. In one test, it even turned at the end of the corridor to go into the smaller corridor on the right of the map. Fig. 16 shows two trajectories, together with some sample images and the corresponding angular speed command.

We also tested this evolved algorithm in another corridor, visually different from the previous one. In one direction the robot reaches the end of the corridor without problem. On the return trip it failed against two obstacles as shown on Fig. 17. Nevertheless this result is encouraging since this corridor is very different from the one that was recorded in the learning base. The second and third images of Fig. 17 show the 2 obstacles that the algorithm failed to avoid.

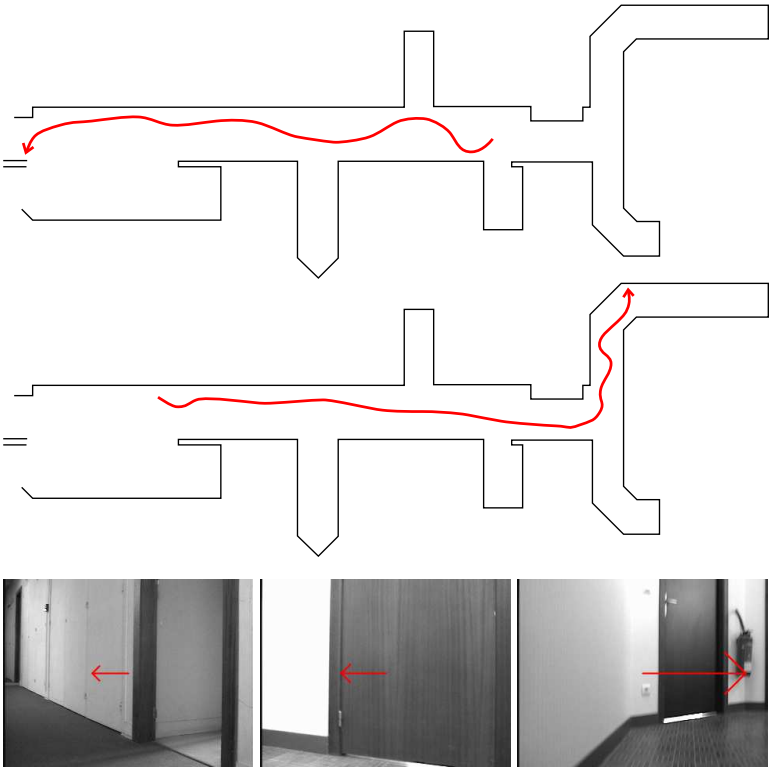


Fig. 16: Top: Trajectories followed by the robot when driven by an evolved algorithm. Bottom: Example images and commands issued by the algorithm in the generalization tests.

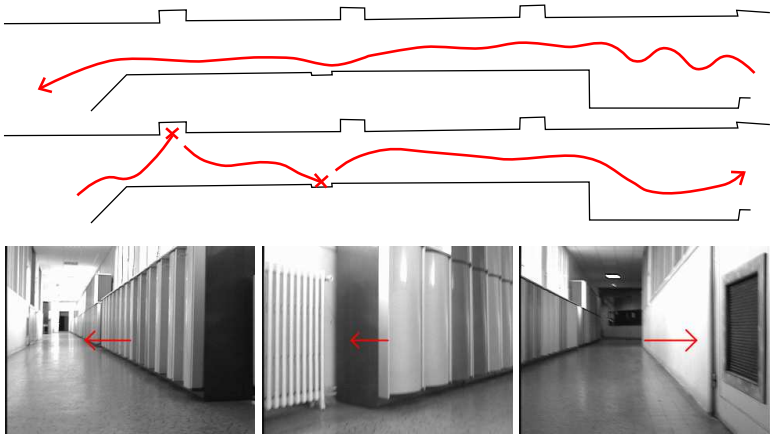


Fig. 17: Top: Trajectories followed by the robot driven by the evolved algorithm in another corridor. Bottom: Images and commands issued by the evolved algorithm in this other corridor.

7 Conclusion

We have presented in this article a genetic programming system to automatically design vision algorithms from a collection of primitives and a set of construction rules. The originality of our approach lies in the fact that the visual task is considered as a whole, with the lowest levels of processing also taking part of the learning process. From a computational point of view, the collection of primitives could be ideally identified with the instruction set(s) of the processor(s) used to compute the algorithm. We have made a more realistic choice from a combinatorial point of view, which consists in constructing the primitive collection from the earliest stages of visual perception (mostly, spatio-temporal filtering). Nonetheless, the choice of the primitives and their complexity level is an interesting open problem that should be addressed in the future.

Although we have concentrated here on the obstacle avoidance problem, we believe that the proposed system can be adapted to automatically design other artificial vision tasks. This can be easily done as long as an objective evaluation can be determined. Typically, such automatic design can be envisaged for: visual categorization, salient features detection, room recognition, etc.

The results we have obtained in simulation have shown that our system was able to provide interpretable controllers adapted to the visual environment. The 2-phase evolution has proven an efficient way to guide the evolution towards more promising solutions. Regarding the generalization capabilities, the restricted structure controllers behaved better than the free structured ones, but at the price of an important *a priori* in the structure of the algorithms. Finding a trade-off between goal accession and obstacle avoidance remains a difficult problem. In that sense, it is possible that the combination between two objectives with very different cognitive levels (obstacle avoidance and goal accession) constitutes a fundamental difficulty.

Unexpectedly, the final results obtained on the real robot were better than those obtained in simulation, possibly due to the removal of the goal accession constraint. The task to achieve was simpler (wandering in a corridor while avoiding the walls), but the results, particularly in generalization were much better. As it turns out, this type of imitation based learning is both easy to implement and promising in real environment, thus we will continue to investigate future solutions in a similar manner.

One limitation of our system is that it is subject to bloating. Several solutions have been proposed to limit bloating in genetic programming. One such example is the inclusion of program size as an independent criterion in a multi-objective evolutionary algorithm, which has been shown to produce efficient and small-sized programs (see [4] for instance). Such adaptation should be envisaged in the future.

Finally the most important perspective of our work is the adaptation to on-line learning. Presently, the proposed system only performs off-line evolution. To reach the global objective of our research, which is providing the mobile robots with more autonomy, we must also integrate a certain level of reactive adaptation. Our method can be adapted to that in several ways. A first level of adaptation can be experimented while keeping the structure of the existing off-line evolution system: a collection of evolved algorithms can be plugged into the robot and a subsequent on-line learning is used to select and parameterize the algorithm which is best adapted to the current context. Another level which would to be interesting to pursue is to modify the structure of the algorithms, which is essentially bottom-up, in order to explicitly allow the implementation of top-down mechanisms. Typically, the size and position of the integration windows for the extraction primitive could vary according to the image content; the na-

ture of the spatio-temporal filters could also be made variable according to the context. Such mechanisms are not excluded by our current system, but could be more explicitly taken into account in the structure of the controllers. In this purpose, active vision and attention mechanisms [22,12,31] provide frameworks that could help to improve the automatic design of the algorithms and hence will be investigated in our future works.

Acknowledgements This work was supported by the French Armaments Procurement Agency (DGA). The authors would like to thank the anonymous reviewers for their helpful comments, and Mr Toby Low for scientific and English proofreading.

References

1. http://playerstage.sourceforge.net/index.php?src=gazebo.
2. http://www.ode.org/.
3. http://www.opengl.org/.
4. S. Bleuler, M. Brack, L. Thiele, and E. Zitzler. Multiobjective Genetic Programming: Reducing Bloat Using SPEA2. *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, 1:536–543, 2001.
5. S. Cagnoni. Evolutionary computer vision: a taxonomic tutorial. In *Eighth International Conference on Hybrid Intelligent Systems*, pages 1–6, Los Alamitos, CA, 2008. IEEE Computer Society.
6. D. Coombs, M. Herman, T.H. Hong, and M. Nashman. Real-time obstacle avoidance using central flow divergence, and peripheral flow. *IEEE Transactions on Robotics and Automation*, 14(1):49–59, 1998.
7. K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.
8. E. Dunn, G. Olague, and E. Lutton. Parisian camera placement for vision metrology. *Pattern Recognition Letters*, 27(11):1209–1219, 2006.
9. M. Ebner. On the evolution of interest operators using genetic programming. In R Poli, W.B. Langdon, M. Schoenauer, T. Fogarty, and W. Banzhaf, editors, *Late Breaking Papers at EuroGP'98: the First European Workshop on Genetic Programming*, pages 6–10, Paris, France, April 1998. The University of Birmingham, UK.
10. M. Ebner and A. Zell. Evolving a task specific image operator. *Evolutionary image analysis, signal processing and telecommunications: First european workshop, EVOIASP*, pages 74–89, 1999.
11. M. Ebner and A. Zell. Centering behavior with a mobile robot using monocular foveated vision. *Robotics and Autonomous Systems*, 32(4):207–218, 2000.
12. D. Floreano, T. Kato, D. Marocco, and E. Sauser. Coevolution of Active Vision and Feature Selection. *Biological Cybernetics*, 90(3):218–228, 2004.
13. C. Gagné, M. Schoenauer, M. Parizeau, and M. Tomassini. Genetic Programming, Validation Sets, and Parsimony Pressure. In *Proceedings of EuroGP 2006*, volume 3905 of *Lecture Notes in Computer Science*, pages 109–120. Springer, 2006.
14. F. Gomez and R. Miikkulainen. Incremental Evolution of Complex General Behavior. *Adaptive Behavior*, 5(3-4):317–342, 1997.
15. B.K.P. Horn and B.G. Schunck. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1981.
16. I. Horswill. Polly: A vision-based artificial agent. *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pages 824–829, 1993.
17. L. Lacassagne, A. Manzanera, J. Denoulet, and A. Mérigot. High performance motion detection: some trends toward new embedded architectures for vision systems. *Journal of Real-Time Image Processing*, 4(2):127–146, 2009.
18. Y. LeCun, U. Muller, J. Ben, E. Cosatto, and B. Flepp. Off-road obstacle avoidance through end-to-end learning. In *Proceedings of the Conference on Neural Information Processing Systems*, pages 739–746, 2006.
19. L.M. Lorigo, R.A. Brooks, and W.E.L. Grimson. Visually-guided obstacle avoidance in unstructured environments. *Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1:373–379, 1997.

20. T. Low and G. Wyeth. Learning to avoid indoor obstacles from optical flow. In *Proceedings of the 2007 Australiasian Conference on Robotics and Automation*, pages 1–10, Brisbane, Australia, 2007.
21. B.D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. *Proc. DARPA Image Understanding Workshop*, pages 121–130, 1981.
22. D. Marocco and D. Floreano. Active vision and feature selection in evolutionary behavioral systems. *From Animals to Animats*, 7:247–255, 2002.
23. M.C. Martin. Evolving visual sonar: Depth from monocular images. *Pattern Recognition Letters*, 27(11):1174–1180, 2006.
24. J. Michels, A. Saxena, and A.Y. Ng. High speed obstacle avoidance using monocular vision and reinforcement learning. *Proceedings of the 22nd international conference on Machine learning*, pages 593–600, 2005.
25. L. Muratet, S. Doncieux, Y. Brière, and J.-A. Meyer. A contribution to vision-based autonomous helicopter flight in urban environments. *Robotics and Autonomous Systems*, 50(4):195–209, 2005.
26. R.C. Nelson and J. Aloimonos. Obstacle avoidance using flow field divergence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(10):1102–1106, 1989.
27. G. Olague and C. Puente. Parisian evolution with honeybees for three-dimensional reconstruction. *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 191–198, 2006.
28. O. Pauplin, J. Louchet, E. Lutton, and A. De La Fortelle. Evolutionary Optimisation for Obstacle Detection and Avoidance in Mobile Robotics. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 9(6):622–629, 2005.
29. C.W. Reynolds. An evolved, vision-based model of obstacle avoidance behavior. *Artificial Life III*, pages 327–346, 1994.
30. A. Saxena, S.H. Chung, and A.Y. Ng. 3-D Depth Reconstruction from a Single Still Image. *International Journal of Computer Vision*, 76(1):53–69, 2008.
31. M. Suzuki. *Enactive Robot Vision*. PhD thesis, École Polytechnique Fédérale de Lausanne (EPFL), 2007.
32. L. Trujillo and G. Olague. Synthesis of interest point detectors through genetic programming. *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 887–894, 2006.
33. I. Ulrich and I. Nourbakhsh. Appearance-based obstacle detection with monocular color vision. *Proceedings of AAAI Conference*, pages 866–871, 2000.
34. J. Walker, S. Garrett, and M. Wilson. Evolving controllers for real robots: A survey of the literature. *Adaptive Behavior*, 11(3):179–203, 2003.
35. P.A. Whigham. Grammatically-based genetic programming. *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 33–41, 1995.