Generalization performance of vision based controllers for mobile robots evolved with genetic programming

Renaud Barate, Antoine Manzanera

Generalization Performance of Vision Based Controllers for Mobile Robots Evolved with Genetic Programming

Renaud Barate
ENSTA - UEI
Paris, France
renaud.barate@ensta.fr

Antoine Manzanera
ENSTA - UEI
Paris, France
antoine.manzanera@ensta.fr

ABSTRACT

We present a genetic programming system to evolve vision based obstacle avoidance algorithms. In order to develop autonomous behavior in a mobile robot, our purpose is to design automatically an obstacle avoidance controller adapted to the current context. Using a simulation environment, several trajectories are used to evaluate the performance of an evolving population of controllers. Two different methods are proposed and compared. In the first one (structure-free controller), we discard any *a priori* in the structure of the algorithm. We show that the evolved controllers behave well in the evolution environment, but have very limited generalization capabilities. In the second method (structure restricted controller), the compromise between the two antagonist functions of the robot navigation, i.e. obstacle avoidance and target reaching, is explicitely integrated in the structure of the algorithms. We show that controllers evolved with this second method are more generic. We give hints in the discussion to detect overlearning earlier in the evolution process, so as to quickly evolve controllers with good generalization abilities.

Categories and Subject Descriptors

I.2.2 [**Artificial Intelligence**]: Automatic Programming—*program synthesis*; I.4.8 [**Image Processing and Computer Vision**]: Scene Analysis—*depth cues*

General Terms

Algorithms

Keywords

Vision, obstacle avoidance, robotic simulation, generalization

.

1. INTRODUCTION

Our goal is to automatically design vision based controllers for robots. We chose obstacle avoidance as target application as it is a necessary task for any mobile robot. One commonly used method to perform obstacle avoidance using a single camera is based on the calculation of optical flow. Optical flow represents the perceived movement of the objects and can be used to compute a fairly accurate approximation of their distance. It has been used recently for obstacle avoidance with an autonomous helicopter for instance [12]. However, systems based on optical flow don't cope well with thin or lowly textured obstacles. On the other hand, more simple primitives can be used to extract useful informations in the image, generally based on texture. For instance, Michels implemented a system to estimate depth from texture information in outdoor scenes [11]. Other obstacle avoidance systems use this kind of information to discriminate the floor from the rest of the scene and calculate obstacle distances in several directions [6, 7, 16]. Nevertheless those methods suppose that the floor may be clearly discriminated and they neglect potentially useful information from the rest of the scene.

As none of those methods is robust enough to deal with all kind of environments, our system should be able to automatically select the kind of visual features that allow obstacle detection in a given environment. More, it should adapt and parameterize the whole obstacle avoidance controller to use those features efficiently. For that, we propose to use genetic programming as it allows us to evolve algorithms with little *a priori* on their structure and parameters. For now, the evolution is an offline process using a simulation environment for the evaluations where the robot must reach a target point while avoiding obstacles.

Evolutionary techniques have already been used for robotic navigation and the design of obstacle avoidance controllers but in general vision is either overly simplified or not used at all. For instance, Harvey used a 64×64 pixels camera but simplified the input to a few average values on the image [5]. Marocco used only a 5×5 pixels retina as visual input [8]. A description of several other evolutionary robotics systems can be found in [10, 17] but most of them rely only on range sensors. On the other hand, genetic programming has been proved to achieve human-competitive results in image processing systems, e.g. for the detection of interest points [3, 15]. Parisian evolution has also been shown to produce very good results for obstacle detection and 3D reconstruction but those systems need two calibrated cameras [14, 13].

To our knowledge, only Martin tried evolutionary tech-

niques with monocular images for obstacle avoidance [9]. 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 image. 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.

We first tried to evolve controllers with very little *a priori* on their algorithmic structure. In this case, all the algorithm is designed by the evolutionary process. The advantage of this freedom is that every kind of algorithm can potentially be evolved this way, even those that we didn't think of while designing the system. Nevertheless the compromise between moving toward the target point and avoiding obstacles is very difficult to achieve with this kind of free structure. In general the evolution will either find controllers moving toward the target point but with limited obstacle avoidance abilities, or controllers that avoid obstacles but that won't move toward the target location. Therefore this kind of controllers won't generalize well to slightly different conditions.

To overcome this problem, we designed a more restricted algorithmic structure that facilitates the compromise between avoiding obstacles and moving toward the target location. We will show that those controllers are more generic as they can reach the target point while avoiding obstacles even when the environment changes.

2. STRUCTURE OF THE CONTROLLERS

2.1 The Vision Algorithms

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. Then these features are extracted, i.e. represented by a small set of scalar values. Finally these values are used for a domain dependent task, here 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. Then we 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, the root being output command, 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. 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, Laplacian, threshold, Gabor, difference of Gaussians, Sobel and subsampling filter.
- **Temporal filters** (*input: image, output: image*): pixel-to-pixel min, max, sum and difference of the last two frames, and recursive mean operator.
- **Optical flow** (*input: image, output: vector field*): Horn and Schunck global regularization method, Lucas and Kanade local least squares calculation and simple block matching method [1]. 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 or Manhattan norm computation, and 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:

     $$R = \alpha_0 + \sum_{i=1}^{n} \alpha_i \mu_i$$
     $$\mu_i = \mu_{\mathrm{L}i} + \mu_{\mathrm{R}i} \text{ or } \mu_i = \mu_{\mathrm{L}i} - \mu_{\mathrm{R}i}$$

     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 and a temporal mean calculation.
- **Command generation** (*input: two scalars, output: command*): The motor command is represented by two scalar values: the requested linear and angular speeds.

Most of those primitives use parameters along with the input data to do their calculations (for example, the standard deviation value for the Gaussian filter). Those parameters are specific to each algorithm; they are randomly generated when the corresponding primitive is created by the genetic programming system described in Sect. 3.4.

2.2 Structure-Free Controllers

For this first kind of controllers, all the algorithmic tree will be built by the evolutionary process. We add two scalar input variables: the target distance and the target direction, and one scalar operator: an if-then-else test. This allows the evolutionary process to create more complex and non-linear combinations between those scalar input variables and the

scalars issuing from the vision part of the algorithm. Fig. 1 shows an example controller built with this structure. The algorithm is in this case a single tree, and all the primitives between the input data and the resulting motor command are created and parameterized by the evolution.

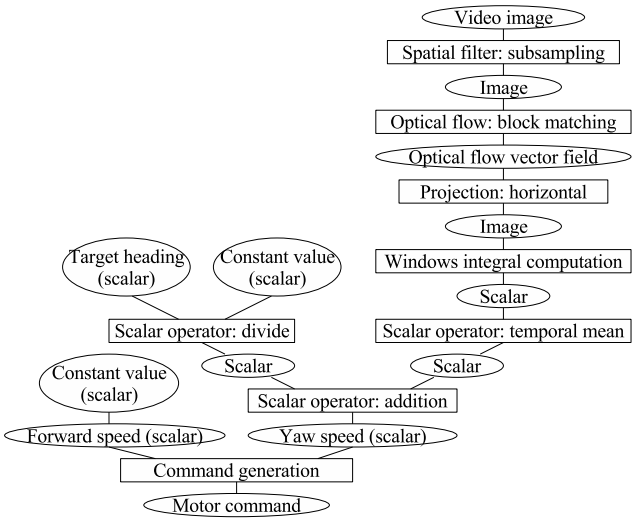

Figure 1: Example of a structure-free controller for obstacle avoidance based on optical flow. Rectangles represent primitives and ellipses represent data.

2.3 Structure-Restricted Controllers

Here, we restrict the global structure of the controllers to facilitate the compromise between avoiding obstacles and going toward the target point. 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.

The first vision algorithm (obstacle detection) is in fact always linear, 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.

The second vision algorithm (command generation) can use as 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. Here, several scalar values can be produced with different filter chains and these values are combined using scalar operators. The final step will generate a motor command using the resulting scalar value(s). Fig. 2 illustrates this restricted structure with an example.

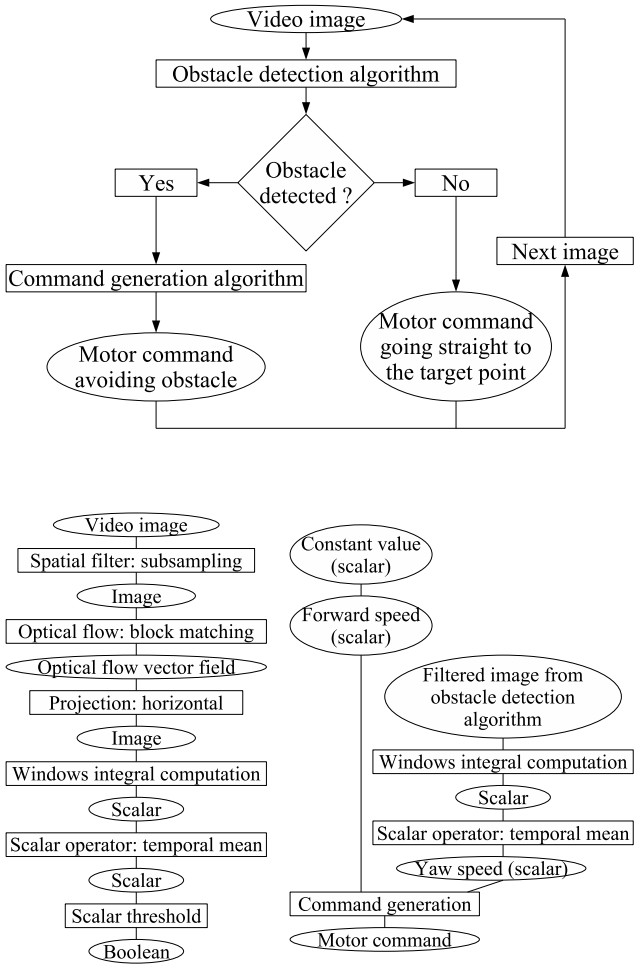

Figure 2: Example of a structure-restricted controller. Top: overview of the global fixed structure. Bottom left: an example of obstacle detection algorithm. Bottom right: an example of command generation algorithm. Rectangles represent primitives and ellipses represent data.

3. THE EVOLUTION PROCESS

3.1 Simulation Environment

For the evaluation of the different obstacle avoidance algorithms, 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. The simulator uses ODE physics engine for the movement of the robot and collisions detection and OpenGL 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. 3). All the obstacles are immovable to prevent the robot from just pushing them instead of avoiding them. In each experiment, the goal of the robot is to go from a given starting point to a goal location without hitting obstacles.

Figure 3: Snapshot of the simulation environment, containing three bookshelves in a 6 m $\times$ 6 m closed room.

3.2 First Phase: Evolution of Algorithms that Imitate a Behavior

Due to the simulation time and mainly to the complexity of the vision algorithms, we're limited to about 40,000 evaluations to keep the evolution time acceptable (a few days at most). We therefore face a common problem with genetic programming systems, that is the state space is immense compared to the number of evaluations. More, the fitness landscape is very chaotic so the evolution can easily get stuck in a local minima of the fitness function. To overcome those problems, we use a two-phase evolution process: the first phase is based on imitation to quickly guide the evolution toward promising solutions, while the second phase will evolve more robust controllers by using a functional evaluation in the simulation environment.

With this kind of problem, it is very difficult to manually design even a mediocre controller but it is very easy to manually guide the robot toward the target point while avoiding obstacles. So we can obtain a good example of an efficient behavior by recording the video sequence and command parameters while we guide the robot. In this first phase, we will try to evolve algorithms that imitate this efficient example behavior.

The population is initialized with random algorithms and the evolution lasts for 50 generations. For the evaluation of the algorithms, we replay the recorded sequence and compare the command issued by the evaluated 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:

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

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.

3.3 Second Phase: Evolution of Efficient Solutions

The goal of this second phase is to produce more robust and generic controllers. For that, we will evaluate them on their ability to really avoid obstacles and reach the target location in the simulation environment. The population is initialized with the final population of the first phase and the evolution lasts for 50 more generations. For the evaluation, we place the robot at the fixed starting point and let it move in the environment during 30 s 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:

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

where t_G is the time needed to reach the goal in seconds, $t_{\max}$ is the maximum time in seconds (here 30 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 the two phases of evolution, the evaluations consist in fact in four runs with different starting points and goal locations. Final scores are the means of the scores obtained for the four runs. The starting points are fixed because we want to evaluate all algorithms on the same problem. We chose four runs as a tradeoff between generalization behavior and evaluation time but more tests would be needed to determine the optimal number of runs and the precise consequence on generalization.

3.4 The Genetic Programming System

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 [18] 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.

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

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

Table 1: 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.

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 issuing from identical non-terminal nodes in two different individuals. The mutation consists in replacing a subtree by a newly generated one. Table 1 presents the exhaustive grammar that we used in all our experiments.

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 (F and Y in the first phase, G and C in the second phase). 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[2].

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 15 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. Those parameters were determined empirically with a few tests using different values. Due to the length of the experiments, we didn't proceed to a thorough statistical analysis of the influence of those parameters.

4. EXPERIMENTS AND RESULTS

4.1 Analysis of the Evolved Controllers

In total, each experiment lasts for 100 generations, that is 40,000 evaluations. Fig. 4 presents the final Pareto fronts for the evolution of structure-free and structure-restricted controllers. Those final results are comparable in performance for the two kinds of controllers. Structure-restricted controllers generally come closer to the target points but structure-free controllers are better for avoiding obstacles.

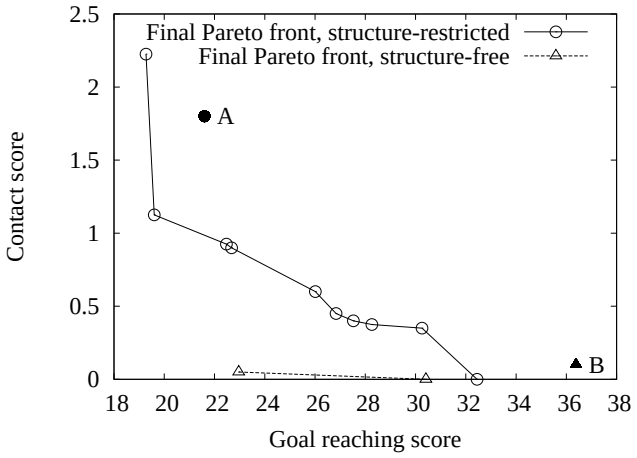

Figure 4: Final Pareto fronts of the solutions obtained with the two kinds of controllers. Those fronts are created by taking the best individuals in the final population in all the four islands and keeping only non-dominated ones. Individuals A and B are the examples used for the trajectories in the next figures.

The real difference appear when we compare the trajectories of the controllers. For that we take an example solution in the final population of each experiment (individuals A and B in Fig. 4). Those individuals are not on the Pareto

front but they show good generalization abilities (see Sec. 4.2).

Fig. 5 represents the trajectories of the robot in the simulation environment used during the evolution with the structure-restricted controller A. We can see that it applies a very simple back and forth strategy: it goes forward toward the goal as long as there is no obstacle in front of it, and it goes backward turning on itself when it faces an obstacle. Nevertheless this simple strategy is sufficient as the robot reaches the target point in each case and with only a small number of contacts.

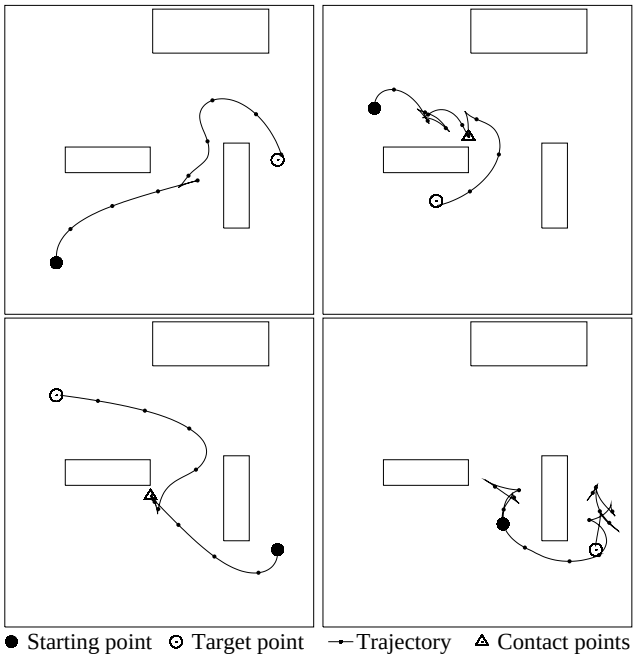

Figure 5: Trajectories of the robot with the structure-restricted controller A in the environment used during the evolution. The small dots are placed every 2 seconds and thus indicate the speed of the robot.

Fig. 6 represents the trajectories in the same environment with the structure-free controller B. Here the robot moves much faster than in the previous case but it doesn't use the target direction information at all, so it will never find the target point unless by chance.

4.2 Generalization Performance

The previous experiment showed the performance of the evolved controllers in the environment that was used for the evolution process but it does not prove that they really can avoid obstacles in a generic way. They could just as well be adapted only to reproduce the trajectory used during the evolution. In this case, they would have learned a trajectory and not a generic obstacle avoidance behavior.

In order to test the genericity of the evolved controllers, we designed a second environment with the same appearance but where the obstacles and the start and target points have been moved around. We also have four different runs with different start and target points. We evaluate all the individuals of the last generation from the previous evolution

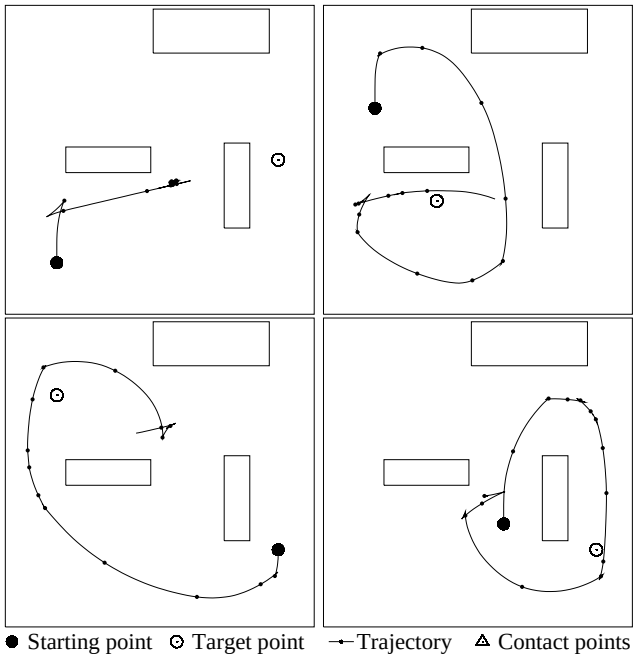

Figure 6: Trajectories with the structure-free controller B in the evolution environment

process. Fig. 7 shows the performance of the individuals evolved before on this new test environment. We chose individuals A and B as they show the best performance in this test environment, but that doesn't mean that they are on the Pareto front with the evolution environment as shown on Fig. 4.

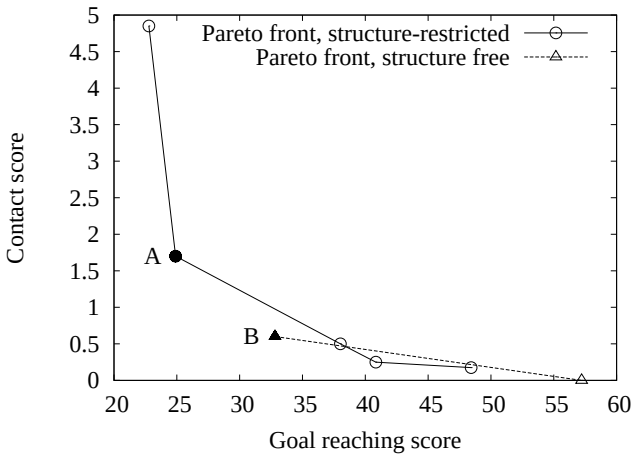

Figure 7: Pareto front of the evolved individuals in the generalization environment.

In this environment, the structure-restricted controllers generally outperform the structure-free ones. This is even clearer when we consider the trajectories of the robot with the two kinds of controllers (Fig. 8 and 9). The structure-restricted controller A adapts quite well to those new conditions and reaches the target point in three out of four test

cases, with only a few more contacts with the obstacles than before. The structure-free controller B still avoids the obstacles correctly but since it does not move toward the target point, it finds it by chance only in one of the test cases.

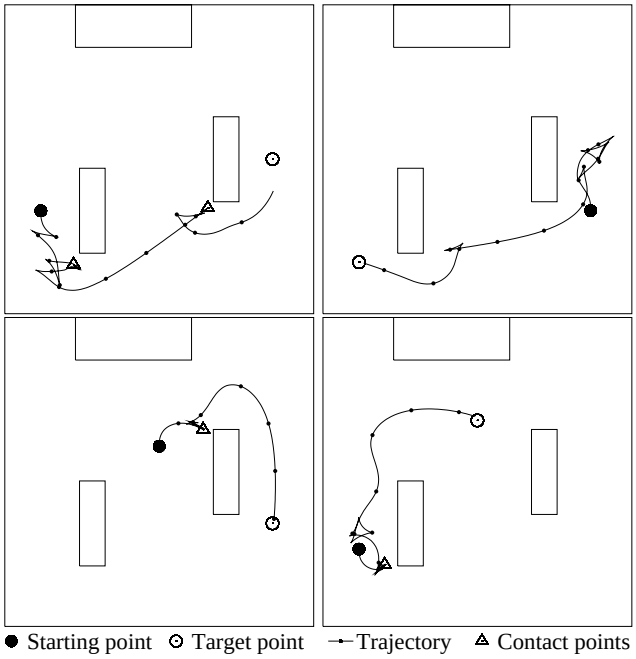

Figure 8: Trajectories in the generalization environment with the structure-restricted controller A

Those experiments show that using a customized structure for the evolution of algorithms can bring significant benefits, especially when we must find a compromise between two different behaviors. Indeed we can evolve very efficient obstacle avoidance controllers or controllers that move toward the goal with a free structure, but only by using a kind of rule-based structure we can evolve controllers that move toward a target point while avoiding obstacles.

Another interesting result is that the best individuals in the environment used during the evolution process are generally not the best ones when we change the environment. This can be explained by the fact that those individuals overlearn the environment used to evaluate them and this decreases their generalization performance. Therefore the question is: how to evolve and select individuals with good generalization abilities? We partially answered this question by using four different test cases in the evaluation environment. This limits the overlearning problem as the evolved controllers must be adapted for the four different trajectories. By evaluating the whole final population on a new environment and selecting individuals performing well with those new conditions, we also choose the solutions that are more likely to generalize well.

But this does not tell us when we should stop the evolution process to avoid overlearning. Gagné [4] proposed to use a three data sets methodology, already commonly used in the Machine Learning community. This consists in testing the individuals on the Pareto front after each generation on a validation set to detect when overlearning begins to occur and therefore to stop the evolution process. But as we have

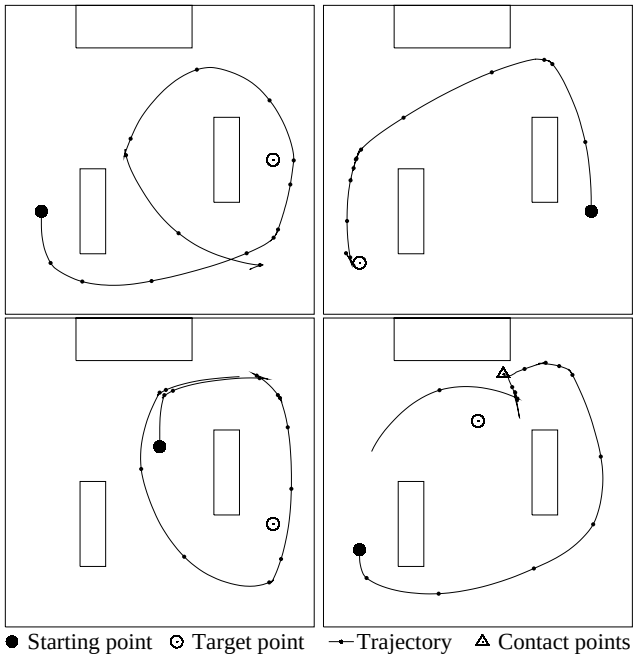

Figure 9: Trajectories with the structure-free controller B in the generalization environment

seen before, the more generic individuals are not always on the Pareto front. We could evaluate the whole population on a new environment after each generation but this would double the evolution time and we would select individuals that are "by chance" adapted to our validation set. Nevertheless we plan to test and adapt this kind of technique in future work in order to select more generic individuals and to shorten the evolution time.

5. CONCLUSION

In this paper, we presented the benefits of using a specific algorithmic structure for the evolution of vision based obstacle avoidance controllers with genetic programming. This structure allows us to facilitate the compromise between going toward a target point and avoiding obstacles. Therefore the controllers evolved with this structure are more generic than structure-free controllers.

Our goal is now to further improve this system to facilitate the detection of different kind of obstacles and to detect overlearning earlier in the evolution process. We will then adapt the system to create controllers for a real robot. Finally, we want to design a high-level controller able to select its behavior depending on the context in real-time, in order to have a fully autonomous and adaptive robot.

6. REFERENCES

[1] J. Barron, D. Fleet, and S. Beauchemin. Performance of optical flow techniques. *International Journal of Computer Vision*, 12(1):43–77, 1994.

[2] 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.

[3] 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.

[4] 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.

[5] I. Harvey, P. Husbands, and D. Cliff. Seeing the Light: Artificial Evolution, Real Vision. *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pages 392–401, 1994.

[6] I. Horswill. Polly: A vision-based artificial agent. *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pages 824–829, 1993.

[7] L. Lorigo, R. Brooks, and W. 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.

[8] D. Marocco and D. Floreano. Active vision and feature selection in evolutionary behavioral systems. *From Animals to Animats*, 7:247–255, 2002.

[9] M. Martin. Evolving visual sonar: Depth from monocular images. *Pattern Recognition Letters*, 27(11):1174–1180, 2006.

[10] M. Mataric and D. Cliff. Challenges in evolving controllers for physical robots. *Robotics and Autonomous Systems*, 19(1):67–83, 1996.

[11] J. Michels, A. Saxena, and A. Ng. High speed obstacle avoidance using monocular vision and reinforcement learning. *Proceedings of the 22nd international conference on Machine learning*, pages 593–600, 2005.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] I. Ulrich and I. Nourbakhsh. Appearance-based obstacle detection with monocular color vision. *Proceedings of AAAI Conference*, pages 866–871, 2000.

[17] J. Walker, S. Garrett, and M. Wilson. Evolving controllers for real robots: A survey of the literature. *Adaptive Behavior*, 11(3):179–203, 2003.

[18] P. Whigham. Grammatically-based genetic programming. *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 33–41, 1995.