Automatic Design of Vision-Based Obstacle Avoidance Controllers Using Genetic Programming

Renaud Barate, Antoine Manzanera

Automatic Design of Vision-based Obstacle Avoidance Controllers using Genetic Programming

Renaud Barate and Antoine Manzanera

ENSTA - UEI, 32 bd Victor,
75739 Paris Cedex 15, France
renaud.barate@ensta.fr, antoine.manzanera@ensta.fr

Abstract. The work presented in this paper is part of the development of a robotic system able to learn context dependent visual clues to navigate in its environment. We focus on the obstacle avoidance problem as it is a necessary function for a mobile robot. As a first step, we use an offline procedure to automatically design algorithms adapted to the visual context. This procedure is based on genetic programming and the candidate algorithms are evaluated in a simulation environment. The evolutionary process selects meaningful visual primitives in the given context and an adapted strategy to use them. The results show the emergence of several different behaviors outperforming hand-designed controllers.

1 Introduction

Obstacle avoidance is an essential function for any mobile robot. Range sensors are the most commonly used devices for detecting obstacles but the accuracy of sonars depends on the angle of reflection and the material of the detected object and laser range sensors are expensive and can be harmful. More, both are active sensors which is undesirable for military applications for instance. On the contrary, video cameras are now cheap, low consumption and high resolution sensors. Nevertheless detecting obstacles with a single camera is a difficult problem. Different solutions exist, using either optical flow or more simple contextual information, but none is generic enough to deal with changing environments.

Ideally, the robot should be able to adapt itself dynamically to the current context and to select its behavior (here, the obstacle avoidance algorithm) in real time depending on its environment. The robot would learn those most efficient behaviors only by interacting with the environment. As a first step towards this goal, we propose here an offline method based on genetic programming for the automatic design of obstacle avoidance controllers within a given environment. The next steps will be to develop a set of algorithms adapted to different environments and to design a high level controller able to select in real time the best algorithm for the current environment. Our system uses a simulation environment to test different algorithms. We will show that the evolved solutions use relevant visual primitives and that they perform better than hand-designed controllers.

2 Inspiration and Principles

2.1 Vision Based Obstacle Avoidance

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 in the camera field of view. It is commonly represented by a vector field, each vector representing the displacement of the corresponding pixel. According to the parallax principle, the perceived displacements will be greater for nearer objects when the camera has a translational movement. A simple and robust approach for obstacle avoidance is to move the robot forward for a few seconds, to calculate the mean of the optical flow on the right and left sides of the image, then to turn the robot in order to balance the flow on each side. This technique is inspired by the behavior of bees and has been successfully applied to center a mobile robot in a corridor [1, 2]. More recently it has been used for the control of an autonomous helicopter [3, 4]. However, systems based on optical flow don't cope well with thin or lowly textured obstacles.

On the other hand, simple primitives can be used to extract different informations about the image. Those include threshold, Gaussian and Laplacian filters as well as orientation and frequency filters. In many cases, combinations of some of those primitives can deliver enough information to discriminate potential obstacles. For instance, Michels implemented a system to estimate depth from texture information in outdoor scenes [5]. 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 [6–8]. Nevertheless those methods suppose that the floor may be clearly discriminated and they neglect potentially useful contextual information from the rest of the scene. Ideally, a good obstacle avoidance system should be able to use any visual clue.

2.2 Vision in Evolutionary Robotics

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 64x64 pixels camera but simplified the input to a few average values on the image [9]. Marocco used only a 5x5 pixels retina as visual input [10]. A description of several other evolutionary robotics systems can be found in [11, 12] 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 [13, 14]. Ebner also developed a navigation system for a robot using range sensors [15] but he didn't combine the two approaches for a vision-based navigation system. Parisian evolution has also been shown to produce very good results for obstacle detection and 3D reconstruction but those systems need two calibrated cameras [16, 17].

To our knowledge, only Martin tried evolutionary techniques with monocular images for obstacle avoidance [18]. 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. In our work, the vision algorithms are not limited to a particular technique and the selection is based on a functional evaluation of the algorithms, that is their ability to avoid obstacles in the simulation environment.

3 Material and Methods

3.1 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. Several scalar values can be produced with different filter chains, these values are then combined using scalar operators. A final step will generate a motor command using the resulting scalar value(s). We extended this purely vision based algorithmic structure by adding two scalar input variables: the goal location distance and heading relative to the robot position. These variables can be used for command generation along with the other scalar values.

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. Fig. 1 shows an example program for a simple obstacle avoidance behavior based on 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 [19].

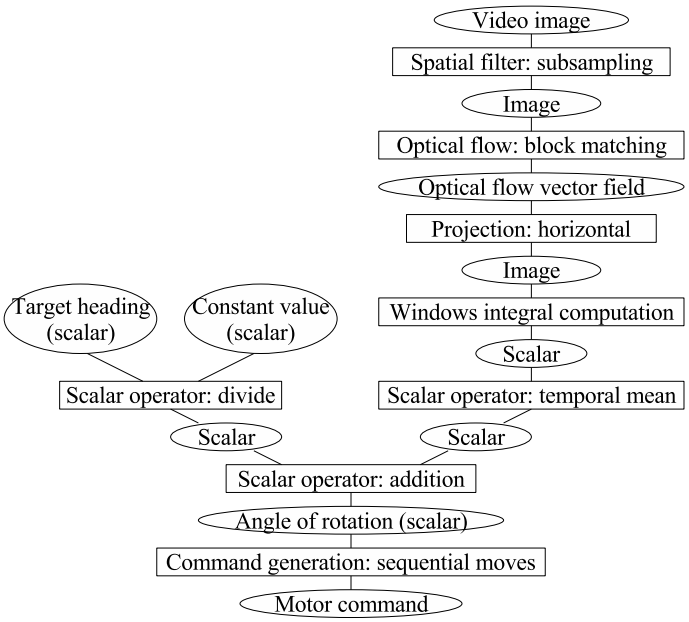

Fig. 1. Algorithmic tree of a program example for obstacle avoidance. Rectangles represent primitives and ellipses represent data.

- **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, temporal mean calculation and simple if-then-else test.
– **Command generation** (*input: scalar(s), output: command*): The motor command is represented by two scalar values: the requested linear and angular speeds. We created two operators to generate these values:
  • Direct generation: The required linear and angular speeds are two input scalar values.
  • Sequential moves: The movement is decomposed in straight moves and in-place rotations. This facilitates the usage of optical flow based strategies, since optical flow exploitation is more straightforward when the movement has no rotational part. This operator uses one input scalar value, which will be the angle of rotation at the end of a straight move.

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. 3.3.

3.2 Evaluation of Algorithms

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 simulation environment is a closed room of 36 m^2 area (6 m x 6 m) with block obstacles or furniture items depending on the experiments.

The simulation 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 320x160 representing a field of view of 120° x 60°. This large field of view reduces the dead angles and hence facilitates obstacle detection and avoidance. 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. For that, we place the robot at the fixed starting point and let it move in the environment during 60 s driven by the obstacle avoidance algorithm. Two scores are attributed to the algorithm depending on its performance: a goal-reaching score G_1 rewards algorithms reaching or approaching the goal location, whereas score C_1 rewards the individuals that didn't hit obstacles on their way. Those scores are calculated with the following formulas:

$$G_1 = \begin{cases} t_\mathrm{G} \text{ if the goal is reached} \\ t_\mathrm{max} + d_\mathrm{min}/V \text{ else} \end{cases}$$
$$C_1 = t_\mathrm{C}$$

where t_G is the time needed to reach the goal in seconds, $t_{\max}$ is the maximum time in seconds (here 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).

We proceed then to a second run with a different starting point and a different goal location. Scores G_2 and C_2 are calculated the same way and the total scores G_{T} and C_{T} are obtained by adding the scores from the two runs:

$$G_{\mathrm{T}} = G_1 + G_2$$
$$C_{\mathrm{T}} = C_1 + C_2$$

The goal is hence to minimize those two scores G_{T} and C_{T}. Performing two different runs favors algorithms with a real obstacle avoidance strategy while not increasing evaluation time too much. The starting points are fixed because we want to evaluate all algorithms on the same problem.

3.3 The Evolution Process

We use genetic programming to evolve obstacle avoidance algorithms with the least possible *a priori* on the structure of the algorithm. Genetic programming, as introduced by J. Koza [20], is the evolution of computer programs by means of artificial evolution. Like other evolutionary algorithms, it is based on a selection-breeding process inspired by biological evolution which creates better algorithms by combining the primitives of the best algorithms of previous generations.

As usual with evolutionary algorithms, the population is initially filled with randomly generated individuals. The difficulty that arises with algorithms that use different data types is to ensure that the generated algorithms are valid. Montana introduced strongly-typed genetic programming to overcome this problem [21] and Whigham extended it by using a grammar to generate the algorithms [22]. We decided to use this grammar-based genetic programming as 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 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. For clarity and space reasons, we cannot present the whole derivation process here but detailed explanations can be found in the paper from Whigham [22]. 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

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

Table 1. Grammar used in the genetic programming system for the generation of the algorithms.

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 → ifThenElse(REAL,REAL,REAL,REAL) for example).

As described in the previous section, we wish to minimize two criteria G_{T} and C_{T}. 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 introduced by K. Deb. We will not describe this algorithm here, as detailed and thorough explanations can be found in [23].

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. The population size is 100 individuals and the experiments last for 100 generations. 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 Experiment in a Simple Environment with Block Obstacles

The experiment described in this subsection uses only simple block obstacles. The goal is to cross the environment diagonally without hitting those block obstacles. We use a single brick texture for all the walls, floor, ceiling and blocks. We also manually created a simple obstacle avoidance controller for this environment based on optical flow. It manages to avoid some obstacles but it is quite slow and gets stuck in some situations. For this problem, the evolution created more efficient solutions, as shown on Figure 2.

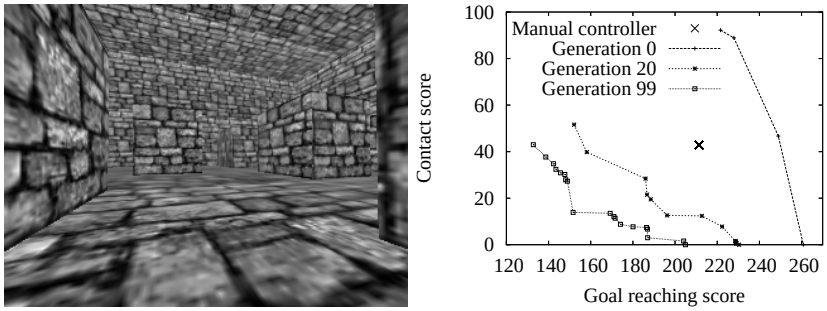

Fig. 2. Left: Snapshot of the simple environment with textured block obstacles. Right: Evolution of the best algorithms along the generations. Only the Pareto front for each generation is shown here.

Different kinds of controllers emerged, from slow ones only covering a few meters to fast ones quickly reaching the goal but bouncing on obstacles rather than avoiding them. The interesting point is that all those controllers use either an optical flow based technique, which is the most straightforward, or a Gabor filter detecting near obstacles. The evolution didn't find the ideal compromise between obstacle avoidance and speed but it selected reasonable and usable visual primitives for the environment efficiently. The resulting trajectories for the reference controller and two evolved ones are shown on Fig. 3.

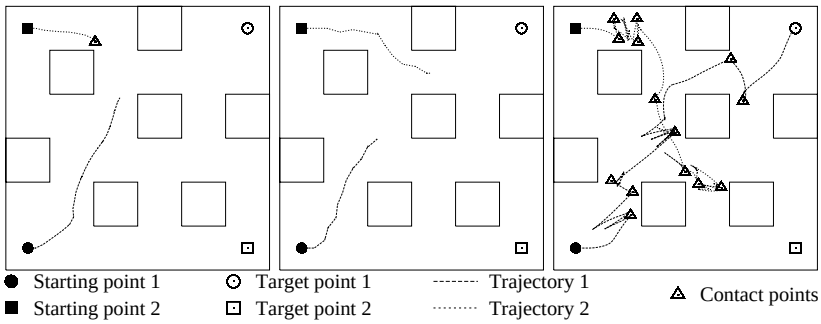

Fig. 3. Left: Trajectory of the reference controller. Middle: Trajectory of an evolved controller with a careful behavior. Right: Trajectory of a faster evolved controller bouncing on obstacles.

The two corresponding evolved controllers are shown on Fig. 4. It is interesting to note that these controllers use only a few primitives. In this kind of experiment, several primitives are seldom selected: edge detectors for example (Laplacian or Sobel) are almost never used, probably because the integral com-

putation step is not adapted for the extraction of this kind of feature. On the contrary, optical flow is very often used but mainly to detect near obstacles because the flow computation is very imprecise at high speed or with a rotary movement. Gabor filter is also often selected as it can detect textured obstacles at a given distance.

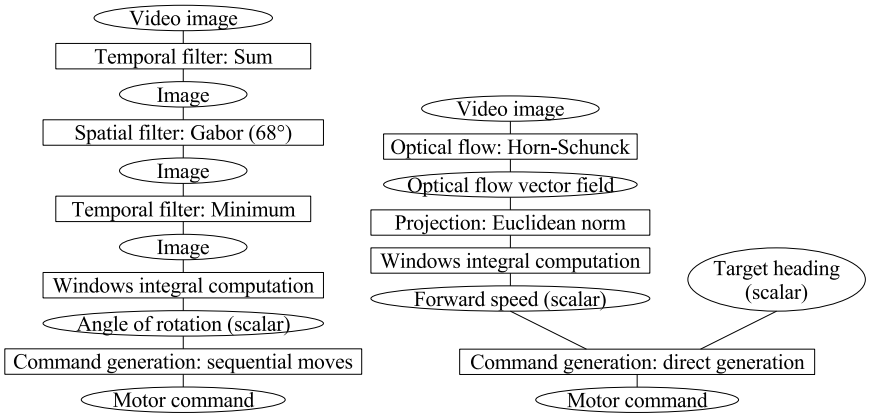

Fig. 4. Left: Algorithm of an evolved controller with a careful behavior. Right: Algorithm of a faster evolved controller.

4.2 Experiment in a more Realistic Environment

In this experiment, the block obstacles have been replaced by bookshelves. The floor and the walls have different textures. The environment is less cluttered but the obstacles are larger than in the previous experiment. As there is more free space and the target location is nearer from the starting point, the experiment time is reduced to 30 seconds. We manually designed another reference controller to compare with the evolution results. It is also based on optical flow but it has been slightly changed to perform better in this environment. Figure 5 shows the environment and the performance of these controllers.

Once again, one of the evolved controllers avoids obstacles correctly but it is far from perfect as it doesn't reach the goal from the first starting point. Other evolved controllers reach the goal quickly but hit obstacles and walls several times on their way. This time, almost all the evolved controllers use direct integral computations on the image to detect obstacles. When approaching the bookshelves, the bottom of the shelves becomes more visible. Those areas are darker because the light comes from the ceiling in these experiments, thus with a simple integral computation in the upper part of the image, the robot brakes when approaching an obstacle. This obstacle detection technique is coupled with a target heading behavior and a seemingly random back and forth move that

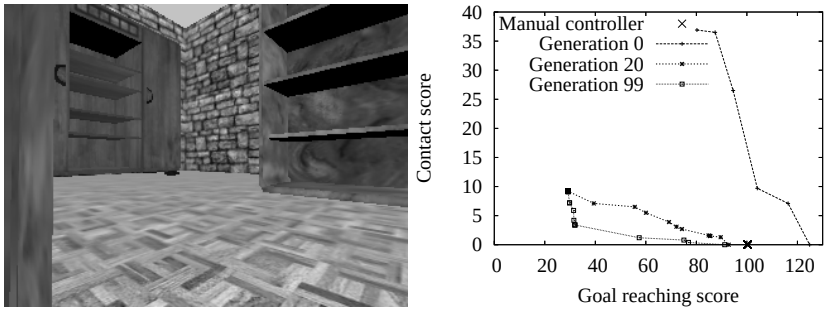

Fig. 5. Left: Snapshot of the environment with several furniture items. Right: Evolution of the best algorithms along the generations.

partially prevents the robot from getting stuck behind large obstacles. The combination of those three techniques achieves a rather good obstacle avoidance performance, as shown on Fig. 6.

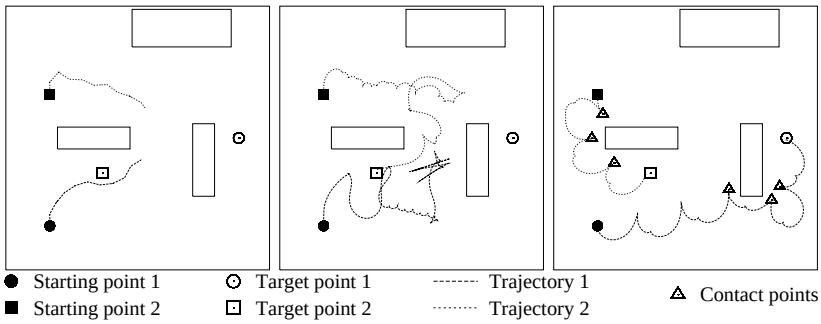

Fig. 6. Left: Trajectory of the hand-designed controller. Middle: Trajectory of an evolved controller avoiding obstacles. Right: Trajectory of another evolved controller quickly reaching the goal but not avoiding obstacles.

4.3 Discussion

The results of these two experiments show that the evolution process introduced in this paper is able to produce interesting solutions, generally outperforming hand-designed controllers. The evolution caused the emergence of original solutions to the obstacle avoidance problem depending on the context. The visual primitives selected in each case are coherent and can be used to avoid obstacles. This is promising for the next necessary step which will be the validation of the evolved algorithms on a real robot.

Nevertheless it should be noted that these performances are far from perfect. More, in other experiments with an even simpler environment where hand-designed controllers are really efficient, evolution gets stuck in local minima and achieves far worse performance. This underlines the major drawback of our approach at the moment: the population is much too small compared to the size of the search space, so evolution only explores a small part of it and hence misses many interesting solutions. We currently investigate solutions to overcome this problem without increasing the evolution time too much. The first possible solution would be to introduce a diversity metric to prevent premature convergence. This solution has been proved efficient on several problems but it is not straightforward to define a good diversity metric for tree based algorithms. An easier and probably more efficient solution would be to change population parameters. We could either split the population in several independent subpopulations or change the population size during the evolution to explore more solutions in the beginning. A combination of these two solutions is also possible and is likely to improve the results greatly. Another drawback is that our feature extraction operator is not adapted for linear or punctual features (edges or corners for instance). The addition of others feature extraction operators should increase the diversity of usable features.

5 Conclusion

In this paper, we used multi-objective genetic programming to create obstacle avoidance controllers making use of visual information. We use a simulation environment with computer rendered images to evaluate the candidate controllers. The evolution allowed the emergence of original strategies using relevant visual primitives in the environment. For future research, we intend to improve the results by controlling more precisely the population parameters of the evolution process. We also plan to design a more realistic simulation environment and robot model to evolve more robust controllers and to test them on a real robot. We will then adapt our system for the online selection of relevant controllers in order to develop strategies adapted to the visual context in real time. This will bring us closer to our goal of an adaptive and reactive robot able to face the complex and ever-changing environments of the real world.

References

1. Santos-Victor, J., Sandini, G., Curotto, F., Garibaldi, S.: Divergent stereo for robot navigation: learning from bees. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition. (1993) 434–439
2. Ebner, M., Zell, A.: Centering behavior with a mobile robot using monocular foveated vision. Robotics and Autonomous Systems **32**(4) (2000) 207–218
3. Muratet, L., Doncieux, S., Brière, Y., Meyer, J.A.: A contribution to vision-based autonomous helicopter flight in urban environments. Robotics and Autonomous Systems **50**(4) (2005) 195–209

4. Hrabar, S.: Vision-Based 3D Navigation for an Autonomous Helicopter. PhD thesis, University of Southern California (2006)
5. Michels, J., Saxena, A., Ng, A.: High speed obstacle avoidance using monocular vision and reinforcement learning. Proceedings of the 22nd international conference on Machine learning (2005) 593–600
6. Horswill, I.: Polly: A vision-based artificial agent. Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93) (1993) 824–829
7. Lorigo, L., Brooks, R., Grimson, W.: Visually-guided obstacle avoidance in unstructured environments. Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robots and Systems **1** (1997) 373–379
8. Ulrich, I., Nourbakhsh, I.: Appearance-based obstacle detection with monocular color vision. Proceedings of AAAI Conference (2000) 866–871
9. Harvey, I., Husbands, P., Cliff, D.: Seeing the Light: Artificial Evolution, Real Vision. From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior (1994) 392–401
10. Marocco, D., Floreano, D.: Active vision and feature selection in evolutionary behavioral systems. From Animals to Animats **7** (2002) 247–255
11. Mataric, M., Cliff, D.: Challenges in evolving controllers for physical robots. Robotics and Autonomous Systems **19**(1) (1996) 67–83
12. Walker, J., Garrett, S., Wilson, M.: Evolving controllers for real robots: A survey of the literature. Adaptive Behavior **11**(3) (2003) 179–203
13. Ebner, M., Zell, A.: Evolving a task specific image operator. Evolutionary image analysis, signal processing and telecommunications: First european workshop, evoiasp (1999) 74–89
14. Trujillo, L., Olague, G.: Synthesis of interest point detectors through genetic programming. Proceedings of the 8th annual conference on Genetic and evolutionary computation (2006) 887–894
15. Ebner, M., Zell, A.: Evolving a behavior-based control architecture-From simulations to the real world. Proceedings of Genetic and Evolutionary Computation Conference **2** (1999) 1009–1014
16. Pauplin, O., Louchet, J., Lutton, E., De La Fortelle, A.: Evolutionary Optimisation for Obstacle Detection and Avoidance in Mobile Robotics. Journal of Advanced Computational Intelligence and Intelligent Informatics **9**(6) (2005) 622–629
17. Olague, G., Puente, C.: Parisian evolution with honeybees for three-dimensional reconstruction. Proceedings of the 8th annual conference on Genetic and evolutionary computation (2006) 191–198
18. Martin, M.: Evolving visual sonar: Depth from monocular images. Pattern Recognition Letters **27**(11) (2006) 1174–1180
19. Barron, J., Fleet, D., Beauchemin, S.: Performance of optical flow techniques. International Journal of Computer Vision **12**(1) (1994) 43–77
20. Koza, J.: Genetic Programming: On the Programming of Computers by Natural Selection. MIT Press, Cambridge, MA, USA (1992)
21. Montana, D.: Strongly Typed Genetic Programming. Evolutionary Computation **3**(2) (1995) 199–230
22. Whigham, P.: Grammatically-based genetic programming. Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications (1995) 33–41
23. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation **6**(2) (2002) 182–197