Evolving a Vision-Driven Robot Controller for Real-World Indoor Navigation

Paweł Gajda and Krzysztof Krawiec

Institute of Computing Science
Poznan University of Technology, Poznań, Poland

Abstract. In this paper, we use genetic programming (GP) to evolve a vision-driven robot controller capable of navigating in a real-world environment. To this aim, we extract visual primitives from the video stream provided by a camera mounted on the robot and let them to be interpreted by a GP individual. The response of GP expressions is then used to control robot's servos. Thanks to the primitive-based approach, evolutionary process is less constrained in the process of synthesizing image features. Experiments concerning navigation in indoor environment indicate that the evolved controller performs quite well despite very limited human intervention in the design phase.

1 Introduction and Related Work

Using genetic programming (GP) to evolve robot controllers, whether for real or for virtual environments, may be traced back to the very beginning of GP [8,11]. Typical robotic tasks successfully solved by means of GP include wall following [7], mapping sensor readings to robot locations [4], and learning an obstacle avoidance strategy [6], to mention a few representative contributions. Also, much research has been done on evolving cooperative behaviors of robots, for instance to control a team of robots playing soccer [1]. An extensive review on evolving controllers for real robots may be found in [14].

In most of the aforementioned contributions, the evolved GP programs usually process scalar data coming from distance sensors (or virtual distance sensors in case of simulation). In this paper, we are particularly interested in evolving a *vision-driven* real-world robot controller. Past research within this area includes several contributions. In [3], Ebner used genetic programming to evolve edge detectors for robotic vision. Graae, Nordin, and Nordahl [5] evolved a stereoscopic vision system for a humanoid robot using GP. Langdon and Nordin used machine code GP to evolve hand-eye coordination for a humanoid robot [9]. Seok, Lee, and Zhang applied a variant of linear GP and FPGA hardware to evolve the behavior of locating light sources and avoiding obstacles [13]. Wolff and Nording made use of visual feedback for evolving gait controllers of a bipedal robot [15]. There are also other reports on GP-based robotic vision in an virtual environment, which is not considered in this paper (e.g., in [2] an OpenGL framework is used to simulate the visual environment of a physical robot solving the task of line following).

M. Giacobini et al. (Eds.): EvoWorkshops 2008, LNCS 4974, pp. 184–193, 2008.

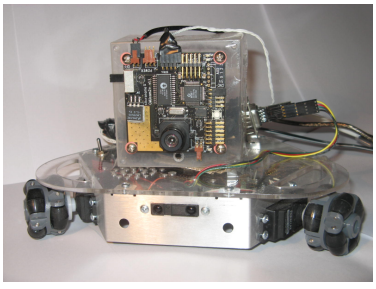

Fig. 1. The PPRK robot with the mounted CMUCam2+ camera

This paper demonstrates the possibility of evolutionary learning of a high-level behavioral pattern directly from low-level visual sensory input. In our approach, we avoid defining a fixed repertoire of high-level visual features to be used by the learner. Rather than that, we feed a low-level, though non-raster, visual data into GP learners. Next, using an appropriately defined fitness function, we entice the evolving learners to navigate in a real-world indoor environment. As our robot's perception is not tuned to the particular task or visual target, the evolutionary learning has to build up an appropriate and effective higher-level representation of visual patterns. An experimental evaluation on a real-world robotic platform in an indoor environment shows the ability of our approach to solve the task of approaching a visual target.

2 The Hardware Platform

We used Palm Pilot Robot Kit (PPRK) as the hardware platform for our approach. PPRK is a small, three-wheeled, autonomous robot designed by the Robotics Institute at the Carnegie Mellon University and produced by Acroname [12]. As its name suggests, it may be controlled by a handheld computer (Palm Pilot or PocketPC), but it is also equipped with a built-in controller called *BrainStem* that is able to store and execute simple behavioral patterns. BrainStem is based on PIC18C252 processor clocked at 40MHz.

PPRK uses *holonomic* drive, composed of three equidistant wheels arranged in a circle and propelled by three independent servos. Each of its 'omni-wheels', thanks to built-in rolls, may move freely in directions that do not lay in its plane of rotation; this happens, for instance, when the wheel is dragged by the other wheels. This kind of drive allows for arbitrary rotation and translation, so that the total number of robot's degrees of freedom (3) is equal to the number of controllable degrees of freedom. For instance, applying the same potential v to all three servos causes the robot to rotate in place. Applying potentials v, $-v$, and 0 to servos #1, #2, and #3, respectively, makes the robot move perpendicularly to the section connecting wheels #1 and #2, dragging the wheel #3 behind it.

We mounted a CMUCam2+ board on robot's chassis (see Fig. 1). The CMUCam2+ card, also provided by Acroname, uses Omnivision's CMOS OV6620

camera controlled by SX52 microcontroller. Though it implements some simple image processing and feature tracking functionalities, they have not been used in our setup, and the digitized video stream was directly sent to the controlling computer. The three infrared proximity sensors included in PPRK were also inactive in our experiments.

The camera board acquires video data at resolution of 87×143 pixels and is able to perform some elementary image analysis at 25-50 frames per second. Unfortunately, it communicates with other modules (including the BrainStem) via standard serial interface, which limits the maximum data transfer rate to 115200 bits per second. This seriously reduces the number of frames that can be sent to the image analysis module in real time. This is why, in the following experiments the actual number of frames processed per seconds amounts to approximately 1, though the evolved GP expressions could easily handle two orders of magnitude higher frame rates.

In contrast to most of related research, we do not extract any predefined high-level image features to help evolution to learn the desired robot's behavior. Rather than that, we rely on GP-based processing of low-level visual primitives, an approach described in the following section.

3 Using GP Trees to Process Visual Primitives

Our GP-based approach to visual processing, originally proposed in [16] and later extended in [17], has the following rationale. The volume of raw raster data is usually too large to make it direct subject to evolutionary learning. To reduce these data, one commonly uses a predefined set of visual features extracted from the training images; the evolutionary process works then with such features only. There is, however, a significant risk that the features pre-selected by the human are not the best ones to cope with the particular visual task. Also, the sole process of defining and implementing such features may be time-consuming and difficult.

To keep the amount of visual training data within reasonable limits on one hand and avoid arbitrary pre-selection of visual features on the other, our approach relies on *visual primitives* (VP). We define the visual primitive as a local salient feature extracted from an image location characterized by a prominent gradient. In the beginning of processing, each VP is described by three scalars called hereafter *attributes*; these include two spatial coordinates (x and y) and orientation of the local gradient vector. The complete set P of VPs is usually much more compact than the original image s in terms of information content, yet it well preserves the overall sketch of the visual input.

The right-hand part of Figure 2 presents the VPs extracted from an exemplary frame of the video sequence used in the following experiment, with each primitive depicted as a short section. Note, however, that the individuals described in the following learn from training examples that are technically *sets* of VPs, each of them described by a triple of numbers. In other words, learners do not explicitly 'perceive' the image as a two-dimensional raster.

Fig. 2. An exemplary input image (left) and the corresponding visual primitives (right)

Each learner L is a GP expression written in a form of a tree, with nodes representing *functions* that process sets of VPs. The feed of image data for the tree has been implemented by introducing a special terminal function (named *ImageNode*) that fetches the set of primitives P derived from the input image s. The consecutive internal nodes process the primitives, all the way up to the root node. We use strongly-typed GP, so child node's output type must match parent node's input type. The list of types includes numerical scalars, sets of VPs, attribute labels, binary arithmetic relations, and aggregators.

The functions, presented in Table 1, may be divided into (a) selectors, which select some VPs based on their attributes, (b) iterators, which process VPs one by one, (c) grouping operators, which group VPs based on their attributes and features, e.g., spatial proximity, and (d) scalar arithmetic functions. In addition,there is a group of functions that compute simple set operations in the domain of VPs, like set union (*SetUnion*), set difference (*SetMinus*), or symmetric difference (*SetMinusSym*). Implementation of most functions is straightforward. For instance, the *SelectorMin* function applied to a set of primitives S and attribute label p_y selects from S the VP (or VPs) with the minimal value of attribute (coordinate) y. The *ForEach* function iterates over the set of elements returned by its left child node, and passes each of them through the subtree rooted in its right child node, finally grouping the results into one set. The semantics of the remaining functions may be decoded from their mnemonics.

It is worth emphasizing that in this process, VPs and sets of VPs are used interchangeably. Technically, a set may contain both VPs and other nested sets of VPs. Therefore, the processing carried out by an individual-learner L applied to the input image s boils down to building a hierarchy of VP sets derived from s. Each invoked tree node creates a new set of VPs built upon the elements (VPs or sets of VPs) provided by its child node(s). When needed, we recursively compute an attribute value of a VP set as an average of its elements.

To control robot's actuators (servos), our trees must compute real-valued responses at the root node. To this aim, we employ an advanced feature of our approach, namely its ability to create new attributes, apart from the pre-defined ones (x, y, and orientation). The new attribute may be computed from and attached to VPs or VP sets. Technically, this is implemented by the *AddAttribute*

Table 1. The GP operators

Type	Operator
$\Re$	ERC – Ephemeral Random Constant
Ω	*Input*() – the VP representation P of the input image s
A	p_x, p_y, p_o,
R	*Equals*, *Equals5Percent*, *Equals10Percent*, *Equals20Percent*, *LessThan*, *GreaterThan*
G	*Sum*, *Mean*, *Product*, *Median*, *Min*, *Max*, *Range*
$\Re$	+($\Re$,$\Re$), −($\Re$,$\Re$), *($\Re$,$\Re$), /($\Re$,$\Re$), *sin*($\Re$), *cos*($\Re$), *abs*($\Re$), *sqrt*($\Re$), *sgn*($\Re$), *ln*($\Re$), *AttributeValue*(Ω,A)
Ω	*SetIntersection*(Ω,Ω), *SetUnion*(Ω,Ω), *SetMinus*(Ω,Ω), *SetMinusSym*(Ω,Ω), *SelectorMax*(Ω,A), *SelectorMin*(Ω,A), *SelectorCompare*(Ω,A,R,$\Re$), *SelectorCompareAggreg*(Ω,A,R,G), *CreatePair*(Ω,Ω), *ForEach*(Ω,Ω), *ForEachCreatePair*(Ω,Ω,Ω), *Ungroup*(Ω), *GroupHierarchyCount*(Ω,$\Re$), *GroupHierarchyDistance*(Ω, $\Re$), *GroupProximity*(Ω, $\Re$), *GroupOrientationMulti*(Ω, $\Re$), *AddAttribute*(Ω,$\Re$), *AddAttributeToEach*(Ω,$\Re$)

and *AddAttributeToEach* functions (see Table 1). For instance, the *AddAttribute* function takes the VP set S returned by its left child node and passes it through its right child subtree. Due to syntactic constraints imposed by strong typing, the right child subtree is forced to return a scalar value computed using ERCs, scalar functions (e.g., +, -, *, /, *abs*), and the values of existing attributes fetched from S using, among others, the *AttributeValue* function. The computed value is attached as a new attribute to S, which is subsequently returned by the *AddAttribute* function (the VP hierarchy in S remains therefore unchanged). The *AddAttributeToEach* function operates similarly, however, it repeats the steps listed here recursively for each element of S.

Given this functionality, we expect evolution to elaborate individuals that define a new attribute, different from p_x, p_y, and p_o, which should be attached to the set of VPs returned by the root node as the final response of the tree. The returned value (essentially a very specific image feature) is subsequently evaluated by the fitness function which compares it to the desired value.

4 The Experiment

4.1 The Task

Our task consists in navigating the robot in a small (3.2×3.2m) indoor environment, in varying lighting conditions. More specifically, the robot has to find and approach a 15×15cm diamond-shaped marker placed close to the floor on one of the room's walls. To make the task non-trivial, we placed some other objects in the environment; these included a chair and a cupboard. Other artifacts often visible in robot's field of view include power outlets and floor-wall boundaries.

4.2 The Training Procedure

From an evolutionary perspective, the most desirable approach to evolve our controllers would be to directly bind the evolutionary process to the real-world and estimate the fitness of each controller (individual) by downloading it to the physical robot and letting it navigate in our environment. Such an procedure would be obviously implausible. An alternative approach of evaluating individuals in a virtual simulator has been often criticized as being far from perfect in terms of fidelity to the real-world. Because our objective was to evolve a controller that operates in real-world, we took another way.

To devise a computationally feasible experimental setup without abandoning the real-world data, we came up with the following teacher-driven approach. The training data has been collected by guiding the robot by a human operator, starting from random initial locations and ending close to the target marker. For each such experiment, we recorded the sequence of video frames together with the synchronized values of potentials applied to the servos. Specifically, servo values have been recorded at one second delay with respect to the video stream to compensate for operator's reaction time.

When preparing to data acquisition, we found out that direct controlling of the holonomic drive is cumbersome and non-intuitive for humans. Therefore, we designed a more handy set of controls, composed of 4 buttons: *Forwards*, *Backwards*, *Left*, and *Right*. A single click on a button increases the intensity of particular action; e.g., each click on the *Forwards* button increases the speed of forward movement; after that, a couple of clicks on the *Backwards* button are needed to stop the robot.

The buttons determine the values of two intermediate variables: linear speed and angular speed. The values of these variables are subsequently mapped to voltages to be applied to particular servos. Therefore, each video frame is accompanied by three desired effector values.

Figure 3 shows the typical training frames selected from one of the recording sessions. After a couple of such sessions that a few hours in total (during which the lighting conditions changed significantly), the total number of collected frames amounted to 734. As using such a number of frames for evolutionary learning would be prohibitive from the viewpoint of computational burden (even when using visual primitives instead of raw raster data), we included only 30 representative frames in the final training set. These 30 frames were subject to extraction of visual primitives (see Section 3). In this process, we created VPs only for image locations for which the response of the gradient exceeded 150 (on the scale 0..255), and enforced a lower limit on their mutual distance (5 pixels). The number of primitives extracted from one image (i.e., $|P|$) was not allowed to exceed 300. To speed up calculation, this procedure has been carried out only once, prior to the evolutionary run, and the resulting VPs have been cached in the memory.

As the complete controller of our robot has to provide three output (effector) values, one for each wheel, we carried out three separate evolutionary runs. The

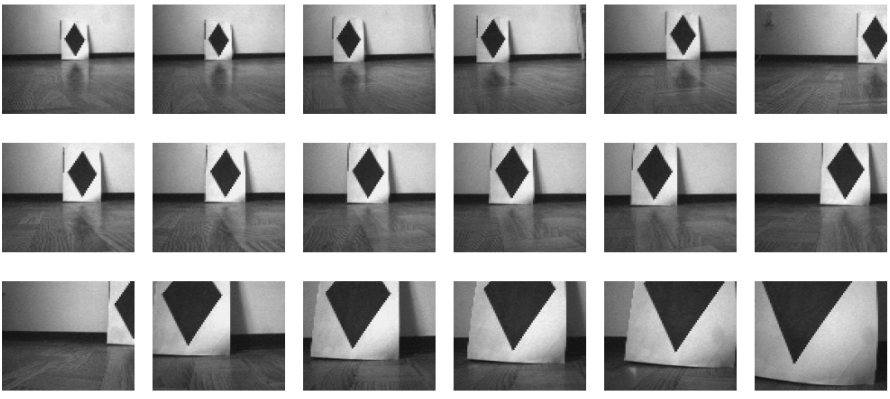

Fig. 3. Selected frames from the training set

final controller has been built by combining the best-of-run individuals from these runs.

In each of those runs, a tree is expected to return a set of visual primitives with a new attribute attached to it (cf. end of Section 3). The value of that attribute, $r(t)$, computed by the individual for the frame $\#t$, is subsequently interpreted as the voltage to be applied to the corresponding servo. In the training phase, these values were compared to the desired values $d(t)$ provided by the human controller using fitness function that aggregates the SSDs of the individual's responses $r(t)$ with respect to the desired values $d(t)$ over the entire set of 30 training frames:

$$\sum_{t=1}^{30}(r(t) - d(t))^2.$$

In each of the evolutionary runs, the following parameter values have been used: population size: 5000, number of generations: 100, tournament selection with tournament size 3, probability of crossover: 0.8, probability of mutation: 0.2, maximal tree depth: 6. The remaining parameters were set to their default values as provided in the ECJ software package [10]. Evolving controller for each servo took about 10 hours on a Pentium PC computer with 1GHz processor.

4.3 Testing the Evolved Controller

The best evolved controller, shown partially in Figure 4 (left-wheel tree only), has been subsequently used for real-time control of the robot in the testing phase. Thirty testing sessions have been carried out. Among them, 14 ended successfully, with the robot reaching the marker. Figure 5 presents a typical correct trajectory traversed by the robot. In the remaining cases, at some stage of approaching the marker the robot usually executed an extensive turn, probably distracted by spurious visual primitives resulting from image noise. Having lost the marker

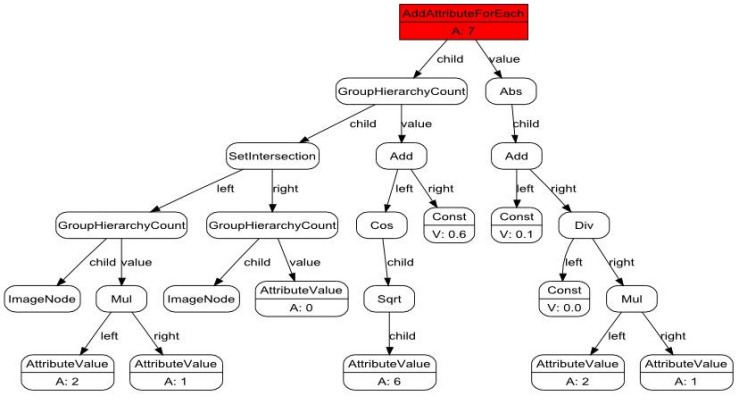

Fig. 4. The best controller evolved for the left wheel

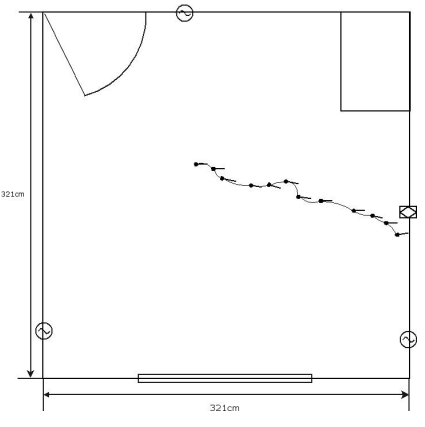

Fig. 5. A typical correct trajectory of the robot

from its field of view, it then started to move randomly. In some cases, it was able to redirect its camera towards the marker and continue the journey. Though this deficiency in robot's behavior could be circumvented, e.g., by introducing an extra heuristic, such intervention was beyond the subject of this study.

In the presence of other objects, like power sockets or chairs, the robot was usually able to navigate correctly towards the marker. However, in about 40% of trials, it erroneously targeted at the distracting objects. On the other hand, we have observed many times that a navigation error caused by one of the trees was corrected by the others. In other words, the controllers of particular servos seemed to cooperate well, despite the fact that their training was carried out independently.

5 Conclusions

Though the performance of our evolved controller is far from perfect, we think that the proposed approach is a step in the right direction. By operating in the domain of visual primitives, our learners are less constrained when defining visual features and may come up with vision-driven procedures that a human would never think of. The entire process of evolving the complete path from stimulus to action is therefore much less prone to the potential subjectivity of human designer than it is the case in more conventional approaches. By making visual learning largely independent from the particular task, we hope to make it applicable also to other behaviors like tracking or avoiding obstacles.

From the technical viewpoint, the experiment reported here is a proof-of-concept, demonstrating that well-established EC/GP software libraries like ECJ may be quite easily integrated with a real-world robotic systems. Although, as explained earlier, the learning process does not take place on-line here, an extension of our experimental setup to on-line learning is straightforward and will be most likely the subject of subsequent study. On the other hand, we have also learned hard lessons concerning robot hardware. In spite of our expectations, it turned out to be nearly impossible to implement many of the required functionalities on the robotic platform (e.g., BrainStem or PocketPC), due to technical difficulties (e.g., lack of the API for PocketPC built-in camera). Thus, in the end, the robot is a kind of mobile 'thin client' in our setup, merely carrying the camera and executing simple commands. All the actual robot's intelligence is hosted by a nearby laptop computer. On the other hand, we appreciated such setup as less constraining for the evolutionary part of the system.

Acknowledgement

This research has been supported by the Ministry of Science and Higher Education grant # N N519 3505 33.

References

1. Ciesielski, V., Wilson, P.: Developing a team of soccer playing robots by genetic programming. In: McKay, B., Tsujimura, Y., Sarker, R., Namatame, A., Yao, X., Gen, M. (eds.) Proceedings of The Third Australia-Japan Joint Workshop on Intelligent and Evolutionary Systems. School of Computer Science Australian Defence Force Academy, Canberra, Australia, November 22-25, 1999, pp. 101–108 (1999)
2. Dupuis, J.-F., Parizeau, M.: Evolving a vision-based line-following robot controller. In: The 3rd Canadian Conference on Computer and Robot Vision (CRV 2006), p. 75. IEEE Computer Society Press, Los Alamitos (2006)
3. Ebner, M.: On the evolution of edge detectors for robot vision using genetic programming. In: Groß, H.-M. (ed.) Workshop SOAVE 1997 - Selbstorganisation von Adaptivem Verhalten, VDI Reihe 8 Nr. 663, Düsseldorf, pp. 127–134. VDI Verlag (1997)

4. Ebner, M.: Evolving an environment model for robot localization. In: Langdon, W.B., Fogarty, T.C., Nordin, P., Poli, R. (eds.) EuroGP 1999. LNCS, vol. 1598, pp. 184–192. Springer, Heidelberg (1999)
5. Graae, C.T.M., Nordin, P., Nordahl, M.: Stereoscopic vision for a humanoid robot using genetic programming. In: Oates, M.J., Lanzi, P.L., Li, Y., Cagnoni, S., Corne, D.W., Fogarty, T.C., Poli, R., Smith, G.D. (eds.) EvoWorkshops 2000. LNCS, vol. 1803, pp. 12–21. Springer, Heidelberg (2000)
6. Harding, S., Miller, J.F.: Evolution of robot controller using cartesian genetic programming. In: Keijzer, M., Tettamanzi, A.G.B., Collet, P., van Hemert, J.I., Tomassini, M. (eds.) EuroGP 2005. LNCS, vol. 3447, pp. 62–73. Springer, Heidelberg (2005)
7. Koza, J.R.: Evolution of a subsumption architecture that performs a wall following task for an autonomous mobile robot via genetic programming. In: Hanson, S.J., Petsche, T., Rivest, R.L., Kearns, M. (eds.) Computational Learning Theory and Natural Learning Systems, June 1994, vol. 2, pp. 321–346. MIT Press, Cambridge (1994)
8. Koza, J.R., Rice, J.P.: Automatic programming of robots using genetic programming. In: Proceedings of Tenth National Conference on Artificial Intelligence, pp. 194–201. AAAI Press/MIT Press (1992)
9. Langdon, W.B., Nordin, P.: Evolving hand-eye coordination for a humanoid robot with machine code genetic programming. In: Miller, J., Tomassini, M., Lanzi, P.L., Ryan, C., Tetamanzi, A.G.B., Langdon, W.B. (eds.) EuroGP 2001. LNCS, vol. 2038, pp. 313–324. Springer, Heidelberg (2001)
10. Luke, S.: ECJ 15: A Java evolutionary computation (2006), `http://cs.gmu.edu/~library.eclab/projects/ecj/`
11. Nordin, P., Banzhaf, W.: Genetic programming controlling a miniature robot. In: Siegel, E.V., Koza, J.R. (eds.) Working Notes for the AAAI Symposium on Genetic Programming, November 10–12, 1995, pp. 61–67. MIT Press, Cambridge (1995)
12. Reshko, G., Mason, M.T., Nourbakhsh, I.R.: Rapid prototyping of small robots. Technical Report CMU-RI-TR-02-11 (2002)
13. Seok, H.-S., Lee, K.-J., Zhang, B.-T.: An on-line learning method for object-locating robots using genetic programming on evolvable hardware. In: Sugisaka, M., Tanaka, H. (eds.) Proceedings of the Fifth International Symposium on Artificial Life and Robotics, Oita, Japan, January 26-28, 2000, vol. 1, pp. 321–324 (2000)
14. Walker, J., Garrett, S., Wilson, M.: Evolving controllers for real robots: A survey of the literature. Adaptive Behavior 11(3), 179–203 (2003)
15. Wolff, K., Nordin, P.: Evolution of efficient gait with humanoids using visual feedback. In: Proceedings of the 2nd IEEE-RAS International Conference on Humanoid Robots, pp. 99–106. Institute of Electrical and Electronics Engineers, Inc (2001)
16. Krawiec, K.: Learning High-Level Visual Concepts Using Attributed Primitives and Genetic Programming. In: Rothlauf, F., Branke, J., Cagnoni, S., Costa, E., Cotta, C., Drechsler, R., Lutton, E., Machado, P., Moore, J.H., Romero, J., Smith, G.D., Squillero, G., Takagi, H. (eds.) EvoWorkshops 2006. LNCS, vol. 3907, pp. 515–519. Springer, Heidelberg (2006)
17. Krawiec, K.: Generative Learning of Visual Concepts using Multiobjective Genetic Programming. Pattern Recognition Letters 28(16), 2385–2400 (2007)