On the Evolution of Edge Detectors for Robot Vision using Genetic Programming

Marc Ebner
Eberhard-Karls-Universität Tübingen
Wilhelm-Schickard-Institut für Informatik
Computer Architecture
Köstlinstraße 6, 72074 Tübingen, Germany
ebner@informatik.uni-tuebingen.de

Abstract

Genetic programming has been applied to the task of evolving edge detectors. Using simple elementary functions we have evolved edge detectors using the squared difference between the individual's response and Canny's edge detector as a fitness measure. We present the best evolved individual and two individuals which specialized themselves to detect only vertical edges.

1 Motivation

Image processing is usually done by chaining a series of well known operators to solve a particular problem at hand. We are trying to automate the process by evolving the necessary program using Genetic Programming [11, 12] which consists of several elementary operators. Therefore one only needs to specify what the desired response should be but not how the image operators are actually applied.

Currently we are trying to equip a mobile robot with feature detectors which have adapted themselves to extract relevant features from the environment. These features can then be used during control and navigational tasks. As a first step we wanted to see if Genetic Programming is able to develop certain high-level operators given the low-level operators as the set of elementary functions. In this paper we present our results with the evolution of edge detectors.

2 Background

In this section we briefly review the related work of other approaches to the task of adaptively determining edges and the application of evolutionary algorithms to solve image processing tasks.

Joshi and Lee [3] modeled retinal ganglion cells and used backpropagation to train their model to detect edges. The trained neural net approximated the Laplacian of the Gaussian function. Barrow [5] modeled processing in the retina and the lateral geniculate nucleus by a convolution with a difference of Gaussians kernel. Using a competitive learning rule, Barrow found that the weights converged into patterns resembling edge or bar masks. Barrow used real world images as input for his model. Linsker [13] has shown that the layers of a four layer linear feed-forward network adapt themselves to develop averaging cells, center-surround cells and orientation selective cells using random snow as input and a Hebbian

learning rule. His emphasis was on the self organisation of a perceptual network where no desired output is presented. Kohonen [10] introduced an adaptive-subspace self-organizing map which developed Gabor-type filters if the colored noise input patterns are translated. Jeong and Kim [9] developed a method for adaptively determining the size of the Gaussian filter used in edge detectors like the Marr-Hildreth edge detector or Canny's edge detector. The optimal size of the Gaussian is determined for every pixel by minimization of an energy function which has been defined such that the size is large at uniform intensity areas and small wherever there is a significant change in intensities and that the size does not change abruptly from pixel to pixel.

Katz and Thrift [1] have used Genetic Algorithms [8] to evolve image filters for recognition and classification of images. Roth and Levine [7] have applied Genetic Algorithms to the task of geometric primitive extraction from images. Lohmann [14] used structure evolution a variant of an evolution strategy [17] to evolve detectors which determine the Euler number of an image. Andre [2] used Genetic Programming to evolve 2-dimensional feature detectors using hit-miss matrixes and applied them to the task of letter recognition. Andre used binarized images for processing. Johnson et al. [15] used Genetic Programming to evolve visual routines. The evolved visual routines were able to find the hands in images showing only the silhouette of a person. Johnson et al. defined a high-level set of terminals such as the centroid point of the silhouette and upper left and lower right points of the bounding box.

3 Evolution of edge detectors using Genetic Programming

To apply Genetic Programming [11, 12] to the task of evolving edge detectors we have to specify the set of terminals, the set of elementary functions, the fitness measure, the parameters for controlling the run and a criterion to terminate the run [11].

3.1 Set of terminals

We have converted the input image into a gray scale image with the range [0,1] and used this single band image as the single terminal for the results presented here. It would also have been possible to define the set of terminals as the set of the three individual color bands red, green and blue. However we feel that automatically defined functions [12] have to be used to evolve feature detectors using this low level set of terminals.

3.2 Set of primitive functions

We have used the complete set or a subset of the following primitive functions for the problem of edge detection. Let I be the input image for the unary functions and I_1 and I_2 be the input images for the binary functions. I_R is the resulting image. Image coordinates are specified by x and y.

3.2.1 Unary functions

- Negation (Neg) negates the image I: $I_R(x,y) = -I(x,y)$
- Absolute Value (Abs) calculates image with absolute pixel values: $I_R(x,y) = |I(x,y)|$
- Shift left (ShiftL) shifts the image I one pixel to the left: $I_R(x,y) = I(x+1,y)$
- Shift right (ShiftR) shifts the image I one pixel to the right: $I_R(x,y) = I(x-1,y)$

- Shift up (ShiftU) shifts the image I one pixel up: $I_R(x,y) = I(x,y+1)$
- Shift down (ShiftD) shifts the image I one pixel down: $I_R(x,y) = I(x,y-1)$
- Threshold (UThr) sets all pixels with values greater than zero to one and to zero otherwise: $I_R(x,y) = \begin{cases} 1 & I(x,y) > 0 \\ 0 & \text{otherwise} \end{cases}$

3.2.2 Binary functions

- Threshold (Thr) sets all pixels in I_1 with values greater than the corresponding value in I_2 to one: $I_R(x,y) = \begin{cases} 1 & I_1(x,y) > I_2(x,y) \\ 0 & \text{otherwise} \end{cases}$
- Subtraction (-) subtracts image I_2 from I_1: $I_R(x,y) = I_1(x,y) - I_2(x,y)$
- Division (/) divides every pixel value from image I_1 by the pixel value from I_2: $I_R(x,y) = I_1(x,y)/I_2(x,y)$
- Multiplication (*) multiplies every pixel value from image I_1 with the pixel value from I_2: $I_R(x,y) = I_1(x,y) \cdot I_2(x,y)$
- Addition (+) adds image I_2 to I_1: $I_R(x,y) = I_1(x,y) + I_2(x,y)$
- Mininum (Min) calculates for every pixel the minimum of both pixel value of I_1 and I_2: $I_R(x,y) = \min\{I_1(x,y), I_2(x,y)\}$
- Maximum (Max) calculates for every pixel the maximum of both pixel value of I_1 and I_2: $I_R(x,y) = \max\{I_1(x,y), I_2(x,y)\}$

3.3 Fitness measure

To calculate the fitness of an edge detector we summed up the squared differences of the pixel values between actual and desired output of the detector. We used five fitness cases for the problem presented here. The five images we used for the fitness calculations were all taken in our lab under everyday conditions and had the size 128×128. Raw fitness is calculated as $\text{fitness}_{\text{raw}}(\text{Ind}) = \sum_{i=1}^{5} \left(U(\text{Ind}(I_i)) + \frac{1}{n} \sum_{p \in I_i} (\text{Ind}(I_i(p)) - \text{Edges}(I_i(p)))^2 \right)$ where the five images are given as $\{I_1, \ldots, I_5\}$, $\text{Ind}(I)$ denotes the resulting image produced by the individual on the image I and $\text{Edges}(I)$ are the desired edges which should be extracted from the image I, p is an image point and n is the number of points in the image. $U(\text{Ind}(I_i))$ evaluates to a large value for images with uniform pixel values and zero otherwise. We added the term $U(\text{Ind}(I_i))$ to avoid the evolution of detectors which simply return a uniform image with pixel values like zero or another small value. This "detector" might otherwise have a quite good fitness since most pixels are not edge pixels and therefore have a value of zero.

4 Experiments

In our experiments we used the Canny edge detector [16] as the desired edge detector. The image with the detected edges was thresholded to give a binary image where the edge pixels have the value 1.0 and all others have been set to zero. The Canny edge detector uses a Gaussian smoothing and nonlocal maxima suppression which was not included in the set of elementary functions. Thus we did not expect to develop a 100% accurate Canny edge detector.

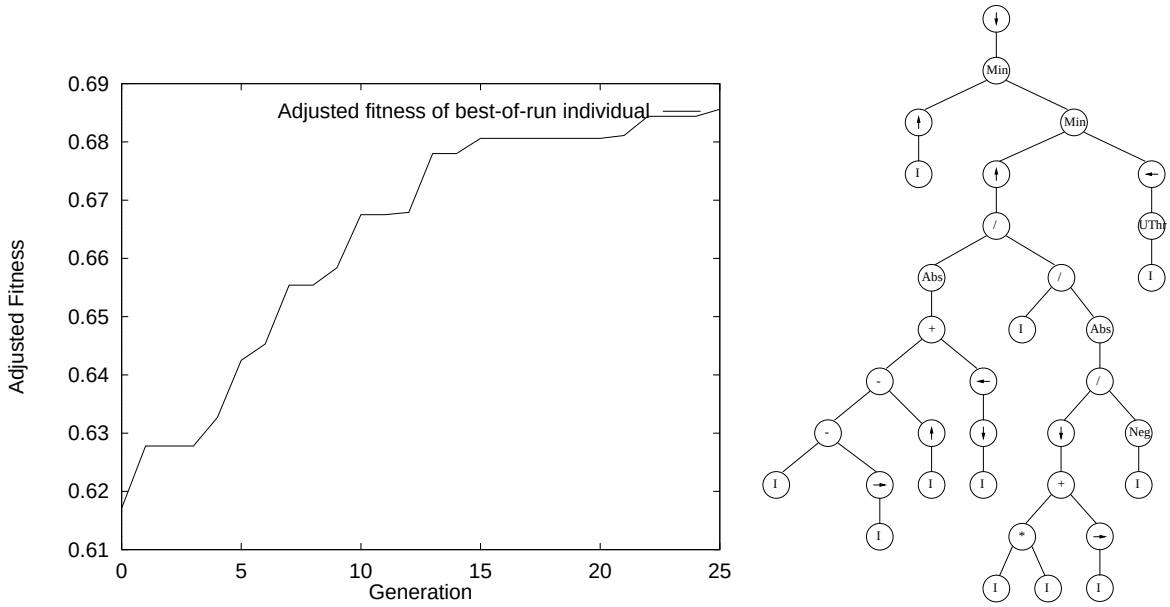

Figure 1: Fitness statistics for best individual and structure of best individual (manually simplified). I denotes the input image.

We experimented with different sets of elementary functions and terminals. We used the following base set of functions and terminals for all our experiments.

$$\text{BASE} = \{\text{Neg}, \text{Abs}, \text{ShiftL}, \text{ShiftR}, \text{ShiftU}, \text{ShiftD}, -, /, *, +, \text{Max}, \text{Min}, \text{inBandI}\}$$

where inBandI denotes the terminal intensity image with the range [0, 1]. In addition we used the following extensions to the base set:

$$\text{EXT} = \{\text{UThr}, \text{Thr}, \mathcal{R}\}$$

where $\mathcal{R}$ is the ephemeral random constant with the range $[-1.0, 1.0)$. This ephemeral random constant denotes a uniform image where all pixel values are set to the value of the ephemeral random constant. We did not use a wrapper such as binarizing the image. We wanted to see if Genetic Programming is able to determine the threshold automatically using the elementary functions and the ephemeral random constant of the extension set.

We constructed eight different function sets using the base set and all possible subsets of the extension set. Due to the long running times involved we were only able to perform two runs for each of the different sets. We have used a population size of 4000 individuals for each of the runs. For the first run we imposed a limit of 1000 nodes and a maximum possible depth of the trees of 17 and used fitness proportionate selection and the parameters $(p_{\text{cross}} = 0.9, p_{\text{rep}} = 0.1)$. For the second run we did not impose a limit on the number of nodes and the maximum depth and used fitness proportionate selection with over-selection and the parameters $(p_{\text{cross}} = 0.85, p_{\text{rep}} = 0.1, p_{\text{mut}} = 0.05)$.

Since we did not expect to evolve a perfect individual we terminated the runs after 25 generations. The second of the two runs always produced a fitter individual than the first. The evolved edge detectors were tested on five additional images which have not been used for the fitness calculations. The individual which achieved the best results on the previously unseen images was evolved using the set of elementary functions $\text{BASE} \cup \{\text{UThr}\}$ during the second of the two runs. The structure of the individual is as follows:

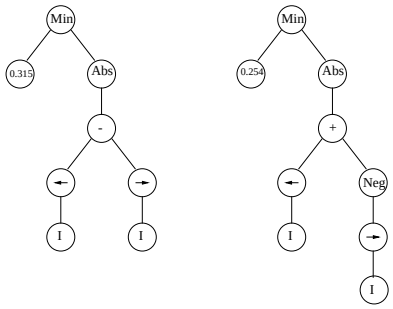

Figure 2: Detectors for vertical edges (manually simplified). I denotes the input image.

(ShiftD (Min (Abs (- (- inBandI inBandI) (ShiftU inBandI))) (Min (ShiftU (/ (Abs (+ (- (- inBandI (Min (/ inBandI inBandI) (ShiftR inBandI))) (ShiftU inBandI)) (ShiftU (ShiftL (ShiftD (ShiftD inBandI)))))) (/ inBandI (Abs (Min (/ (ShiftD (Max (+ (* inBandI inBandI) (ShiftR inBandI)) (Min (Min (ShiftU inBandI) (+ (+ (Max (- (Min inBandI inBandI) (ShiftR (ShiftD (* (ShiftD inBandI) (ShiftL inBandI))))) (* (* inBandI inBandI) (Min (Abs (Abs (+ inBandI inBandI))) (ShiftU (- (Abs inBandI) (Neg inBandI)))))) (ShiftU inBandI)) (ShiftU (ShiftL (ShiftL (ShiftU (/ (* (* (ShiftR inBandI) (- inBandI inBandI)) (Neg (ShiftL inBandI))) (Neg (Neg inBandI))))))))) (* inBandI inBandI)))) (Neg inBandI)) (/ inBandI inBandI)))))) (ShiftR (ShiftL (ShiftL (UThr inBandI)))))))

The parse tree of the same individual after manual simplifications have been applied is shown in figure 1. The response of the detector to the images used for the fitness calculations is shown in figure 3 and the response of the detector to the images used to test the generalization capabilities of the detector is shown in figure 5.The fitness statistics are shown in figure 1.

The response of the best individual of the first random population of the run which evolved the best individual is shown in figure 4. Its structure is:

(ShiftD (Min (Abs (+ (- (- inBandI inBandI) (ShiftU inBandI)) (ShiftU (* inBandI inBandI)))) (Min (ShiftU (/ (+ inBandI inBandI) (ShiftR inBandI))) (+ (UThr (ShiftL inBandI)) (UThr (ShiftD inBandI))))))

On the first of the two runs two interesting individuals emerged with the sets BASE $\cup$ $\{\text{UThr}, \mathcal{R}\}$ and BASE $\cup$ $\{\text{UThr}, \text{Thr}, \mathcal{R}\}$. The individuals simply take the absolute difference between the pixels on the left and right side for every pixel. Finally both compute the result as the minimum value between the absolute value of the difference and a small ephemeral random constant. These detectors are obviously good at detecting vertical edges and do not detect any horizontal edges. The minimum operator acts like a threshold which sets large edge values to a fixed value. The structures of both detectors are shown in figure 2 after manual simplifications have been applied.

5 Conclusion and ongoing research

The results presented here have shown that genetic programming is able to evolve an edge detector. Having shown this, we are interested in the question if Genetic Programming also evolves a structure capable of detecting edges given a fitness measure which does not directly incorporate the edges as the desired output. We will also apply Genetic Programming to the task of evolving general feature detectors which can be used during robot control and navigation tasks.

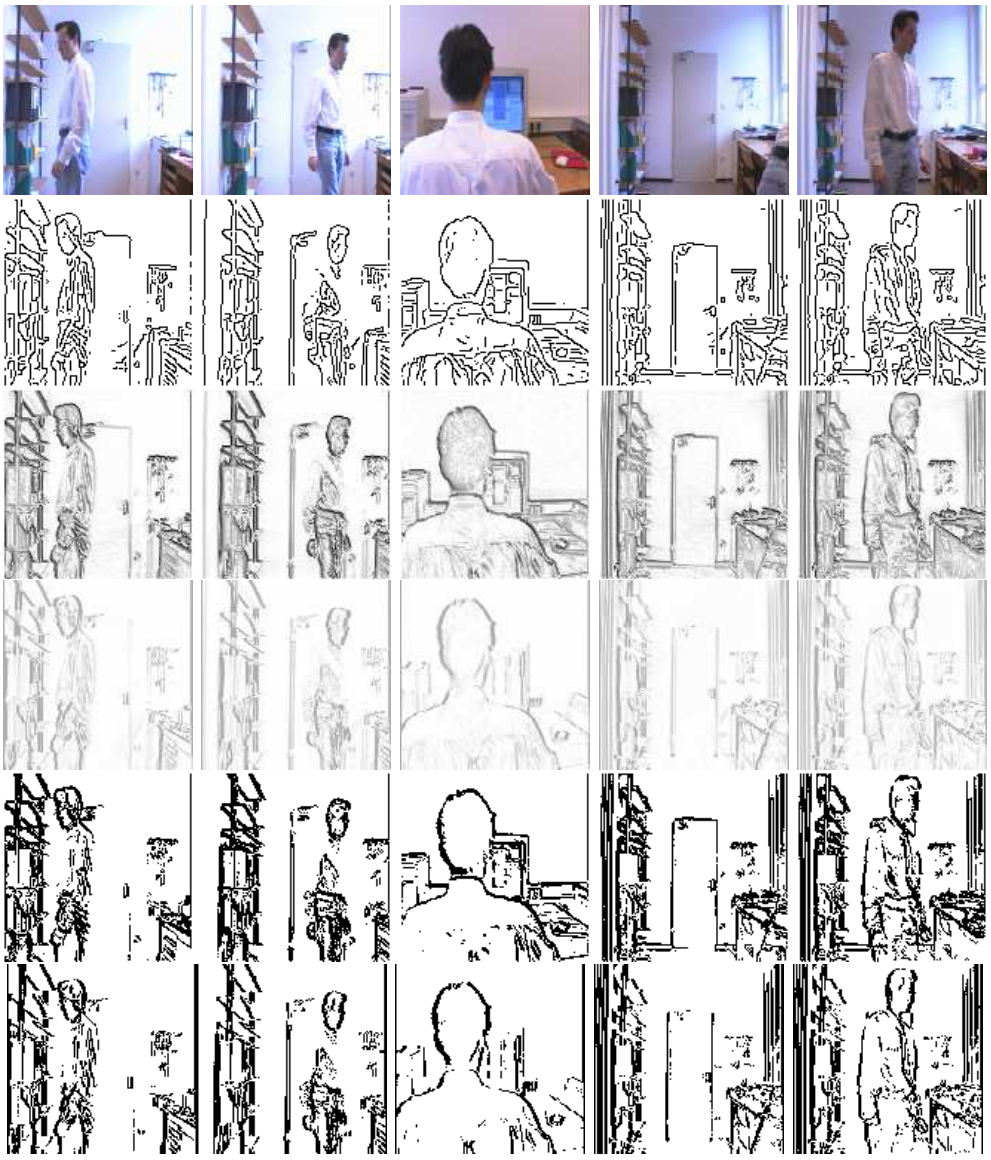

Figure 3: The first line of images shows the images used for the fitness calculations. The second line shows the edges detected with the Canny edge detector. The third line shows the response of the best individual found. The fourth line shows the response of the first of the two detectors for vertical edges. Finally the fifth and sixth lines show the binarized images of line three and four. The threshold has been manually chosen at 0.25 and 0.1 respectively.

Figure 4: Response of the best individual generated during initialization of the run which finally evolved the best individual of all experiments. This individual is not able to perform edge detection.

Figure 5: The first line of images show the set of images which were used to test the generalization capabilities of the detectors. The second line shows the output of the Canny edge detector. The third line shows the response of the best evolved edge detector. Finally the fourth line shows the binarized response of the best evolved edge detector with a manually set threshold of 0.25.

6 Acknowledgements

The author is currently supported by a scholarship according to the Landesgraduiertenförderungsgesetz.

For our experiments we have used the lil-gp Programming System, version 1.01 [6]. lil-gp was written by Douglas Zongker under the direction of Bill Punch. For image processing we have used the Vista software environment [4].

References

[1] A. J. Katz and P. R. Thrift. Generating image filters for target recognition by genetic learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(9):906–910, September 1994.

[2] David Andre. Automatically defined features: The simultaneous evolution of 2-dimensional feature detectors and an algorithm for using them. In Jr. Kenneth E. Kinnear, editor, *Advances in Genetic Programming*, pages 477–494, Cambridge, Massachusetts, 1994. The MIT Press.

[3] Anupam Joshi and Chia-Hoang Lee. Backpropagation learns marr's operator. *Biological Cybernetics*, 70:65–73, 1993.

[4] Arthur R. Pope and David G. Lowe. Vista: A software environment for computer vision research. In *CVPR'94*, 1994.

[5] Harry. G. Barrow. Learning receptive fields. In *Proceedings of the 1st International Conference on Neural Networks*, volume 4, pages 115–121. IEEE, 1987.

[6] Douglas Zongker and Bill Punch. *lil-gp 1.01 User's Manual (support and enhancements Bill Rand)*. Michigan State University, March 1996.

[7] Gerhard Roth and Martin D. Levine. Geometric primitive extraction using a genetic algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(9):901–905, September 1994.

[8] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.

[9] Hong Jeong and C. I. Kim. Adaptive determination of filter scales for edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(5):579–585, May 1992.

[10] Teuvo Kohonen. Emergence of invariant-feature detectors in the adaptive-subspace self-organizing map. *Biological Cybernetics*, 75:281–291, 1996.

[11] John R. Koza. *Genetic Programming, On the Programming of Computers by Means of Natural Selection.* The MIT Press, Cambridge, Massachusetts, 1992.

[12] John R. Koza. *Genetic Programming II, Automatic Discovery of Reusable Programs.* The MIT Press, Cambridge, Massachusetts, 1994.

[13] Ralph Linsker. Self-organization in a perceptual network. *Computer*, 21:105–117, 1988.

[14] Reinhard Lohmann. Selforganization by evolution strategy in visual systems. In Hans-Michael Voigt, Heinz Mühlenbein, and Hans-Paul-Schwefel, editors, *Evolution and Optimization '89*, pages 61–68. Akademie-Verlag, 1990.

[15] Michael Patrick Johnson, Pattie Maes, and Trevor Darrell. Evolving visual routines. In Rodney A. Brooks and Pattie Maes, editors, *Artificial Life IV, Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, pages 198–209, Cambridge, Massachusetts, 1994. The MIT Press.

[16] Ramesh Jain, Rangachar Kasturi, and Brian G. Schunck. *Machine Vision.* McGraw-Hill, Inc., New York, 1995.

[17] Ingo Rechenberg. *Evolutionsstrategie '94.* frommann-holzboog, Stuttgart, 1994.