Recognizing Speed Limit Sign Numbers by Evolvable Hardware

Jim Torresen[1], Jorgen W. Bakke[1] and Lukas Sekanina[2]

[1] Department of Informatics, University of Oslo
P.O. Box 1080 Blindern, N-0316 Oslo, Norway
[2] Faculty of Information Technology, Brno University of Technology
Bozetechova 2, 612 66 Brno, Czech Republic
E-mail: {jimtoer,jorgenwa}@ifi.uio.no, sekanina@fit.vutbr.cz

Abstract. An automatic traffic sign detection system would be important in a driver assistance system. In this paper, an approach for detecting numbers on speed limit signs is proposed. Such a system would have to provide a high recognition performance in real-time. Thus, in this paper we propose to apply evolvable hardware for the classification of the numbers extracted from images. The system is based on incremental evolution of digital logic gates. Experiments show that this is a very efficient approach.

1 Introduction

For the last years there has been an increasing focus on enhancing traffic safety by applying intelligent systems in vehicles to assist drivers. One of the reasons for this becoming possible is the recent advance in computer hardware providing affordable technology with a high processing capability. In the work presented in this paper, we consider recognizing speed limit signs. Such a system could assist drivers on signs they did not notice before passing them. It will inform drivers about the present speed limit as well as possibly giving an alert if a car is driven faster than the speed limit. More actively, it could be applied to avoid using today's physical obstacles in the road. This would require that the system could control the vehicle so that it becomes impossible to drive faster than the speed limit. This will mainly be relevant on roads with low speed limits. In the future, autonomous vehicles would have to be controlled by automatic road sign recognition. We have found very little work on speed limit sign classification. There exists a system based on Global Position System (GPS) and digital road maps with speed limits included [4]. However, such a system depends on much external infrastructure. Further, problems like lack of updated speed limits on road maps question the reliability of such a system.

This paper concerns detection of the *numbers* on speed limit signs specifically. This is by classifying numbers extracted from the sign. This will be undertaken in digital logic gates configured by evolution (evolvable hardware – EHW). The complete image recognition system will be presented shortly as well.

The goal of the design is to provide an architecture with a high classification performance at a high speed. This is important since images in such a real-time system are input from a normally *fast moving* vehicle. To obtain sufficient performance and speed at an affordable cost, we plan to migrate computational demanding parts of the software into

dedicated hardware. This paper shows the first step in this direction by undertaking the *number classification* part in dedicated (evolved) hardware.

To make the system affordable, expensive hardware is not applicable. A promising technology which also allows for adaptation and upgrades is reconfigurable technology. Thus, we believe Field Programmable Gate Arrays (FPGA) would be an appropriate technology for hardware implementation. Other parts of the system would run as software on a processor. Earlier research has shown that classification can be conducted effectively in evolvable hardware [7]. We are not aware of any other work combining evolvable hardware and road sign classification.

One of the main problems in evolving hardware systems seems to be the difficulties in evolving systems for complex real-world application. The limitation seems to be in the chromosome string length [2, 9]. A long string is normally required for solving a complex problem. However, a larger number of generations is required by the evolutionary algorithm as the string increases. This often makes the search space becoming too large. Thus, work has been undertaken to try to diminish this limitation. One promising approach – incremental evolution of EHW, was first introduced in [5] for a character recognition system. The approach is a divide-and-conquer on the evolution of the EHW system. It consists of a division of the *problem* domain together with incremental evolution of the hardware system. Evolution is first undertaken individually on a set of basic units. The evolved units are the building blocks used in further evolution of a larger and more complex system. The benefits of applying this scheme is both a *simpler* and *smaller* search space compared to conducting evolution in one single run. In this paper, it is applied to evolve a number classification architecture applied on extracted bit arrays from road images. An EHW architecture as well as how incremental evolution is applied are described.

The next section introduces the speed limit signs recognition. This is followed by an outline of the proposed evolvable hardware for classification in Section 3, respectively. Results from the implementation is given in Section 4. Finally, Section 5 concludes the paper.

2 Speed Limit Sign Recognition System

Speed limit signs have features making them feasible for automatic detection. First, there is a limited number of signs to distinguish. The following speed limit signs – see Fig. 1,

Fig. 1. Speed limit signs

are used in the experiments: 30, 40, 50, 60, 70 and 80 km/h (other speed limit signs are not included due to lack of images). The outer circle of a sign is in *red* color.

Second, there are rules (named a road grammar) for how signs can be placed along the road. After a “80” sign you will never find a “30” sign, but rather another “80” sign or a “60” sign. Third, the numbers on the signs are positioned vertically making the matching

simpler. In curves only, the signs may be marginally tilted. Forth, each time a *new* speed limit is given, there are speed limit signs at *both* sides of the driving lane. These features make it promising to undertake speed limit sign detection with a very high rate of correct prediction. A typical input image is shown in Fig. 2.

Fig. 2. An example of an input image

The algorithm is divided into *three* parts: 1) Image filtering to emphasize the red parts of the sign(s), 2) Template matching to locate possible sign(s) in an image and 3) Sign number recognition [3]. These will be presented below.

In the first part, a specialized robust color filter is applied on the image to emphasize the *red circle* surrounding the speed limit numbers. Further, this part of the algorithm effectively limits the number of red pixels in the image to be further processed.

In the second part, possible signs are located in the image by searching for the *red circle* surrounding the numbers. The search is based on matching with a set of circular templates of various size. In addition to locating the sign, the algorithm tries to reject objects that are not signs and signs that are not speed limit signs. That is, to improve recognition at the same time as reducing the computation time.

2.1 Sign Number Recognition

The last part of the algorithm is to detect the speed limit *number* on a sign. This is conducted as follows:

1. Clean the space defined by the best template (remove RED and surrounding colors), but keep the numbers.
2. Find boundaries of numbers (width and height).
3. Work only with the first number (the second is always zero).
4. Create a 7 (rows) x 5 (columns) bit array of the given number (down-scaling): Set each bit of the array to 1 if there is more BLACK than WHITE pixels, else set the bit to 0.
5. Classify the bit array using the evolved classifier circuit (described in the next section).

Some randomly picked examples of bit arrays with different numbers are included in Fig. 3.

Fig. 3. Examples of extracted arrays from real images to be classified by EHW

Number Classification in EHW To be able to correctly classify the bit array containing the extracted number, some kind of classification tool is needed. We would like to show that evolved digital logic gates are appropriate. The initial reason for this is the format of the array: it is *small* and contains *binary* values. If applying other approaches like artificial neural network (ANN), a large number of floating point operations would be needed. The EHW architecture, on the other hand, could be implemented in combinational logic where classification is performed in parallel. The next section presents the architecture applied for classification.

3 An Evolvable Architecture for Classification

In this section, the proposed classifier architecture is described. This includes the algorithm for undertaking the incremental evolution. Earlier experiments have shown that the number of generations required for evolution by this architecture can be substantially reduced compared to evolving a system directly in one operation. In experiments for prosthetic hand control [7], better classification results than for artificial neural networks were obtained.

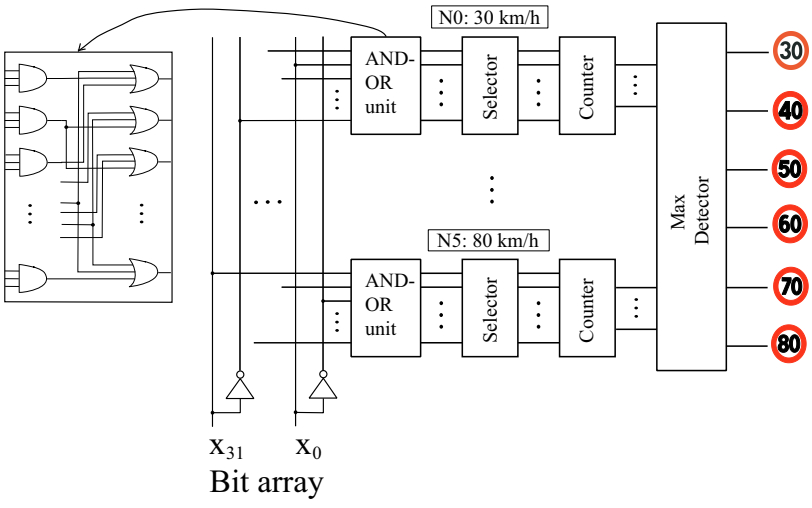

Fig. 4. The digital gate based architecture of the sign number classifier

The architecture is illustrated in Fig. 4. It consists of one subsystem for *each* of the six numbers to be classified. In each subsystem, the binary inputs $x_0 \ldots x_{31}$ (3 bits from the array are not used) are processed by a number of different units, starting by the AND-OR unit. This is a layer of AND gates followed by a layer of OR gates. Each gate has three inputs. The outputs of the OR gates are routed to the Selector. This unit selects *which* of these outputs that are to be counted by the succeeding counter. That is, for each new input, the Counter is counting the number of *selected* outputs being "1" from the corresponding AND-OR unit. Finally, the Max Detector outputs which counter – corresponding to *one* specific number, is having the largest value. If the Counter having the *largest* value corresponds to the correct number, the input has been correctly classified. E.g. if a bit array with "8" is input, we want as *many* outputs as possible of the lowermost (N5) AND-OR unit in Fig. 4 being "1". For the other five subsystems, we want as *few* outputs as possible outputs being "1". The reason for using more than one output for each number is to provide generalisation.

One of the motivations for introducing the selectors is to be able to adjust the *number* of outputs from each AND-OR unit in a flexible way. Earlier experiments have shown that this effectively increases the performance. A scheme, based on using multi-input AND gates together with counters, has been proposed earlier [10]. However, the architecture used in this paper is distinguished by including OR-gates, together with the selector units involving incremental evolution. The incremental evolution of this system can be described by the following steps:

1. **Step 1 evolution.** Evolve the AND-OR unit for each subsystem *separately* one at a time. Apply *all* vectors in the training set for the evolution of each subsystem. There are no interaction among the subsystems at this step, and the fitness is measured on the output of the AND-OR units.
2. **Step 2 evolution.** Assemble the six AND-OR units into one system as seen in Fig. 4. The AND-OR units are now fixed and the *Selectors* are to be evolved in the assembled system – in one common run. The fitness is measured using the same training set as in step 1 but the evaluation is now on the output of the Max Detector.
3. The system is now ready to be applied.

In the first step, subsystems are evolved separately, while in the second step these are evolved together. The motivation for evolving separate subsystems – instead of a single system in one operation, is that earlier work has shown that the evolution time can be substantially reduced by this approach [5, 6].

The layers of AND and OR gates in one AND-OR unit consist of 32 gates each. This number has been selected to give a chromosome string of about 1000 bits which has been shown earlier to be appropriate. A larger number would have been beneficial for expressing more complex Boolean functions. However, the search space for evolution could easily become too large. For the step 1 evolution, each gate's *inputs* are determined by evolution. The evolution is based on gate level building blocks. However, since several output bits are used to represent one number, the resolution becomes increased from the two binary levels. For the step 2 evolution, each line in each selector is represented by *one* bit in the chromosome. This makes a chromosome of 32 x 6 bits= 192 bits. If a bit is "0", the corresponding line should *not* be input to the counter, whereas if the bit is "1", the line *should* be input.

Fitness Measure

In step 1 evolution, one would normally think about measuring fitness on all the 32 outputs of each AND-OR unit. However, several earlier experiments have shown a great benefit when the fitness is measured on a *limited* number (16 are used here) of the outputs [8]. That is, each AND-OR unit still has 32 outputs but – as seen in Fig. 5, only 16 are included in the computation of the fitness function:

$$\text{Fitness} = \sum_{i=1}^{16} \text{Output OR gate } i \tag{1}$$

The 16 outputs not used are included in the chromosome and have *random* values. That is, their values do not affect the fitness of the circuit. After evolution, all 32 outputs are applied for computing the performance:

$$\text{Performance} = \sum_{i=1}^{32} \text{Output OR gate } i \tag{2}$$

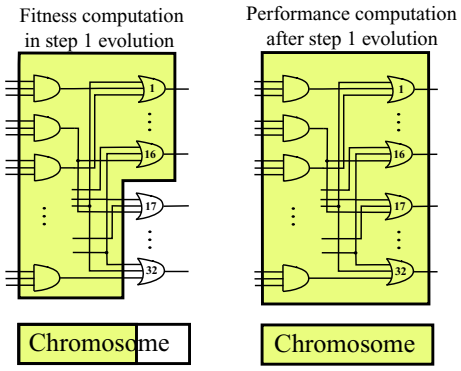

Fig. 5. A "fitness measure" equal to 16

Since 16 OR gates are used for fitness computation, the "fitness measure" equals 16. In the figure, gate 1 to 16 are used for the fitness function. However, in principle any 16 gates out of the 32 can be used. Other numbers than 16 were tested in earlier experiments but 16 showed to give the best performance results and was used in the following reported experiments. This approach has shown to be important to improve the generalisation of the circuit [7]. Only the OR gates in the AND-OR unit are "floating" during the evolution since all AND gates may be inputs to the 16 OR gates used by the fitness function. The 16 "floating" OR-gates then provide additional combination of these *trained* AND gates.

Fitness Function

The fitness function is important for the performance when evolving circuits. For the step 1 evolution, the fitness function – applied for each AND-OR unit separately, is as follows

for the number n ($n \in \{0,\ldots,5\}$) unit:

$$F_1(n) = \frac{1}{s} \sum_{(n \text{ not active})} \sum_{i=1}^{O} x_i + \sum_{(n \text{ active})} \sum_{i=1}^{O} x_i \tag{3}$$

$$\text{where } x_i = \begin{cases} 0 \text{ if } y_{i,j} \neq d_{n,j} \\ 1 \text{ if } y_{i,j} = d_{n,j} \end{cases}$$

where $y_{i,j}$ in the computed output of OR gate i and $d_{n,j}$ is the corresponding target value of the training vector j. As mentioned earlier, each subsystem is trained for *one* number (the last expression of F_1). This includes outputting "0" for input vectors for other numbers (the first expressions of F_1). The s is a scaling factor to implicitly emphasize on the vectors for the number the given subsystem is assigned to detect. Since there is a variable number of training vectors for each number, this constant was set specifically for each subsystem (as seen in the result section). The O is the number of outputs included in the fitness function and is 16 in the following experiments (referred to as "fitness measure" in the previous section).

The fitness function for the step 2 evolution is applied on the complete system and is given as follows:

$$F_2 = \sum_{j=0}^{P-1} x_j \tag{4}$$

$$\text{where } x_j = \begin{cases} 1 \text{ if } d_{n,j} = 1 \text{ and } n = i \text{ for which } \max_{i=0}^{5}(Counter_i) \\ 0 \text{ else} \end{cases}$$

This fitness function counts the number of training vectors for which the target *output* being "1" *equals* the *id* of the counter having the maximum output (only *one* output bit is "1" for each training vector). P is the total number of vectors in the training set ($P = 100$ in the following experiments).

The Evolutionary Algorithm

The simple Genetic Algorithm (GA) – given by Goldberg [1], was applied for the evolution with a population size of 50. For each new generation an entirely new population of individuals is generated. Elitism is used, thus, the best individuals from each generation are carried over to the next generation. The (single point) crossover rate is 0.8, thus the cloning rate is 0.2. Roulette wheel selection scheme is applied. The mutation rate – the probability of bit inversion for each bit in the binary chromosome string, is 0.01.

Various experiments have been undertaken to find appropriate GA parameters. The ones that seemed to give the best results were selected and fixed for all the experiments. This was necessary due to the large number of experiments that would have been required if GA parameters should be able vary through all the experiments. The initial experiments indicated that the parameter setting was not a major critical issue.

The proposed architecture fits into most FPGAs. The evolution is undertaken off-line using software simulation. However, since no feed-back connections are used and the number of gates between the input and output is limited, the real performance should equal the simulation. Any spikes could be removed using registers in the circuit.

For each experiment presented, four different runs of GA were performed. Thus, *each* of the four resulting circuits from step 1 evolution is taken to step 2 evolution and evolved for four runs.

Effective Processing in EHW

In the last part of Section 2.1, the benefits of the EHW architecture compared to ANN were introduced. In this section more details will be given. The EHW architecture do classification in *parallel* for all the different numbers. Each path consists of *two* layers of gates in the AND-OR unit. The selector could be implemented with a *one* gate layer. The counter could effectively be implemented as a tree of gates. The Max Detector would consist of a comparator structure. Thus, all can be implemented in combinational logic. However, to improve the speed if necessary, registers could be introduced in the architecture together with pipelined processing. Thus, the architecture should be able to process one bit array *each* clock cycle. A neural network would typically have 32 inputs, 32 hidden units and 6 outputs. This would result in at least (32x32 + 32x6 = 1216) multiply-accumulate operations and (32+6) sigmoid function operations. This would normally have to be executed as serial floating point operations on a processor. This seems to be a waste of resources since the input would still be binary values.

4 Results

This section reports the results from the experimental work undertaken. A database of 198 images from traffic situations were used in the experiments. 115 contained a speed limit sign and 83 contained other signs or no sign at all. Many of the images were in various ways "difficult" (different brightness on sign, faded color on sign etc). The results were as follows (before number recognition was undertaken):

- Is there a speed limit sign? A speed limit sign was found in 100 of the 115 images (87%). In those not found, the system stated that a speed limit sign was not present in the image.
- 78 of the 83 images without a speed limit was correctly refused (94%). Thus, only five images were sent to the final sign number recognition.

For *all* the 100 images (P) that the system correctly detected a speed limit sign, the extracted number arrays were applied for evolving the hardware to perform classification. The number of arrays for the different speed limits is given in Table 1. The values for fitness scaling (see equation (3)) are also included. They are computed according to the following equation:

$$s = \frac{P}{P_n} \cdot k = \frac{100}{P_n} \cdot 0.7 = \frac{70}{P_n}$$

P_n is the number of arrays for the given class (column two in the table) and k is a constant determining the ratio between the fitness from P_n training vectors (active) and $(P - P_n)$ training vectors (not active). I.e. for speed limit 40, the *total* fitness for the 11 arrays counts a factor 0.7 less than the *total* of the fitness for the other 89 arrays. The value of k was determined by experiments.

Table 1. The number of extracted arrays classified for each speed limit

Speed limit	Number of arrays (P_n)	Fitness scaling factor (s)
30	4	18
40	11	7
50	32	2
60	35	2
70	12	7
80	6	12

Table 2 summarizes the classification performance. All the experiments are based on a "fitness measure" equal to 16. In the table, "A" shows the performance when only the outputs applied in the fitness function are applied in computing the performance (after step 1 evolution is finished). However, we see that it is better to apply all 32 outputs ("B") since the average performance is about 8% higher. Thus, 16 OR gates with random input connections improve the performance. We see the importance of the step 2 evolution in "C" in which the performance is substantially increased. The average performance is more than 15% higher than for "B". The best classifier provided a performance of 96%.

Table 2. The correct number classification performance in %

Type of system	Step of evolution	Min	Max	Average
A: Performance of 16 outputs	1	67.6	75.7	71.44
B: Performance of 32 outputs	1	71.8	90.9	79.2
C: Performance of 32 outputs	2	92.9	96.0	94.5

The performance for each speed limit for the best classifier is shown in Table 3. Only two speed limits have less than 100% correct classification.

Table 3. Performance of the best classifier architecture evolved

Speed Limit	30	40	50	60	70	80
Performance (in %)	100	100	84.4	91.4	100	100

In step 1 evolution, each AND-OR unit was evolved for 25,000 generations. Step 2 evolution needed less than 500 generations before the performance stopped increasing. We have not yet studied the performance of the 5 images that were wrongly sent to the number recognizer. However, these could probably be eliminated by a combination of matching on the "0" number and applying road grammar (as explained in Section 2). We have not fully explored all possible parameter settings for the architecture. Thus, there is a potential for further improving the performance. Further, a threshold unit could be introduced to avoid misclassifying numbers by rather indicating no distinct match.

In addition to achieve a high rate of correct classification, we have made much effort at the same time to reduce the processing time. This would be highly needed in a future real-time system. By applying digital logic gates for the number classification, we have almost eliminated the processing time for this operation. To have a robust and reliable system, more images should be analyzed and this is a part of our future work. With the promising results so far, future work also consists of further improving the algorithms and implementing the most computational demanding parts in special hardware. More image

data should be applied to better test the generalisation performance. Further, possible inclusion of evolution in other parts of the system is of interest as well. Finally, a prototype to be applied in a vehicle will be implemented.

5 Conclusions

The paper has presented a novel approach to detect extracted numbers from speed limit signs using evolvable hardware. It is focused on reducing computation time at the same time as getting high recognition performance. The results show that performance can be substantially increased by applying incremental evolution. In average, 94.5 % correct classification is achieved in the experiments.

Acknowledgments

The research is funded by the Research Council of Norway through the project *Biological-Inspired Design of Systems for Complex Real-World Applications* (project no 160308/V30). The research is also performed with funding from the Grant Agency of the Czech Republic under No. 102/03/P004 *Evolvable hardware based application design methods*.

References

1. D. Goldberg. *Genetic Algorithms in search, optimization, and machine learning*. Addison–Wesley, 1989.
2. W-P. Lee, J. Hallam, and H.H. Lund. Learning complex robot behaviours by evolutionary computing with task decomposition. In A. Birk and J. Demiris, editors, *Learning Robots: Proc. of 6th European Workshop, EWLR-6 Brighton*, volume 1545 of *Lecture Notes in Artificial Intelligence*, pages 155–172. Springer-Verlag, 1997.
3. L. Sekanina and J. Torresen. Detection of Norwegian speed limit signs. In *Proc. of the 16th European Simulation Multiconference (ESM2002)*, pages 337–340. SCS Europe, June 2002.
4. R. Thomas. Less is more [intelligent speed adaptation for road vehicles]. *IEE Review*, 49(5):40–43, May 2003.
5. J. Torresen. A divide-and-conquer approach to evolvable hardware. In M. Sipper et al., editors, *Evolvable Systems: From Biology to Hardware. Second International Conference, ICES 98*, volume 1478 of *Lecture Notes in Computer Science*, pages 57–65. Springer-Verlag, 1998.
6. J. Torresen. Scalable evolvable hardware applied to road image recognition. In J. Lohn et al., editor, *Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware*, pages 245–252. IEEE Computer Society, Silicon Valley, USA, July 2000.
7. J. Torresen. Two-step incremental evolution of a digital logic gate based prosthetic hand controller. In *Evolvable Systems: From Biology to Hardware. Fourth International Conference, (ICES'01)*, volume 2210 of *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag, 2001.
8. Jim Torresen. A scalable approach to evolvable hardware. *Journal of Genetic Programming and Evolvable Machines*, 3(3):259–282, 2002.
9. X. Yao and T. Higuchi. Promises and challenges of evolvable hardware. In T. Higuchi et al., editors, *Evolvable Systems: From Biology to Hardware. First International Conference, ICES 96*, volume 1259 of *Lecture Notes in Computer Science*, pages 55–78. Springer-Verlag, 1997.
10. M. Yasunaga et al. Genetic algorithm-based design methodology for pattern recognition hardware. In *Evolvable Systems: From Biology to Hardware. Third International Conference, ICES 2000*, volume 1801 of *LNCS*, pages 264–273. Springer-Verlag, 2000.