Is Depth Information and Optical Flow Helpful for Visual Control?

Johannes Hansen and Marc Ebner
Ernst-Moritz-Arndt-Universität Greifswald
Institut für Mathematik und Informatik
Walther-Rathenau-Straße 47, 17487 Greifswald, Germany
Tel: (+49)3834/86-4646, Fax: (+49)3834/86-4640
marc.ebner@uni-greifswald.de

***Abstract*—The human visual system was shaped through natural evolution. We have used artificial evolution in order to investigate whether depth information and optical flow are helpful for visual control. Our experiments were carried out in simulation. The task was controlling a simulated racing car. We have used The Open Racing Car Simulator TORCS for our experiments. Genetic Programming was used to evolve visual algorithms that transform input images (color, optical flow or depth information) to control commands for a simulated racing car. We found that significantly better solutions were found when color, depth and optical flow were available as input together compared to either color, depth or optical flow alone.**

***Index Terms*—Visual Control, Genetic Programming, Optical Flow, Depth Map**

I. Introduction

With this contribution, we investigate whether depth and motion provide an evolutionary advantage compared to color alone. As a test environment, we have used The Open Racing Car Simulator TORCS [1]. Simulated evolution [2] was used to evolve control algorithms for the racing car. These algorithms use screen grabs from the racing car simulator. The screen grabs are processed using elementary computer vision operators. The output of these algorithms control the steering wheel as well as gas/brakes of the car [3]. OpenCV [4], [5], an open source library for computer vision, was used for image processing. Genetic Programming [6], [7], [8] was used to evolve visual algorithms. We will see that significantly better solutions are found if color, depth and optical flow are all available.

This article is structured as follows. The next section gives a brief introduction to the visual system. Section III describes the racing car simulator. Section IV explains how data from the simulator is used to compute optical flow. A brief introduction to Genetic Programming is given in Section V. Related work on visual control using Genetic Programming is discussed in Section VI. How we have used genetic programming to evolve visual control algorithms is explained in Section VII. Our results are described in Section VIII. We provide our conclusions in Section IX.

II. The Visual System

The human visual system was shaped through natural evolution [9], [10]. Visual processing starts with light entering the eye. This light is measured by two different types of receptors inside the retina [11], [12]: rods and cones. The rods mediate vision when little light is available. They have a much higher sensitivity than the cones. Cones are in charge of color vision. Three different types of cones exist which absorb light mainly in the red, green and blue parts of the visual spectrum.

Some preprocessing occurs inside the retina. Information flows from the retinal receptors to the retinal ganglion cells. This information exists the eye at the blind spot, passes through the lateral geniculate nucleus and then reaches the primary visual cortex or area V1. Area V1 is highly structured [13]. Cells within ocular dominance segments respond primarily to stimuli from one or the other eye. V1 contains columns with orientation sensitive cells. Blobs with color or lightness sensitive cells also exist. In other words, visual information is analyzed using a retinotopic map with respect to different aspects. Indeed the entire visual cortex is highly structured.

Color, shape and motion appear to be processed by separate visual areas [14], [15]. Color is a product of the brain. It is processed in visual area V4. Shape is processed in V3 and motion is processed in V5. A dedicated area for face and object recognition also exist. It is also interesting that color and motion is not perceived synchronously. Moutoussis and Zeki [16] demonstrated that color is perceived earlier than motion. The brain appears to bind visual attributes that are perceived together.

III. TORCS

TORCS [1], is an open source racing car 3D simulation. A sample screenshot is shown in Figure 1. We have used this simulator as a test environment in order to evaluate whether depth information and optical flow are helpful for visual control. Currently, Berhard Wymann maintains the TORCS project. Its original creators were Eric Espié and Christophe Guionneau. The TORCS simulator provides several different racing tracks. A player can choose among different cars when playing the game. Several different opponents are available to race against. A split screen mode is also available. Up to four human players are supported.

Supported controls are a joystick, mouse and keyboard. Some steering wheels are also supported. The game features realistic 3D graphics, lighting, smoke and skid marks. Game

Figure 1. The Open Racing Car Simulator

physics include simulation of collisions, tire and wheel properties and a simple damage model. It even includes a simple aerodynamic model with slip-streaming and ground effects. The Open Racing Car Simulator was used in several different scientific competitions [17], [18]. For these competitions, participants are developing AI methods that drive the racing car along its track. Usually, a client-server architecture is used and competition participants develop a client that sends its control commands to the TORCS server.

We have modified TORCS slightly in order to use it for our experiments. We extract screen grabs from the graphics buffer. A dense depth map is extracted form the so called Z-buffer of the graphics library [19], i.e. both color and depth are readily available form the graphics context. In addition to color and depth, we also provide optical flow to the visual control algorithms. How we compute optical flow is described in the next section.

IV. Computing Optical Flow

Optical flow describes the pattern of motion of objects which are seen on the screen. This information is very helpful for visual control. We would like to estimate a dense flow field, i.e. optical flow for each image pixel. Let $\mathbf{v}(x, y, t) = (v_x, v_y)$ be the optical flow, estimated for image pixel (x, y) in an image taken at time t, then the image content in the vicinity of pixel (x, y) will be found at position $(x + v_x\Delta t, y + v_y\Delta t)$ in an image taken at time $t + \Delta t$ provided that the object is moving with constant velocity across the screen.

Several different methods have been developed to compute optical flow from visual input [20], [21], [22], [23]. In recent years, the accuracy of the methods have been considerably improved. Many methods for computing optical flow are based on partial derivatives. However, block based methods are also used. Block based methods search for pixels within an area of a given pixel in a subsequent image to determine image motion.

Estimating optical flow from two subsequent images is an expensive image operation. Therefore, we have used the depth map and the known motion of the race car to compute optical flow. The depth map is readily available from the graphics library. It is a by-product of rendering the scene. The depth map contains, for each image pixel, the distance from the object to the camera along the Z-axis. The motion of the race car is available directly from the race car simulator.

Let $d(x, y) = d_{x,y}$ be the depth map of the current image shown on the screen. We assume that all screen coordinates are specified relative to the center of the screen. Let f be the focal length of the camera. The location of an object of the scene which is shown at image pixel (x, y) has coordinates (X_S, Y_S, Z_S) inside the camera coordinate system centered on the car driver.

$$\begin{bmatrix} X_S \\ Y_S \\ Z_S \end{bmatrix} = \frac{d_{x,y}}{f} \begin{bmatrix} x \\ y \\ f \end{bmatrix} \tag{1}$$

The coordinates (X_S, Y_S, Y_S) are relative to the viewer sitting inside the car, i.e. these are eye-coordinates.

Let $\mathbf{R}$ be the inverse 3×3 rotation matrix that describes the rotatory motion of the racing car from one time step of the simulation to the next. Let $\mathbf{D}$ be the inverse vector which describes the translatory motion of the racing car from one time step of the simulation to the next. Hence, after the racing car has moved, the point (X_S, Y_S, Y_S) will have moved to a location (X'_S, Y'_S, Z'_S) relative to the eye of the driver.

$$\begin{bmatrix} X'_S \\ Y'_S \\ Z'_S \end{bmatrix} = \mathbf{R} \begin{bmatrix} X_S \\ Y_S \\ Z_S \end{bmatrix} + \mathbf{D} \tag{2}$$

The coordinate (X'_S, Y'_S, Z'_S) can be projected onto the screen using the known focal length of the camera. Let (x', y') be the screen coordinates of (X'_S, Y'_S, Z'_S), then we have

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = f \begin{bmatrix} X'_S/Z'_S \\ Y'_S/Z'_S \end{bmatrix} \tag{3}$$

Optical flow can then be computed by subtracting the screen coordinate before the racing car has moved from the screen coordinate after the car has moved.

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix} - \begin{bmatrix} y \\ y \end{bmatrix} \tag{4}$$

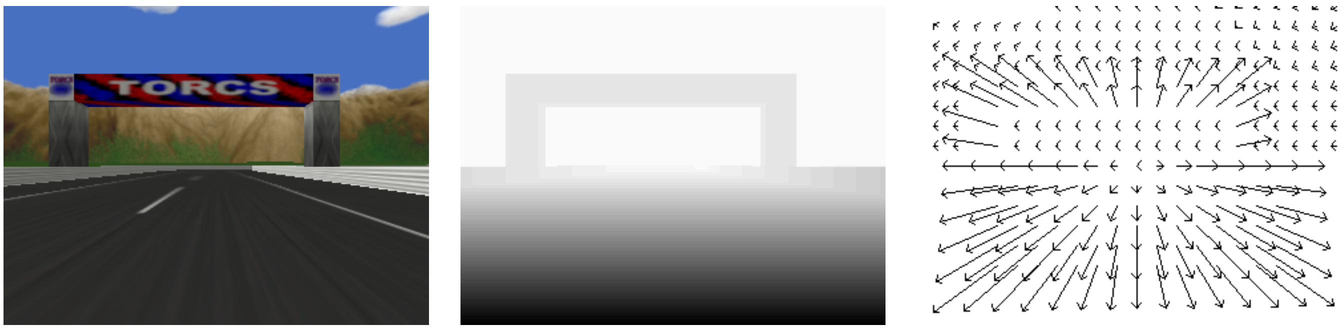
Since the depth map and the known motion of the racing car is correct, the optical flow will also be correct. Figure 2 shows the computed optical flow for a sample image.

It would be possible to compute optical flow directly from the input images using an algorithm based on partical derivatives or using block based methods. However, this would take considerably more computing resources and also would have the disadvantage that the estimated optical flow would not be 100% correct for all image pixels.

Next, we will describe Genetic Programming, which we have used to evolve visual control algorithms.

V. Genetic Programming

Genetic programming [6], [7], [8] is an evolutionary algorithm. Evolutionary algorithms use simulated evolution to solve optimization problems. Such algorithms work with a population of individuals. Each individual represents a possible



(a) (b) (c)

Figure 2. Computing optical flow from game angine data. (a) input image (b) depth map (c) optical flow computed using the depth map and the known ego-motion of the car.

solution to the optimization problem. Darwinian selection is used to select above average individuals in order to create an offspring population, i.e. a new generation of individuals.

In Genetic Programming, individuals are represented as trees. The fitness of an individual describes how well this individual solves the given problem. The main operators of an evolutionary algorithms are selection, reproduction and variation. Above average individuals are selected to create offspring. For our experiments we have used four genetic operators: reproduction, mutation, ERC-mutation and crossover. Each genetic operator is applied with a certain probability p_{rep}, p_{mut}, $p_{\text{ERC-mut}}$, p_{cross} respectively. These four probabilities sum to one.

Individuals of the first generation are created using the so called ramped-half-and-half initialization [6]. In order to create offspring, one genetic operator is randomly selected (using the four probabilities). The reproduction operator simply creates a copy of the genetic material of the individual, i.e. the tree. Mutation, ERC-mutation and crossover create offspring that are similar but not identical to their parents. Depending on the type of operator, one or two parents are selected from the population. Typically, tournament selection is used to select new parents. For tournament selection, n_T individuals are selected with uniform probability from the population. These n_T individuals form a tournament. The individual with highest fitness is the winner of the tournament and becomes a parent. This parent will then create offspring using one of the genetic operators.

The genetic operators are illustrated in Figure 3. Figure 3(a) shows the reproduction operator which is applied with probability p_{rep}. Figure 3(b) shows the mutation operator which is applied with probability p_{mut}. Figure 3(c) shows the ERC-mutation operator which is applied with probability $p_{\text{ERC-mut}}$. Figure 3(d) shows the crossover operator which is applied with probability p_{cross}.

If the reproduction operator is applied, then a copy of the parent individual is created. Next, a node of the tree is randomly selected. Internal nodes are selected with probability 0.9, while external nodes are selected with probability 0.1. Finally, the selected node is replaced with a randomly generated sub-tree. The method which is used to create this sub-tree is the same method that is used to create the individuals of the first generation. For ERC-mutation, a single node which contains a so called ephemeral random constant (ERC) is selected. All ERCs located within this subtree are muated. We originally intended to use Gaussian mutation (like an evolution strategy [24]) to slightly alter this ERC value, i.e. $v := v \cdot e^{0.01z}$ where v is the original value of the constant and z is a normally distributed random value with mean 0 and standard deviation 1. However, we have actually used $v := v \cdot z \cdot 0.01$ which pulls the ERC towards zero and may also change the sign of the constant. For crossover, two parent individuals are selected. Next, two nodes are randomly selected (one for each tree). Again, internal nodes are selected with probability 0.9, while external nodes are selected with probability 0.1. Then the two sub-trees (together with the selected nodes) are exchanged between the two individuals. Trees are limited to a maximum depth of 17.

Whenever an offspring is created, it is inserted into the next generation of individuals. The process of selecting a genetic operator and creating offspring continues until the next generation is filled. Usually an evolutionary run is terminated after a certain number of generations have been created. The individual with highest fitness that was found during all these generations is the solution that solves our problem best.

For our experiments, we have used the Evolutionary Computation Library ECJ developed by Sean Luke [25]. Development of ECJ started in 1998 and is a mature library for evolutionary computation.

VI. Visual Control using Genetic Programming

Our racing car is controlled through visual input alone. We only use the images obtained from screen grabs and the optical flow. Data from the game engine is not used to control the car. It is only used to compute optical flow as described above.

Genetic Programming has been used by Winkler and Manjunath [26] for object detection. Johnson et al. used it to evolve visual routines [27]. Ebner and Tiede [28] have previously evolved controllers for TORCS using Genetic Programming. However, for this work, input was taken directly from the game engine and not from screen grabs. Koutnik et al. [29] have used an evolutionary algorithm to evolve compressed encodings of a recursive neural network to control a racing car in TORCS. Tanev and Shimohara [30], [31] have used a genetic algorithm

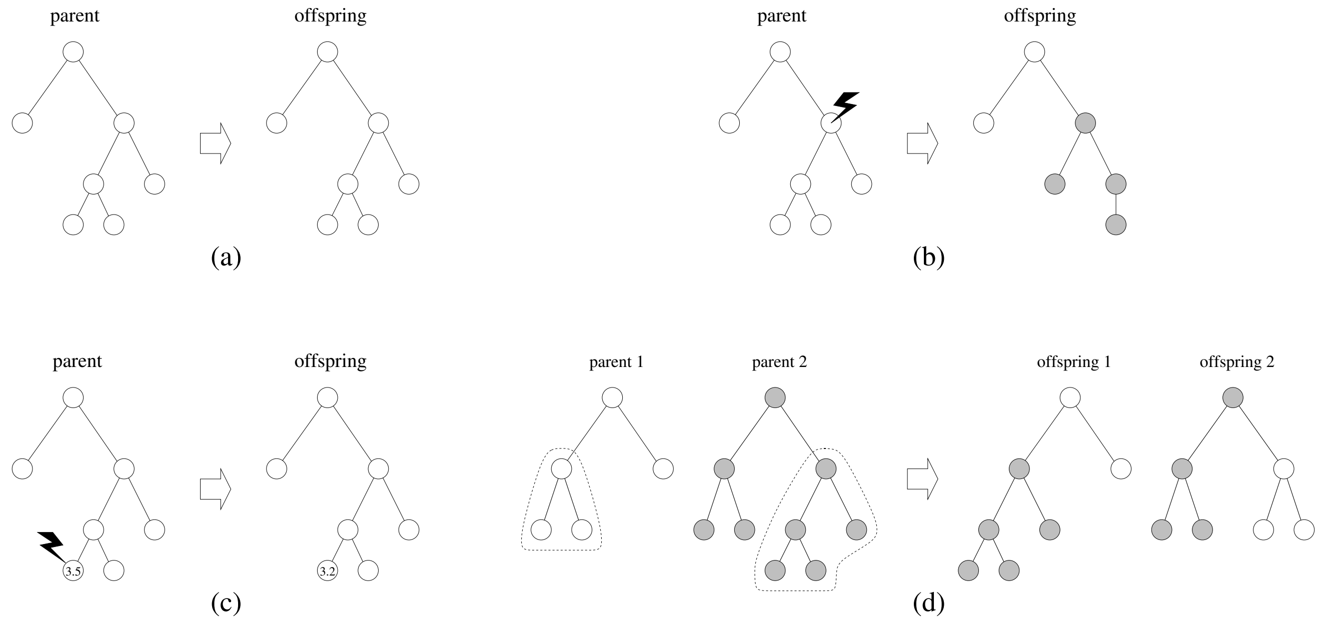


Figure 3. Genetic operators. (a) reproduction (b) mutation (c) ERC-mutation (d) crossover

to evolve parameters that will control an actual scale model of a car using visual input from an overhead camera.

Other researchers have used Atari Video games for training game players [32]. Hausknecht et al. [33], [34] evaluated neuro evolutionary methods for general game playing of Atari video games. They found that HyperNEAT was the only neuro-evolution algorithm able to play based on raw-pixel input from the games. Minh et al. [35], [36] created a deep neural network that was trained using reinforcement learning. It was able to achieve a level comparable to human players. Deep learning in combination with Monte-Carlo tree search planning was used by Guo et al. [37]. Parker and Bryant [38], [39] evolved controllers for Quake II which used only visual input.

VII. Materials and Methods

We are using Strongly Typed Genetic Programming [40] to evolve two trees. The first tree is used to control the steering wheel. The second tree is used to control the velocity of the racing car. The terminal symbols are shown in Table I. We work with two return types: float and image. The only terminal symbol returning a floating point value is an ephemeral random constant. An ephemeral random constant is a random floating point value from the range $[0, 1]$. Once a node with an ephemeral random constant is created, it stays constant throughout the life of the node. It may be modified by the ERC-mutation operator, though.

The remaining terminal symbols provide access to visual information obtained via screen grabs from the game engine. This screen grab is scaled down to one third of its original size. All pixel values are transformed to the range $[0, 1]$. All terminal symbols returning image data provide single band images: red channel (`imageR`), green channel (`imageG`), blue channel (`imageB`), cyan channel (`imageC`), magenta channel (`imageM`), yellow channel (`imageY`), gray channel (`imageGray`). The depth map of this input image is available through the terminal symbol (`depthMap`). Optical flow is computed using the depth map and the known ego-motion of the car which is available from the game engine. Since optical flow is a two-dimensional vector, the x-component of this vector is made available through the terminal `opticalFlowX` and the y-component is made available through the terminal `opticalFlowY`. All image data is downscaled to one third of this size of the original image. Pixel values are scaled to the range $[0, 1]$.

The set of elementary functions is shown in Table II and Table III. Table II shows elementary functions which return a floating point value. Table III shows elementary functions which return an entire image. We have used standard arithmetic functions like addition and multiplication, computation of minimum and maximum. We have also included functions which search for maximum and minimum values inside the image. These functions return either the x or the y coordinate of the position where the extremum was found. A Gaussian filter is also available. If we apply a Gaussian filter to a gray scale input image and then a function which locates the maximum, we can locate the brightest point in the image. The function `extractNAME` extracts smaller regions from the input image. The size of the region is one third of the image. The location of the region can be specified through the parameters of the function. All bands (see terminal symbols) are available for this extraction operation.

This functions is very useful to control the steering wheel of the car. This is illustrated in Figure 4. Suppose we extract a region from the depth map from the left hand side of the image and we extract another region from the depth map from the right hand side of the image. If we apply the `avg` function which computes the average depth within these two areas then we can compare both average depths to control the steering

Table I
TERMINAL SYMBOLS.

Name	Return Type	Description
`ERC`	float	Ephemeral random constant in the range $[0, 1]$
`imageR`	Image	input image (red channel)
`imageG`	Image	input image (green channel)
`imageB`	Image	input image (blue channel)
`imageC`	Image	input image (cyan channel)
`imageM`	Image	input image (magenta channel)
`imageY`	Image	input image (yellow channel)
`imageGray`	Image	input image (gray channel, average RGB)
`depthMap`	Image	input image (depth map)
`opticalFlowX`	Image	optical flow (horizontal component)
`opticalFlowY`	Image	optical flow (vertical component)

Table II
ELEMENTARY FUNCTIONS. THE RETURN VALUE OF THE NODE IS o.

Name	Output Type	Description
`abs(float v)`	float	absolute value, $o = \|v\|$
`round(float v)`	float	round function, $o = \text{round}(v)$
`floor(float v)`	float	floor function, $o = \lfloor v \rfloor$
`ceil(float v)`	float	ceil function , $o = \lceil v \rceil$
`neg(float v)`	float	negate input, $o = -v$
`sqrt(float v)`	float	square root, $o = \sqrt{v}$
`minLocX(Image c)`	float	x-coordinate (range $[0, 1]$) of minimum $c(x, y)$
`minLocY(Image c)`	float	y-coordinate (range $[0, 1]$) of minimum $c(x, y)$
`maxLocX(Image c)`	float	x-coordinate (range $[0, 1]$) of maximum $c(x, y)$
`maxLocY(Image c)`	float	y-coordinate (range $[0, 1]$) of maximum $c(x, y)$
`avg(Image c)`	float	average value of all pixels, $o = \sum_{x,y} c(x, y)$
`min(float a,float b)`	float	minimum value, $o = (a < b)?b : a$
`max(float a,float b)`	float	maximum value, $o = (a > b)?b : a$
`q-quantile(Image c,float q)`	float	q-quantile of the image
`add(float a,float a)`	float	addition, $o = a + b$
`mult(float a,float b)`	float	multiplication, $o = a \cdot b$

Table III
ELEMENTARY FUNCTIONS. THE RETURN VALUE IS $o(x, y)$ FOR EACH PIXEL (x, y) OF THE OUTPUT IMAGE.

Name	Output Type	Description
`abs(Image c)`	Image	absolute value $o(x, y) = \|c(x, y)\|$
`sqrt(Image c)`	Image	square root, $o(x, y) = \sqrt{c(x, y)}$
`min(Image c)`	Image	minimum value, $o(x, y) = \min_{x,y} c(x, y)$
`max(Image c)`	Image	maximum value, $o(x, y) = \max_{x,y} c(x, y)$
`add(image a,image b)`	Image	addition, $o(x, y) = a(x, y) + b(x, y)$
`constImage(float v)`	Image	constant image, $o(x, y) = v$
`invert(Image c)`	Image	image, $o(x, y) = \max - c(x, y)$ where $\max$ is the maximum value of all image pixels
`gauss(float v,Image c)`	Image	gaussian filter with kernel $e^{-x^2/(2\sigma^2)}$ where $\sigma = 0.3(\|v\| - 1) + 0.8$
`median(float v,Image c)`	Image	median filter with size $[2\|v\| + 1]$
`binary(Image c,float v)`	Image	binary threshold, $o(x, y) = (c(x, y) > v)?1 : 0$
`clamp(Image c,float v)`	Image	binary threshold, $o(x, y) = (c(x, y) > v)?v : c(x, y)$
`thresholdPass(Image c,float v)`	Image	threshold $o(x, y) = (c(x, y) > v)?c(x, y) : 0$
`thresholdZero(Image c,float v)`	Image	threshold $o(x, y) = (c(x, y) > v)?0 : c(x, y)$
`avgThreshold(Image c)`	Image	average threshold, $o(x, y) = (c(x, y) > a)?1 : 0$ with $a = \sum_{x,y} c(x, y)$.
`localThreshold(Image c)`	Image	local average threshold, $o(x, y) = (c(x, y) > a(x, y))?1 : 0$ where $a(x, y)$ is obtained by convolving the input image with a Gaussian kernel $e^{-x^2/(2\sigma^2)}$ with standard deviation $\sigma = 1.1$
`extractNAME(float s,float t)`	Image	10 variants of this elementary function exist, with `NAME` $\in$ {`R, G, B C, M, Y, Gray, DepthMap, OpticalFlowX, OpticalFlowY`}. The parameters (s, t) specify the upper left corner of a rectangular area which is extracted from the current input image (as specified by `NAME`). The width and height of this area is one third of the input image.

wheel. This simple control algorithm will drive the car in the direction where more space is available.

We have carried out four sets of experiments in order to evaluate whether depth information is helpful in controlling the racing car. For our experiments, we used the same basic set of elementary functions, but varied the input information that was made available to the evolved individuals. For experiment A), only color information was available. For experiment B), only the depth map was available, for experiment C), only optical flow was provided and finally, for experiment D), all of the visual information (color, depth, and optical flow) was provided. The terminal symbols and elementary functions for the 4 experiments are shown in Table IV. Note that some of the functions listed under "All" accept both floating point values and images as input.

The track that we have used for all of our experiments is shown in Figure 5. The red arrow illustrates the direction in which the race will start. A path taken by an evolved individual is shown overlayed on this track (green line). The end of this path is marked with a green cross. At this point the evolved driver lost control of its car and crashed into the border of the track.

The task is to evolve visual controllers that will drive the car along the track. Therefore, fitness is computed by considering distance traveled along the track. In addition, the damage attained is also used for the fitness computation. Individuals which stay away from the border of track and manage to avoid damage will receive higher fitness values. Let d be the distance traveled along the track (in meters). Let a be the amount of damage attained (with range $[0, 1]$ where 1 is a completely damaged car). Then fitness f_i of individual i is given as

$$f_i = \begin{cases} -50 & \text{if controller is disqualified} \\ d \cdot (1-a)^2 & \text{not disqualified,} \\ & \text{only single sided steering} \\ \max\{2d \cdot (1-a)^2, 0\} & \text{not disqualified,} \\ & \text{steering toward left and right} \end{cases} \tag{5}$$

A controller is disqualified if it (a) drives along the track in the wrong direction, or (b) does not use the steering wheel, i.e. the first tree returns a constant value or (c) does not use the gas pedal, i.e. the second tree returns a constant value. If the controller is disqualified, it will be penalized with a fitness value of -50. Otherwise it will receive a fitness of $d \cdot (1-a)^2$. This fitness value is doubled if the controller turns the steering wheel to the left as well as to the right.

For each experiment, 10 runs with different initializations of the random number generator were performed. For each run, a population of 200 individuals was evolved for 99 generations. Individuals were selected using tournament selection with $n_T = 7$. Crossover was applied with probability $p_{\text{cross}} = 0.2$, mutation was applied with probability $p_{\text{mut}} = 0.4$, ERC mutation was applied with probability $p_{\text{ERC-mut}} = 0.3$ reproduction was applied with probability $p_{\text{rep}} = 0.1$. Ramped half-and-half initialization was used to initialize the individuals of the first generation with depth ranging from 2 to 6.

VIII. Results

Figure 6 shows the best fitness values obtained for all four experiments. By far the highest fitness values were reached when only depth information was available. It is clear (as we have described above) that this type of information is helpful in controlling the car. Optical flow also seemed to be helpful. Indeed, it is well known that bees use optical flow to achieve centering behavior [41], [42]. This ability (comparison of lateral optical flow) has also been used for visual control in robotics [43], [44].

Figure 7 shows the average best fitness for all four experiments. Using only color information resulted, on average, in less average best fitness compared to using depth information, optical flow or all three after 99 generations. Average best fitness at generation 99 is summarized in Table V. This table also shows the overall best fitness obtained in all of the 40 runs (ten per experiment) which we have carried out.

We have used a Mann Whitney U Test to compare these averages as shown in Table VI. We confirmed that the maximum fitness at the end of 99 generations was not normally distributed using a Kolmogorov-Smirnov-Test. Results when using depth information or optical flow were not significantly better compared to when only visual information was used. Average fitness improved considerably when depth information or optical flow was used for visual control. However, the difference was not significantly different because the Mann Whitney U Test compares results on an ordinal scale.

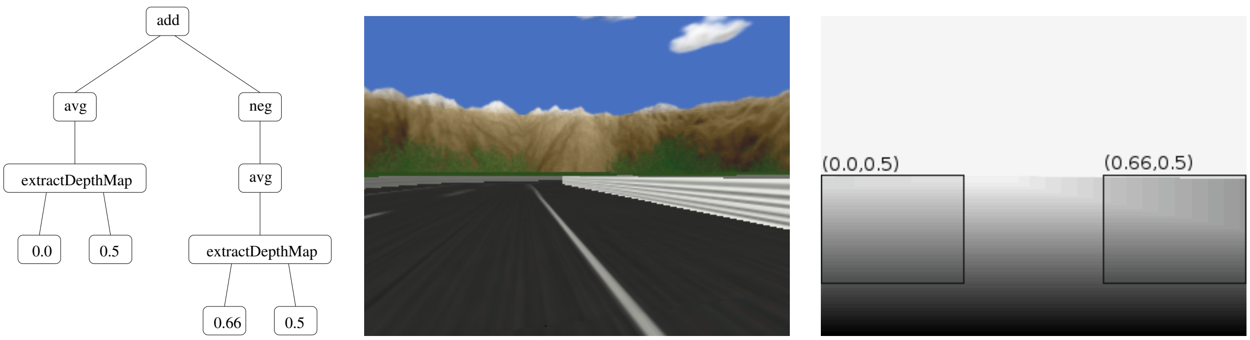
Only when all visual data (color, depth, and optical flow) is combined, do we get significantly better results than when either visual input is used alone. Interestingly, the human brain evaluates its visual input with respect to color (in V4) and motion (in V5) [45], [11]. Ocular dominance columns are found in V1. Depth information can be computed from disparity information, i.e. a lateral offset of visual information between the two eyes [46]. The human visual system somehow combines this visual information in higher areas to achieve visual control.

IX. Conclusion

We have used an evolutionary algorithm (Genetic Programming) to evolve image processing algorithms to control a racing car. These algorithms are able to process various types of visual information: color, depth information or optical flow. Each individual consists of two trees. The first tree is used to control the steering wheel and the second tree is used to control the acceleration of the car. Several elementary visual operations like Gaussian smoothing, addition, multiplication, threshold or locating a maximum or minimum response are also provided. These are all elementary operations that can be performed easily by a network of spiking neurons. In our experiments, we found that significantly better results in driving a racing car along its track are obtained when color, depth and optical flow are provided together.

References

[1] B. Wymann, C. Dimitrakakis, A. Sumner, E. Espié, and C. Guinneau, "TORCS, The Open Racing Car Simulator," http://www.torcs.org, Mar. 2015.

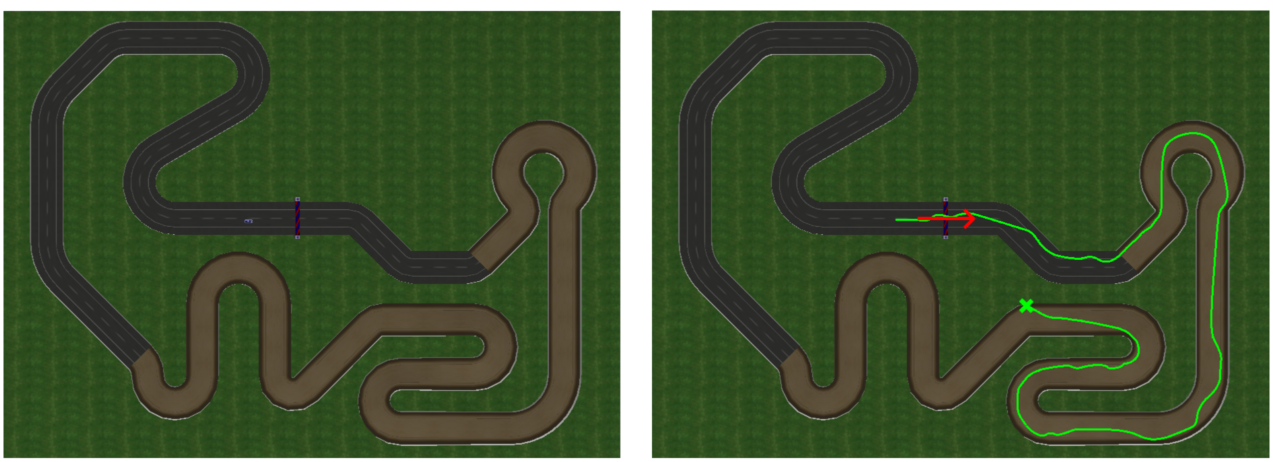


(a) (b) (c)

Figure 4. Extracting sub-regions from the visual input is helpful for visual control. (a) sample tree which evaluates information from the depth map. (b) input image (c) depth map with regions.

Table IV
TERMINAL SYMBOLS AND ELEMENTARY FUNCTIONS USED FOR OUR EXPERIMENTS.

Experiment	Terminal Symbols and Elementary Functions
A	`imageR`, `imageG`, `imageB`, `imageC`, `imageM`, `imageY`, `imageGray`, `extract{R,G,B,C,M,Y,Gray}`
B	`imageDepthMap`, `extractDepthMap`
C	`opticalFlowX`, `opticalFlowY`, `extractOpticalFlowX` `extractOpticalFlowY`
D	`imageR`, `imageG`, `imageB`, `imageC`, `imageM`, `imageY`, `imageGray`, `extract{R,G,B,C,M,Y,Gray}`, `imageDepthMap`, `extractDepthMap`, `opticalFlowX`, `opticalFlowY`, `extractOpticalFlowX` `extractOpticalFlowY`
All	`ERC`, `abs`, `round`, `floor`, `ceil`, `neg`, `sqrt`, `minLocX`, `minLocY`, `maxLocX`, `maxLocY`, `avg`, `min`, `max`, `q-quantile`, `add`, `mult`, `constImage`, `invert`, `gauss`, `median`, `binary`, `clamp`, `thresholdPass`, `thresholdZero`, `avgThreshold`, `localThreshold`



(a) (b)

Figure 5. Track used for our experiments. (a) track (b) path taken by an evolved individual (green line). Driving direction (red arrow).

[2] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. Cambridge, Massachusetts: The MIT Press, 1992.

[3] J. Hansen, "Visuelle Steuerung eines simulierten Rennfahrzeugs mit Hilfe von genetischer Programmierung," Master's thesis, Ernst Moritz Arndt Universität Greifswald, Faculty of Mathematics and Natural Sciences, Greifswald, Germany, Dec. 2014.

[4] G. Bradski, "The OpenCV library," *Dr. Dobb's Journal*, Nov. 2000.

[5] K. Pulli, A. Baksheev, K. Kornyakov, and V. Eruhimov, "Real-time computer vision with OpenCV," *Communications of the ACM*, vol. 55, no. 6, pp. 61–69, Jun. 2012.

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

[7] ——, *Genetic Programming II. Automatic Discovery of Reusable Programs*. Cambridge, Massachusetts: The MIT Press, 1994.

[8] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic Programming - An Introduction: On The Automatic Evolution of Computer Programs and Its Applications*. San Francisco, California: Morgan Kaufmann Publishers, 1998.

[9] C. Darwin, *The Origin of Species. Edited with an Introduction by Gillian Beer*. Oxford, England: Oxford University Press, 1996.

[10] D.-E. Nilsson and S. Pelger, "A pessimistic estimate of the time required for an eye to evolve," *Proc. R. Soc. Lond. B*, vol. 256, pp. 53–58, 1994.

[11] M. J. Tovée, *An introduction to the visual system*. Cambridge: Cambridge University Press, 1996.

[12] J. E. Dowling, *The retina: an approachable part of the brain*. Cambridge, Massachusetts: The Belknap Press of Harvard University Press,

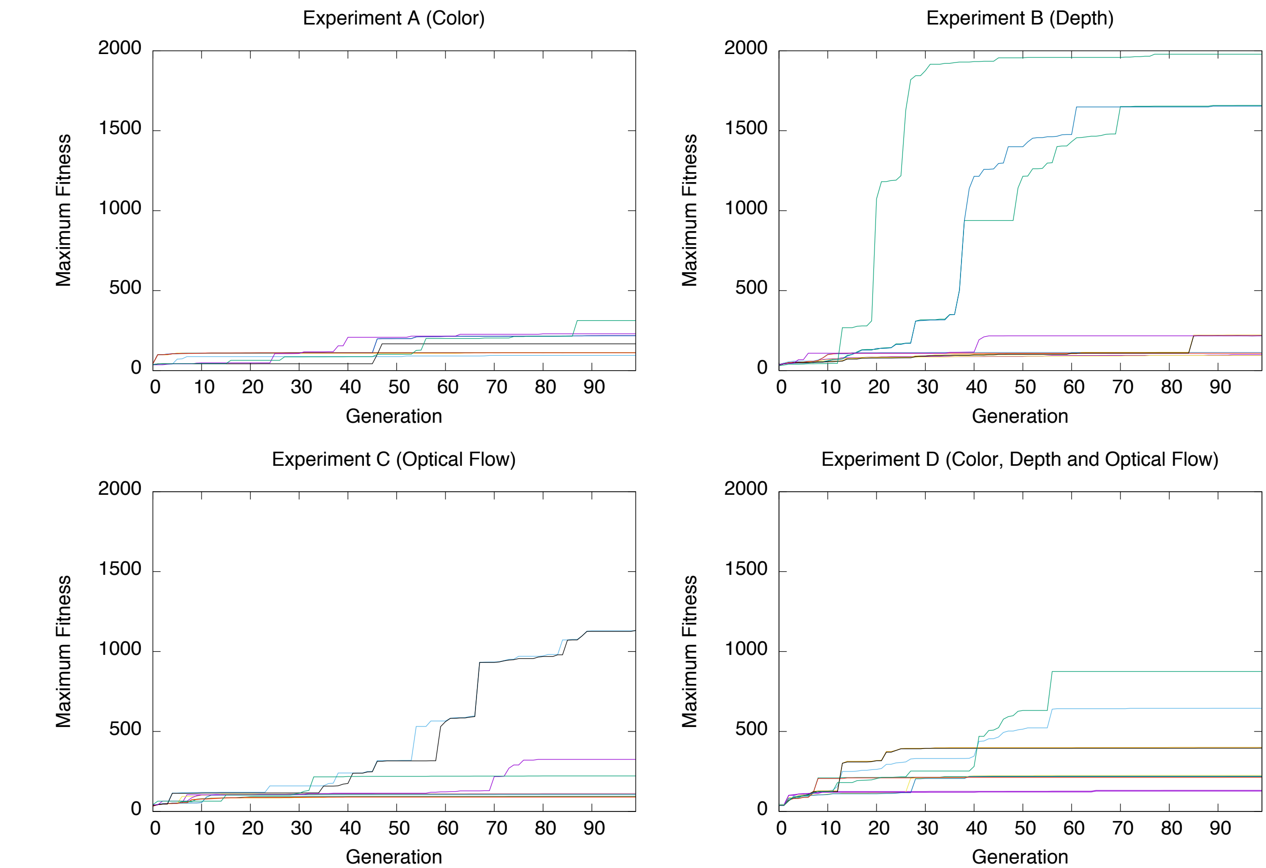


Figure 6. Best fitness values obtained for all four experiments. 10 runs were conducted for each experiment. Depth information seems to provide an evolutionary advantage. For some of the runs it produced exceptionally high fitness individuals. Optical flow also seems to provide an evolutionary advantage.

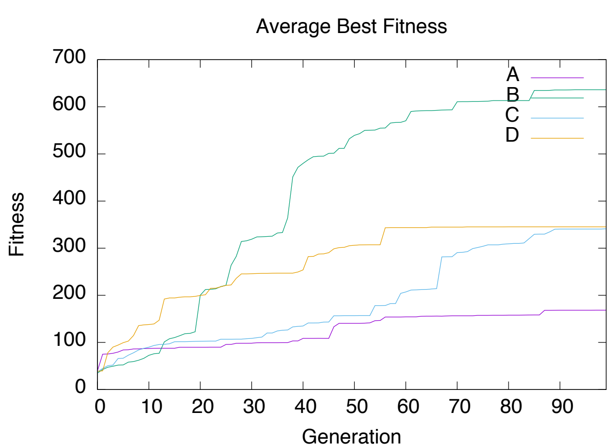


Figure 7. Average best fitness for all four experiments.

Table V
EXPERIMENTAL RESULTS FOR ALL FOUR EXPERIMENTS. OVERALL BEST FITNESS IF THE BEST FITNESS OBTAINED OVER ALL 10 RUNS FOR EACH EXPERIMENT. THE AVERAGE BEST FITNESS OBTAINED IN GENERATION 99 IS SHOWN IN THE LAST COLUMN.

experiment	overall best fitness	avg best fitness f
A	312.8	168.4
B	1978.6	636.2
C	1129.7	341.2
D	875.0	345.4

Table VI
COMPARISON OF AVERAGE BEST FITNESS IN GENERATION 99 USING THE MANN WHITNEY U TEST. SIGNIFICANT DIFFERENCES ARE SHOWN IN BOLD FACE.

hypothesis	p value
$H_0 : f_A = f_B, H_1 : f_A < f_B$	0.30
$H_0 : f_A = f_C, H_1 : f_A < f_C$	0.65
$H_0 : f_A = f_D, H_1 : f_A < f_D$	**0.02**

1987.

[13] M. S. Livingstone and D. H. Hubel, “Anatomy and physiology of a color system in the primate visual cortex,” *The Journal of Neuroscience*, vol. 4, no. 1, pp. 309–356, Jan. 1984.

[14] S. M. Zeki, “Review article: Functional specialisation in the visual cortex of the rhesus monkey,” *Nature*, vol. 274, pp. 423–428, Aug. 1978.

[15] S. Zeki, *Inner Vision. An Exploration of Art and the Brain*. Oxford: Oxford University Press, 1999.

[16] K. Moutoussis and S. Zeki, “A direct demonstration of perceptual asynchrony in vision,” *Proc. R. Soc. Lond. B*, vol. 264, pp. 393–399, 1997.

[17] D. Loiacono, L. Cardamone, and P. L. Lanzi, “Simulated car racing championship. competition software manual,” http://arxiv.org/abs/1304.1672, Apr. 2013.

[18] D. Loiacono, J. Togelius, P. L. Lanzi, L. Kinnaird-Heether, S. M. Lucas, M. Simmerson, D. Perez, R. G. Reynolds, and Y. S. (2008), “The wcci

2008 simulated car racing competition," in *Proceedings of the 2008 IEEE Symposium on Computational Intelligence and Games, December 15-18, Perth, Australia*. IEEE, 2008.

[19] R. S. Wright, Jr., N. Haemel, G. Sellers, and B. Lipchak, *OpenGL SuperBible. Comprehensive Tutorial and Reference*, 5th ed. Upper Saddle River, NJ: Addison-Wesley, 2011.

[20] B. K. P. Horn, *Robot Vision*. Cambridge, Massachusetts: The MIT Press, 1986.

[21] H. Bülthoff, J. Little, and T. Poggio, "A parallel algorithm for real-time computation of optical flow," *Nature*, vol. 337, no. 9, pp. 549–553, Feb. 1989.

[22] H. Wang, M. Brady, and I. Page, "A fast algorithm for computing optic flow and its implementation on a transputer array," in *Proceedings of the British Machine Vision Conference*, Zisserman, Ed. Oxford: British Machine Vision Association, 1990, pp. 175–180.

[23] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert, "High accuracy optical flow estimation based on a theory for warping," in *Proceedings of the 8th European Conference on Computer Vision, Part IV,Prague, Czech Republic, May, 2004*, T. Pajdla and J. Matas, Eds. Berlin: Springer-Verlag, 2004, pp. 25–36.

[24] I. Rechenberg, *Evolutionsstrategie '94*. Stuttgart: frommann-holzboog, 1994.

[25] S. Luke, *The ECJ Owner's Manual. A User Manual for the ECJ Evolutionary Computation Library*, 2015.

[26] J. F. Winkeler and B. S. Manjunath, "Genetic programming for object detection," in *Genetic Programming 1997, Proceedings of the Second Annual Conference, July 13-16, 1997, Stanford University*, J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, Eds. San Francisco, California: Morgan Kaufmann Publishers, 1997, pp. 330–335.

[27] M. P. Johnson, P. Maes, and T. Darrell, "Evolving visual routines," in *Artificial Life IV, Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, R. A. Brooks and P. Maes, Eds. Cambridge, Massachusetts: The MIT Press, 1994, pp. 198–209.

[28] M. Ebner and T. Tiede, "Evolving driving controllers using genetic programming," in *IEEE Symposium on Computational Intelligence & Games, September 7-10, Politecnico di Milano, Milano, Italy*. IEEE, 2009, pp. 279–286.

[29] J. Koutník, G. Cuccu, J. Schmidbuber, and F. Gomez, "Evolving large-scale neural networks for vision-based reinforcement learning," in *Proceedings of the Genetic and Evolutionary Computation Conference, July 6-10, 2013, Amsterdam, The Netherlands*. ACM, 2001.

[30] I. T. Tanev and K. Shimohara, "On human competitiveness of the evolved agent operating a scale model of a car," in *Proceedings of the IEEE Congress on Evolutionary Computation, Singapore, September 25-28*. IEEE, 2007, pp. 3646–3653.

[31] I. Tanev and K. Shimohara, "Evolution of agent, remotely operating a scale model of a car through a latent video feedback," *Journal of Intelligent Robotic Systems*, vol. 52, pp. 263–283, 2008.

[32] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents (extended abstract)," in *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015, pp. 4148–4152.

[33] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone, "A neuroevolution approach to general Atari game playing," *IEEE Transactions on Computational Intelligence and AI in Games*, pp. 355–366, Dec. 2014.

[34] M. Hausknecht, P. Khandelwal, R. Miikkulainen, and P. Stone, "Hyperneat-ggp: A hyperneat-based atari general game player," in *Proceedings of the Genetic and Evolutionary Computation Conference, Philadelphia, Pennsylvania, July 7–11*, 2012.

[35] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[36] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," in *NIPS Deep Learning Workshop*, 2013.

[37] X. Guo, S. Singh, H. Lee, R. Lewis, and X. Wang, "Deep learning for real-time atari game play using offline monte-carlo tree search planning," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 3338–3346.

[38] M. Parker and B. D. Bryant, "Visual control in quake ii with a cyclic controller," in *Proceedings of the 2008 IEEE Symposium on Computational Intelligence and Games*, P. Hingston and L. Barone, Eds. Piscataway, NJ: IEEE Press, 2008, pp. 151–158.

[39] ——, "Visual control in quake ii with a cyclic controller," in *Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games*. Piscataway, NJ: IEEE Press, 2009, pp. 287–293.

[40] D. J. Montana, "Strongly typed genetic programming," *Evolutionary Computation*, vol. 3, no. 2, pp. 199–230, 1995.

[41] M. V. Srinivasan, "How bees exploit optic flow: behavioural experiments and neural models," *Phil. Trans. R. Soc. Lond. B*, vol. 337, pp. 253–259, 1992.

[42] ——, "Distance perception in insects," *Current Directions in Psychological Science*, vol. 1, no. 1, pp. 22–26, Feb. 1992.

[43] J. Santos-Victor, G. Sandini, F. Curotto, and S. Garibaldi, "Divergent stereo for robot navigation: Learning from bees," in *Proceedings of Computer Vision and Pattern Recognition, New York*, 1993, pp. 434–439.

[44] M. Ebner and A. Zell, "Centering behavior with a mobile robot using monocular foveated vision," *Robotics and Autonomous Systems*, vol. 32, no. 4, pp. 207–218, 2000.

[45] S. Zeki, *A Vision of the Brain*. Oxford: Blackwell Science, 1993.

[46] P. A. Arndt, H. A. Mallot, and H. H. Bülthoff, "Human stereovision without localized image features," *Biological Cybernetics*, vol. 72, pp. 279–293, 1995.