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
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. Block based methods search for pixels within an area on partial derivatives. However, block based methods are also 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.

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

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

Let \( R \) be the inverse \( 3 \times 3 \) rotation matrix that describes the rotatory motion of the racing car from one time step of the simulation to the next. Let \( D \) 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 \((X'_S, Y'_S, Z'_S)\) will have moved to a location \((X''_S, Y''_S, Z''_S)\) relative to the eye of the driver.

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

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 partial 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
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 \( p_{\text{rep}} \), \( p_{\text{mut}} \), \( p_{\text{ERC-mut}} \), \( p_{\text{cross}} \) 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 \( p_{\text{rep}} \). Figure 3(b) shows the mutation operator which is applied with probability \( p_{\text{mut}} \). 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 \( p_{\text{cross}} \).

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 mutated. We originally intended to use Gaussian mutation (like an evolution strategy [24]) to slightly alter this ERC value, i.e. \( v := v + e^{0.01z} \) 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 \( v := v - 0.01z \) 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.
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 ($\text{imageR}$), green channel ($\text{imageG}$), blue channel ($\text{imageB}$), cyan channel ($\text{imageC}$), magenta channel ($\text{imageM}$), yellow channel ($\text{imageY}$), gray channel ($\text{imageGray}$). The depth map of this input image is available through the terminal symbol ($\text{depthMap}$). 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 $\text{opticalFlowX}$ and the y-component is made available through the terminal $\text{opticalFlowY}$. All image data is downsampled 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 $\text{extractNAME}$ 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 function 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 $\text{avg}$ function which computes the average depth within these two areas then we can compare both average depths to control the steering
Table I

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

Table II

**ELEMENTARY FUNCTIONS.** The return value of the node is 0.

<table>
<thead>
<tr>
<th>Name</th>
<th>Output Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs(float v)</td>
<td>float</td>
<td>absolute value, o =</td>
</tr>
<tr>
<td>round(float v)</td>
<td>float</td>
<td>round function, o = round(v)</td>
</tr>
<tr>
<td>floor(float v)</td>
<td>float</td>
<td>floor function, o =</td>
</tr>
<tr>
<td>ceil(float v)</td>
<td>float</td>
<td>ceil function, o =</td>
</tr>
<tr>
<td>neg(float v)</td>
<td>float</td>
<td>negate input, o = −v</td>
</tr>
<tr>
<td>sqrt(float v)</td>
<td>float</td>
<td>square root, o = √</td>
</tr>
<tr>
<td>minLocX(Image c)</td>
<td>float</td>
<td>x-coordinate (range [0, 1]) of minimum c(x,y)</td>
</tr>
<tr>
<td>minLocY(Image c)</td>
<td>float</td>
<td>y-coordinate (range [0, 1]) of minimum c(x,y)</td>
</tr>
<tr>
<td>maxLocX(Image c)</td>
<td>float</td>
<td>x-coordinate (range [0, 1]) of maximum c(x,y)</td>
</tr>
<tr>
<td>maxLocY(Image c)</td>
<td>float</td>
<td>y-coordinate (range [0, 1]) of maximum c(x,y)</td>
</tr>
<tr>
<td>avg(Image c)</td>
<td>float</td>
<td>average value of all pixels, o = ∑x,y c(x,y)</td>
</tr>
<tr>
<td>min(float a, float b)</td>
<td>float</td>
<td>minimum value, o = (a &lt; b)?b : a</td>
</tr>
<tr>
<td>max(float a, float b)</td>
<td>float</td>
<td>maximum value, o = (a &gt; b)?a : b</td>
</tr>
<tr>
<td>q-quantile(Image c, float q)</td>
<td>float</td>
<td>q-quantile of the image</td>
</tr>
<tr>
<td>add(float a, float b)</td>
<td>float</td>
<td>addition, o = a + b</td>
</tr>
<tr>
<td>mult(float a, float b)</td>
<td>float</td>
<td>multiplication, o = a · b</td>
</tr>
</tbody>
</table>

Table III

**ELEMENTARY FUNCTIONS.** The return value is o(x,y) for each pixel (x,y) of the output image.

<table>
<thead>
<tr>
<th>Name</th>
<th>Output Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs(Image c)</td>
<td>Image</td>
<td>absolute value o(x,y) =</td>
</tr>
<tr>
<td>sqrt(Image c)</td>
<td>Image</td>
<td>square root, o(x,y) = √</td>
</tr>
<tr>
<td>min(Image c)</td>
<td>Image</td>
<td>minimum value, o(x,y) = minx,y</td>
</tr>
<tr>
<td>max(Image c)</td>
<td>Image</td>
<td>maximum value, o(x,y) = maxx,y</td>
</tr>
<tr>
<td>add(Image a, Image b)</td>
<td>Image</td>
<td>addition, o(x,y) = a(x,y) + b(x,y)</td>
</tr>
<tr>
<td>constImage(float v)</td>
<td>Image</td>
<td>constant image, o(x,y) = v</td>
</tr>
<tr>
<td>invert(Image c)</td>
<td>Image</td>
<td>image, o(x,y) = max −</td>
</tr>
<tr>
<td>gauss(float v, Image c)</td>
<td>Image</td>
<td>gaussian filter with kernel $e^{-x^2/(2\sigma^2)}$ where $\sigma = 0.3(</td>
</tr>
<tr>
<td>median(float v, Image c)</td>
<td>Image</td>
<td>median filter with size</td>
</tr>
<tr>
<td>clamp(Image c, float v)</td>
<td>Image</td>
<td>binary threshold, o(x,y) = (c(x,y) &gt; v)?1 : 0</td>
</tr>
<tr>
<td>thresholdPass(Image c, float v)</td>
<td>Image</td>
<td>threshold o(x,y) = (c(x,y) &gt; v)?c(x,y) : 0</td>
</tr>
<tr>
<td>thresholdZero(Image c, float v)</td>
<td>Image</td>
<td>threshold o(x,y) = (c(x,y) &gt; v)?0 : c(x,y)</td>
</tr>
<tr>
<td>avgThreshold(Image c)</td>
<td>Image</td>
<td>average threshold, o(x,y) = (c(x,y) &gt; a)?1 : 0 with a = ∑x,y c(x,y).</td>
</tr>
<tr>
<td>localThreshold(Image c)</td>
<td>Image</td>
<td>local average threshold, o(x,y) = (c(x,y) &gt; a(x,y))?1 : 0</td>
</tr>
<tr>
<td>extractNAME(float s, float t)</td>
<td>Image</td>
<td>10 variants of this elementary function exist, with NAME ∈ {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.</td>
</tr>
</tbody>
</table>
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

\[
f_i = \begin{cases} 
-50 & \text{if controller is disqualified} \\
 d \cdot (1-a)^2 & \text{not disqualified,}
\end{cases}
\]

\[
\max\{2d \cdot (1-a)^2, 0\} & \text{not disqualified, } \\
\text{steering toward left and right}
\]

\[(5)\]

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 \( p_{c_{max}} = 0.2 \),
mutation was applied with probability \( p_{mut} = 0.4 \). ERC mutation
was applied with probability \( p_{ERC-mut} = 0.3 \) reproduction
was applied with probability \( p_{rep} = 0.1 \). Rammed half-and-half
initialization was used to initialize the individuals of the first
generation with depth ranging from 2 to 6.

\section*{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 exper-
iments. 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 max-
imum 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 informa-
tion 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.

\section*{IX. Conclusion}

We have used an evolutionary algorithm (Genetic Program-
ming) 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.

\begin{thebibliography}{10}
\bibitem{1} B. Wymann, C. Dimitrakakis, A. Sumner, E. Espié, and C. Guinneau,
2015.
\end{thebibliography}
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>
<thead>
<tr>
<th>Experiment</th>
<th>Terminal Symbols and Elementary Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>imageDepthMap, extractDepthMap</td>
</tr>
<tr>
<td>C</td>
<td>opticalFlowX, opticalFlowY, extractOpticalFlowX, extractOpticalFlowY</td>
</tr>
<tr>
<td>All</td>
<td>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</td>
</tr>
</tbody>
</table>

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


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

<table>
<thead>
<tr>
<th>experiment</th>
<th>overall best fitness</th>
<th>avg best fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>312.8</td>
<td>168.4</td>
</tr>
<tr>
<td>B</td>
<td>1978.6</td>
<td>636.2</td>
</tr>
<tr>
<td>C</td>
<td>1129.7</td>
<td>341.2</td>
</tr>
<tr>
<td>D</td>
<td>875.0</td>
<td>345.4</td>
</tr>
</tbody>
</table>

Table VI

<table>
<thead>
<tr>
<th>hypothesis</th>
<th>p value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$H_0 : f_A = f_B$, $H_1 : f_A &lt; f_B$</td>
<td>0.30</td>
</tr>
<tr>
<td>$H_0 : f_A = f_C$, $H_1 : f_A &lt; f_C$</td>
<td>0.65</td>
</tr>
<tr>
<td>$H_0 : f_A = f_D$, $H_1 : f_A &lt; f_D$</td>
<td>0.02</td>
</tr>
</tbody>
</table>

1878.


