Fast Developmental Stereo-Disparity Detectors

J. Arden Knoll*†, Van-Nam Hoang¶, Jacob Honer*‡, Samuel Church*‡, Thanh-Hai Tran¶‖, Juyang Weng*†§

*GENISAMA LLC, Okemos, MI 48864 USA
†Department of Computer Science and Engineering, ‡Department of Electrical and Computer Engineering,
§Cognitive Science Program and Neuroscience Program, Michigan State University, East Lansing, MI, 48824 USA
¶MICA International Research Institute, ‖School of Electronics and Telecommunications,
Hanoi University of Science and Technology, Hanoi, Vietnam

***Abstract*—Traditional methods for stereo-disparity detection use explicit search between the left and right images. Although such methods are simple and intuitive for understanding, they suffer from degeneracies when the search window contains weak texture. Developmental Networks (DNs) are task-nonspecific and modality-nonspecific learning engines. Because they are general-purpose learners, they have a potential to deal with many types of degeneracies in intelligent systems. This work presents two novel mechanisms to deal with degeneracies: volume dimension and subwindow voting. While developmental stereo-disparity detection has been tested on simulated stereo images in our prior publications, it has never been tested on the real world. This paper reports our system, 3DEye, which is the first to have filled this void. The algorithm, software, graphical user interface, training, performance, and update rates on CPU and GPU, respectively, on a Sony G8142 mobile phone are reported. Many deep learning methods that use error back-propagation suffer from the controversy of "post-selection" using the test set [1], to select one from many networks to report. In contrast, all randomly initialized DNs are performance-equivalent, no "post-selection" using test set. Possible future improvements for practical real-world and real-time applications are discussed.**

I. Introduction

Artificial Intelligence (AI) has reflected two schools that study natural intelligence of human minds — nativist and connectionist. Closely related to the *nativist* school in natural intelligence, we have seen the *symbolic* school in AI. The connectionist schools in AI and natural intelligence have shared the same term "connectionist".

A. Symbolic school

Symbols are used in many AI methods (e.g., states in HMMs, Graphical Models and SLAM), because they are useful for computers and intuitive to programmers. However, symbolic methods have limitations.

The symbolic school typically assumes a micro-world in 4D space-time in which an object, apple-1, is unique in space-time, represented by a series of symbols $\{\text{symbol-1}(t) \mid t_0 \leq t < t_1\}$. The correspondences among all these symbols of the same object across different times are known as "the frame problem" [2] in AI. In computer vision, the symbolic school assumes a single symbol "apple-1" for all its 3D $\mathbf{x}$-positions at time t in the 3D trajectory $\{\mathbf{x}(t) \mid t_0 \leq t \leq t_1\}$ and uses techniques, such as feature tracking through video. Therefore, the symbolic school is fact based, and, as the name "nativist" implies, the facts are native and inborn.

Stereo-specific rules, such as left and right matching using a sliding window, are typical in engineering versions of stereo systems [3], [4].

For symbolic stereo-disparity detection, at each image position, the agent extracts a patch $\mathbf{p}$ on the left image at row-column position (r, c) and searches along a horizontal line in the right image for the best match $\mathbf{p}'$ at (r, c'). The disparity is defined by the difference of columns $d = c' - c$.

Other motion-stereo methods explicitly compute dense disparity vectors using an array of handcrafted features [5], [6]. However, such systems are not real-time as computations along a dense pixel-grid take time when the number of grid points is large.

For many symbolic stereo-disparity detection systems, the match between left and right becomes degenerate when all the pixels in the patch have the same, or nearly the same, RGB values (i.e., weak texture). Therefore, multiple disparity values may indicate a strong match. Because of this, symbolic systems tend to have high brittleness and perform poorly in natural settings.

B. Connectionist school

In contrast, the connectionist school is egocentric — meaning that the agent must learn from its *world* without a handcrafted, *world-centered* object model. Although connectionist methods often assume some task-specific symbols, e.g., a static set of object labels, they typically do not assume a micro-world model. Therefore, a connectionist model typically needs to sense and learn using a network.

Networks, however, require a larger number of computations, typically higher than a corresponding symbolic system, as static concepts in a symbolic system must be learned by the network.

Genetic algorithms offer an approach to such learning. These algorithms study changes in genomes across different lives, but many genetic algorithms do not deal with lifetime development [7]. We argue that handcrafting functions of a genome for development seems to be a clean and tractable problem which avoids the extremely high cost of evolution on developmental algorithms.

Compared with methods of symbolic schools, a developmental method must learn through a lifetime, starting with a static genome called a *developmental algorithm*. Because a

developmental system does not assume a given-task, it tends to have a relatively higher adaptability to natural settings than methods of the symbolic school.

C. Developmental stereo-disparity detection

The neuroscience literature [8], [9], [10], [11], [12], has reported that neurons in V1, V2, V5/MT and V4 are tuned to different amounts of disparities. The shift of a pattern is called "phase". Our LCA (Lobe Component Analysis) theory indicates that "phase" is a resulting phenomenon caused by statistics-based competition in LCA. Such a "phase" phenomenon has already been reported by our earlier work [13] using a large number of locally shifted monocular natural images as inputs to simulate stereo effects.

Here, the stereo-disparity detection is a system that is trained from a general-purposed DN running on VCML-100 hardware for real-time development. A DN is task-nonspecific following the idea of AMD (Autonomous Mental Development) [7], as its output can be trained to generate actions other than disparity, such as navigation actions, naming a recognized object, responding to a recognized sound, or replying to sentences of natural languages [14].

For stereo-disparity detection, we consider output actions as binocular disparities. However, actions from a DN can be anything muscles can carry out, not only disparities.

Using horizontal shifts from monocular natural images to simulate binocular images, Solgi & Weng 2009 [13] reached a sub-pixel accuracy for developmental stereo-disparity detection. The work here is the first for testing developmental stereo-disparity detection for real-world images.

The major *novelty* of the work is to task-independently deal with weak texture, mainly using two mechanisms:

The first *new* mechanism is called *volume dimension*. This entails adding a new dimension to each neuron's input that is high when the other entries in the normalized input vector are low. Volume dimension avoids degeneracies when a nearly zero input vector (weak texture) of each neuron is normalized to a unit vector — resulting in noise explosions.

The second *new* mechanism is *subwindow voting*. Within a mask of 3×3 subwindows, each subwindow is analyzed by a column of many hidden neurons. The output disparity is the voting result from the $3 \times 3 = 9$ subwindows, weighted according to each subwindow's analyzed certainty.

The remainder of the paper is organized as follows. Sec. II describes the system architecture and the selection of the major parameters of the system for the stereo-disparity detector. The DN algorithm is given in Sec. III, and Sec. IV discusses how the system is trained and tested. Sec. V explains how the system works for stereo-disparity detection when a DN is accordingly trained. The experimental results are reported in Sec. VI and Sec. VII provides concluding remarks.

II. System Architecture

This system utilizes the VCML-100 from GENISAMA, which is equipped with a real-time stereo camera, a controller for training or testing the learner, and binocular goggles for viewing stereo videos or movies on a mobile phone screen.

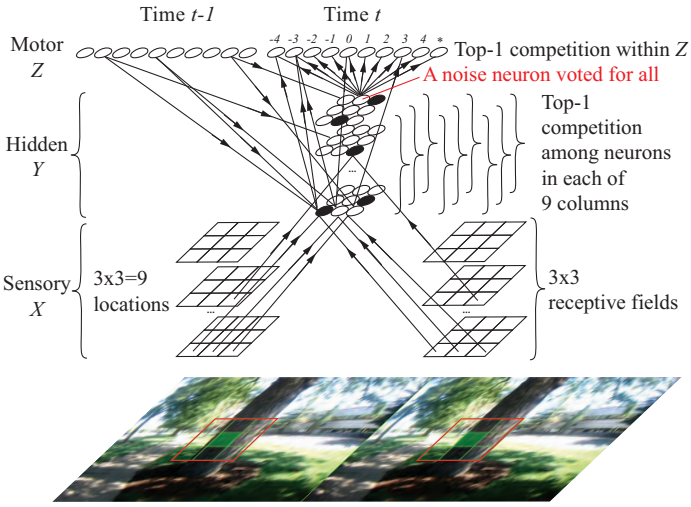

Fig. 1. The architecture of the 3DEye network. The $3 \times 3 = 9$ subwindows (green) correspond to the mask (red) in which 3×3 columns will be computed in the hidden area. The initial connections from the motor area at time $t-1$ to hidden area are complete. The same is true from the hidden area to the motor area at the current time t. The illustration only provides schematic connection patterns, not all initial complete connections.

A. Design considerations

The main design considerations for stereo-disparity detection using the VCML-100 include the following mutually conflicting constraints: (1) real-time speed (2) cost, and (3) mobility.

Because the actions from the DN software must be in real time (e.g., taking less than 30ms after a binocular image pair is grabbed), we need to limit the number of computations for each binocular image frame. This is especially important for a mobile phone, such as the Sony G8142 that was used primarily because of its high-resolution screen display suited for 3D stereo-image views.

Taking into account the above constraints, we chose the following DN parameters. (1) The input is grabbed from a small, 135×135 pixel, mask on both the left and right images (red square in Fig. 1). (2) The hidden neurons are grouped into nine columns (inhibition zones), each column sharing the same initial receptive fields. (3) The input image is divided into 3×3 non-overlapping subwindows (Fig. 2), of size 45×45 pixels, where each subwindow is the receptive field for the neurons in its respective column. (4) The number of hidden neurons is limited. Such a choice of DN architecture parameters affects performance, but a practical system must be real-time with limited computational power (i.e., a smart phone). To help achieve real-time speeds, we designed a more optimal version using the GPU and multiple CPU cores of the phone.

We would like to use more subwindows and columns. However, this would require more computations for each network update, which would make real-time response unreachable. The decision to limit the region for disparity detection to the mask, defined by the $3 \times 3 = 9$ subwindows, was based on human intuitive estimation and is not optimized.

Fig. 2 shows the graphical user interface (GUI) of the

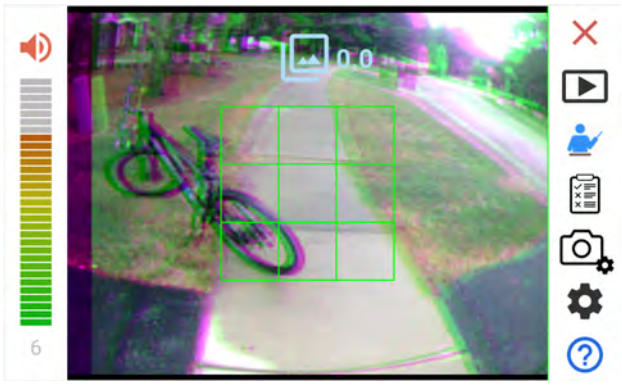

Fig. 2. The graphical user interface of 3DEye in the stereo disparity mode, displayed on the touch screen of a smart phone. Left RGB image: overlaid luminance as green; right RGB image: overlaid luminance as purple. The disparity value is heard through the earphone, and marked also visually by the left vertical bar.

3DEye software. The left image (green) and the right image (purple) overlap, allowing for intuitive labeling of disparity.

B. Binocular vergence

It is important that the absolute disparity value between the left and right receptive fields accounts for a small proportion of the subwindow size so that a given neuron's input from the left and right subwindows overlaps.

The two eyes of a human dynamically converge by a proper amount to make the 3D fixation point to have a nearly zero binocular disparity. This process is important to enable the brain to use its limited neuronal resources effectively, because dealing with very large binocular disparity will greatly divert neuronal resources.

In contrast, the binocular cameras of 3DEye are fixed and parallel. This means that all disparities are positive and only infinity has a zero disparity.

As infinity is not a common fixation point, we select a 3D fixation point that is near the center of the most likely fixation region, called a calibrated 3D fixation point. We approximate the effect of the converge motor using the following approximation method. Globally shift both images horizontally so that the calibrated 3D point has a zero disparity. The 3D points should have smaller absolute values of disparities than without such a global shift, because the disparities values now take both negative and positive values.

III. DN Algorithm

The DN, detailed in [15], takes Z as the state/action and X as input. It has a hidden area Y in which each hidden neuron has different receptive fields in X and in Z. We should consider Z as a state in a finite automaton (FA).

The receptive field from Z to each Y hidden neuron (effective receptive field) is globally connected so that each hidden neuron receives inputs from all Z neurons, as illustrated in Fig. 1.

Each hidden neuron in Y takes $\mathbf{x}(t) = (\mathbf{x}_l(t), \mathbf{x}_r(t))$ where $\mathbf{x}_l \in X_l$ and $\mathbf{x}_r \in X_r$ denote the left image and right image, respectively, and X_l and X_r are the space of left images and the space of right images, respectively. Although not explicitly stated, each hidden neuron only takes a small patch of $\mathbf{x}(t) = (\mathbf{x}_l(t), \mathbf{x}_r(t))$.

The DN algorithm is presented below:

Algorithm 1 (DN Algorithm): Areas: X: sensory; Y: hidden (internal); Z: motor

Input areas: X and Z.

Output area: Z.

The dimension and representation of the X and Z areas are based on the sensors and effectors of the agent. Y is skull-closed, meaning it is not directly accessible from the outside.

1) For the Y area, initialize the adaptive part $N_y = (V, G)$ and response vector $\mathbf{y}$, where V is the synaptic weights and G is the neuronal ages. All Y neurons have been initialized with random weights and zero firing ages. The Z area initializes its adaptive part N_z and the response vector $\mathbf{z}$ in a similar way.
2) At time $t = 0$, supervise initial state $\mathbf{z}$. Grab the first sensory input $\mathbf{x}$.
3) At time $t = 1, 2, 3, ...$, repeat the following steps forever (executing steps 3a, 3b in parallel, before doing step 3c):
   a) All Y neurons compute in parallel:
   $$(\mathbf{y}', N_y') = f_y(\mathbf{z}, \mathbf{x}, N_y) \quad (1)$$
   where f_y is the Y area function to be explained below, which computes the response vector $\mathbf{y}'$ and updates the adaptive part N_y' of the Y area. The area Y performs neuron splitting (mitosis) if the best matched Y neurons do not match the input vector y sufficiently well.
   b) Components in $\mathbf{z}'$ are supervised if they are never fired. Otherwise, Z neurons use the following expression to compute the Z area's response vector $\mathbf{z}'$ and the adaptive part N_z' in parallel:
   $$(\mathbf{z}', N_z') = f_z(\mathbf{y}, N_z) \quad (2)$$
   where f_z is the Z area function to be explained below.
   c) Then, replace for asynchronous update: $\mathbf{y} \leftarrow \mathbf{y}'$, $\mathbf{z} \leftarrow \mathbf{z}'$, $N_y \leftarrow N_y'$ and $N_z \leftarrow N_z'$. Supervise input $\mathbf{x}$.

The DN is in asynchronous mode and must update at least twice before the effects of each new input patterns in X and Z, respectively, go through one update in Y and then one update in Z to appear in X (if DN predicts also X) and Z.

A. Area function details of the DN algorithm

We use Lobe Component Analysis (LCA) [16] as the neuron's learning algorithm. The area function f_y in Eq.(1) and area function f_z in Eq.(2) are computed by the following procedures. These procedures have complex mechanisms which contribute to computation of response vectors $\mathbf{y}'$ and $\mathbf{z}'$ and the maintenance of adaptive parts N_y' and N_z' for the Y area and the Z area, respectively. The procedures of the area function are described below.

1) Initialization: The random Grounded DN (GDN) initialization method [17] is used for our implementation. Whenever the network takes an input, add a volume dimension and normalize per the specifications in [15] and compute the pre-responses in Y. If the top-1 winner in Y has a pre-response much lower than a perfect match, simulate a mitosis-equivalent process by adding a new neuron to learn the new inputs from X and Z. In this way, Y neurons fully store patterns from beginning cases in training.

2) Pre-response computation: Each Y neuron i computes its bottom-up pre-response and top-down pre-response, respectively, by doing an inner product of normalized input and its stored pattern. (We denote $\dot{\mathbf{v}}$ as the vector of $\mathbf{v}$ with a unit Euclidean norm: $\dot{\mathbf{v}} = \mathbf{v}/\|\mathbf{v}\|$.) The bottom-up pre-response $r_{b,i}$ of each neuron i is calculated as follows:

$$r_{b,i} = \dot{\mathbf{v}}_{b,i} \cdot \dot{\mathbf{x}}_i \tag{3}$$

where $\dot{\mathbf{x}}_i$ is the sensory input vector from X area and $\dot{\mathbf{v}}_{b,i}$ is the bottom-up weight of that neuron. This computes the similarity between each neuron's stored weight pattern and the input vector. The top-down response $r_{t,i}$ for each neuron i is computed in the same way.

After each Y neuron i computes its bottom-up pre-response and top-down pre-response, the neuron sets its pre-response value to be the sum of the two values:

$$r'_i = w_b r_{b,i} + w_t r_{t,i} \tag{4}$$

where w_b and w_t are the weights for bottom-up and top-down, respectively. We select w_t, $0 \leq w_t \leq 1$, and then w_b is determined by $w_b = 1 - w_t$.

The pre-response computation of Z neurons is similar, but each Z neuron only has bottom-up input from Y. The Y area always has $3 \times 3 = 9$ winners as voters, one from each of the 3×3 columns.

3) Top-k competition: The neurons within each of the 3×3 columns compete to fire. We use top-k competition as a simulation of dynamic inhibition among the neurons. The k neurons with the highest pre-response value fire while other neurons are suppressed. We adjust the pre-response value of each winner neuron i based on their ranking, and obtain final response value r_i:

$$r_i = \begin{cases} (r'_i - r'_{k+1})/(r'_1 - r'_{k+1}) & r'_{k+1} \leq r'_i \leq r'_1 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

where r'_1 is the highest response value; r'_{k+1} is the $k+1$-th high response value. The adjusted responses are used to update corresponding firing neurons' weights.

4) Hebbian learning: Hebbian learning is known to exist in biological brains. Adaptations occur when a neuron fires. The input that triggers firing in that neuron would be recorded as an incremental average to the neuron's weight vector.

The firing Y neuron i would update its bottom-up weight and top-down weight respectively, using the Hebbian learning rule. The update formula of Y neuron's bottom-up weight is as follows:

$$\mathbf{v}_{b,i} \leftarrow \beta_1 \mathbf{v}_{b,i} + \beta_2 r_i \dot{\mathbf{x}}_i \tag{6}$$

where β_1 is the retention rate and β_2 is the learning rate of the neuron i.

$$\beta_1 = \frac{m_i - 1 - \mu(m_i)}{m_i}, \beta_2 = \frac{1 + \mu(m_i)}{m_i} \tag{7}$$

with $\beta_1 + \beta_2 \equiv 1$, where m_i is the neuron's firing age.

The Y neuron's top-down weight is updated similarly. The firing Z neurons also update their bottom-up weights in the same way.

The amnesic parameter μ, modeling effects of genes, is a monotonically increasing function of the neuron's age (m_i) that prevents the learning rate β_2 from being zero so that no neuron stops learning (e.g., let $t_1 = 20, t_2 = 200$, $c = 2$, $\tau = 2000$):

$$\mu(m_i) = \begin{cases} 0, & \text{if } m_i < t_1 \\ c(m_i - t_1)/(t_2 - t_1), & \text{if } t_1 \leq m_i \leq t_2 \\ c + (m_i - t_2)/\tau, & m_i > t_2. \end{cases} \tag{8}$$

IV. Training and Testing

Although we were able to train and test a DN in real-time using the 3DEye software, we elected to collect the data, in real-time, to then train and test the DN in batch, varying parameters to determine optimal performance.

A. Real-time data collection and labeling

The 3DEye software allows for continuous data collection and labeling of the disparity of binocular images. For our experiments, the 3D fixation point was calibrated at two meters. The images and their corresponding disparity labels were collected at 5Hz. Simultaneously, the trainer used the VCML-100 controller to align the left and right images so that the nearest point of the nearest object comes into focus within one of the subwindows on the GUI. The amount the images were shifted when an image was grabbed gave the disparity label for those images. This is done with the application of collision avoidance in mind.

Following data collection, we played back the images, shifted with their labeled disparity, to validate the data and make changes where required.

B. Training and testing in batch

The data collected as described above was split into two groups. The odd frames were used for training while the even frames where used for testing. Because it takes two frames for the input $\mathbf{x}(t-1)$ to reach $\mathbf{z}(t+1)$, the training and testing process must take into account this 2-frame delay. For testing, this means the motor response at $\mathbf{z}(t)$ must be compared to the label for $\mathbf{x}(t-2)$ to test the DN's prediction accuracy. Training and testing with batch data allowed us to test multiple DNs with different parameters.

Note when any neuron happens to win for the first time (age 1), the learning rate is 1 and retention rate is zero so that its random weights will only affect which neuron wins but not the resulting operational part of the DN — every DN is equivalent. Because of this, we only ever have to test one DN for each set of parameters.

V. Theoretical Analysis

A. Hidden area of the DN

Suppose there are 100 hidden neurons for each of the $3\times3 = 9$ initial receptive fields. Because Hebbian learning will make some weights become zero or nearly zero, each neuron will change the location and shape of its receptive field. There are a total of $3 \times 3 \times 100$ receptive fields in each image, each corresponding to their respective $3 \times 3 = 9$ subwindows. The network computes the top-1 winner neuron among the 100 neurons in each column. This results in nine winners whose input is used for the Z area competition.

Suppose neuron n_{ij} is dedicated to a subwindow s_i, $i = 1, 2, ..., 9$, $j = 1, 2, ..., 100$. Let the bottom-up weight vector of neuron n_{ij} be denoted by $\mathbf{w}_{ij}$. Then, $\mathbf{w}_{ij}$ is an LCA vector [16] learned from all binocular sample sub-images s_{ik}, $k = 1, 2, ...n$ when the neuron fires, where $n \gg 100$ is a large but finite number indicating the number of lifetime training images. In other words, from n image pairs, the DN incrementally developed $3 \times 3 \times 100$ feature vectors, $\mathbf{w}_{ij}$, $i = 1, 2, ..., 9$, and $j = 1, 2, ..., 100$.

If $k = 1$ for top-k competition within each neuronal column, the LCA vectors are not necessarily well distributed in the sample space, especially if the same images arrive in a biased way, e.g., early images do not show in later ages. After all, an early experience that is not reviewed later in life tends to be forgotten. It is desirable for $k > 1$ for each sub-image s_i, $i = 1, 2, ..., 9$, however, review of early experiences is still important if early experiences need to be recalled in later ages.

B. Explanation of DN weights

We divide this material into three subsections, bottom-up weights of hidden neurons, bottom-up weights for motor neurons, and top-down weights for hidden neurons, all illustrated in Fig. 1.

1) Bottom-up weights for hidden neurons: As discussed above, the bottom-up weight for a neuron n_{ij} is an LCA feature from subwindow s_i that neuron n_{ij} happens to learn through competition. If the bottom-up receptive field of neuron n_{ij} has a large overlap between the left subwindow and right subwindow, the Hebbian learning expression in (6) is able to trim the left and right receptive fields so that the resulting binocular weight vector $(\mathbf{w}_l, \mathbf{w}_r)$ corresponds to a pattern in which $\mathbf{r}_l$ is similar to $\mathbf{r}_r$ except that their locations are different. Let us see an example: $\mathbf{w}_l = (*, 1.01, 2.02, *, *)$ and $\mathbf{w}_r = (*, *, 1.02, 2.01, *)$, where $*$ denotes values that do not match. This location difference between $\mathbf{w}_l$ and $\mathbf{w}_r$ is called *binocular disparity tuned by this neuron*. The similar patterns $(1.01, 2.02)$ in $\mathbf{w}_l$ and $(1.02, 2.01)$ in $\mathbf{w}_r$ are called *binocular patterns tuned by this neuron*. Therefore, the binocular weight vector $(\mathbf{w}_l, \mathbf{w}_r)$ contains both disparity information and pattern information.

Thus, with [17] we have proven the following theorem:

Theorem 1 ("Glass box" disparity): The disparity of a binocular pattern in each hidden neuron is not independent of the binocular pattern, but rather is embedded inside it. If all the hidden neurons link to the motor using Hebbian learning, the motor reports disparities if it is supervised so.

Hebbian learning automatically decides the scope and scale of the pair of binocular receptive fields based on the presence of such wide variety of differences. No stereo-specific rules are necessary in this biologically inspired DN.

2) Bottom-up weights for motor neurons: According to the DN theorem [17], each weight of a motor neuron is the probability that the connected hidden neuron fired, given the motor neuron fired. This result is for a single hidden area where only one neuron fires at each time. Here, we have $3 \times 3 = 9$ neuronal columns, each of which has one best-matched neuron firing.

Among these $3\times3 = 9$ columns, some columns are looking at weak texture and others are not, but the network does not know which are and which are not. This is a typical puzzle for us to solve in order to understand the emergent behavior of the DN.

Under each image input, consider two columns: A column that has a weak-textured input, e.g. a uniform gray area, which we call a noise column, and another column that has a strong-textured input, which we call a reliable column.

The noise column has a winner neuron whose weights happened to match the uniform gray area best. Whenever this "noise" neuron fires, the firing motor neuron is independent of this "noise" neuron. In other words, this "noise" neuron links to many motor neurons. Because the neurons in the motor area (Z) compete to win according to their pre-action potential, summed from the $3 \times 3 = 9$ columns, the "noise" Y neuron does not contribute selectively to the competition in Z.

Contrarily, a "reliable" neuron is very different from a "noise" neuron. Because its input texture is strong, whenever it fires, it contributes reliably only to the single neuron in the motor area that is supervised to fire.

In summary, a "reliable" neuron contributes to only a few specific motor neurons but a "noise" neuron widely contributes to many motor neurons. Because motor neurons compete to win, a "reliable" neuron plays a greater role in deciding motor outputs than a "noise" neuron.

3) Top-down weights for hidden neurons: In Section V-B1, we discussed only the bottom-up weights for hidden neurons. In fact, each hidden neuron has two parts of input, bottom-up and top-down. As proven in [17], the top-down part corresponds to the state match in a finite automaton (FA), and the bottom-up part corresponds to the input match in the FA. In order for each hidden neuron to successfully compete, it must match well for both parts, as a transition in an FA must match both the state and input in the FA transition table.

In a DN, the state context is the last motor pattern at the previous time. In other words, the DN is a spatiotemporal learning machine. It considers the context of all past history condensed into the disparity value in the motor area.

VI. Experiments and Results

We used a sequence of 1400 stereo images, recorded in real-time from the natural world, mostly outdoor like the one in

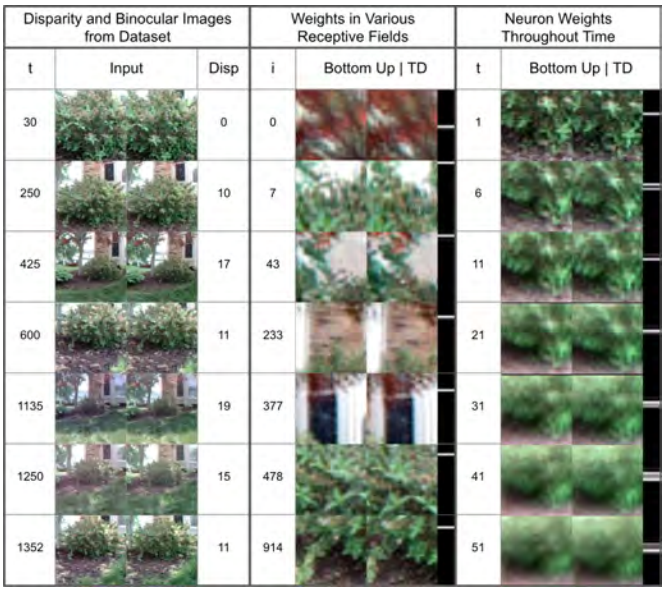

Fig. 3. Visualization of some examples: Left: Input images; Center: bottom-up and top-down (TD) weights of hidden neurons; Right: Weights updated through time t of a single neuron. Hidden neurons are indexed by i.

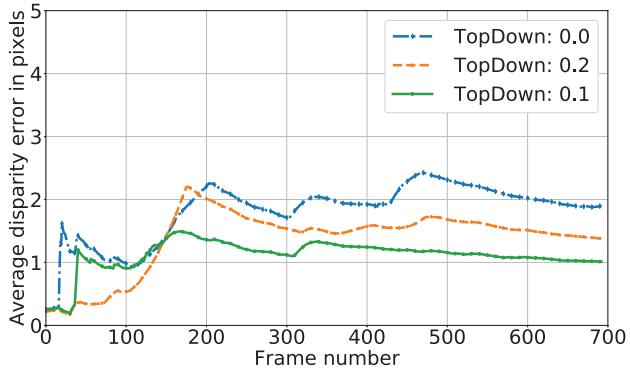

Fig. 4. Average disparity error over every testing time t from $t = 0$ by a DN that has learned 700 stereo frames. TopDown: (i.e., w_t in Eq. (4)) the weight of top-down match contributed to the pre-action potential of each hidden neuron. The total number of hidden neurons: 900, 100 per column.

Fig. 1. Fig. 3 gives the visualization. The DN was trained on frames from odd indices and tested on even frames.

The incrementally computed average errors from time zero of all disjoint tests are shown in Fig. 4 with the even indices renumbered as 0, 1, 2 ... for convenience.

When the top-down weight was 0.1, the average error was the smallest, reaching 1.0 pixels after 700 frames had been learned. For this experiment, every DN had 100 neurons per column (900 total), top-8 competition in each of the $3 \times 3 = 9$ columns, and a global top-3 competition in the motor area.

This table reports the average speed data, in network update rates, while the same DN ran on a Sony G8142 mobile phone.

Version	Training rate	Frozen-testing rate
CPU	0.96 Hz	1.00 Hz
GPU	7.16 Hz	10.47 Hz

In real-world stereo situations, a pair of the matching left and right image patches exhibits more complex distortions beyond a simple shift. Future versions of the 3DEye will use more neurons, optimized subwindow dimensions to more finely learn such differences.

VII. Conclusions

This paper presented the first developmental stereo-disparity detection system for the real world (implemented through 3DEye). The average error in detected disparities reached around 1.0 pixel in our experiments. In general, the more hidden neurons the DN has, the less the average error, since DNs do not have the problem of post-selection [1] mentioned in the abstract. We will study larger memories in the future.

The DN, as "glass box", seems to be suited for integrating the rich information redundancy available in color stereo images. Compared with traditional window-search methods in disparity detection, the developmental systems here for real-world and real-time stereo-disparity detection seem to present a superior robustness for driverless cars and blind visual aids, competing with laser scanners in robustness and cost.

References

[1] J. Weng. Life is science (36): Did Turing Awards go to fraud? Facebook blog, March 8 2020. www.facebook.com/juyang.weng/posts/10158319020739783.

[2] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Upper Saddle River, New Jersey, 3rd edition, 2010.

[3] W. E. L. Grimson and D. Marr. A computer implementation of a theory of human stereo vision. In L. S. Baumann, editor, *Proc. ARPA Image Understanding Workshop*, pages 41–45, 1979.

[4] U. R. Dhond and J. K. Aggarwal. Structure from stereo: A review. *IEEE Trans. Systems, Man and Cybernetics*, 19(6):1489–1510, Nov. - Dec. 1989.

[5] D. J. Fleet, A. D. Jepson, and M. R. M. Jenkin. Phase-based disparity measurement. *CVGIP: Image Understanding*, 53(2):198–210, 1991.

[6] J. Weng. Image matching using the windowed Fourier phase. *International Journal of Computer Vision*, 11(3):211–236, 1993.

[7] J. Weng, J. McClelland, A. Pentland, O. Sporns, I. Stockman, M. Sur, and E. Thelen. Autonomous mental development by robots and animals. *Science*, 291(5504):599–600, 2001.

[8] A. Anzai, I. Ohzawa, and R. D. Freeman. Neural mechanisms underlying binocular fusion and stereopsis: position vs. phase. *Journal of Cognitive Neuroscience*, 94:5438–5443, 1997.

[9] F. Gonzalez and R. Perez. Neural mechanisms underlying stereoscopic vision. *Progress in Neurobiology*, 55(3):191–224, 1998.

[10] I. Ohzawa, G. C. DeAngelis, and R. D. Freeman. Stereoscopic depth discrimination in the visual cortex: Neurons ideally suited as disparity detectors. *Science*, 249:1037–1041, 1990.

[11] C. W. Tyler. Representation of stereoscopic structure in human and monkey cortex. *Trends in Neurosciences*, 27(3):116–118, 2004.

[12] A. J. Parker. Binocular depth perception and the cerebral cortex. *Nature Reviews Neuroscience*, pages 379–391, 2007.

[13] M. Solgi and J. Weng. Developmental stereo: Emergence of disparity preference in models of visual cortex. *IEEE Trans. Autonomous Mental Development*, 1(4):238–252, 2009.

[14] J. Weng, Zejia Zheng, Xiang Wu, and Juan Castro-Garcia. Auto-programming for general purposes: Theory and experiments. In *Proc. International Joint Conference on Neural Networks*, pages 1–8, Glasgow, UK, July 19-24 2020.

[15] J. Weng, Z. Zheng, and X. Wu. Developmental network two, its optimality, and emergent turing machines. U.S. Provisional Patent Application Serial Number: 62/624,898, Feb. 1 2018. Published.

[16] J. Weng and M. Luciw. Dually optimal neuronal layers: Lobe component analysis. *IEEE Trans. Autonomous Mental Development*, 1(1):68–85, 2009.

[17] J. Weng. Brain as an emergent finite automaton: A theory and three theorems. *International Journal of Intelligent Science*, 5(2):112–131, 2015.