# Anatomy and Physiology of an Artificial Vision Matrix

Andrew Vardy and Franz Oppacher

School of Computer Science
Carleton University
Ottawa, K1S 5B6, Canada
Fax: +1 (613) 520-4334
avardy@scs.carleton.ca
http://www.scs.carleton.ca/~avardy

**Abstract.** We present a detailed account of the processing that occurs within a biologically-inspired model for visual homing. The *Corner Gradient Snapshot Model (CGSM)* initially presented in [1] was inspired by the *snapshot model* [2] which provided an algorithmic explanation for the ability of honeybees to return to a place of interest after being displaced. The concept of *cellular vision* is introduced as a constraint on processing. A *cellular vision matrix* processes visual information using retinotopically arranged layers of low-level processing elements interacting locally. This style of processing reflects general principles known of visual processing throughout the animal kingdom. From a technical standpoint, this style of processing is inherently parallel. Here we describe a cellular vision matrix which implements CGSM and illustrate how this matrix obeys cellular vision. Some new comparative results are presented and it is found that CGSM's performance degrades gracefully with environmental modification and occlusion.

## 1 Introduction

In [1] we presented a biologically-inspired model for visual homing. This model was inspired by the *snapshot model*, proposed to explain the ability of honeybees to return to a place of interest after displacement [2]. Our model, called the *Corner Gradient Snapshot Model (CGSM)*, was found to achieve successful homing on a dataset of real-world panoramic images. It outperformed a similar biologically-inspired model [3,4] on the same images. CGSM has two distinguishing features. First, it operates on real-world two-dimensional images, in contrast to most other models inspired by the snapshot model which operate on one-dimensional images, often taken of simulated or simplified worlds [5,6,7,8,9,10, 11,3]. Secondly, all of CGSM's processing is constrained to involve only local low-level interactions between neuron-like elements. We call this constraint *cellular vision*. Here our main purpose is to show how CGSM adheres to cellular vision by providing detailed wiring diagrams. We also present some new results which indicate that CGSM's performance will degrade gracefully when the environment is significantly modified.

### 1.1 Visual Homing

A critical competence for an agent, whether animal or robot, is the ability to repeatedly return to important places in the environment such as the nest or food sources. If this is achieved by exploiting visual cues then we call it *visual homing.* The snapshot model (described below) is a possible model for visual homing in the honeybee. With the snapshot model, a single "snapshot" image taken from the goal location is all that is stored to represent the environment. A number of alternative approaches have been proposed for visual homing, particularly in robotics. Basri et. al employ formal computer vision techniques to determine the epipolar geometry relating the snapshot and current locations and move directly home [12]. Memory-based approaches represent the environment by a set of images and associated home vectors [13,14,15]. Some memory-based approaches employ panoramic imaging systems as we do here (e.g. [14,15]). If the agent never has to leave the vicinity of the goal then the feature tracking approach of [16] can be employed for homing (this approach also uses corners as we do here). However, none of these more technical approaches consider the biologically-plausibility of their algorithms. Thus, we are inspired here by the snapshot model and we have attempted to provide an implementation of it which is both biologically-plausible and efficient.

### 1.2 The Snapshot Model

The snapshot model was developed to match data of honeybee search patterns [2]. A model agent is placed at the goal and allowed to capture a snapshot image. It is then displaced and allowed a return attempt. The disparity between the current and snapshot images is used to guide the return. One key requirement of the snapshot model is that the agent maintains a constant orientation. There is evidence to suggest that bees take on the same orientation when returning to a remembered place as they took when originally learning the layout of that place [17,18,19]. A robot homing via the snapshot model must employ some sort of compass system to maintain or compensate for changes in orientation.

### 1.3 Cellular Vision

The neurophysiology of visual homing in insects has not yet been probed [11]. How then can we posit an insect-inspired neural architecture to implement visual homing? The answer is to step back and look at the overall principles that seem to govern the processing of visual information in insects and other animals.

The lower layers in human visual cortex are composed of neurons exhibiting differently structured *receptive fields* [20]. In their pioneering work on cat visual cortex, Hubel and Wiesel state, *"the receptive field of a cell in the visual system may be defined as the region of retina (or visual field) over which one can influence the firing of that cell."* [21]. Further, visual processing neurons generally appear to be organized into *columns* with the retina on the bottom and increasingly elaborate, yet overlapping, receptive fields found as one ascends. This arrangement is *retinotopic*: *"each level of the system is organized like a map of the retina."* [20]. These observations also hold for insects. In describing motion processing within the framework of the fly's visual system Egelhaaf *et al* state *"the visual system is organized in a retinotopic way by columnar elements"* [22]. They describe how local image motion is detected by small-field neurons which subsequently connect to wide-field neurons. The small-field cells, called Elementary Motion Detectors (EMDs), compute local motion through lateral interactions between neighbouring cells on the same level within adjacent columns. To summarize, both vertebrates and invertebrates have visual systems that are structured according to the following general principles: Vertical arrangement of retinotopic columns; Horizontal arrangement into layers of heterogeneous function; Local processing of neurons that interact with other neurons in nearby columns and layers.

We refer to the use of these principles in artificial vision as *cellular vision*, and a matrix of processing elements adhering to these principles as a *Cellular Vision Matrix* (CVM). The CVM explored here was designed to produce movement vectors from images. Thus, at some point the dimensionality of the processing matrix must be brought down to the two dimensions of a vector describing motion in the plane. This is done in the final two layers of the CVM which exhibit very wide receptive fields. This aspect is also incorporated into the cellular vision concept and is inspired by the increasingly wide receptive fields found as one ascends from the retina into the visual processing areas of many animals.

A number of other researchers who have built computational and/or robotic models inspired by insects seem to be implicitly adhering to cellular vision. In [23] a robot inspired by the fly motion detection system has an array of hardware-implemented EMD units arranged retinotopically in a ring around the robot. Similar fly-inspired examples can be found in [24,25]. [26] presents a retinotopic neural model for the escape reflex of locusts. Additional examples of retinotopic neural models of insect vision include two others based also on the snapshot model [10,11]. Of these, the *neural snapshot model* [10] is of primary interest here because it provides the inspiration for CGSM. The primary difference being that the neural snapshot model operates only in a simplified simulation environment on one-dimensional binary images.

Two notes before concluding this section. First, adopting cellular vision rules out a host of more traditional computational methods. Chief among these is search. Interestingly, most of the existing implementations of the snapshot model utilize search and are therefore not readily implementable in a CVM (exceptions in [10,11]). Second, a CVM is inherently parallel. Here and in [1] the CVM for CGSM is run on a serial computer. However, parallel implementation must be imagined to allow a fair comparison with non-CVM methods. If simulating a CVM with $m$ layers at resolution $n \times n$ on a serial computer, the time complexity to complete processing of one image would be $O(mn^2)$. However, if the implementation is parallel the time complexity is just $O(m)$.

## 2 The CVM for CGSM

The CVM which implements the CGSM homing method consists of 29 layers. Each layer is composed of a grid of elements equal in size to the input image, all sharing the same function. A group of adjacent layers that together performs some higher-level function is called a *segment*. We begin by describing the processing matrix at a high-level and then descend to describe how segments and groups of segments achieve the necessary high-level functions in low-level terms.

A flow chart of the CGSM processing matrix is depicted in figure 1. An input image is fed into the processing matrix. It is first smoothed and then corners are extracted as features. If the agent is at the goal then the image of features is stored as the snapshot image. Gradients are formed around each feature and these gradients are locally compared with features in the snapshot image to generate vectors which indicate the direction that these features have moved in. These vectors specifying motion in the image are mapped onto vectors specifying the corresponding motion of the agent. Finally, this last set of vectors is summed to create the agent's home vector.

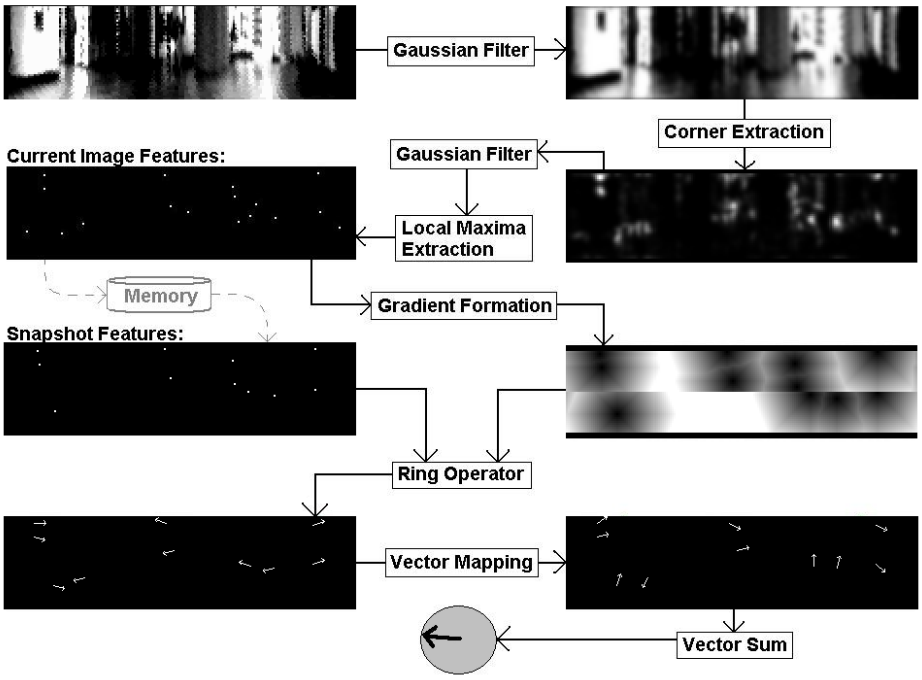


**Fig. 1.** Processing applied by the CGSM matrix. Adapted from [1].

Before describing the CVM for CGSM in depth we look at it from a more removed perspective. Figure 2 shows the entire CVM, visualized in 3-D with OpenGL. A single column of the CVM is shown. Note how many layers are displaced diagonally. These displacements have been added to make clearly visible the connections between layers. Processing flows from the bottom right up to the upper left. All subsequent figures of the CVM will follow this convention. These figures are all zoomed-in views of particular parts of figure 2 with additional labels and effects added. Note the circled summation sign at the upper left of the matrix. This indicates that the next-to-last and last layers are to be summed to form the $x$ and $y$ components (respectively) of the final agent motion vector that is output from the CVM. The next section provides detail on how the low-level structure of a CVM is to be described. Subsequent sections present each high-level component of the processing matrix in low-level terms.

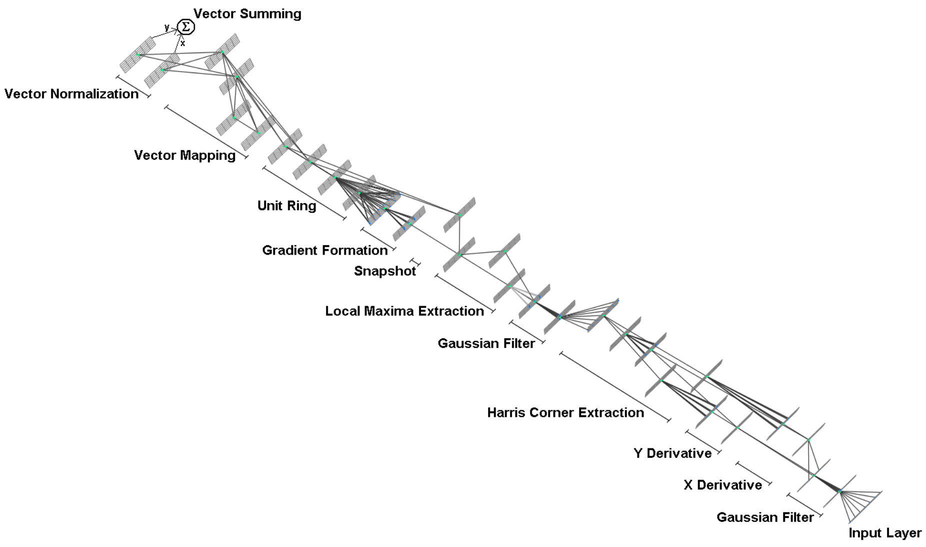


**Fig. 2.** Overview of CVM for CGSM, visualized in OpenGL.

## 3 CVM Structure

We can describe a CVM by giving just the neighbourhood and function of each processing element along a column that ascends from the input layer to the last layer of the CVM. The input layer can be considered the ‘retina’. By the time processing reaches the CVM’s last two layers the input image will have been transformed into a vector which will directly control the agent’s behaviour. The neighbourhood of an element within a CVM describes which of its peers it receives input from and the weighting of their connections with that element. The function achieved by each processing element is one of the following:

`Conv.` The sum of all neighbours' weighted activation levels is returned. This is the convolution operation, prevalent in signal processing.

`Mult.` The product of all neighbours' weighted activation levels is returned.

`Max.` The highest weighted activation level is returned.

`IsMax.` If the *special neighbour* has the highest weighted activation of all neighbours then 1 is returned. Otherwise 0 is returned. The "special neighbour" is a single neighbour designated to have a special status. For `IsMax` it is just the one neighbour that all other neighbours are compared to.

`MaxDX.` The horizontal displacement of the neighbour with the highest weighted activation level is returned.

`MaxDY.` The vertical displacement of the neighbour with the highest weighted activation level is returned.

`Thold(`$f$`).` If the *special neighbour*'s weighted activation level is above $f$ then 1 is returned. Otherwise 0.

`Pow(`$f$`).` The *special neighbour*'s weighted activation level is raised to power $f$.

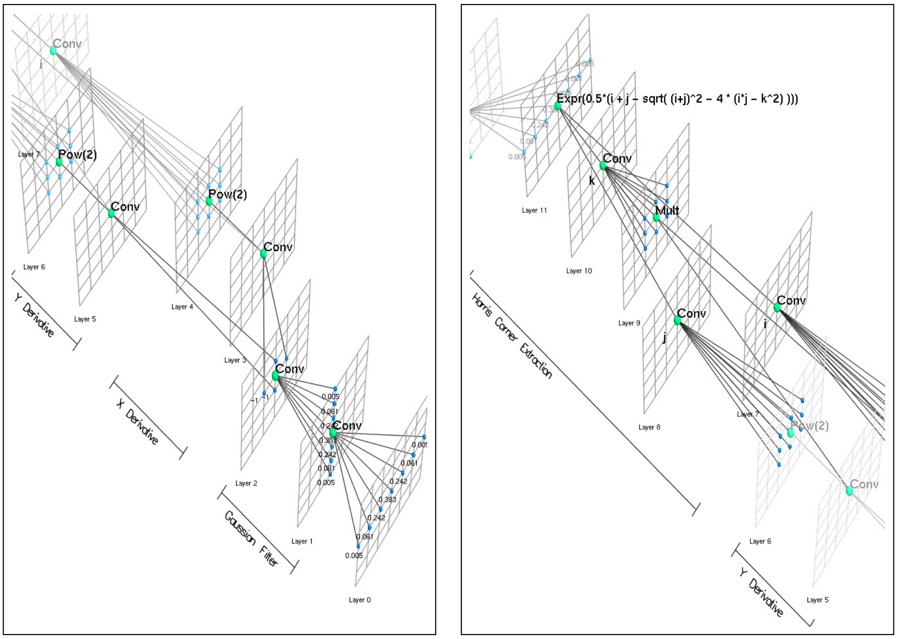
`Expr.` Each neighbour may have a symbol associated with it whose purpose is to stand in for that neighbour's activation level in an algebraic expression. The algebraic expression includes the usual operators and common numerical functions ( $+$, $-$, $*$, $/$, ^, `sin`, `cos`, `sqrt`, and the constant `pi` ). Also, the following special symbols are used to stand for certain element-specific parameters or global constants.

$c$ column index of current element in image (i.e. $x$-coordinate)
$r$ row index of current element in image (i.e. $y$-coordinate)
$w$ image width
$h$ image height
$o$ value set to +1 if the current element is in the top half of the image, and to -1 in the bottom half

An element applies its function and then takes on the resultant value. All elements on a single layer are updated synchronously. In this implementation images are horizontally panoramic but vertically bounded. If an element's neighbourhood stretches horizontally past the image boundary then it is wrapped around toroidally to the opposite side of the image. If an element's neighbourhood stretches vertically past the image boundary then the element will be given a special *out-of-bounds* value, causing any subsequent elements that connect to this element to also become out-of-bounds. Effectively the image being processed will shrink vertically. This approach allows all functions to handle the border in a uniform manner.

### 3.1 Gaussian Filter

The input image is assumed to be corrupted with some amount of high-frequency noise. Some of this noise is removed with a 5x5 Gaussian filter. Figure 3(a) depicts a 2-layer implementation of this filter. Layer 0 is the input layer and Layers 1 and 2 form the filter. The Gaussian kernel is separable, meaning that it can be split into two identical, but orthogonal layers [27]. After processing, Layer 2 will be a smoothed version of Layer 1.



(a) Gaussian, X Deriv, Y Deriv (b) Y Deriv, Harris-Corner-Extr

**Fig. 3.** CVM segments from input image to corner image. Larger spheres are elements, smaller spheres show connections. Numerical labels show connection weights. Symbols used in the `Expr` function are shown adjacent to corresponding elements.

### 3.2 Corner Extraction

We briefly review the general method of corner extraction first before describing how it is implemented here. The method is known as the Harris detector [27] (pages 82-85) and was originally presented in [28]. Consider an image point, $p$, and a small window centred on $p$. If the window is shifted then we can compare the amount of change between the old and new windows. A corner is defined as an image point where this degree of change is large for all possible shifts. A simplified check for this condition is that, for the shift producing the smallest change, the level of change must still be large. An analysis is presented in [28] which equates this minimum change with the smaller eigenvalue of matrix $\mathbf{M}$,

$$\mathbf{M} = \begin{bmatrix} i & k \\ k & j \end{bmatrix} \qquad \text{where } i = \sum D_x^2, \quad j = \sum D_y^2, \quad k = \sum D_x D_y$$

$D_x$ and $D_y$ are the spatial derivatives of the image in the $x$ and $y$ directions. The summations above cover the pixels of a small window centred on $p$. After a few algebraic steps we can arrive at an equation for the smaller eigenvalue, $\lambda_s$,

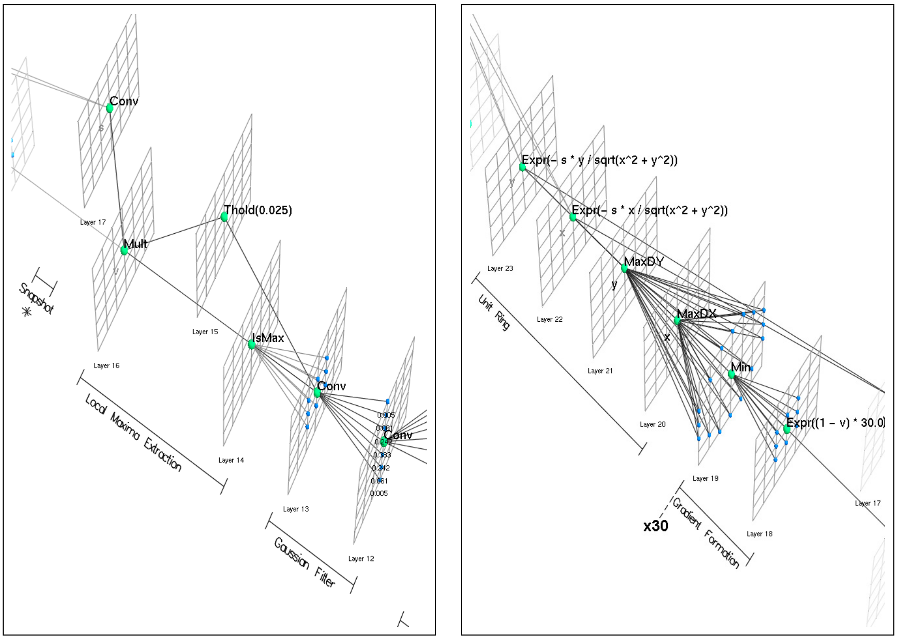
$$\lambda_s = \frac{i + j - \sqrt{(i+j)^2 - 4(ij - k^2)}}{2} \qquad (1)$$

$\lambda_s$ will be the output from the Harris Corner Extraction segment. The first step in arriving at this value is to estimate spatial derivatives. Figure 3(a) shows the X Derivative and Y Derivative segments which do just that. We indicate a particular element at image position $(x, y)$ and layer index $l$ as $V_{x,y,l}$. The value of an element at position $V_{x,y,3}$ is computed as $V_{x+1,y,2} - V_{x-1,y,2}$ which is the central-difference approximation of the first derivative in the $x$ direction. The `Conv` function of Layer 3 combined with its -1 and +1 connections to Layer 2 performs this approximation. Layer 4 squares Layer 3. That is, $V_{x,y,4} = V_{x,y,3}^2$. This provides us with the value $D_x^2$ necessary to calculate $\lambda_s$. The Y Derivative segment is similar to the X Derivative and provides us with $D_y^2$ in Layer 6.

The Harris Corner Extraction segment is shown in figure 3(b). Layer 7 does a summation from Layer 4 of the X Derivative segment. This calculates $i = \sum D_x^2$. Similarly, Layer 8 calculates $j = \sum D_y^2$. The inputs to Layer 9 travel back from figure 3(b) to 3(a). They connect to Layers 3 and 5, the first layers of the X Derivative and Y Derivative segments. Multiplying them yields $D_x D_y$. Layer 10 then does the summation to achieve $k = \sum D_x D_y$. Finally, Layer 11 implements equation 1 to calculate $\lambda_s$.

## 3.3 Local Maxima Extraction

Layer 11 holds an image of $\lambda_s$ values indicating the quality of corner at each position. This may be described as a ‘hilly corner image’. To extract discrete point features the peaks of these hills must be found. Figure 4(a) shows how this is achieved. First, another Gaussian Filter is applied in Layers 12 and 13 to further reduce residual noise as well as any noise introduced by the corner extraction process. Next local maxima will be detected. The `IsMax` function in Layer 14 will set its element to 1 only if the corresponding element in Layer 13 has a value strictly greater than all of its immediate neighbours. The `Thold` function in Layer 15 will set its element to 1 only if the corresponding element in Layer 13 has a value greater than the threshold (0.025). Finally, Layer 16 multiplies these quantities to perform a boolean AND. That is, a point in the smoothed corner image (Layer 13) must be both a local maximum AND exceed a threshold if it is to be extracted as a corner.



(a) Loc-Max-Extr, Snapshot (b) Gradient Formation, Unit Ring

**Fig. 4.** CVM segments from corner image, to gradient image, and finally to image vectors. (a) The `IsMax` function has one dark line to show connection to the special neighbour. The '*' symbol beneath *Snapshot* indicates that this layer requires an outside signal. (b) "x30" indicates that layer 19 is applied for 30 iterations.

## 3.4 Snapshot

Figure 4(a) also depicts the Snapshot layer, Layer 17. This layer simply performs a copy of Layer 16. However, the Snapshot layer requires an external signal to operate. This signal is made active when the agent is positioned at the goal position and allowed to take a snapshot image. If the signal is not active then the Snapshot layer will just maintain its storage.

## 3.5 Gradient Formation

Before describing the implementation of gradient formation in the CVM, we first review the purpose of forming gradients around detected features of the current image. The CGSM homing method requires that correspondences be established between current image features (Layer 16) and snapshot image features (Layer 17), and that these correspondences should be in the form of vectors which point from the snapshot image feature to the corresponding current image features.

From the perspective of an individual element within a CVM, how can these vectors be determined? There can be no 'search for similar elements' because search is a global operation not permitted within a CVM. One answer is to form a gradient image which rises from zero at the position of a current image feature and increases in proportion to distance from the feature. Then, correspondence is established by assuming that a snapshot image feature matches whatever current image feature was responsible for generating the particular 'well' of the gradient image which the snapshot feature is sitting on. The downhill direction of the gradient gives this disparity vector, or *image vector*.

We use Hassoun and Sanghvi's method to form the gradient image [29]. The intended goal of their algorithm is to determine the optimal path between two points on a grid, and to compute this path with a set of processors which are spatially distributed on the same grid. To this end, the first stage of their algorithm distributively computes a potential surface whose height gives the cost of moving to the source point[1]. We use just this aspect of their algorithm here, with cost defined as distance on the grid. In this case, however, there are multiple sources for the potential surface. The grid is initialized to 0 where there are features detected in the feature detection layer, and to some arbitrarily large value $G_{\max}$ everywhere else[2]. $V_{x,y}$ will indicate the current value of a grid element and $V'_{x,y}$ will indicate the new value. The set of neighbours of $(x, y)$ is $\Pi_{x,y}$. The distance from $(x, y)$'s neighbour $(p, q) \in \Pi_{x,y}$ to $(x, y)$ is $d_{p,q}$. This whole method of gradient formation boils down to the following update rule,

$$V'_{x,y} = \min \left\{ V_{x,y}, \min_{(p,q) \in \Pi_{x,y}} (V_{p,q} + d_{p,q}) \right\} \tag{2}$$

Figure 4(b) shows the Gradient Formation segment. This segment is unique in this CVM in that its second layer (Layer 19) is applied in a loop for 30 iterations. On the first iteration this layer is initialized from Layer 18, but thereafter it is recurrently connected to itself (Layer 18 is executed only once). Ideally, a number of iterations equal to the largest dimension of the image would be used so that the gradient would be guaranteed to spread throughout the whole image. However, this is a computationally expensive operation and 30 was chosen as a tradeoff value which seemed to yield good performance. Layer 18 provides the initial conditions for Layer 19. Specifically, it initializes all non-feature points to $G_{\max}$ and all feature points to zero. Layer 19 implements equation 2.

See figure 1 for an example gradient image produced by this segment.

## 3.6 Ring Operator

Figure 4(b) also shows the Unit Ring segment. This segment calculates vectors describing the downhill direction of the gradient image at positions where a

[1] A device-efficient hardware implementation of this same idea is presented in [30].

[2] $G_{\max}$ is set here to 30, the number of iterations of the gradient formation layer.

feature in the snapshot layer exists. Layers 20 and 21 have ring-shaped neighbourhoods connecting to the output of the Gradient Formation segment. These two layers determine the $x$ and $y$ components (respectively) of the required vector. Using the symbols in the figure [3], this vector is $[x, y]$. However, this vector should only exist where there is a snapshot feature. Thus, Layers 22 and 23 multiply the $x$ and $y$ components by $s$ which stands for a connection to the snapshot layer. If $s$ is 0, meaning that there is no snapshot feature at this position, then the vector $[x, y]$ will be zeroed. These expressions also divide $[x, y]$ by its length, making it a unit vector. The negative signs in Layers 22 and 23 invert the image vector $[x, y]$ so that it points downhill on the gradient image.

### 3.7 Vector Mapping

The last two layers of the Unit Ring segment encode *image vectors* expressing the movement of features within the image. The task of the Vector Mapping segment is to transform these image vectors into *agent motion vectors*. An image vector $\overrightarrow{v}$ is the movement of a feature in the image given the movement of the agent in the plane $\overrightarrow{V}$ (agent motion vector). An approximate method was presented in [1] to obtain $\overrightarrow{V}$ from $\overrightarrow{v}$ and $\theta_S$, the angular position of the snapshot feature within the image. Unfortunately, space limitations present repeat coverage of this method here. The method is achieved by equation 3, which has been implemented in this CVM as shown in figure 5(a).

$$\overrightarrow{V} = \begin{bmatrix} v_y \sin\theta_S - v_x \sin(\theta_S + 90^o) \\ v_y \cos\theta_S - v_x \cos(\theta_S + 90^o) \end{bmatrix} \tag{3}$$

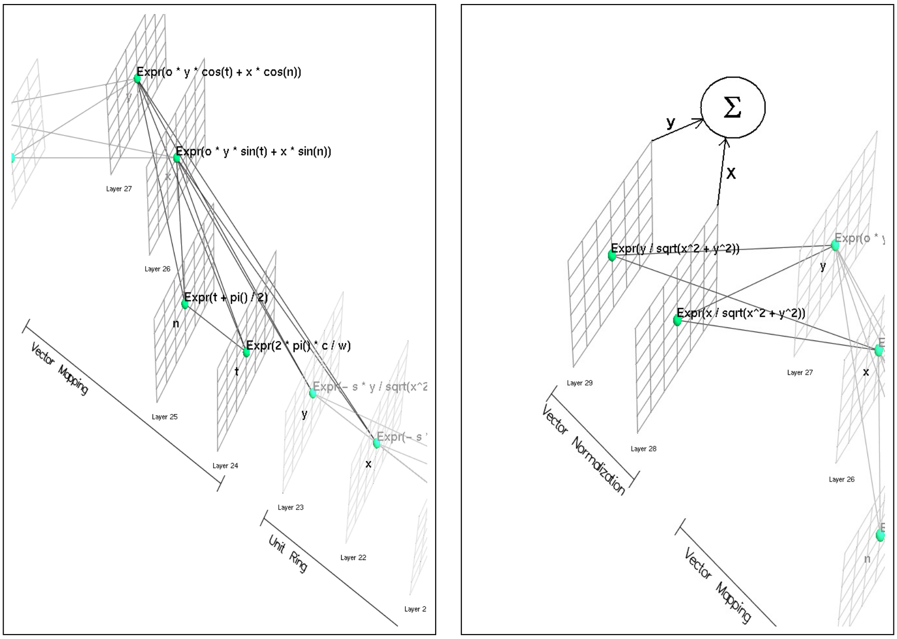
### 3.8 Vector Normalization and Summing

In figure 5(b), the agent motion vectors produced in the Layers 26 and 27 are normalized. This is the final step of the processing chain. Now the final output of the CVM is determined by summing the normalized agent motion vectors stored in Layers 28 and 29. This is illustrated by the large circle in figure 5(b) which produces the two-dimensional vector used to move the agent.

## 4 Results

In [1] CGSM was compared with a model we referred to as the *eXtracted Average Landmark Vector* (XALV) model (from [3]) and found to exhibit superior performance. Here we compare the performance of CGSM with Franz et. al's *warping method* [7]. Other biorobotics researchers have made comparisons between their own methods and the warping method, both favourable and unfavourable. For example, Weber et. al found that their method exceeded the performance of

[3] Variables used in the `Expr` function are recycled. There are several places where the values used are $x$ and $y$ but these values are local to each `Expr` function.

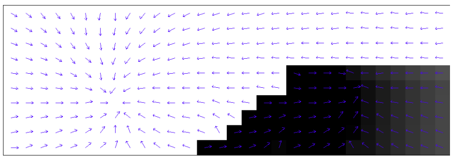


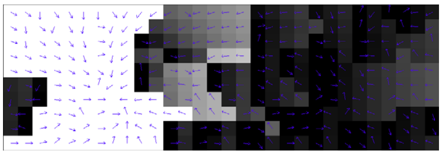
(a) Vector Mapping (b) Vector Normalization, Summing

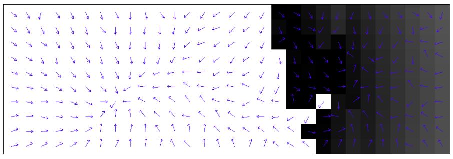
**Fig. 5.** CVM segments to produce (a) agent motion vectors; (b) normalized agent motion vectors. The large circle in (b) indicates the summation of Layers 28 and 29 to produce the $x$ and $y$ components of the final home vector.

the warping method—however, the comparison was made only in simulation [9]. Möller compared robotic implementations of the snapshot model, XALV, and the warping method and found that the warping method generally performed the best and was the least sensitive to parameter settings [31]. The warping method searches through the space of possible movements away from the home position. For each possible movement the snapshot image is warped to appear as if the robot had performed that movement away from the snapshot location. The parameters of the warped image that are most similar to the current image are used to compute the home vector.

As in [1] we compare the two methods on the 300 panoramic images used in [3] (see the top left image in figure 1 for an example image from this database). This database of images was taken using an omnidirectional imaging system on a $9m$ x $3m$ grid of positions in an unmodified university building entrance hall. The unwrapped panoramic images used here are 180 x 48 pixels in size. As a thorough test of performance each of the 300 images is used in turn as the snapshot image, with each such round of trials generating a set of home vectors such as in figure 6. Once all home vectors are generated we attempt to find a path home from every grid position. The overall average number of positions where homing can succeed, over all snapshot images, is the *return ratio*. Table 4 reports the return ratio for CGSM and the warping method. As the left-most column of numbers in this table indicates, the warping method exhibits far superior homing performance. Some example homing maps are shown in figure 6 for both methods. These particular maps are for the snapshot position at coordinates $(210cm, 180cm)$.


(a) Warping Method


(b) Warping Method, One Block


(c) CGSM

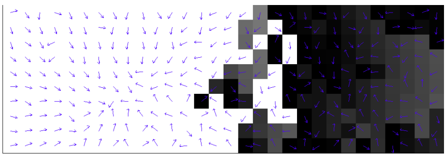

(d) CGSM, One Block

**Fig. 6.** Homing vectors for position $210cm$ x $180cm$. "One Block" means corruption by one block (see text). White cells indicate successful homing. Grey cells give the *approach index* for unsuccessful homing trips (see [1]).

There are, however, at least two particular aspects of the experiment so far which are unrealistic from the perspective of insect modelling. Firstly, an insect such as a honeybee does not have truly panoramic vision. Honeybees in particular have a 'blind spot' of about $50^o$ in the rear [32]. Secondly, an insect's world is dynamic. We expect disruptions in the environment to impair homing performance, however robustness in the face of modified surroundings is critical.

To model both of these conditions we have constructed three corrupted versions of the 300 image database. The corruption is applied simply by blotting out (setting to zero) one, two, or three randomly positioned 40 x 48 vertical blocks within the image. If the blocks happen not to coincide, then this represents a loss of 22.2%, 44.4%, or 66.6% of the information possibly contained in the image (for one, two, and three blocks respectively). Each round of homing ($300^2$ individual trials) is now based on snapshot images from the original image database and current images from the disrupted database. The final three columns of table 4 show how the warping method's performance is severely affected by these disruptions while CGSM exhibits graceful degradation and superior performance to the warping method for all the disrupted cases. Figure 6 shows how the home vectors for both methods change when a single corruption block is added.

**Table 1.** Return ratios for the two methods under varying conditions.

| | **Conditions** | | | |
|---|---|---|---|---|
| **Method** | Undisturbed | 1 Block | 2 Blocks | 3 Blocks |
| CGSM | 69.5% | **63.8%** | **57.1%** | **52.5%** |
| Warping Method | **94.3%** | 34.2% | 19.0% | 15.3% |

## 5 Discussion

The warping method achieves its excellent result on the undisturbed image database by exploiting global image information. All parts of the image contribute to the calculated home vector. While CGSM does not perform nearly as well as the warping method on the undisturbed database, it remains relatively unaffected by large transient changes to the image database. This is due to CGSM's inherent functional parallelism. Approximate home vectors are generated for each feature of the snapshot image. The computation of these home vectors occurs independently and in parallel. These vectors are finally summed to yield the overall result, but this is intended only to offset the possible negative impact of incorrect vectors. The final summation is the first and only time when these multiple results converge.

That CGSM's parallelism should turn out to provide robust performance is an interesting result. Functional parallelism was not one of the intended features of CGSM. Instead, the purpose of CGSM's parallel computing style was to ensure biological-plausibility, and to enable the possibility of parallel implementation for improvement of algorithm complexity. However, we are now encouraged to look to parallelism for performance advantages as well.

## 6 Conclusions

We have presented here a detailed account of the structure and function of a cellular vision matrix for visual homing. It has been shown how operations such as corner extraction, local maxima extraction, gradient formation, and the vector operations could all be achieved by simple locally-connected elements arranged retinotopically. Experiments compared the performance of CGSM with the warping method and found that while CGSM's performance is not as strong as the warping method under ideal conditions, its performance degrades far more gracefully under conditions of large environmental modifications and occlusion.

Much work remains for developing and exploring CGSM. Homing performance has not yet been tested live on a free-roving robotic platform. Also, a thorough analysis of error conditions remains to be completed. Some work in-progress includes the development of an exact vector mapping method and alternative means for locally determining feature correspondence. Finally, it will be interesting to see what other sorts of projects the cellular vision concept can be applied to. Cellular vision was inspired by research on biological systems described at a particular level of detail (low-level, but above that of neural wiring). It remains to be seen how useful this concept may be both in the creation of artificial vision systems—and perhaps—in the understanding of natural vision systems.

**Acknowledgments.** Many thanks to Ralf Möller for reviews and continued guidance, as well as the C++ implementation of the warping method. This work has been partially supported by NSERC Postgraduate Scholarship B - 232621 - 2002.

## References

1. Vardy, A., Oppacher, F.: Low-level visual homing. In Banzhaf, W., Christaller, T., Dittrich, P., Kim, J.T., Ziegler, J., eds.: Advances in Artificial Life - Proceedings of the 7th European Conference on Artificial Life (ECAL), Springer Verlag Berlin, Heidelberg (2003) 875–884
2. Cartwright, B., Collett, T.: Landmark learning in bees. Journal of Comparative Physiology **151** (1983) 521–543
3. Möller, R., Lambrinos, D., Roggendorf, T., Pfeifer, R., Wehner, R.: Insect strategies of visual homing in mobile robots. In Webb, B., Consi, T., eds.: Biorobotics - Methods and Applications. AAAI Press / MIT Press (2001)
4. Roggendorf, T.: Visuelle Landmarkennavigation in einer natürlichen, komplexen Umgebung. Master's thesis, Universität Bielefeld (2000)
5. Hong, J., Tan, X., Pinette, B., Weiss, R., Riseman, E.: Image-based homing. In: Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacremento, CA. (1991) 620–625
6. Röfer, T.: Controlling a whellchair with image-based homing. In: Proceedings of AISB Workshop on Spatial Reasoning in Mobile Robots and Animals, Manchester, UK. (1997)
7. Franz, M., Schölkopf, B., Mallot, H., Bülthoff, H.: Where did i take that snapshot? scene-based homing by image matching. Biological Cybernetics **79** (1998) 191–202
8. Lambrinos, D., Möller, R., Labhart, T., Pfeifer, R., Wehner, R.: A mobile robot employing insect strategies for navigation. Robotics and Autonomous Systems, Special Issue: Biomimetic Robots (1999)
9. Weber, K., Venkatesh, S., Srinivasan, M.: Insect-inspired robotic homing. Adaptive Behavior **7** (1999) 65–97
10. Möller, R., Maris, M., Lambrinos, D.: A neural model of landmark navigation in insects. Neurocomputing **26-27** (1999) 801–808

11. Möller, R.: Insect visual homing strategies in a robot with analog processing. Biological Cybernetics **83** (2000) 231–243
12. Basri, R., Rivlin, E., Shimshoni, I.: Visual homing: Surfing on the epipoles. Technical Report MCS99-10, Weizmann Institute of Science, Mathematics & Computer Science (1999)
13. Nelson, R.: From visual homing to object recognition. In Aloimonos, Y., ed.: Visual Navigation. Lawrence Earlbaum (1996) 218–250
14. Ulrich, I., Nourbakhsh, I.: Appearance-based place recognition for topological localization. In: IEEE International Conference on Robotics and Automation. Volume 2. (2000) 1023–1029
15. Jogan, M., Leonardis, A.: Robust localization using panoramic view-based recognition. In: International Conference on Pattern Recognition. (2000) 136–139
16. Argyros, A., Bekris, C., Orphanoudakis, S.: Robot homing based on corner tracking in a sequence of panoramic images. In: Computer Vision and Pattern Recognition Conference. (2001)
17. Collett, T., Baron, J.: Biological compasses and the coordinate frame of landmark memories in honeybees. Nature **368** (1994) 137–140
18. Zeil, J., Kelber, A., Voss, R.: Structure and function of learning flights in bees and wasps. Journal of Experimental Biology **199** (1996) 245–252
19. Judd, S., Collett, T.: Multiple stored views and landmark guidance in ants. Nature **392** (1998) 710–714
20. Pinel, J.: 7. In: Biopsychology. Allyn and Bacon (1997)
21. Hubel, D., Wiesel, T.: Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. Journal of Physiology **160** (1962) 106–154
22. Egelhaaf, M., Kern, R., Krapp, H., Kretzberg, J., Kurtz, R., Warzecha, A.K.: Neural encoding of behaviourally relevant visual-motion information in the fly. Trends in Neurosciences **25** (2002) 96–102
23. Franceschini, N., Pichon, J., Blanes, C.: From insect vision to robot vision. Philosophical Transactions of the Royal Society of London B **337** (1992) 283–294
24. Harrison, R.: Fly-inspired VLSI vision sensors. In J. Ayers, J.D., Rudolph, A., eds.: Neurotechnology for Biomimetic Robots. MIT Press (2002)
25. Huber, S., Franz, M., Bültoff, H.: On robots and flies: Modeling the visual orientation behavior of flies. Robotics and Autonomous Systems **29** (1998) 227–242
26. Rind, F.: Collision avoidance: from the locust eye to a seeing machine. In Srinivasan, M., Venkatesh, S., eds.: From living eyes to seeing machines. Oxford University Press (1997)
27. Trucco, E., Verri, A.: Introductory Techniques for 3-D Computer Vision. Prentice Hall (1998)
28. Harris, C., Stephens, M.: A combined corner and edge detector. In: Proceedings of the Fourth Alvey Vision Conference. (1988) 147–151
29. Hassoun, M., Sanghvi, A.: Fast computation of optimal paths in two- and higher-dimensional maps. Neural Networks **3** (1990) 355–363
30. Möller, R.: Path planning using hardware time delays. IEEE Transactions on Robotics and Automation **15** (1999) 588–591
31. Möller, R.: Habilitationsschrift, Wirtschaftswissenschaftliche Fakultät der Universität Zürich (2002)
32. Land, M.: Variations in the structure and design of compound eyes. In Stavenga, Hardie, eds.: Facets of Vision. Springer-Verlag (1989)