Appendix

Appendix A

Implementation Details

A.1 Creating the Gaussian Pyramid

In chapter 2 Gaussian pyramids are used in order to compute center-surround differences of various features at various scales. The conventional way of creating the levels of the pyramid consists of two separate steps, convolution with a separable Gaussian filter followed by decimation (Burt and Adelson 1983; Itti 2000). This process is illustrated in figure A.1A for the one-dimensional case, using a convolution kernel of length 3. The first row symbolizes a one-dimensional input image with only one active pixel; the second row shows the result of the convolution; in the third row the image is decimated by discarding all even-numbered pixels (marked with a red cross); and the remaining pixels constitute the resulting subsampled image (fourth row).

It becomes apparent from the figure that computational resources are wasted for computing the convolution for pixels that are later dropped (the gray pixels in this example). This can be avoided by combining the two operations into one integral process, in which convolution is only performed for those pixels that survive subsequent decimation. This saves half of all convolution operations and therefore half of all multiplications. We have implemented this integral operation with separable filters in the iNVT toolkit as well as the SaliencyToolbox. Computing the saliency map for an input image was sped up by 8 % on average by using the functionally equivalent combined operation instead of separate filtering and decimation.

A second problem with the operation as shown in figure A.1A is that the apparent location of the active pixel in the result image is shifted to the right by 1/2 pixel with respect to the input image. If this operation is applied repeatedly, then the systematic error accumulates from layer to layer. In figure A.2A, for instance, the activity caused by the single active pixel at the center of the input image at level 1 moves more and more to the bottom-right corner of the pyramid levels. By level 4, the center is no longer the most active location.

An intuitive solution would be using the average of two neighboring pixels as the pixel activation of the new map, instead of dropping one pixel and retaining the other. In discrete space, averaging

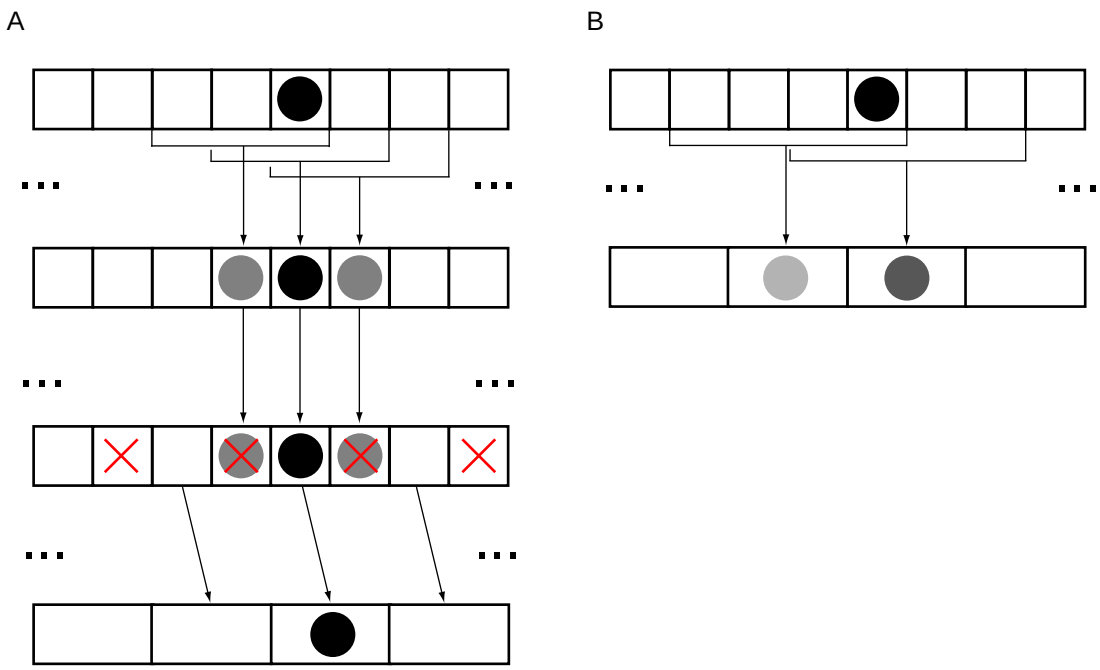

Figure A.1: Illustration of one-dimensional filtering and subsampling. (A) convolution with a filter of length 3 (first to second row), followed by decimation by a factor of 2 (third and fourth row) – the pixels marked with a red cross are removed; (B) integral operation of convolution with a filter of length 4 and decimation by a factor of 2.

over two pixels is equivalent to convolution with the kernel $K_a = [1\ \ 1]/2$. Since filtering itself is a convolution with a kernel K_f, and since convolution is associative, both operations can be combined into convolution with a new filter kernel $K'_f = K_f * K_a$. This new kernel K'_f is one entry larger than the old kernel K_f. Figure A.1B illustrates this principle by using a kernel of length 4, compared to the length 3 kernel in figure A.1A. As explained above, convolution results are only computed for pixels that survive decimation in this integral operation.

Instead of using full two-dimensional convolution for two-dimensional images, separable filters are used that can be applied in the x and y directions separately. Itti (2000), for instance, uses a separable 5×5 convolution filter $K_f = [1\ \ 4\ \ 6\ \ 4\ \ 1]/16$. With dec_x and dec_y being decimation of the image in the x and y directions by a factor of two, pyramid level L_{i+1} would be computed from level L_i as

$$L_{i+1} = \mathrm{dec}_x \left[K_f * \mathrm{dec}_y \left(K_f^T * L_i \right) \right]. \tag{A.1}$$

With $\circ$ being the combined convolution and decimation, we propose to replace eq. A.1 with

$$\begin{aligned} L_{i+1} &= (K_f * K_a) \circ \left[(K_f * K_a)^T \circ L_i \right] \\ &= K'_f \circ \left[\left(K'_f \right)^T \circ L_i \right], \end{aligned} \tag{A.2}$$

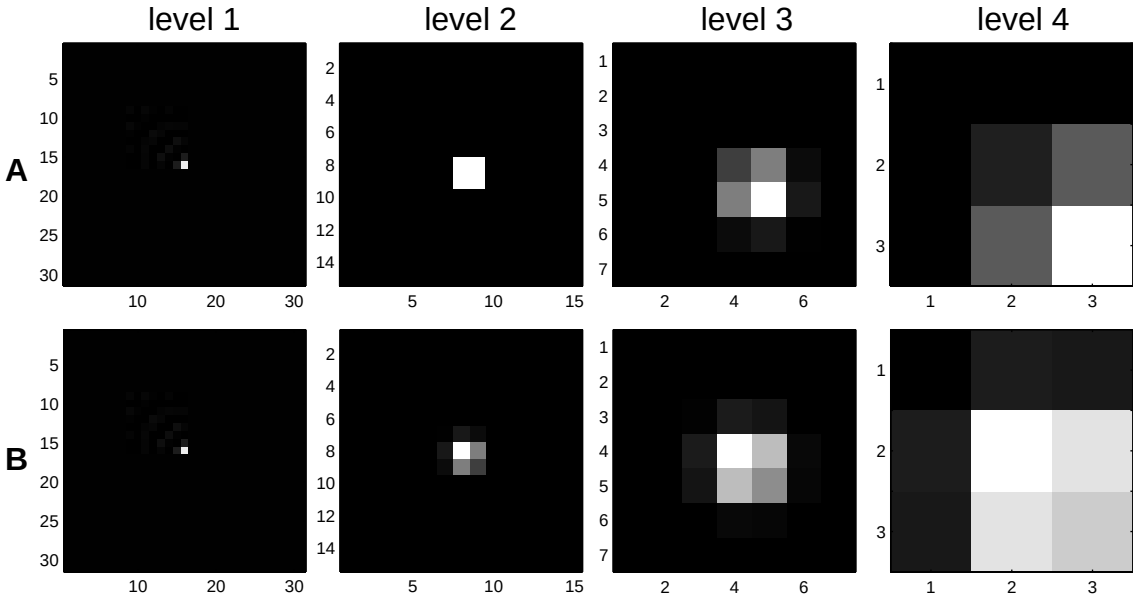

Figure A.2: Example of repeated filtering and subsampling of an image of size 31×31 pixels with only one pixel activated with: (A) a 5×5 filter with subsequent subsampling; and (B) a 6×6 filter with integrated subsampling. Bright pixels indicate high and dark pixels low activity.

where

$$\begin{aligned} K'_f &= K_f * K_a \\ &= [1 \;\; 4 \;\; 6 \;\; 4 \;\; 1]/16 * [1 \;\; 1]/2 \\ &= [1 \;\; 5 \;\; 10 \;\; 10 \;\; 5 \;\; 1]/32 \end{aligned} \tag{A.3}$$

is the new 6×6 separable filter that includes the averaging step. As a general rule, shifting artifacts are best avoided if subsampling by an even factor is accompanied by filtering with a filter of an even length, and subsampling by an odd factor with a filter of an odd length.

We have implemented subsampling according to eq. A.2 with the 6×6 separable kernel from eq. A.3 as part of our SaliencyToolbox (see appendix B). At the image border the filter kernel is truncated. In figure A.2 we show a 31×31 pixels test image with only one active pixel at position $(16, 16)$ being subsampled repeatedly with eq. A.1 (A) and eq. A.2 (B). In (A), the activity due to the active pixel moves toward the bottom-right corner until, at level 4, the bottom-right pixel is the most active instead of the pixel at the center. In (B) the activity caused by the pixel does not move, although activation still appears to spread to the bottom-right quadrant to some extent.

A.2 Color Opponencies for Bottom-up Attention

Red-green and blue-yellow color opponencies are central to modeling the contribution of color to saliency. To this end, the RGB values of the color input image need to be mapped onto a red-green and a blue-yellow opponency axis in a way that largely eliminates the influence of brightness. Hurvich and Jameson (1957) showed that these two opponency axes can cover the entire visible light.

With RGB pixel values (r, g, b), Itti (2000) defines illumination independent hue values for red (R_i), green (G_i), blue (B_i), and yellow (Y_i) as

$$R_i = \left[\frac{r-(g+b)/2}{I_i}\right]_+, \tag{A.4a}$$

$$G_i = \left[\frac{g-(r+b)/2}{I_i}\right]_+, \tag{A.4b}$$

$$B_i = \left[\frac{b-(r+g)/2}{I_i}\right]_+, \text{ and} \tag{A.4c}$$

$$Y_i = \left[\frac{r+g-2\left(|r-g|+b\right)}{I_i}\right]_+ \text{ with} \tag{A.4d}$$

$$I_i = \frac{r+g+b}{3}, \tag{A.4e}$$

where $[\cdot]_+$ denotes rectification. For numerical stability, R_i, G_i, B_i, and Y_i are set to zero at locations with low luminance, i.e., $I < 1/10$, assuming a dynamic range of $[0, 1]$. Red-green (RG_i) and blue-yellow (BY_i) opponencies are defined as

$$RG_i = R_i - G_i \tag{A.5a}$$

$$BY_i = B_i - Y_i, \tag{A.5b}$$

and their center-surround differences are computed across scales as shown in eq. 2.3.

Some of the problems with this definition are illustrated by the examples in table A.1. Orange $(1, 0.5, 0)$, which is perceived as consisting of equal parts of red and yellow, for instance, yields $RG_i = 1.5$, which is half of the value for pure red $(1, 0, 0)$, and $BY_i = -1$, which is only one third of the value for pure yellow $(1, 1, 0)$. For magenta $(1, 0, 1)$, one would expect full positive response from both opponency values, but they are only 0.75 compared to 3.0 for fully saturated red and blue, respectively. Decreasing color saturation by half should lead to a decrease of the color opponency values by half as well. However, for desaturated red $(1, 0.5, 0.5)$, $RG_i = 0.75$ compared to 3.0 for full saturation, while desaturated yellow $(1, 1, 0.5)$ gives $BY = -1.2$, compared to -3.0 for the fully saturated color.

Two main problems need to be addressed to find a better definition of the color opponencies:

Table A.1: Color opponency values for several colors.

Color	(r, g, b)	Itti's definition (eq. A.4) RG_i	BY_i	our definition (eq. A.6) RG_w	BY_w
red	$(1, 0, 0)$	3.0	0.0	1.0	0.0
green	$(0, 1, 0)$	–3.0	0.0	–1.0	0.0
blue	$(0, 0, 1)$	0.0	3.0	0.0	1.0
yellow	$(1, 1, 0)$	0.0	–3.0	0.0	–1.0
orange	$(1, 0.5, 0)$	1.5	–1.0	0.5	–0.5
magenta	$(1, 0, 1)$	0.75	0.75	1.0	1.0
cyan	$(0, 1, 1)$	–0.75	0.75	–1.0	1.0
white	$(1, 1, 1)$	0.0	0.0	0.0	0.0
desaturated red	$(1, 0.5, 0.5)$	0.75	0.0	0.5	0.0
desaturated yellow	$(1, 1, 0.5)$	0.0	–1.2	0.0	–0.5

the definition of yellow and normalization with brightness.

Yellow is perceived as the overlap of red and green in equal parts, so that the amount of yellow contained in an RGB pixel is given by $min(r, g)$. Only amounts of red or green exceeding this value should be counted towards red-green opponency in order to assure independence of the RG and BY opponency axes.

To address the normalization issue, let us consider the colors red $(1, 0, 0)$, blue $(0, 0, 1)$, and magenta $(1, 0, 1)$ for a moment. For red, we would expect $RG = 1$, and for blue, $BY = 1$. Using the average over (r, g, b) as defined in eq. A.4e for normalization leaves us with a normalization factor of 3. For magenta, we would also expect $RG = 1$ and $BY = 1$, but now the normalization factor is 3/2. Cases like these can be reconciled by normalizing with the maximum over (r, g, b) instead of the average.

With these two observations we arrive at our definitions of red-green (RG_w) and blue-yellow (BY_w) color opponencies used in eqs. 2.2 on page 7:

$$RG_w = \frac{r - g}{\max(r, g, b)} \text{ and} \tag{A.6a}$$

$$BY_w = \frac{b - \min(r, g)}{\max(r, g, b)}. \tag{A.6b}$$

RG_w and BY_w are set to zero at locations with $max(r, g, b) < 1/10$.

Table A.1 compares the RG and BY opponencies obtained with eqs. A.4 and A.5 with the values from eqs. A.6. The problems raised with the definition by Itti (2000) are addressed by our new definition of red-green and blue-yellow color opponencies.

Our method of computing color opponencies is implemented as an option in the *iLab Neuromorphic Vision C++ Toolkit* (`http://ilab.usc.edu/toolkit`), and it is the standard method in our SaliencyToolbox (see appendix B).

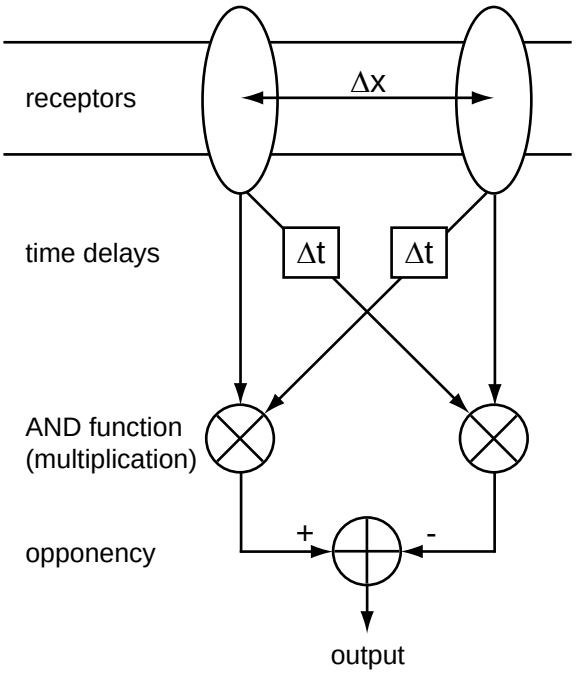

Figure A.3: Schematic of the correlation-based motion detector by Hassenstein and Reichardt (1956). The activation of each receptor is correlated with the time delayed signal from its neighbor. The leftwards versus rightwards opponency operation prevents full field illumination or full field flicker from triggering the motion output signal.

A.3 Motion as a Salient Feature

Motion is a very salient cue for bottom-up attention and should be part of a models of saliency-based attention. Here we describe a simple way of detecting motion that makes use of the existing multi-scale representation of the visual information in pyramids. This work was performed as a SURF project with Chuck Yee during the summer 2002.

Hassenstein and Reichardt (1956) described motion detection in the visual system of the beetle *Chlorophanus* using correlation of signals between adjacent ommatidia with a time delay (figure A.3). Adelson and Bergen (1985) showed the equivalence of the Hassenstein-Reichardt model with a full spatiotemporal energy model of motion under suprathreshold conditions.

Here we adopt this principle to detect motion at multiple scales. If $\mathcal{M}_I(x, y, t)$ is a pixel in the intensity map (eq. 2.1) at position (x, y) and time t, then we can compute the motion opponency maps for left-right ($\leftrightarrow$) and up-down ($\updownarrow$) motion as

$$\mathcal{M}_{\leftrightarrow}(x, y, t) = \mathcal{M}_I(x, y, t - \Delta t) \cdot \mathcal{M}_I(x + \Delta x, y, t) - \mathcal{M}_I(x + \Delta x, y, t - \Delta t) \cdot \mathcal{M}_I(x, y, t), \quad \text{(A.7a)}$$

$$\mathcal{M}_{\updownarrow}(x, y, t) = \mathcal{M}_I(x, y, t - \Delta t) \cdot \mathcal{M}_I(x, y + \Delta y, t) - \mathcal{M}_I(x, y + \Delta y, t - \Delta t) \cdot \mathcal{M}_I(x, y, t). \quad \text{(A.7b)}$$

This is similar in principle, but not in implementation details to the implementation of motion

detection in a Connection Machine by Bülthoff et al. (1989).

Assuming that the origin is in the top-left corner of the image, we obtain maps for individual motion directions as

$$\mathcal{M}_{\leftarrow} = [-\mathcal{M}_{\leftrightarrow}]_+ ; \; \mathcal{M}_{\rightarrow} = [\mathcal{M}_{\leftrightarrow}]_+ ; \; \mathcal{M}_{\uparrow} = \left[-\mathcal{M}_{\updownarrow}\right]_+ ; \; \mathcal{M}_{\downarrow} = \left[\mathcal{M}_{\updownarrow}\right]_+ , \tag{A.8}$$

with $[\cdot]_+$ symbolizing rectification.

The motion detectors in eqs. A.7 are most sensitive to motion velocities $v_x = \Delta x/\Delta t$ or $v_y = \Delta y/\Delta t$, respectively. With Δt fixed to one frame (e.g., 33 ms at a typical frame rate of 30 frames per second), we can obtain a range of velocities by applying eqs. A.7 to multiple levels σ of the intensity pyramid. Because of the dyadic subsampling, a value of $\Delta x = \Delta y = 1$ pixel in level σ will lead to highest sensitivity to motion at a velocity of $v_\sigma = 2^\sigma$ pixels per frame at level σ in the motion detection pyramid.

So far, we have limited the directions of motion to the four cardinal directions, which correspond to motion vectors $(\pm 1, 0)$ and $(0, \pm 1)$. In these cases, eq. A.7 corresponds to correlating an intensity map at time t with a shifted version of the map at time $t - \Delta t$. Shifting by integer pixels is straight forward and easy to implement. It is also possible to consider other directions of motion, e.g., in the diagonal directions with motion vectors $(\pm\sqrt{1/2}, \pm\sqrt{1/2})$. Shifting maps by fractional numbers of pixels is implemented using bilinear interpolation. In our implementation, motion detection with arbitrary motion vectors is possible.

We solve the boundary problem by setting pixels that are shifted into the image to an initial value of zero, and by subsequently attenuating the boundaries in the motion opponency maps to avoid saliency artifacts at the image border.

Allman et al. (1985) report non-classical receptive fields for motion perception in area MT with an excitatory center and an inhibitory surround for a particular direction and speed of visual motion (figure A.4). In their Selective Tuning Model, Tsotsos et al. (1995, 2002, 2005) have modeled these receptive fields using a beam of inhibition around the attended location in a feedback pass.

In the context of sections 2.2 and 2.3, we can model this behavior with the existing center-surround mechanism by applying eq. 2.3 to the motion maps from eq. A.8, redefining L as: $L = L_I \cup L_C \cup L_O \cup L_M$ with $L_M = \{\leftarrow, \rightarrow, \uparrow, \downarrow\}$. Eq. 2.5 can now be applied directly, and eq. 2.6 is extended by a new motion conspicuity map:

$$\mathcal{C}_M = \mathcal{N}\left(\sum_{l \in L_M} \bar{\mathcal{F}}_l\right), \tag{A.9}$$

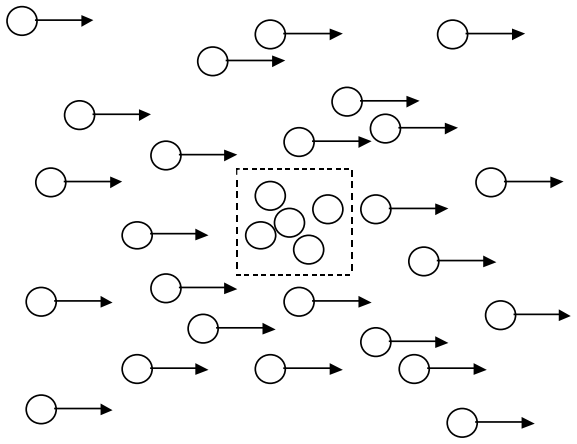

Figure A.4: Illustration of center-surround receptive fields for motion perception. The five dots at the center of the display are salient even though they are stationary because they are surrounded by a field of moving dots.

which contributes to the saliency map, replacing eq. 2.7 by:

$$\mathcal{S} = \frac{1}{4} \sum_{k \in \{I,C,O,M\}} \mathcal{C}_k. \tag{A.10}$$

We have implemented this mechanism for attending to motion as part of the iLab Neuromorphic Vision C++ Toolkit (`http://ilab.usc.edu/toolkit/`), which is available to the public at no cost. Figure A.5 shows an example of motion feature maps (a-d) and the motion conspicuity map (e) for a white bar moving from left to right (f). As expected, $\mathcal{M}_{\rightarrow}$ shows activity at the edges of the bar, while $\mathcal{M}_{\uparrow}$ and $\mathcal{M}_{\downarrow}$ have no activity. The activity in $\mathcal{M}_{\leftarrow}$ is due to aliasing.

The multiscale implementation of the motion detector in a pyramid allows us to detect a large range of speeds. However, this effect is confounded with decreasing spatial resolution for higher levels of the pyramid, leading to difficulties in detecting small, fast moving objects. To decouple speed from spatial resolution, several values for Δx and Δy might be chosen and applied to a pyramid level with sufficient spatial resolution. Furthermore, our model fails to account for the interactions between different motion directions in non-classical receptive fields that were reported by Allman et al. (1985). These interactions would need to be built into the model explicitly.

Our model only detects local motion at various resolutions, equivalent to the spatiotemporal receptive fields in V1. It does not encompass the detection of global motion patterns such as expansion or rotation by areas MT and MST. Tsotsos et al. (2002, 2005) describe detection of such motion patterns as well as mechanisms for attending to them in their Selective Tuning Model.

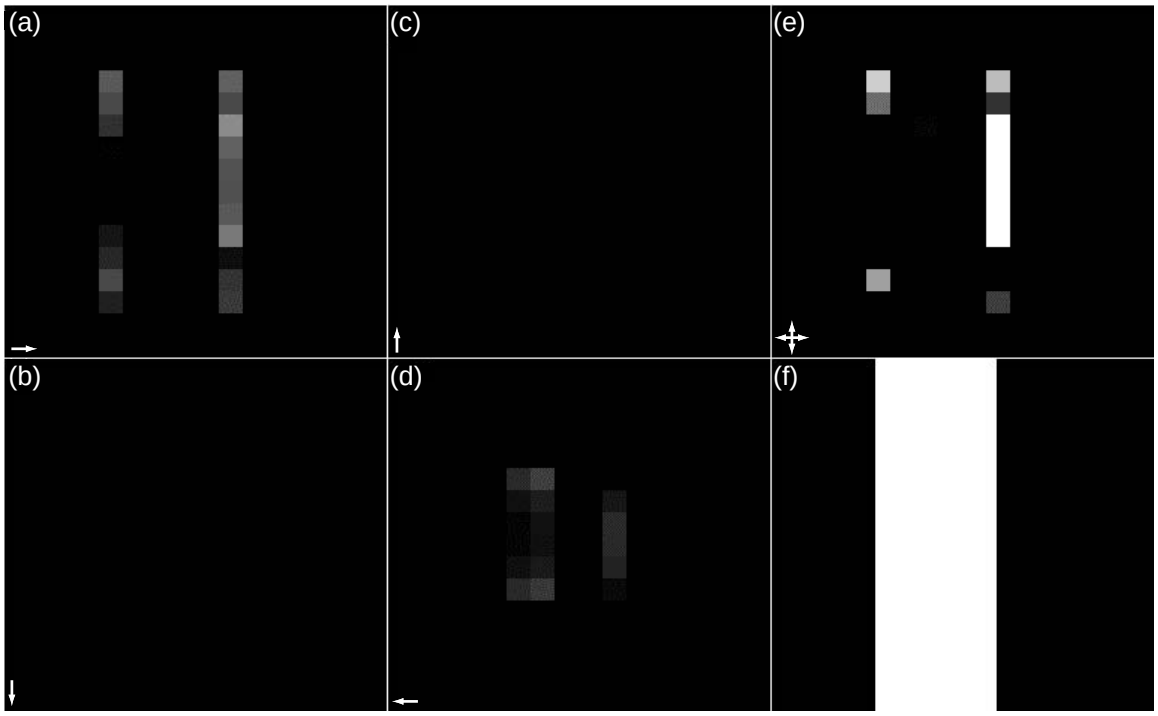

Figure A.5: Feature maps for motion directions right (a), down (b), up (c), left (d), and the motion conspicuity map (e) in response to a rightward moving white bar (f).

A.4 Skin Hue Detection

In chapter 4 we use skin hue as a benchmark for our top-down attention maps because it is known to be a good indicator for the presence of human faces (Darrel et al. 2000). Since we want our model of skin hue to be independent of light intensity, we model it in a simplified color space akin to the CIE color space. If (r, g, b) are the RGB values of a given color pixel, then we compute our (r', g') color coordinates as

$$r' = \frac{r}{r+g+b} \text{ and} \tag{A.11a}$$

$$g' = \frac{g}{r+g+b}. \tag{A.11b}$$

Note that it is not necessary to have a separate value for blue because the blue content of the pixel can be infered from r' and g' at any given light intensity $(r+g+b)$.

We use a simple Gaussian model for skin hue in this color space. For a given color pixel with coordinates $c = (r', g')$, the model's hue response is given by

$$h(c) = \exp\left[-\frac{1}{2}(c-\mu)^T \Sigma^{-1} (c-\mu)\right], \tag{A.12}$$

with $\mu = (\mu_r, \mu_g)$ the center of the distribution and

$$\Sigma = \begin{pmatrix} \sigma_r^2 & \frac{\sigma_r \sigma_g}{\rho} \\ \frac{\sigma_r \sigma_g}{\rho} & \sigma_g^2 \end{pmatrix} \tag{A.13}$$

its covariance matrix with σ_r^2 and σ_g^2 the variances of the r' and g' components, respectively, and ρ their correlation. Substituting eq. A.13 into eq. A.12 yields

$$h(r', g') = \exp\left[-\frac{1}{2}\left(\frac{(r'-\mu_r)^2}{\sigma_r^2} + \frac{(g'-\mu_g)^2}{\sigma_g^2} - \frac{\rho(r'-\mu_r)(g'-\mu_g)}{\sigma_r \sigma_g}\right)\right]. \tag{A.14}$$

To estimate the parameters of the skin hue distribution, we used 1153 color photographs containing a total of 3947 faces from the world wide web[1] and fitted the hue distribution of the faces. The resulting parameters are shown in table A.2. The images used for estimating the skin hue model are a separate set from the set of images used in chapter 4. The images depict humans of many different ages and ethnicities, both female and male. There is a slight bias toward caucasian males, reflecting a general bias of images of humans inthe world wide web. We observed that the skin *hue* does not vary much between different ethnicities, while brightness of the skin shows much more variations.

Table A.2: Parameters of the distribution of skin hue in (r', g') color space.

Parameter	*Value*
μ_r	0.434904
μ_g	0.301983
σ_r	0.053375
σ_g	0.024349
ρ	0.5852

Figure A.6 shows the hue of the training faces and the fitted distribution. An example for applying eq. A.14 to a color image is shown in figure A.7. The skin hue model is implemented as part of the SaliencyToolbox for Matlab (appendix B).

[1] Thanks to Dr. Pietro Perona for providing the images.

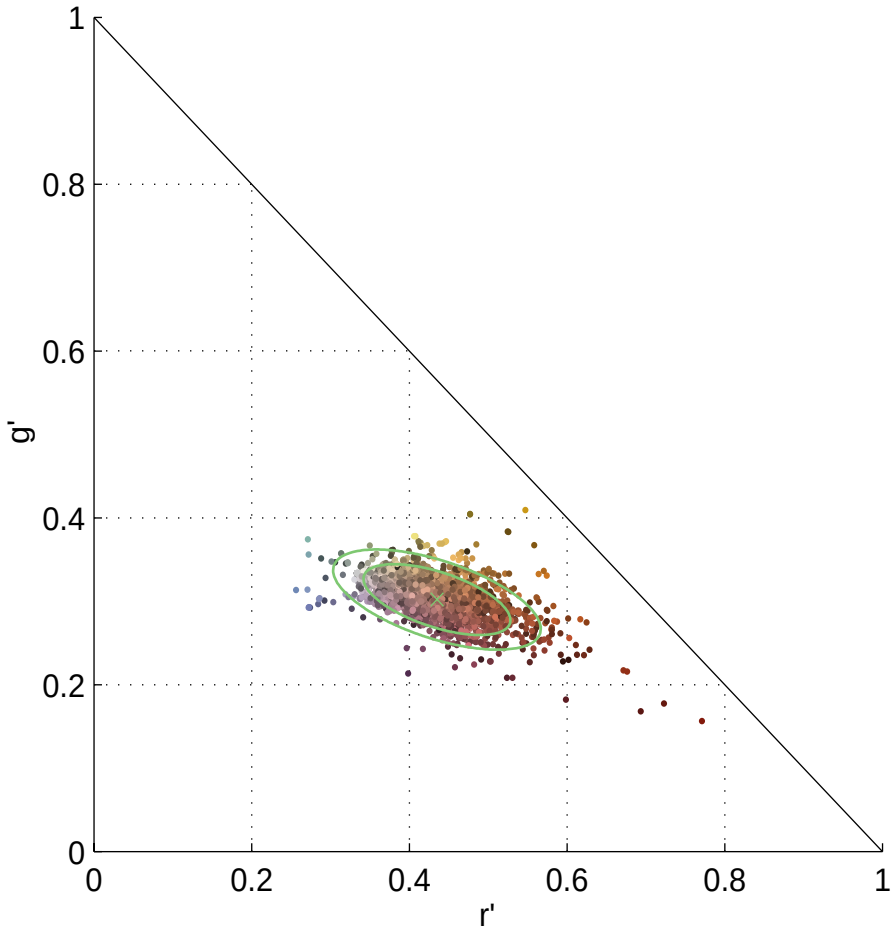

Figure A.6: The Gaussian model for skin hue. The individual training points are derived from 3974 faces in 1153 color photographs. Each dot represents the average hue for one face and is plotted in the color of the face. The green cross represents the mean (μ_r, μ_g), and the green ellipses the 1σ and 2σ intervals of the hue distribution.

Figure A.7: Example of a color image with faces (left) processed with the skin hue model from eq. A.14, using the parameters from table A.2 (right). The color scale on the right reflects how closely hue matches the mean skin hue, marked with a green cross in figure A.6. Note that face regions show high values, but other skin colored regions do as well, e.g. arms and hands or the orange T-shirt of the boy on the right.