Cogn Comput (2011) 3:124–145
DOI 10.1007/s12559-010-9077-9

Dynamic, Task-Related and Demand-Driven Scene Representation

Sven Rebhan · Julian Eggert

Received: 1 May 2010 / Accepted: 6 October 2010 / Published online: 2 November 2010

Abstract Humans selectively process and store details about the vicinity based on their knowledge about the scene, the world and their current task. In doing so, only those pieces of information are extracted from the visual scene that is required for solving a given task. In this paper, we present a flexible system architecture along with a control mechanism that allows for a task-dependent representation of a visual scene. Contrary to existing approaches, our system is able to acquire information selectively according to the demands of the given task and based on the system's knowledge. The proposed control mechanism decides which properties need to be extracted and how the independent processing modules should be combined, based on the knowledge stored in the system's long-term memory. Additionally, it ensures that algorithmic dependencies between processing modules are resolved automatically, utilizing procedural knowledge which is also stored in the long-term memory. By evaluating a proof-of-concept implementation on a real-world table scene, we show that, while solving the given task, the amount of data processed and stored by the system is considerably lower compared to processing regimes used in state-of-the-art systems. Furthermore, our system only acquires and stores the minimal set of information that is relevant for solving the given task.

Keywords Scene representation · Cognitive control · Attention · Visual search

S. Rebhan (✉) · J. Eggert
Honda Research Institute Europe, Carl-Legien-Str. 30, 63073 Offenbach/Main, Germany
e-mail: sven.rebhan@honda-ri.de

J. Eggert
e-mail: julian.eggert@honda-ri.de

Introduction

The visual environment of humans is full of details. To account for a limited computational power and memory capacity, humans selectively process and store those details present in the visual scene. Different experiments show that the selection process is based on the given task and the subject's knowledge about the vicinity and the world. The strong influence of a given task on the way we scan a scene was shown in Yarbus [40]. There, the observed subjects' scan pattern for a visual scene varied, dependent on the given task. Subjects fixated locations containing task-relevant information more frequently, whereas other locations were not visited at all. The results of this experiment was also confirmed by others, e.g. [18, 21, 29, 31]. Beside the task, the knowledge about the current scene and the world in general plays an important role for the way we process a visual scene as experiments conducted by e.g. [9, 19, 21, 34] show. All those experiments suggest, that locations that are relevant for solving a certain task are preferred in contrast to locations containing no such information. However, the attention of humans is not limited to spatial selectivity but also applies to the details stored about objects in the scene such as color, size, form, etc. Experiments presented in Ballard et al. [18, 36] investigate the relation between a given task and the details stored about objects in the scene. They suggest that subjects store only those properties of objects which are relevant to solve a given task. As Triesch put it *"What we see is what we need."* [36]. To summarize, short- and long-term memory as well as the current task bias the attention on objects in the current scene. The experiments have also shown that our attention is not only guided spatially but also in the feature domain.

On the modeling side for visual attention, there exist quite a few architectures for vision systems. As an unbound and data-driven visual search is NP-Complete [37], we here review only models that are based on a top–down guided attention schema. One of the first models describing a top–down guided visual processing was proposed in Bajcsy [4]. The so-called "Active Vision" Aloimonos et al. [2] models interpret vision as an active process, where sensor parameters like zoom, focus, gain or gaze are actively modulated to disambiguate the visual input in a task-specific manner. While earlier work focused on the control of sensory parameters, Ballard [6] and Aloimonos [1] emphasize the ability to control the gaze and thus the spatially selective processing and representation of the scene. Another group of system architectures based on the work of Rensink [28] concentrate on modeling the top–down influence on attention processes. In Navalpakkam and Itti [25], information about the spatial layout of a scene and the knowledge about the world is used to guide attention to locations in the image containing objects relevant for a given task. Here, the knowledge about relevant objects stored in a long-term memory is used to modulate input feature channels to render those objects more salient. In doing so, a coupling of the system's memory and its sensory apparatus is achieved, allowing a system to integrate information about its world over time and reuse it later. Another attention system incorporating top–down knowledge was published in Hamker [16]. Here, the author focuses on a biologically plausible modeling of the top–down influences by incorporating an expectation about the features that should be seen at the target location. Similar approaches using a top–down modulation in the feature space can be found in e.g. Frintrop et al. [13].

All models of vision systems presented so far incorporate the task and the knowledge about the scene and objects in the world. However, the attention in current state-of-the-art systems (see Frintrop et al. [14] for a comprehensive review) is only guided *spatially*. It becomes clear that in contrast to what psychophysical experiments suggest, those models lack attention on *which properties* of an object should be extracted from the scene. That is, once a certain object is attended, in state-of-the-art models *all* properties of this object are extracted, not only those relevant for the task. By not selectively processing the features of objects, these systems neglect potential savings in both the amount of processing and the used memory capacity, which are relevant for resource constraint vision systems. For example, if the task only requires to determine the color of an object, state-of-the-art models nevertheless will run a classifier and store a full-fledged representation of the object as they are built on static processing pathways. In such a processing paradigm, higher level information is computed in pipes from the image pixel up to e.g. an object ID while modulating the different stages in a top–down manner. However, due to the static processing pipelines, a selective extraction of information is not possible as this would require to run and dynamically concatenate subparts of the processing pipelines. In the example of extracting the color of an object, the subparts like saliency computation, segmentation, and color extraction could form a color-extraction process, while not running e.g. the classifier. This example shows that a more flexible and dynamic system architecture is required, allowing for an easy combination of different processing modules. Very early ideas on such a flexible architecture can be found in Ullman [38]. There, the so-called "visual routines" can be seen as highly specialized and independent processing modules like depth computation, color and shape extraction or segmentation which are combined on demand to extract more complex information about an object. Unfortunately, Ullman does not describe how this coupling can be done dynamically in the system. A control mechanism that schedules measurements and a comprising system architecture is missing. However, according to Hayhoe [17], the scheduling of measurements, i.e. when to extract which property of an object dependent on the task, is a fundamental question.

In this paper, we present a flexible system architecture along with a control mechanism that allows for a task-dependent representation of a visual scene. Contrary to existing approaches, our system is able to acquire information selectively according to the demands of the given task. Our system comprises both a short-term and a long-term memory, a spatial saliency algorithm and multiple visual processing modules used to extract visual properties of a focused object. The different visual processing modules operate independently and are specialized in extracting only a particular visual property. However, the dynamic coupling of multiple processing modules allows for the extraction of specific, more complex features that are relevant for solving the given task. The control mechanism we present decides which properties need to be extracted and which processing modules should be coupled. This decision is based on the knowledge stored in the long-term memory of the system. Additionally, the control mechanism ensures that algorithmic dependencies between processing modules are resolved automatically, utilizing procedural knowledge which is also stored in the long-term memory. We evaluate a proof-of-concept implementation of a system constructed according to the architecture and the control mechanism presented in this paper. The experimental evaluation using a real-world table scene shows that while solving the given task, the amount of data processed and stored by our system is considerably lower compared to processing regimes used in state-of-the-art systems. This in turn leads to a noticeable reduction of the computational load and memory demand. The presented work contributes to a task-dependent representation of visual scenes, because only those pieces of information are acquired and stored that are relevant for solving the given task.

In the next section, we present the flexible architecture of the system and briefly discuss the visual routines used in our proof-of-concept implementation. The system's memory architecture as well as important relations in this memory are presented in "Memory architecture". The interplay between the memory and the computation modules is controlled by the attention control mechanism presented in "Attention Control Mechanism". This mechanism schedules the measurement processes and dynamically compiles processing pipelines using the available visual routines. We then use a proof-of-concept implementation to evaluate our ideas. The results of these experiments are shown in "Results". Finally, we discuss these results in "Discusson".

System Architecture

In the following, we will present our flexible system architecture and show an example of the information flow in such a system. Afterward, we briefly discuss important processing parts, called visual routines, that are used in the proof-of-concept implementation of the described architecture.

Architecture

The system architecture we propose comprises three major parts as shown in Fig. 1. On top, there is the relational memory including both the short-term and long-term memory (see 1 in Fig. 1). In the short-term memory, information about the current scene is stored, whereas in the long-term memory the system's knowledge about the world is represented. Here, a relational graph structure is used to store relations between objects and properties along with information about the measurement processes of object properties. More details can be found in "Memory Architecture". Based on the memory and the given task, the attention control mechanism (see 2 in Fig. 1) determines which regions and features will be attended in the scene. To do so, the attention control mechanism is able to modulate features in a saliency map or in visual routines to find new object candidates or measure object properties, respectively. Furthermore, the control mechanism is responsible for scheduling the internal measurement processes and thus the visual routines of the system. Please refer to "Attention Control Mechanism" for more details. Once a property is selected for measurement, the corresponding visual routine is triggered. The visual routines and the saliency map are located in the middle layer of the architecture shown in Fig. 1. The saliency map (see 3 in Fig. 1) is dedicated to finding object candidates in the visual scene that best match the top–down expectations raised by the memory and the attention control mechanism. The visual routines marked with 4 and 5 in Fig. 1 are specialized to compute *one* selected visual property of an object each, similar to the concept proposed in Ullman [38]. Separating the processing into different highly specialized modules allows an easy extension of the system and a free combination of those modules in later stages. Furthermore, it becomes possible to run only those visual routines required to gather the pieces of information relevant for the task. We can arrange the visual routines into three different (partly overlapping) groups. On the left-hand side, there are object-unspecific routines like the

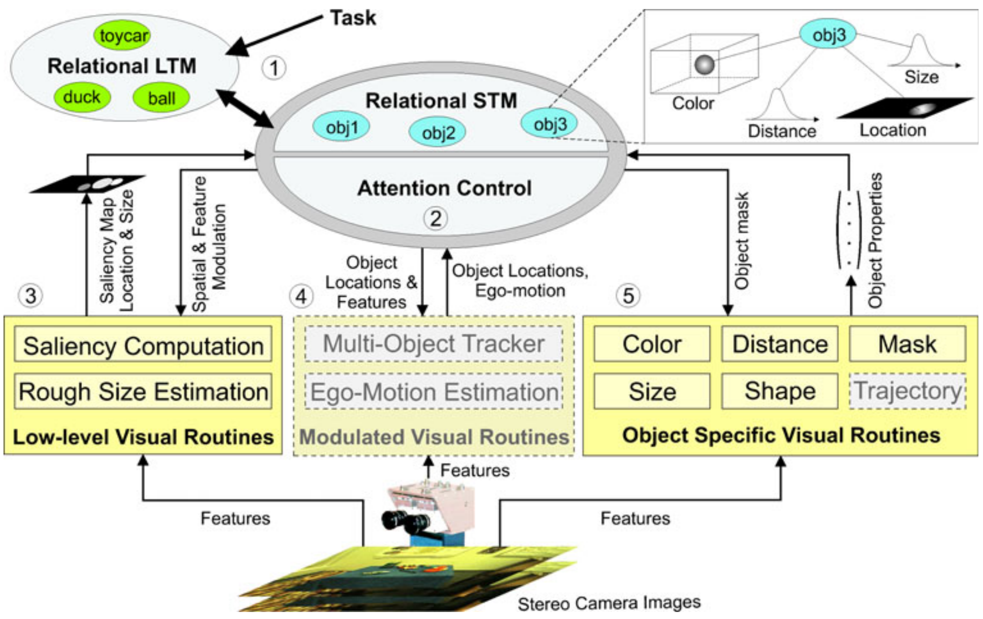

Fig. 1 The system architecture comprises (1) a relational short-term and long-term memory, (2) an attention control mechanism coordinating the processing and information flow of the different visual routines (3–5), (3) object-unspecific visual routines like a saliency computation, (4) multi-object visual routines like multi-object tracking (not implemented here but in Eggert et al. [11]) and (5) object-specific visual routines like segmentation and property measurements

saliency computation (see 3 in Fig. 1). Those routines are bottom-up driven and only influenced by modulatory top–down inputs. The unspecific processing allows parallel computation with respect to the image location. Routines of the second group (see 4 in Fig. 1), work in parallel on a few objects. However, they are tuned to those objects by top–down information like object features or locations. An example for such a routine is a multi-object tracker. Currently, this class of visual routines is not implemented (grayed-out box in Fig. 1) in our system. However, in Eggert et al. [11] that kind of processing has been tested separately in a proof-of-concept system. On the right-hand side of the middle layer (see 5 in Fig. 1), object-specific visual routines are grouped. Those algorithms work sequentially on only one object at a time. The processing itself is very selective and triggered in a top–down manner, including a strong top–down bias. Examples are classification or segmentation algorithms that include for example object biases and form priors.

Processing Flow Example

If the task of the system is for example to search for a “red object”, the processing flow would look like the one shown in Fig. 2. To initiate the search of the requested red object, the task-related properties are activated in the system’s long-term memory. The attention control mechanism now tries to determine which properties have to be measured in the current visual scene to solve the task. In this case, the attention control mechanism needs to trigger the measurement of the object color to solve the task. Furthermore, the attention control mechanism needs to resolve the elemental processing steps required to measure the color which comprise finding a suitable object candidate, extracting an object mask for the found candidate, and finally measuring the color of the object candidate. In each step, the acquired information is compared to the properties known for the searched object. We will describe this part of the processing in “Attention Control Mechanism“. The elemental processing steps

1. Search a suitable object candidate
2. Extract a mask for the object candidate found
3. Extract the color of the object candidate using its mask to make sure it actually has the searched color

will result in the processing flow shown in Fig. 2.

To locate a suitable object candidate, the attention control mechanism triggers the saliency computation (see 1 in Fig. 2). As indicated by the red arrow in Fig. 2, the attention control mechanism provides modulatory inputs for the saliency processing. These inputs are constructed using the knowledge about the searched object. In the example of finding a red object, the system knows the color to look for and thus can provide this information as a modulatory input to the underlying processing. Furthermore, spatial information can be incorporated, e.g. the object is on the table or the object is on the left. Additional to the modulatory color input, a weight for each saliency feature is provided by the attention control mechanism. Now the saliency map is computed using a color map tuned to enhance red objects, while suppressing objects with other colors. This way, the saliency map is tuned for the given task. The resulting saliency map is propagated to the attention control mechanism (blue arrow in Fig. 2) together with a map containing the roughly estimated retinal object sizes at each location

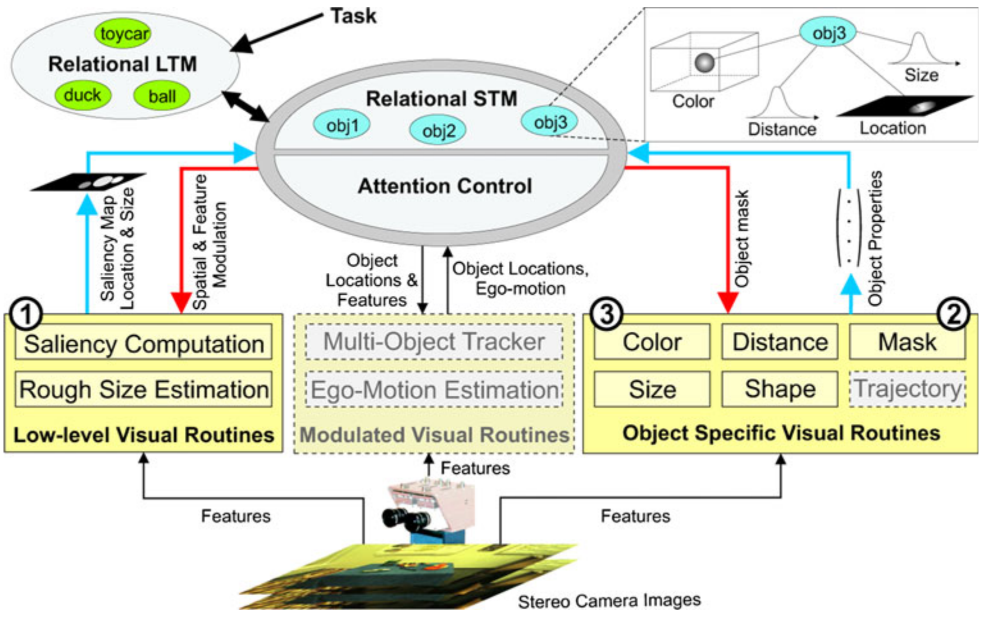

Fig. 2 The processing flow for the simple task “find a red object” is shown here. First, a suitable object candidate is located (1), then the object candidate is segmented (2) and finally the color of the candidate is measured (3). (Color figure online)

(see Rebhan et al. [27] for details). The attention control now selects the most promising object candidate by performing a maximum selection. The selected candidate is attended and stored in the short-term memory, together with its location and rough size. To make sure the attended object candidate actually is the searched object, the system has to measure the object's color. To do so, the attention control mechanism triggers a segmentation routine (see 2 in Fig. 2). Again, the attention control provides modulatory information about the object candidate to the visual routine (red arrow in Fig. 2), namely the location and the rough size previously stored in the short-term memory. The segmentation routine uses this information as a starting condition to segment the object candidate. The resulting mask is then returned to the attention control mechanism (blue arrow in Fig. 2) and stored in the short-term memory together with the already acquired information. At this point of the location, the rough size and the mask for the attended object candidate are known. This is still not enough information to solve the task of finding a red object. However, it enables the system to finally measure the color of the object candidate to make sure it is red. To do so, the attention control mechanism triggers the visual routine for extracting the color of the object candidate (see 3 in Fig. 2). The visual routine requires the location and mask of the object candidate, so the attention control mechanism provides these information as modulatory inputs (red arrow in Fig. 2). The visual routine then extracts and returns the measured color of the object candidate (blue arrow in Fig. 2). The attention control mechanism stores the color in the short-term memory along with the other properties of the candidate and finally compares the measured color with the requested one. If they match (at least to a specified degree), the task is solved. Otherwise a new object candidate needs to be found using the saliency map, while the previous candidate is suppressed (inhibition of return). The evaluation of the measured properties against the properties of the searched object stored in the long-term memory is performed for each newly acquired information. If a mismatch between the stored and measured property is detected, the measurement process can be aborted and a new object candidate will be localized. This whole processing loop will repeat until the searched object is found or all locations in the visual scene have been visited.

Visual Routines

In the following, we describe the different types of processing modules implemented in our proof-of-concept system. The first module described here is the saliency computation, followed by the segmentation module and the group of feature extraction modules. For the last group, we only implemented very basic methods to focus on the attention control mechanism and the system architecture instead of low-level processing. The mentioned processing modules are able to extract the dominant color, the coarse size, and the coarse shape of an object.

Saliency Computation

Our saliency computation incorporates a basic top–down modulation schema on the color feature, a possibility of incorporating spatial modulation and the weighting of different features. Nevertheless, more advanced algorithms as proposed in Hamker [15], Frintrop et al. [13, 26] or Michalke et al. [24] can also be used here. Additional to the saliency map, the processing module computes a rough size estimate (see Rebhan et al. [27] for details). The schematic illustration of the saliency processing is shown in Fig. 3. We can identify three different processing steps: tuning of one or more channels of the input features $\mathbf{F}$ (blue in Fig. 3), calculation of the center-surround contrast (yellow in Fig. 3) and computation of the lateral dynamics (green in Fig. 3).

First, the input features are modulated to enhance the contrast of a searched object with respect to other regions in the scene. Here, we use a method similar to the one proposed in Navalpakkam and Itti [26]. To modulate a certain pixel in the input image, the distribution of an object property is assumed to be Gaussian. Hence, the similarity $\mathbf{t}$ between a selected feature $\mathbf{f}_i$ and the Gaussian distribution (μ, σ) of the searched object property can be calculated for each pixel x, y. In the current implementation, this kind of feature biasing is used for the depth map $\mathbf{z}$ resulting in a biased map $\mathbf{t}_z$ with the top–down provided Gaussian distribution μ_z, σ_z, where μ_z denotes the mean and σ_z the standard deviation, so that

$$t_z(x,y) = e^{-\frac{1}{2\sigma_z^2}(z(x,y)-\mu_z)^2}. \tag{1}$$

For the RGB image $\mathbf{I}^L$ consisting of the channels $\mathbf{i}_r^L, \mathbf{i}_g^L$ and $\mathbf{i}_b^L$, we extend the equation above to the multichannel case. With the Gaussian distribution (μ_r, μ_g, μ_b), $(\sigma_r, \sigma_g, \sigma_b)$, the biased color map $\mathbf{t}_{\text{color}}$ is calculated as

$$t_{\text{color}}(x,y) = e^{-\frac{1}{2}\sum_n \frac{(i_n^L(x,y)-\mu_n)^2}{\sigma_n^2}}. \tag{2}$$

Here, n is the n-th channel of the input features $n = \{r, g, b\}$, μ_n represents the mean values of the Gaussian distribution and σ_n is the standard deviations of the different channels. By applying this kind of modulation, locations with features similar to the searched ones are enhanced, while regions with different features are suppressed. The Gaussian distribution describing features of the searched object is provided by the attention control mechanism based on object knowledge and the given task.

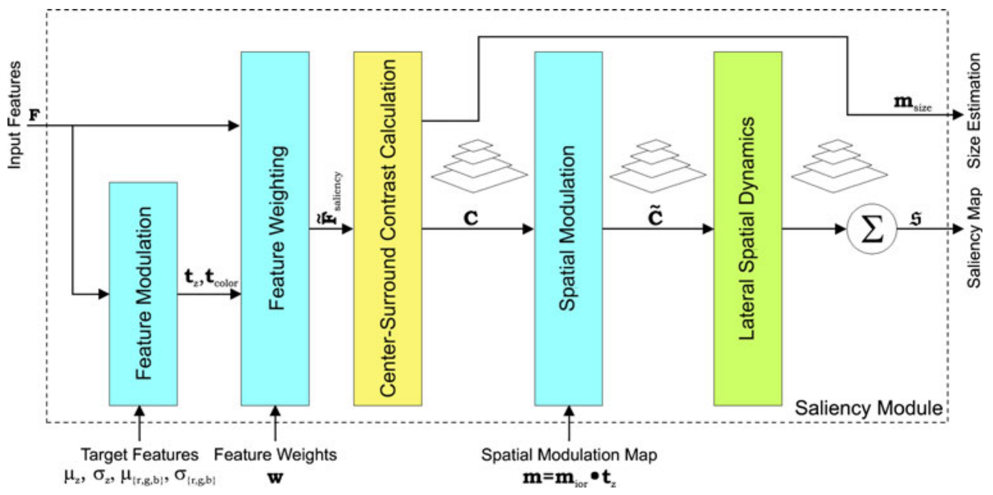

Fig. 3 The input features **F** are modulated and weighted according to the top–down modulatory inputs (*bottom*). After computing the contrast computation, spatial modulation and lateral dynamics, the result is integrated into a final saliency map $\mathfrak{s}$. All these computations work on spatial resolution pyramids. Additionally to the saliency map, the center-surround contrast is used to roughly estimate object sizes

Using the biased and unbiased features, we can construct the feature matrix $\mathbf{F}_{\text{saliency}}$ for the saliency computation consisting of the RGB image, the saturation value **s** and the biased color channel $\mathbf{t}_{\text{color}}$

$$\mathbf{F}_{\text{saliency}} = \left(\mathbf{i}_r^L, \mathbf{i}_g^L, \mathbf{i}_b^L, \mathbf{s}, \mathbf{t}_{\text{color}}\right). \quad (3)$$

The resulting feature matrix $\mathbf{F}_{\text{saliency}}$ is weighted using a top–down weight vector **w** giving the weighted feature matrix $\tilde{\mathbf{F}}_{\text{saliency}}$

$$\tilde{\mathbf{F}}_{\text{saliency}} = \mathbf{F}_{\text{saliency}} \odot \mathbf{w}, \quad (4)$$

where $\mathbf{F}_{\text{saliency}} \odot \mathbf{w}$ denotes the weighting of each feature channel with the corresponding scalar weight.

After biasing and weighting the input features, the center-surround contrast $c_i(x, y)$ for each pixel x, y and center-surround combination i must be computed. Here, i denotes a certain combination of the Gaussian filters $\mathbf{g}_i^{\text{center}}$ and $\mathbf{g}_i^{\text{surround}}$ with σ_i^{center} and $\sigma_i^{\text{surround}}$ respectively. Contrary to other saliency implementations like [20], we use the Euclidean distance between the center and surround feature *vector*. The contrast $\mathbf{c}_i$ is calculated within one combination i as

$$\mathbf{c}_i^2 = \sum_n \left(\left(\mathbf{g}_i^{\text{center}} - \mathbf{g}_i^{\text{surround}}\right) * \tilde{\mathbf{f}}_n\right)^2. \quad (5)$$

The $*$ operator denotes a convolution of the center-surround filter with a single feature channel $\tilde{\mathbf{f}}_n$. We choose the sigmas for the center and surround of the filter combination according to the schema proposed in [20].

The resulting contrast maps $\mathbf{c}_i$ are biased using a spatial modulation map **m**, provided in a top–down manner

$$\tilde{c}_i(x, y) = c_i(x, y) \cdot m(x, y). \quad (6)$$

The top–down spatial modulation map integrates multiple information. First, the modulation map allows a three-dimensional spatial biasing by using the biased distance map $\mathbf{t}_z$ described in Eq. 1. Second, the attention control mechanism can inhibit locations of known objects based on the memory content using the modulation map $\mathbf{m}_{ior}$. This process is known as inhibition of return (IOR) [20]. The multiplicative combination of both maps leads to

$$m(x, y) = m_{ior}(x, y) \cdot t_z(x, y). \quad (7)$$

Another important difference to other saliency implementations is the fact that in our approach, the spatial modulation is done *before* the lateral dynamics. The implications of this slight change become clear when we look at the effect of the lateral dynamics, which act as strong non-linearities in the spatial domain. That is, while different peaks compete with each other, their positions get shifted across the map. When the spatial modulation is applied after that non-linearity, one has to take these shifts into account, which is impossible. By applying the spatial modulation map before the lateral dynamics, we can circumvent this problem.

The next stage of the processing is to perform a lateral spatial competition on the contrast maps $\tilde{\mathbf{c}}_i$ (see Fig. 3). Currently, we apply Amari field dynamics [3] to enhance the signal-to-noise ratio (SNR) in those maps. Finally, the maps of different scales i are combined into one single saliency map $\mathfrak{s}$, containing locations of object candidates. Figure 4 shows some results for an unbiased, a feature biased and a feature and location biased saliency map. It can be seen that the more knowledge is included in the processing, the less object candidates remain in the map. Less object candidates also mean a faster visual search time (see e.g. [26]).

Segmentation

In our system, the object mask plays a major role in the further processing, as it is required to calculate all other object properties. To calculate this mask, we use the multi-cue Level-Set method presented in Weiler and Eggert [39].

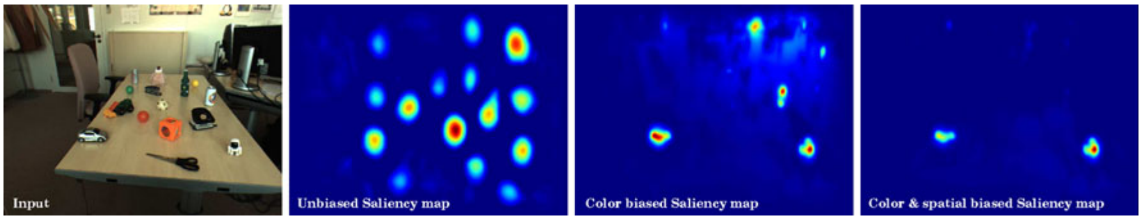
Input
Unbiased Saliency map
Color biased Saliency map
Color & spatial biased Saliency map

Fig. 4 This shows an example of an unbiased, color biased (*white*) and a color and spatially biased (distance bias at 70 cm) saliency map

The mask itself is a function of the retinal object location $\mathbf{x} = (x, y)^T$, the low-level feature vector $\mathbf{F}$ and potential feature modulations $\mathbf{F}_{\text{mod}}$

$$\mathbf{p}_{\text{mask}} = f(\mathbf{x}, \mathbf{F}, \mathbf{F}_{\text{mod}}). \tag{8}$$

Currently, the segmentation is solely based on the color input image. However, it is possible to employ more features like depth or texture which can improve the segmentation results. Furthermore, alternative segmentation algorithms like region-based segmentation Sonka et al. [33] or Graph-cut Boykov and Jolly [8] can be used in our system.

Feature Extraction

Our proof-of-concept system currently comprises four processing modules to extract the dominating color, the distance, a coarse shape and the coarse size of an object. The color and distance properties, $\mathbf{p}_{\text{color}}$ and p_z are approximated using the k-means clustering algorithm [23]. This algorithm estimates the cluster centers $\boldsymbol{\mu}$ in the feature space of a property $\mathbf{p}$ within the given object mask $\mathbf{p}_{\text{mask}}$. In the following, we express the k-means clustering method as $\text{kmeans}(\cdot, \mathbf{p}_{\text{mask}})$, where $\cdot$ is replaced by one or more feature maps $\mathbf{F}$. The second argument is always the binary object mask $\mathbf{p}_{\text{mask}}$

$$\mathbf{p}_{\text{color}} = \boldsymbol{\mu}_{\text{color}} = \text{kmeans}(\mathbf{I}^L, \mathbf{p}_{\text{mask}}) \tag{9}$$

$$p_z = \mu_z = \text{kmeans}(\mathbf{z}, \mathbf{p}_{\text{mask}}). \tag{10}$$

Here, $\mathbf{I}^L$ is the left RGB image of the stereo camera system and $\mathbf{z}$ is the computed depth map. For color and depth, only the dominating cluster, containing the most elements, is passed on to the system's memory. This leads to an estimation of the dominating color or depth while at the same time outliers and locations in the mask, which potentially belong to the background, are ignored.

To determine the size and shape of an object, we calculate the minimal bounding box $\mathbf{p}_{BB}$ for the object mask utilizing the algorithm described in Toussaint [35]. The bounding box is expressed by its width w and height h following the convention $w \geq h$. In the following, the estimation method is denoted with boundingbox($\cdot$) and only gets one argument, the object mask $\mathbf{p}_{\text{mask}}$

$$(w, h)^T = \text{boundingbox}(\mathbf{p}_{\text{mask}}). \tag{11}$$

Using the normalized bounding box parameters w and h, we can calculate the size p_{size} and shape p_{shape} properties as

$$p_{\text{size}} = h \cdot w \tag{12}$$

$$p_{\text{shape}} = h/w. \tag{13}$$

In doing so, we define the size as the area and the shape as the aspect ratio of the bounding box. The resulting coarse shape identifies "compact" objects with $p_{\text{shape}} \approx 1$ and "elongated" objects with $p_{\text{shape}} \ll 1$. Using the pixel size p_{size} and the depth p_z of an object, we can approximate the physical size by

$$p_{\text{physicalsize}} \propto p_{\text{size}} \cdot p_z^2. \tag{14}$$

If a visual routine is triggered by the attention control mechanism, all processing results are passed to the system's short-term memory.

Memory Architecture

To be able to decide which properties of an object should be measured, the system needs to incorporate both the current task and the knowledge about the world. For the visual search tasks we aimed at here, this world knowledge not only comprises information about the object to be found, but also about other objects in the scene. Therefore in this section, we describe our graph-based relational memory, the relations between nodes and the overall memory architecture used in the proof-of-concept system.

Relational Memory

In our approach, the system's memory does not just serve as a "data store" for the world knowledge. More importantly, it constitutes a suitable representation for a control mechanism deciding which properties need to be measured for different objects in order to fulfill a given task. A suited flexible and general memory was proposed in Röhrbein et al. [30]. This memory is capable of representing and operating on large object ontologies. Even though we base our work on this graph-based structure, we present major

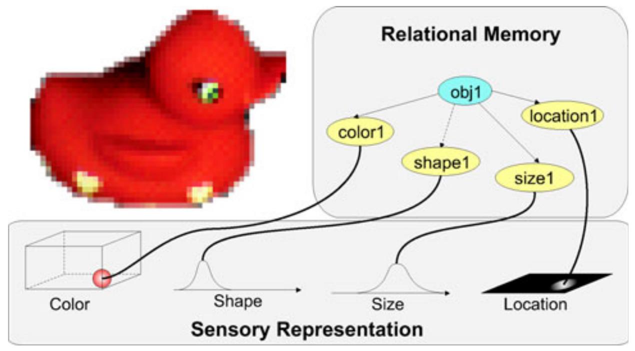

Fig. 5 A duck is represented in the memory graph of the system. Internally its properties are described by nodes, connected by edges that express a "hasProperty" relation. Each node stores a link to the sensory representation

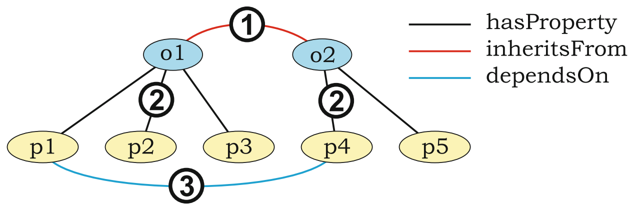

Fig. 6 The relation between the objects o_1 and o_2 (1) denotes an inheritance of information between the two objects. Edges between object o_1 and the properties $p_1 \ldots p_3$ as well as between object o_2 and the properties p_4, p_5 (2) specify visual properties of those objects. Interdependencies of the processing in the system are expressed as edges between e.g. the property p_1 and p_4 (3). For further explanations of these relations, please refer to the text

improvements like representation of procedural knowledge and the memory architecture itself in this paper.

The basic structure of the memory is a graph. Figure 5 shows an exemplary representation of an object in the graph structure of the memory. In Fig. 5, the graph structure on the right-hand side is the system's internal representation of the object on the left. Nodes in the graph represent both properties and the object itself. Edges connecting the nodes specify a "hasProperty" relation between the object and its properties. Importantly, the property nodes are anchored in the sensory representations as shown in Fig. 5. That is, each property node stores the sensory value or distribution it stands for. As an example, the object obj1 in Fig. 5 has a property color1. The color node itself stores a Gaussian distribution of the object's color, in this case the color red (see Fig. 5). In a more formal way, the memory is a graph $G = (V, E)$ where V is the set of vertices or nodes and E is the set of edges connecting these nodes. The nodes V can be further separated into object nodes O and property nodes P where $V = O \cup P$. The sensory information is stored in the property nodes. An object node is then interpreted as a combination of its properties. In this interpretation, the nodes carry the information and the edges represent relations between the different nodes and thus between the different information. It is not only possible to represent a single relation between nodes, but the proposed memory architecture is capable of representing an arbitrary number of relations. That is, we can express relations between objects, between an object and property nodes and between property nodes. These relations are described in more detail below.

Relations

The role of a node is determined by the incoming and outgoing edges, i.e. its graph structure, rather than by an artificial definition. In the same way, we do not define the meaning of an edge, but we rather define a pattern in the memory and its algorithmic interpretation. Here, the edges are bidirectional and implemented as two oppositely directed edges. When applying this schema consequently, we can interpret the memory structure as a bunch of interconnected graphs (one for each relation or edge type). This interpretation allows us to apply well-known graph algorithms. The relations currently used in the system can be seen in Fig. 6. In the following, we define relations between nodes that are necessary to represent both the semantic and procedural knowledge of the system.

Representing Knowledge

The probably most common relation is the "hasProperty" relation. As edges are bidirectional, there also exists an edge into the other direction called "isPropertyOf". Using this relation, properties can be assigned to an object. For example, if we want to express that an object o has a certain color p_{color}, we generate an object node $o \in O$ and a property node $p_{\text{color}} \in P$. Then we define an edge between o and p_{color} as

$$o \xrightarrow{\text{hasProperty}} p_{\text{color}}. \tag{15}$$

At the same time also a relation in the inverse direction called

$$p_{\text{color}} \xrightarrow{\text{isPropertyOf}} o \tag{16}$$

will be generated. In the following examples, only the definition in one direction will be mentioned for readability. However, an edge for the inverse direction is always created simultaneously. As it is common in graphs, one object can have arbitrarily many properties and a property can belong to arbitrarily many objects.

The second type of relation that should be described in more detail here is the "inheritsFrom" and the corresponding inverse "specializesTo" relation. This relation can be established both between two objects or between two properties. When defining the relation $o_1 \xrightarrow{\text{specializesTo}} o_2$, object o_2 will have exactly the same structure and properties as o_1. The same is true for a pair of properties p_1, p_2, which then share the same values and relations. However, based on these inherited properties or values, specializations can take place. That is, inherited properties can be removed or overwritten, new properties can be added or values can be modified. Here is an example: imagine an object o_1 with the color property p_{color} attached ($o_1 \xrightarrow{\text{hasProperty}} p_{\text{color}}$) and an object o_2 without property nodes attached. When writing

$$o_1 \xrightarrow{\text{specializesTo}} o_2, \tag{17}$$

o_2 inherits the color of o_1 and is also assumed to have the color p_{color}. As soon as we attach another property node like the size p_{size} to o_2

$$o_2 \xrightarrow{\text{hasProperty}} p_{\text{size}}, \tag{18}$$

the second object is specialized. That is, it inherits all properties from o_1 but defines additional or modified properties.

Additional to the edges in the graph, property nodes store a link to sensory representation. This is necessary to avoid the so-called "grounding" problem found in classical AI approaches. Here, the difficulty is to relate an abstract sensory node like "red" to a sensory experience of the system, as this information is not stored in the node itself. Contrary to this, by keeping a link to the sensory representation in the property nodes, we do not introduce an artificial abstraction between the sensory side and the memory. Furthermore, the resulting anchoring of the node allows for an easy conversion of the node information back to the sensory representation. This is very important when modulating input features of visual routines as described in "System Architecture". To avoid an excessive usage of memory by storing very detailed sensory representations in the property nodes, we approximate the representations using a k-means clustering. To summarize, by storing a link to the sensory representation, we anchor the property nodes in these representations and thus avoid the "grounding" problem classical AI approaches face. Furthermore, the measurement of an object property can be triggered using the connection of the property node to the actual visual routine (through the sensory representation).

Representing Functional Aspects

Another relation type, the relation "dependsOn" and its inverse relation "influencedBy", accounts for technical aspects of the system and the sequential processing. This relation represents knowledge about the correct ordering of information acquisition processes which is crucial to keep the system functional. If for example a visual routine for acquiring the color p_{color} of an object requires information about the retinal location $p_{x,y}$ of the object, it is important to acquire the location *before* the color routine is called. In current state-of-the-art systems, this is ensured by manually constructing processing pipelines in a static processing regime. By doing so, the whole processing apparatus including e.g. saliency, segmentation, and classification is executed even though for example only the color of an object needs to be determined. The ordering of the visual routines becomes even more important when breaking this static processing as done in this paper. When defining the relation

$$p_1 \xrightarrow{\text{dependsOn}} p_2, \tag{19}$$

we express that the processing of p_1 can only take place if p_2 has already been processed. Currently, the direct dependencies between the different property nodes are defined manually. We will see later that by using these direct definitions a canonical dependency graph can be created automatically. The problem here is very similar to problems found in compiler construction. Even though Ballance et al. state in their paper that "neither switches nor control dependence are required for a demand driven interpretation" Ballance et al. [5, p. 261], some modifiers for dependency relations are required to account for the algorithmic needs of visual routines. Figure 7 shows those required modifiers.

They cover the cases:

a) The operation of node C is optional and not absolutely required for measuring node A, but would improve the result of the measurement. For example a spatial modulation map could constrain the search space for an object, but is not mandatory. If the map is not available, the whole space has to be searched for the object (see Fig. 7a). The type of a dependency is then $type(e \in E_{\text{dependsOn}}) = \{\text{“}optional\text{”}, \text{“}mandatory\text{“}\}$.
b) Property nodes can execute two operations. Each property node can potentially send its information to another node/sensory representation or receive information from another node/sensory representation. The

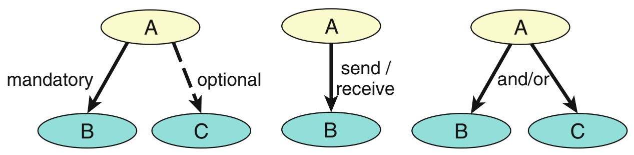

Fig. 7 Different modifiers are required for dependency relations: **a** dependencies can be optional or mandatory, **b** target nodes can execute send and receive operations and **c** all dependencies can be required to be fulfilled or only one dependency needs to be fulfilled

modifier shown in Fig. 7b specifies which operation node B has to execute in order to fulfill the dependency of node A. The operation requested by a dependency is then *operation*($e \in E_{\text{dependsOn}}$) = {“*send*”, “*receive*“}, corresponding to the modulation and measurement of sensory representation by visual routines.

c) There might be alternative ways to measure a certain property, so the system needs to fulfill only one of several dependencies. Think of different segmentation algorithms for estimating the shape of an object, where only one of those algorithms is required to get a shape (see Fig. 7c). A dependency can take one of the following logical modes mode($e \in E_{\text{dependsOn}}$) = {“*and*”, “*or*”}. If no mode is given, the “and” mode is assumed by default.

Based on the dependency relations along with the modifiers, we are able to model functional dependencies between property nodes and thus also between the visual routines bound to those nodes. This distinguishes the memory we use in this paper from other architectures that do not explicitly model relations between processing modules.

Architecture

Using the relational memory described above, we define a memory architecture. This architecture consists of four parts as shown in Fig. 8. At the bottom of Fig. 8, the prototypical structure of the memory is shown. In this part, we define the object structure as “hasProperty” relation patterns between property nodes P and object nodes O. In the example shown in Fig. 8 an object can have the properties shape, color, mask, weight, etc.

We then construct the sensory prototype layer, which *inherits* the structural definition of the layer below as denoted by “inheritsFrom” edge in Fig. 8. By doing so, the relations between the property nodes P and the object nodes O are transferred to the sensory prototype layer without the need to redefine them. Additionally, relations between nodes and the sensory representations are defined, “binding” those nodes to their corresponding sensory representations. The sensory representations themselves are bound to the visual routines that compute the sensory values. That way, we also bind the property nodes to the visual routines of the system. Furthermore, dependency relations between property nodes P are defined in the sensory prototype layer (shown as red edges in Fig. 8) to represent the measurement process as described in “Relations“. By binding the property nodes to the sensory representations, the system can use the knowledge about its own measurement apparatus to estimate costs and information gain of different visual actions.

The long-term memory then *inherits* both the sensory binding along with the dependency structure and the structural definition of the sensory prototype layer. Note that the structural definition of the sensory prototype layer is itself inherited from the structural prototype layer. Contrary to the layers described before, in the long-term memory multiple instances of the prototypical object definition are created. Each of these objects is a specialization of the prototypical one and thus has individual properties bound to the object. These individual property nodes represent for example the color or the shape of an object found

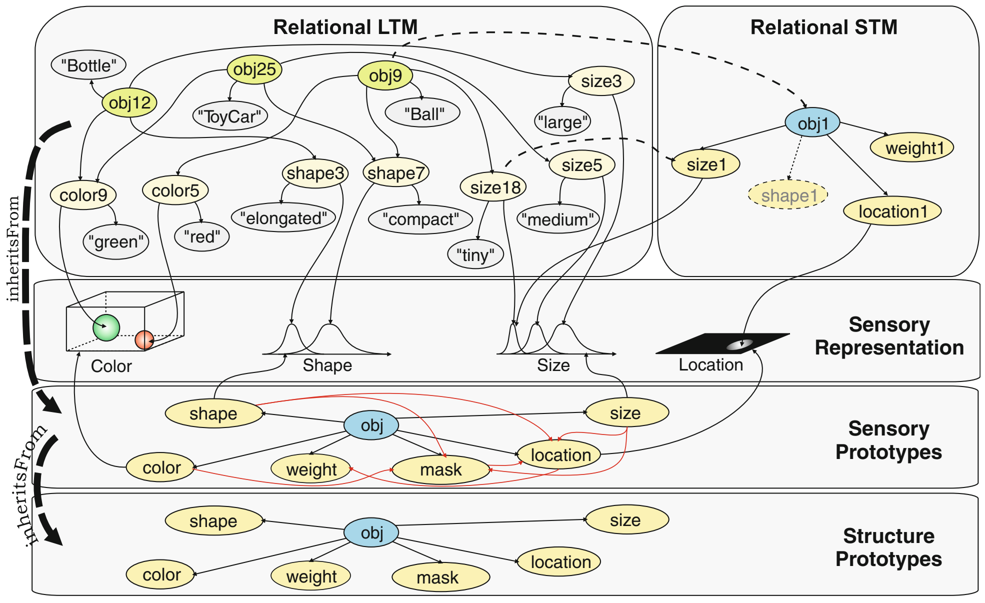

Fig. 8 *Top row*: The long-term and short-term memory store world knowledge and scene information respectively. *Second row*: Sensory information computed by the visual routines. *Third row*: Sensory Prototypes that define the binding of the object property nodes to the corresponding sensor. Furthermore, dependencies between property computations are defined (*red edges*). Objects in the long- and short-term memory inherit information from this layer. *Last row*: The structure prototype definitions express the relations between an object and its properties. (Color figure online)

in the visual world. Looking at Fig. 8, we see that only persistent properties are attached to the object node in the long-term memory. Volatile properties like the objects position are not stored. Furthermore, we see that the nodes in the long-term memory do not contain "names". This illustrates that property nodes and object nodes are only anchored in the sensory representations and are not defined by a label. However, we merge those labels (gray nodes in Fig. 8) in the following illustrations to increase the lucidity and allow for a more intuitive use of the system by humans.

The fourth and last part of the memory architecture is the short-term memory. This part of the system stores knowledge about the current scene. While doing so, information can be *inherited* both from the sensory prototype layer and the long-term memory. If an object is not yet known or identified, the object inherits its information from the sensory prototype object. As soon as the object is identified, the object in the short-term memory inherits information from the long-term memory. By doing so, the system is able to predict properties that are not yet measured. Here, the identical structure of an object in both the long-term and short-term memory eases the transfer of information between these two memory instances. Figure 8 shows an example, where obj1 in the short-term memory is identified to correspond to obj9 of the long-term memory (dashed line in Fig. 8). That is, an object (obj1) was found in the current scene, having similar properties to a known object (obj9) in the long-term memory. As illustrated by the solid outlines of the nodes, size, location and saliency weights of obj1 are measured or sent respectively. However, the shape of obj1 was not yet measured, but can be predicted from the knowledge about obj9. We can use this mechanism to establish a prediction-confirmation loop, where the long-term memory provides the predictions and the measurement apparatus tries to confirm the properties using the visual routines.

To summarize, we propose an architecture that uses multiple layers to separate the structural, procedural, semantic, and sensory definition of an object. A similar separation into procedural (skills) and semantic knowledge is also proposed in Langley et al., 2009 , which uses a rule-based representation. However, this approach misses an anchoring of the object properties in the sensory representation. By splitting the memory architecture, it can be easily adapted to different systems or underlying algorithms by only changing the sensory layer. Here, the structural definition accounts for the sensor pathways and the variety of information available in a system. The sensory definition accounts for the concrete implementation of a sensory pathway and the underlying algorithms. Additionally, the dependencies between different visual routines and thus the dependencies between the underlying algorithms is consistently defined in the sensory prototype layer. Both the long-term and the short-term memory inherit from the sensory prototype layer, which leads to an identical object structure in both parts. This identical structure later eases the transfer of information between the two memory instances. By doing so, a prediction-confirmation loop can be established where the long-term memory predicts properties of objects in the scene and the system tries to confirm these predictions using its visual routines. The question of which properties to measure when is subject of the next section.

Attention Control Mechanism

The attention control mechanism is the key aspect of our system, as it selects the elements of the scene that are perceived. For humans, this process is both selective in the spatial as well as in the feature domain according to the experiments reviewed in the related work. The process of guiding attention spatially is well researched. In this paper, we focus on the guidance of attention in the feature domain, i.e. which properties of a focused object should be measured. Along with the question of which properties to measure, the question on how to organize the acquisition process arises. In this section, we will show how the system's procedural knowledge is used to organize the measurement process and determine the execution order of visual routines.

Information Flow Example

Figure 9 illustrates the process of searching for a "tiny" object and the process of searching for the "toy car". To start the search process, the searched properties or objects are activated in the long-term memory. If a property node is activated directly (as in the "tiny" object example), the visual routine bound to this property node is called on each object candidate in the image. However, for finding a specific object the procedure is more complex. In order to keep the computational and storage demand low, our goal is to find a small subset of measurements ensuring that the attended object is the searched one. This way, the amount of information that needs to be stored in the short-term memory as well as the computation time are kept low. As mentioned earlier, the approach we propose in this paper uses the system's long-term memory to determine characteristic properties of the searched object. Please note that the discriminative power of a certain property strongly depends on concurrently active object hypotheses. In Fig. 9, the "toy car" object has attached the property nodes "green", "medium" and "compact" (see red arrows in Fig. 9). To search for the object, first a suited object candidate needs to be found in the current scene. The saliency computation is triggered, using the properties of the searched object like color and size to bias the input features.

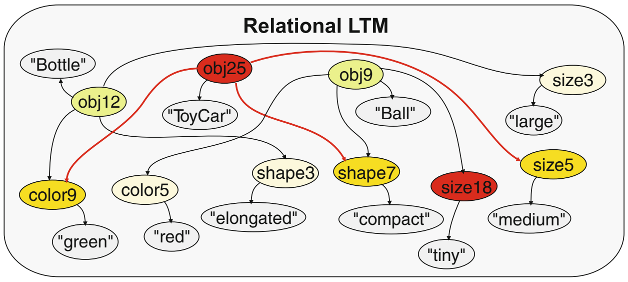

Fig. 9 The system searches for a "tiny" object and the "toy car", triggered by the activation of the corresponding nodes in the long-term memory (nodes *marked red*). (Color figure online)

After finding an object candidate, the system must decide on which property it wants to focus. Given that it actually measures the color "green", there are two valid hypotheses ("bottle" and "toy car"), for the size "medium" only one hypothesis remains ("toy car") and for the shape "compact" two hypotheses ("ball" and "toy car") remain (see Fig. 9). So the information gain is highest for the size measurement, as it reduces the set of possible interpretations to only one. In other examples, the measurement of more than one property is necessary to solve the task. Nevertheless, this example shows the principle of the algorithm: Find the property of the searched object o_s that minimizes the set of remaining interpretations or hypotheses for the attended location. If more than one measurement minimizes the set of hypotheses, the cost of a certain measurement (currently the computation time) is used to decide which measurement to execute first.

Dependency Resolving

The memory we propose is able to represent functional dependencies and modifiers that account for algorithmic needs of the visual routines bound to property nodes. However, in memory, only direct dependencies are modeled. To trigger arbitrary visual routines on demand, the overall information acquisition process needs to be organized. By parsing the direct dependencies stored in memory, an unrolled dependency graph as shown in Fig. 10 is constructed. The unrolling can be achieved by applying a depth-first search algorithm [32] on the graph constructed using the direct dependency definitions. However, this algorithm assumes that the parsed graph is a tree, i.e. it does not contain cycles. For doing so, we introduce a node state marking a node as either "resolving", "visited", "invalid data" or "valid data". The first state means that the node is currently resolving its dependencies. The second state means that the resolving process of the node's dependencies is finished. The third state means that the node has not been visited yet and does not contain up-to-date data. The last state means that the node has not been visited yet, but its data are up-to-date, which is required to reuse already acquired data (state "valid data"). When looking at the edges two, three and four of Fig. 10, one can see that these relations form a circular dependency, as receiving the retinal location depends on sending the spatial modulation map (2), which depends on receiving the object mask first (3). However, receiving the object mask in turn depends on receiving the retinal location (4). This circular dependency violates the tree assumption that was made to be able to unroll the dependency graph. The circular dependency can be detected by marking the nodes as "resolving" when entering them. As soon as the operation on the node is completed, the state is changed to "visited". When entering a node which is already in the state "resolving", a circular dependency is detected. Once detected, the circular dependency needs to be removed to "treeify" the graph. This is possible if one of the edges leading to the circle is cut. But which edge can be cut without breaking the underlying algorithms? Here, the mandatory and optional modifiers introduced earlier come into play. Edge number two of the graph shown in Fig. 10 is optional, denoting that this dependency is not absolutely necessary for the underlying algorithm. By cutting the graph at edge two, the underlying algorithm is still functional, the circular dependency is resolved and the graph is converted to a tree. The depth-first search algorithm can be used. If no optional edge exists, the dependency resolving fails. However, typically this case does not exist as it would imply an iterative algorithm without initial condition.

By combining the depth-first search and the cyclic dependency handling, the following update procedure for nodes applies. As the resolving is a recursive problem, we have chosen a recursive algorithm working on the subgraph $G_{depend} = (P, E_{\text{dependsOn}})$. Here, the property nodes P are $P \quad \in V$ and the dependency edges $E_{\text{dependsOn}}$ are

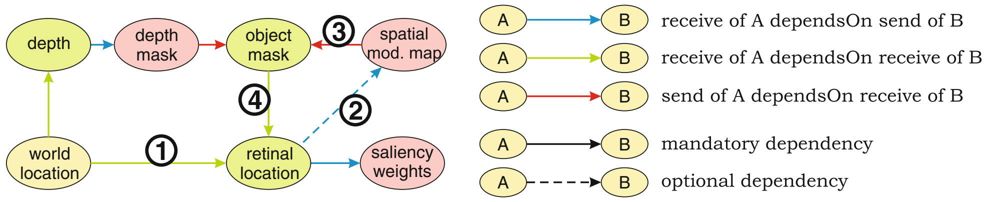

Fig. 10 Unrolled dependency graph for receiving the three dimensional location of an object. Only "dependsOn" relations are shown here. *Dashed lines* are optional dependencies, nodes marked in *red* and *green* need to perform a send or receive operation, respectively. Edge 4 leads to a circular dependency. (Color figure online)

$E_{\text{dependsOn}} \subseteq E$. When triggering the measurement of a certain property node the following update procedure is executed:

```
Procedure UpdateNodeValue:
  (a) Check the ability of the node to run
    (1) Check the current node $p_{\text{self}} \in P$ for valid data state$(p_{\text{self}}) \overset{?}{=}$ "valid data". If $p_{\text{self}}$ already has valid data, skip any operation and return success.
    (2) Check if the current node $p_{\text{self}}$ was already visited and thus indicates a cyclic dependency state$(p_{\text{self}}) \overset{?}{=}$ "resolving". If a cyclic dependency is detected, return the corresponding error.
    (3) Set the state for the current node $p_{\text{self}}$ to "resolving" state$(p_{\text{self}}) =$ "resolving".
  (b) Updating dependencies
    (1) Get the list with all dependencies for the current node $P_{\text{depend}} = \{p \in P | (p_{self}, p) \in E_{\text{dependsOn}}\}$.
    (2) For each dependency (child node) $p_c \in P_{\text{depend}}$ do:
      (2.1) Call the update procedure on the child node UpdateNodeValue$(p_c)$.
      (2.2) If the return code contains a cyclic dependency error and the dependency is mandatory type$((p_{\text{self}}, p_c) \in E_{\text{dependsOn}}) \overset{?}{=}$ "mandatory", return the received error.
      (2.3) Remove the child node from the dependency list $P_{\text{depend}} = P_{\text{depend}} \backslash p_c$ and process the next dependency in the list $P_{\text{depend}}$.
  (c) Execute current node's operation
    (1) call the requested operation operation$((p_{\text{self}}, p_c) \in E_{\text{dependsOn}})$ on the current node $p_{\text{self}}$. If a receive operation was requested operation$((p_{\text{vself}}, p_c) \in E_{\text{dependsOn}}) \overset{?}{=}$ "receive", store the sensor data locally.
    (2) Mark the data of the current node as valid state$(p_{\text{self}}) =$ "valid data'".
    (3) Return success.
```

In the pseudo-code algorithm, the node's state is set to "valid data" when the data of the node is updated. This is done to reduce the computational demand and reuse as much information as possible. When entering a node with the state "valid data", the dependency graph shrinks as the subgraph of this node is not processed. The ability of dynamic pruning distinguishes the algorithm proposed here from similar approaches used in the domain of compiler construction such as [12] and Ballance et al. [5] or the rule-based approach proposed in Langley et al. [22].

Property Selection

As mentioned above, the basic principle of selecting a property is to execute the measurement that reduces the set of remaining hypotheses most. We now mathematically formulate this principle. Our algorithm works on a subgraph of the memory $G_{\text{scheduling}} = (V, E_{\text{hasProperty}})$ consisting of the object nodes O, property nodes P with $V = O \cup P$ and "hasProperty" edges $E_{\text{hasProperty}} \subseteq E$. The measured value $\mathbf{v}$ is then compared to the value $\mathbf{c}_i = \text{value}(p_t)$ of all property nodes $P_t = \{p \in P | \exists (p, p_{\text{type}}) \in E_{\text{inheritsFrom}}\}$ in memory having the same type $p_{\text{type}} = \{p \in P | \exists (p_i, p) \in E_{\text{inheritsFrom}}\}$ as the measured property p_i, in this example the type is size. The following function is used to determine the node activation $a(p_t)$

$$\mathbf{c}_i(p_t) = \text{value}(p_t) \tag{20}$$

$$\hat{a}(p_t) = e^{-\frac{1}{2\sigma^2}\|\mathbf{c}_i(p_t) - \mathbf{v}\|^2} \tag{21}$$

$$a(p_t) = \begin{cases} 0, & \text{for } \hat{a}(p_t) < \xi \\ \hat{a}(p_t), & \text{else} \end{cases}, \forall p_t \in P_t, \tag{22}$$

where ξ is the threshold for the activation. After resetting all activations in memory to zero, the scheduling of the visual routines works as follows:

```
Procedure SearchObject:
  (a) Locate an object candidate
    (1) Activate the searched object $o_s \in O$ and collect its attached properties $P_s = \{p \in P | (o_s, p) \in E_{\text{hasProperty}}\}$.
    (2) Locate an object candidate $o_c$ using the saliency map and initialize the set of remaining hypotheses to all objects $O_r = O$.
  (b) Schedule visual routines
    (1) While $P_s \neq \emptyset$:
      (1.1) Find all remaining competing object hypotheses $O_h$ sharing properties with the searched object $O_h = \{o \in O_r | \exists p \in P_s : (o, p) \in E_{\text{hasProperty}}\}$.
      (1.2) Calculate the discriminative power $d_i = |D_i|^{-1}$ against the remaining hypotheses where $D_i = \{o \in O_h \mid \exists (o, p_i) \in E_{\text{hasProperty}}\}$, $\forall p_i \in P_s$.
      (1.3) Trigger the visual routine on the object candidate $o_c$ for the most discriminative property $p_i : d_i \leq d_j \forall j$ by calling UpdateNodeValue$(p_i)$. If multiple properties minimize the set, select the one with the least cost. Note that at this point, the dependency parsing described in "Dependency Resolving" is used. Remove the selected property from the set $P_s = P_s \backslash p_i$.
      (1.4) Find all property nodes $P_t = \{p \in P | \exists (p, p_{\text{type}}) \in E_{\text{inheritsFrom}}\}$ in memory having the same time $p_{\text{type}} = \{p \in P \mid \exists (p_i, p) \in E_{\text{inheritsFrom}}\}$ as the measured property $p_i$.
      (1.5) Calculate the activation $a(p_t \in P_t)$ for all property nodes of the same kind $p$type (e.g. size) using the activation function shown in Eq. 22.
      (1.6) Propagate the activation $a$ of all activated property nodes $p_m$: $a(p_m) \geq \xi$ to the attached objects $O_m = \{o \in O | \exists (o, p_m) \in E_{\text{hasProperty}}\}$ by calculating $a(O_m)t = a(O_m)_{t-1} \cdot a(p_m)$. Calculate the remaining objects $O_r = O_h \cap O_m : a(O_m)_t \geq \xi$. If the searched object is rejected $o_s \notin O_r$, locate the next object candidate (step (a)). If $O_r = \{o_s\}$, the object is found. Continue with step (c). Otherwise continue with the next property as the object candidate is still ambiguous.
  (c) Link the object to the long-term memory
    (1) If the object $o_s$ was found, link it to the remaining hypothesis $o_r$: $o_r \overset{\text{specializesTo}}{\longrightarrow} o_s$.
```

It is obvious that the number of required measurements strongly depends on the task, the searched object itself, the knowledge stored in the long-term memory and the scene. However, in a worst-case scenario the system measures all available properties of an object, which is equivalent to the behavior of state-of-the-art vision systems.

Results

In this section, we evaluate the proposed attention control mechanism including the scheduling of visual actions experimentally. To do so, we give different tasks to the system, while investigating the internal decision processes concerning which properties should be measured when. Finally, we evaluate the system performance by comparing our system to the state-of-the-art system behavior with respect to savings in memory load and number of required measurements to solve a given task.

Task: Find the Cube

The first task we give to the system is to find the "cube" in the image. Now the system has to find out which property to measure in which temporal order by employing the algorithm proposed in the previous section. To start the search process, we activate the cube (object_36) in the long-term memory as shown on the left hand side of Fig. 11. The activated object is marked red, while the attached properties are marked yellow. The attention control mechanism in each step determines which property reduces the set of hypotheses for the focused object most. On the right hand side of Fig. 11 the number of remaining object hypotheses (top) and the combined cost-gain measure (bottom) is shown as determined by the system. Based on the cost-gain measure (bottom), the minimum is selected, which in this case is the color property. This is due to the fact that the system does not know any further object with the color (orange) of the cube (compare Fig. 11 left).

To measure the color, the system first has to locate an object candidate. The saliency map, the spatial modulation map and the segmentation of the focused object are shown in Fig. 12. Using the known color (orange), the system biases the color feature of the saliency map. As the saliency map and the segmentation in Fig. 12 show, the first fixation is already on the searched cube. After locating the object, the color of the focused candidate is measured. The predicted object properties, the measured color, the distribution of the color cluster activation and the activation of the objects in the long-term memory are shown in Fig. 13.

The predicted color of the cube matches the actually measured color of the object candidate very well, as can be seen in Fig. 13. The color cluster of the cube (marked green) is strongly activated (0.85), which in turn leads to an equally strong activation of the cube object (right hand side of Fig. 13). Because there is no support for any other hypothesis, the system identifies the currently focused object as the searched cube.

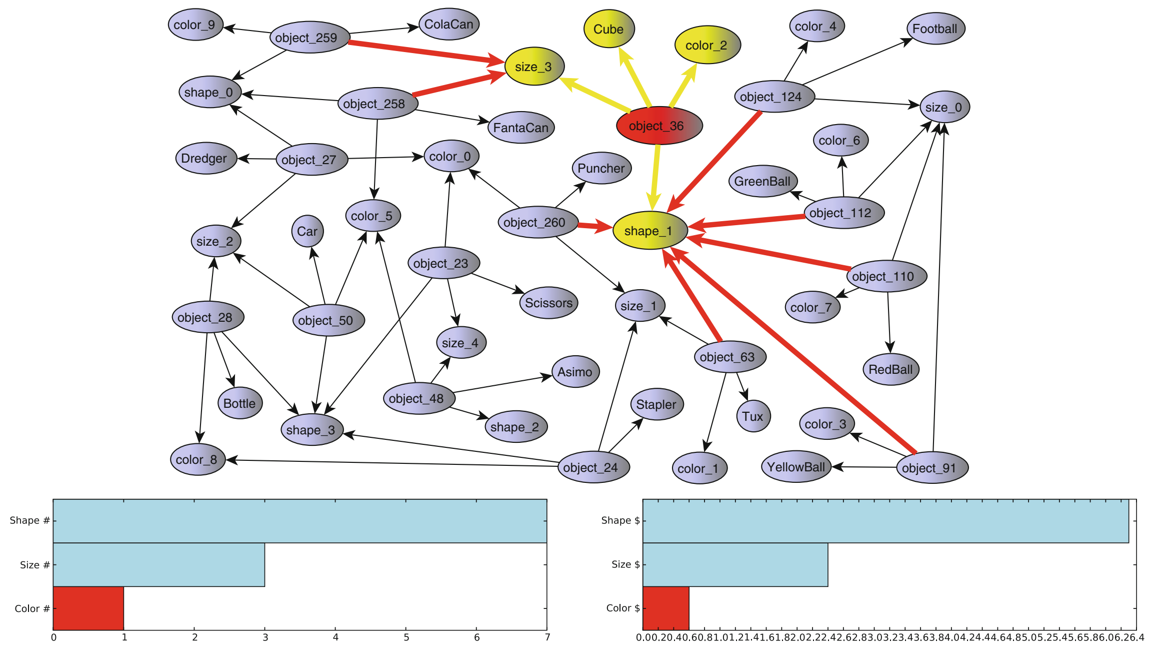

Fig. 11 Task: Find the cube. By activating the cube in memory (*red node*) the features corresponding to that object are activated (*yellow nodes*). The attention control mechanism determines the number of attached nodes (*upper right*) and thus the costs of the measurement (*lower right*) for each property. The property with the least cost (*red bar*), in this case color, is selected to be measured first. (Color figure online)

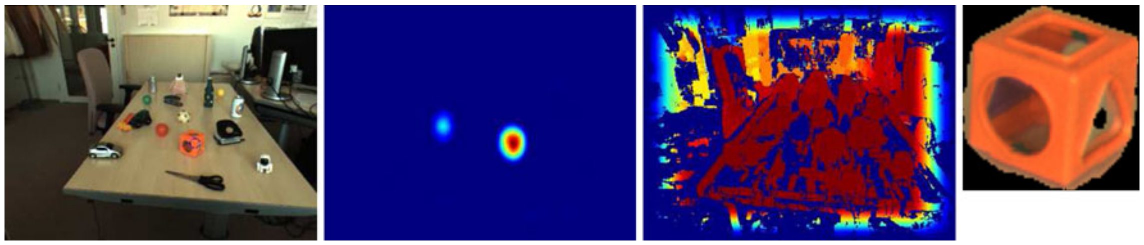

Fig. 12 Task: Find the cube. The input image (*first column*) is biased using the color of the cube known to the system and stored in the long-term memory. Together with the spatial modulation map (*third column*), the saliency map (*second column*) is calculated. The segmentation of the object is shown in the last column

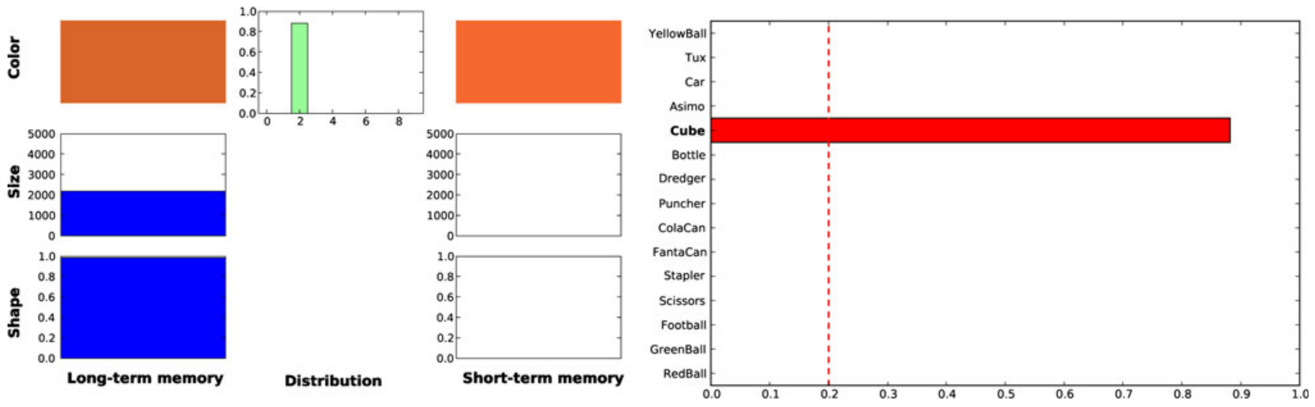

Fig. 13 Task: Find the cube. On the *left*, the predicted properties of the cube as stored in the long-term memory, the activation distribution in the different property clusters and the actually measured properties are shown. The measured color activates only the predicted color cluster. On the *right*: the remaining color cluster only activates the cube in the long-term memory because it is the only one that is known to be *orange*. Thus the system assumes that the currently focused object is the cube. (Color figure online)

Task: Find the Bottle

In the previous task the object could be identified using a single property. To increase the difficulty for the system in this experiment a search of the bottle is requested, which is not identifiable with a single property. As Fig. 14 shows, at least two hypotheses remain for each single property. Again, the searched object is marked red, the corresponding property nodes are marked yellow. When we look at the long-term memory, we see that at least two hypotheses remain, so the system needs to perform at least two measurements to disambiguate the focused object candidate. However, the system first selects the property that minimizes the set of remaining hypotheses. In this case, again the color is the best property to start with, as only two hypotheses will remain, given that the focused object candidate belongs to the predicted color cluster. The measurement process starts again with localizing a suited object candidate in the visual scene. Figure 15 shows the scanpath of the system. Due to multiple peaks in the saliency map, a “wrong” object is focused first (blue cross in Fig. 15). Please note that this object (part of the table) is not even known to the system, i.e no object with corresponding properties exists in the long-term memory. Nevertheless, the system measures the color of the object candidate. The measured color matches the predicted color and five object hypotheses remain (not shown). This is an example that shows that the predicted number of hypotheses not always matches the actual number of remaining hypotheses. To disambiguate, the system selects another property to be measured for the object candidate. According to Fig. 14, the physical size is the property that is supposed to reduce the set of remaining hypotheses best. The measured physical size of the object candidate exceeds the predicted physical size to a great extent. As the part of the table is unknown to the system, no size cluster exists that is activated by the measurement. In turn, all object hypotheses are rejected by the system (activation of all objects equals zero). At this point, the system rejects the currently focused object candidate and tries to locate a new candidate. Again the color feature biased by the predicted color of the searched object is measured. Additionally, all locations of previously focused objects are inhibited using the spatial modulation map. This time the system focuses on the searched object (red cross in Fig. 15) and measures the color of the object candidate. Even though this measurement activates the color cluster of the searched object most, competing object hypotheses remain. To disambiguate those, the system

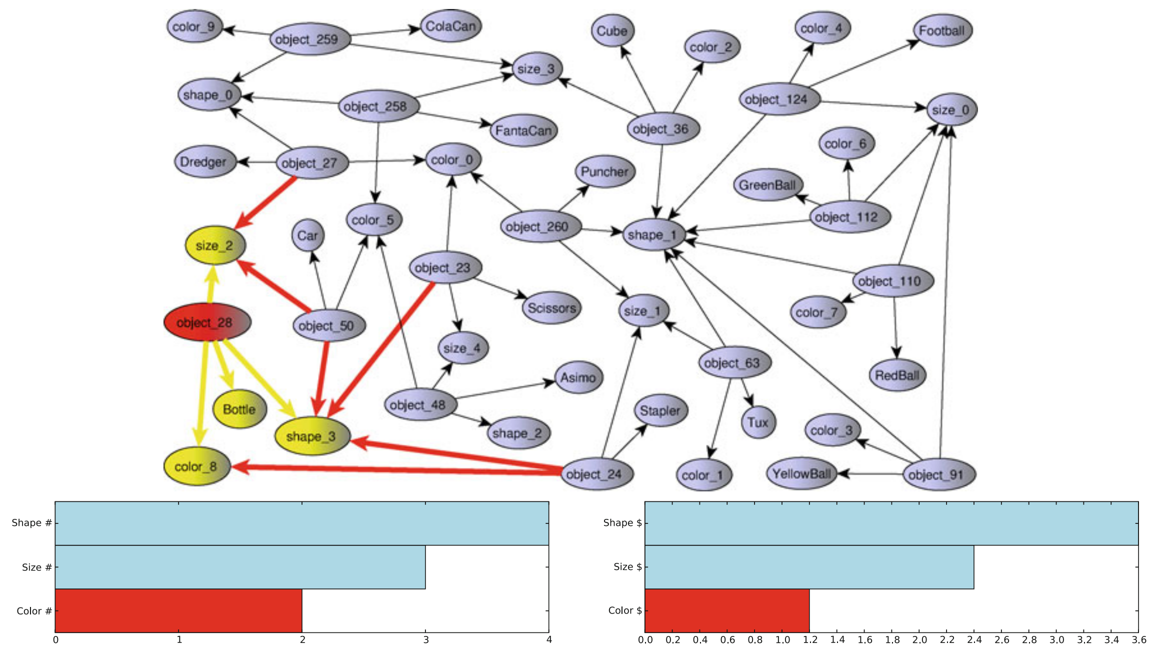

Fig. 14 Task: Find the bottle. The searched object is activated in the long-term memory (*red node*) with its corresponding features (*yellow nodes*). The attention control mechanism determines the number of attached nodes (*upper right*) and the cost of the measurement (*lower right*) for each property. For the measurement with the least cost (*red bar*), in this case color, two hypotheses will remain, which indicates that multiple measurements are required. (Color figure online)

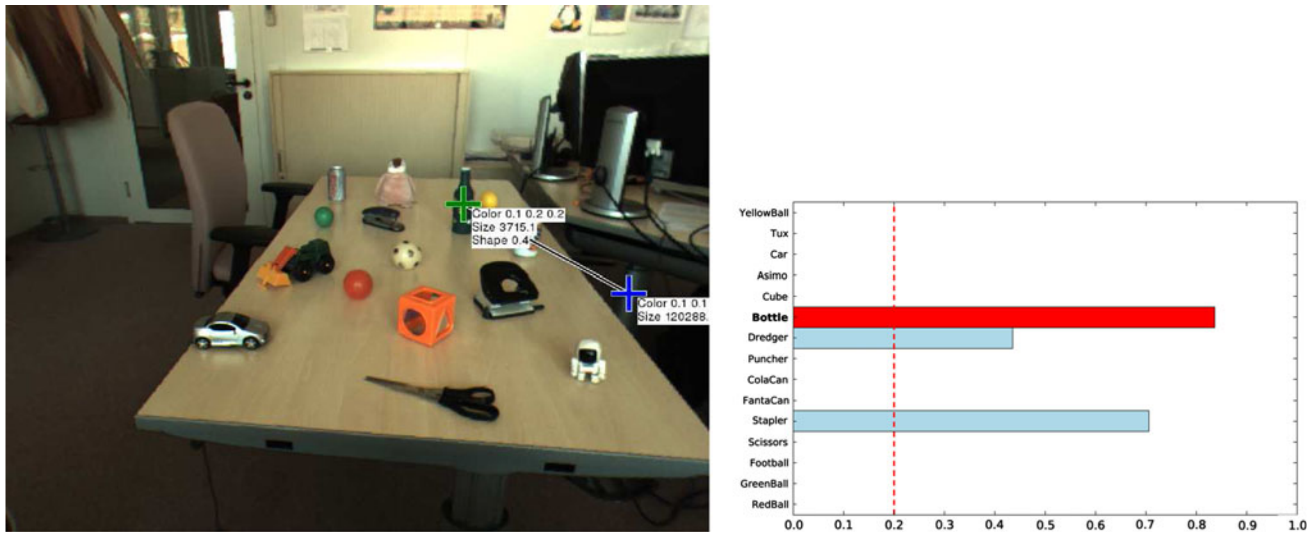

Fig. 15 Task: Find the bottle. The first fixation in the scanpath (*left*) is marked with a *blue cross*, while the final fixation is *marked green*. Due to a size mismatch, the object focused first is rejected, while the bottle is correctly detected after measuring all properties (*right*). (Color figure online)

performs a physical size measurement, which still does not lead to a clear identification of the object. As Fig. 15 shows, even after a third and last measurement (shape), multiple hypotheses remain. However, after exhausting all its possible visual actions, the system identifies the object candidate as bottle, because the bottle has the strongest activation of the remaining hypotheses.

Task: Find the Toy Robot

In the two previous examples, the color was always the most discriminating property for the searched object. When we look at the representation of the toy robot in the long-term memory it becomes clear that the color is not the most discriminating property for this object. In Fig. 16 one can

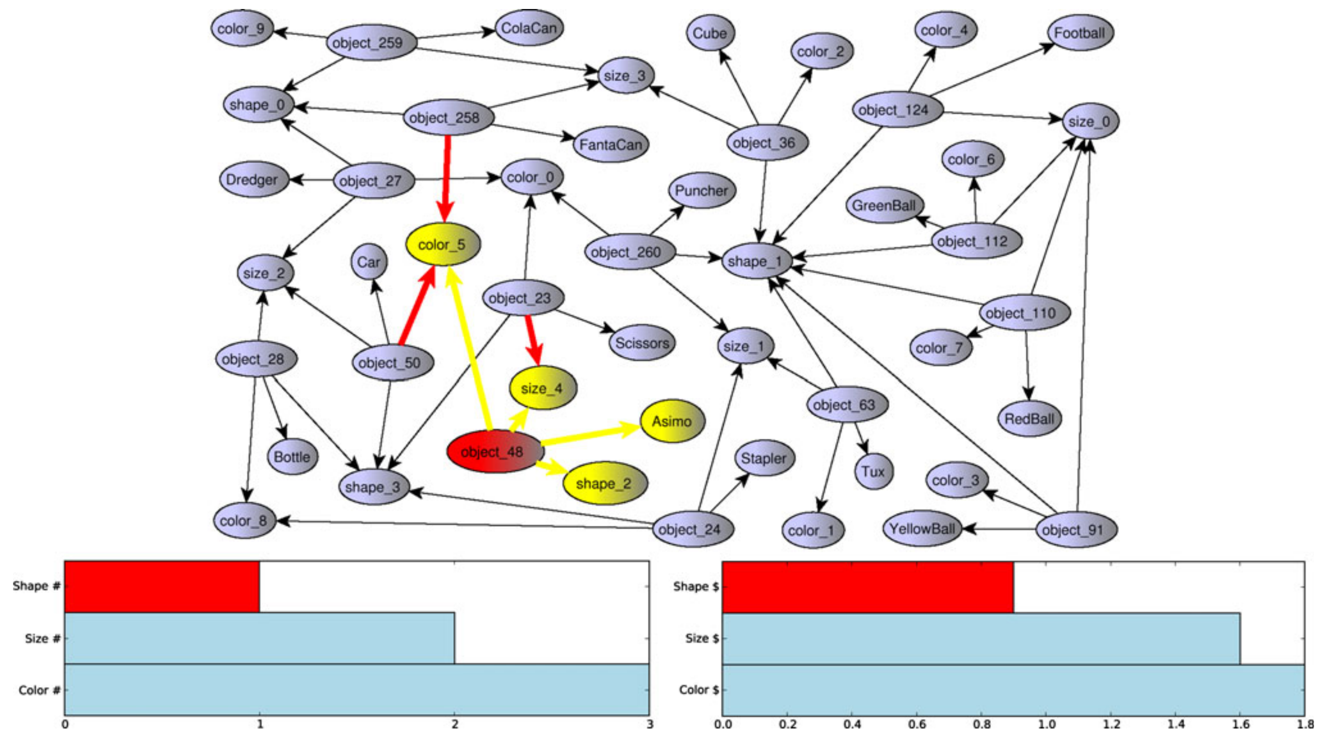

Fig. 16 Task: Find the toy robot. The searched object is activated in the long-term memory (*red node*) with its corresponding features (*yellow nodes*). The attention control mechanism determines the number of attached nodes (*upper right*) and the costs of the measurement (*lower right*) for each property. The shape is most discriminative for the toy robot (*red bar*). (Color figure online)

see that the shape is much better to identify the toy robot. Again, the searched object is marked red, the corresponding properties are marked yellow. The system schedules the measurement of the shape feature first, because only one hypothesis will remain, given that the focused object has the predicted property. The known color of the object is used to localize a suited object candidate, which is difficult as multiple “white” objects exist. According to this prediction, four fixations are necessary to locate the searched object as can be seen in Fig. 17.

At each fixation the shape of the focused object candidate is measured. For the first three fixations, the shape differs too much from the predicted shape. This leads to no activation of the shape cluster attached to the searched object and in turn, the toy robot object hypothesis does not get any support (zero activation). The system interprets this zero activation as a rejection of the object candidate and triggers the location of a new one. Finally in the forth fixation (green cross in Fig. 17), the toy robot shape cluster remains activated. However, another shape cluster also remains activated, creating the need for the system to measure another property to disambiguate the object candidate. The system decides to measure the physical size of the object next, as this is the best choice after the shape measurement according to Fig. 16. By combining the shape and the physical size measurement, the object candidate can be clearly identified as toy robot.

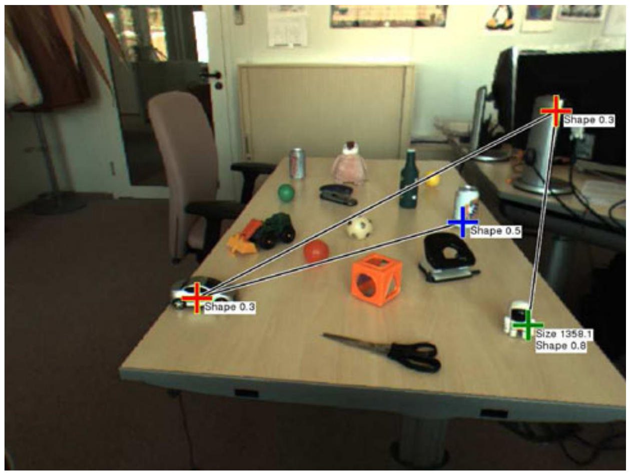

Fig. 17 Task: Find the toy robot. Four fixations are required before the searched toy robot is located (*green cross*). At each fixation the shape is measured (the measured value is shown in the image). Additionally, a physical size measurement was required to confirm that the final object candidate is the toy robot. (Color figure online)

System Performance

In this section we want to cover the overall system performance. These performance measures especially focus on the savings in terms of memory and computational load of our system compared to state-of-the-art systems. Those systems do not selectively process object properties but rather measure all those properties at each fixation point. In the following, all objects in the long-term memory are triggered to evaluate the savings in memory space and computational load. Figure 18 shows the number of nodes stored in the short-term memory for each object, which gives an overview of the memory load for each search process. The experiment is conducted for a system with and without the attention control mechanism proposed in this paper. All other respects (preprocessing, visual routines, etc) are identical. As Fig. 18 shows, the number of stored nodes and thus the memory load is reduced for all objects when employing the attention control mechanism. On average 18% less nodes need to be stored per object compared to the full measurement of all properties. The saving of memory capacity here ranges from 5.9% for the bottle up to 35.3% for the cube.

Additionally, we recorded the number of measured properties during the experiments, which reflects the computational load of the system. The results are shown in Fig. 19. The comparison shows that the number of measurements is reduced by about 44% when using the proposed attention control mechanism. Please note that all state-of-the-art systems lack such a mechanism. To ease the comparison across different objects, Fig. 20 shows the number of saved measurements normalized by the number of fixations.

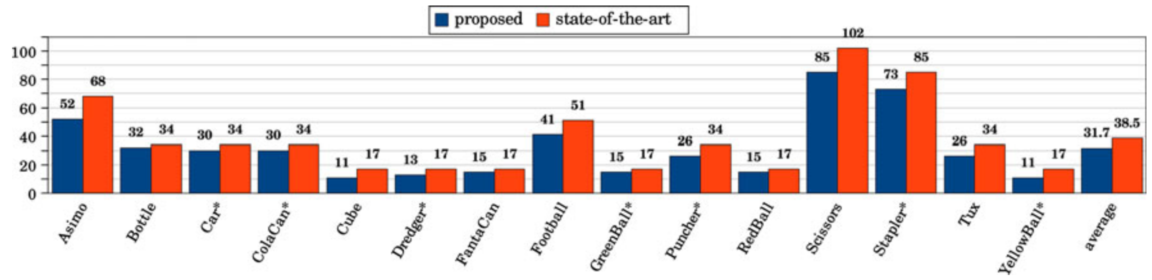

Fig. 18 This shows the number of nodes stored in the short-term memory with the attention control mechanism and without. On average, the system we propose stores about 18% less nodes compared to a system that does not schedule its visual actions but measures all properties at each fixation (*orange*). (Color figure online)

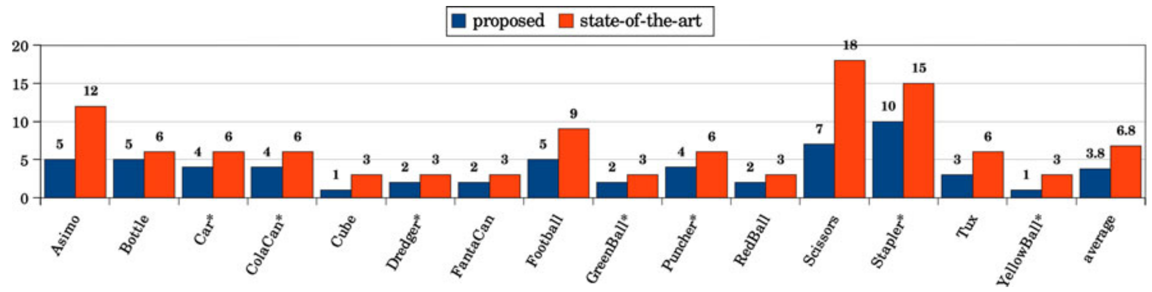

Fig. 19 This shows the number of properties (color, shape, physical size) measured for each search process. On average, our system (*blue*) measures about 44% less properties than a system that measures all properties at each fixation (*orange*). (Color figure online)

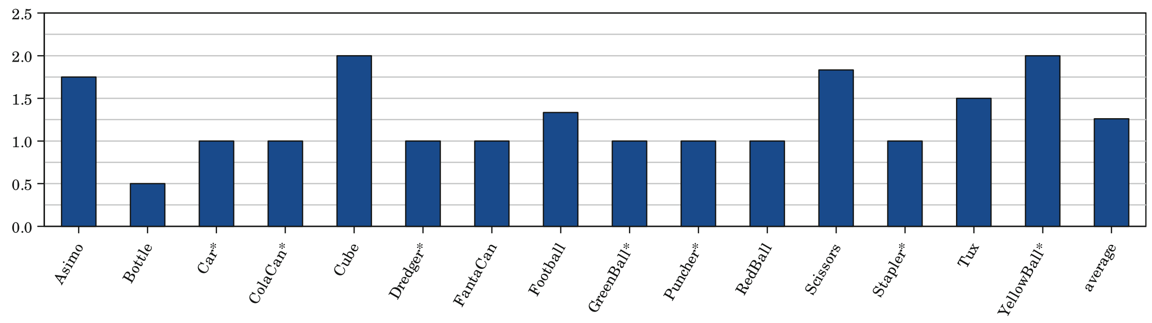

Fig. 20 This shows the reduction in measurements per fixated location when comparing our system to a state-of-the-art system that measures all three properties at each fixation. In nearly all cases at least one measurement per fixation can be saved, on average the saving is about 42%

In nearly every fixation, the system is able to save at least one measurement to decide if an object candidate should be rejected or accepted. Even for the worst case in this scene (Bottle) the average saving is 0.5 measurements per fixation. In general, the worst case scenario for the system would be no saving. For objects with discriminative properties like the cube or yellow ball (color), the toy robot (shape) or the scissors (shape) the number of saved operations is much higher. Please note that the saving also depends on the number of possible properties and the number of objects. If more properties are available the potential saving increases, while more known objects will decrease the savings. However, the gain coming from the number of properties will outbalance the loss coming from the objects because objects are usually only defined by a small subset of properties.

Discussion

In this section, we first discuss the different aspects of the work presented in this paper. We start by discussing details of the presented work like visual routines and the memory of the system concerning on the scalability of the scheduling algorithm. Afterward, we review our approach with respect to its relation to biological findings, before we eventually summarize the main contributions of our work.

Visual Routines

As stated before, the visual routines used in this paper are only very basic. In more complex environments containing e.g. light changes or low contrast these routines will probably yield to insufficient results. However, they can easily be replaced with more advanced state-of-the-art methods and will thus produce more robust results in difficult environments. Additional to the increase in robustness, the color channel should be extended to allow for multicolor objects. This can be done by e.g. additionally transferring the n-largest k-means clusters to memory. In general, the precision of the algorithms used for feature extraction determine the smallest quantization steps of the feature space. That is, it limits the maximal number of property nodes in this sensor modality and in turn has an influence on the memory capacity (see section "Memory Architecture and Scheduling"). This means, that high precision in a visual routine allows for small quantization steps leading to a high potential number of property nodes for this modality. A high number of property nodes then results in a high memory capacity (see section "Memory Architecture and Scheduling").

Due to the basic nature of the algorithms all feature extraction routines for color, size and shape strongly depend on a good segmentation. Missegmentations leading to a strong deformation of the bounding box are especially critical for the size and shape estimation. However, a robust segmentation can be achieved by using additional features (e.g. texture and depth) or object specific knowledge. This knowledge might for example consist of form priors, which mitigates the effects of multicolor objects.

Memory Architecture and Scheduling

When looking at a system's memory, one interesting question is how many distinct objects can be stored. To estimate this number for our memory architecture we need to determine the number of possible combinations of property nodes. Each object in our memory connects to one property node of each sensory pathway. In the example implemented here, one object connects to one color node, one size node and one shape node. Here, each property node is connected to its corresponding sensory prototype node, representing the different sensory pathways, with a "specializedTo" edge, e.g.

$$p_{\text{color}} \xrightarrow{\text{specializesTo}} p_{\text{red}}. \tag{23}$$

We denote the set of sensory prototype nodes with T and the set of property nodes with P. Now, the maximal number of possible connection patterns ψ can be calculated as

$$\psi = \prod_{t \in T} |\{p \in P | (t,p) \in E_{\text{specializesTo}}\}|. \tag{24}$$

The calculated number ψ also represents the maximal number of distinct objects that can be stored in memory.

The optimal set of properties the system has to measure would be the minimal set while still being able to discriminate the searched object $t \in O$ from all stored objects in memory O. The set of property nodes $S_{\max}$, which is connected to the searched object t, can be formulated as $S_{\max} = \{p \in P | \exists (t,p) \in E_{\text{hasProperty}}\}$. The degree of discrimination $d(S)$ of a property set $S \subseteq S_{\max}$ can be calculated as

$$d(S) = \frac{|\Delta(S)|}{|O \setminus t|} \tag{25}$$

with

$$\Delta(S) = \{o \in O | \exists s \in S : (o,s) \notin E_{\text{hasProperty}}\}. \tag{26}$$

The degree of discrimination $d(S)$ depends on the size of the object set O, the size of the selected property set S and the connectivity patterns in memory represented by $E_{\text{hasProperty}}$. Knowing this, we can discuss the worst-case scenario for our scheduling algorithm and the expected scalability of the memory system as a whole. Please note that for large sets of $S_{\max}$ finding the optimal subset S will be computationally expensive. Therefore, we have chosen to implement the greedy scheduling algorithm presented in this paper. However, the following estimations will still be valid.

In the worst-case for our scheduling algorithm the subset $S = S_{\max}$. In this case our system behavior (full measurement of all properties) is identical to current state-of-the-art systems. S equals $S_{\max}$ in case of two indistinguishable objects in memory and approaches $S_{\max}$ with a large number of objects that can only be distinguished by measuring all properties. If the memory contains $n = |O|$ objects, the probability $p(n)$ of two objects sharing identical connections can be calculated as

$$p(n) = 1 - \frac{\psi!}{\psi^n(\psi - n)!} \qquad (27)$$

with ψ being the maximal number of connection patterns according to Eq. 24.[1] As you can see, the probability for a collision increases with the number of objects n, but decreases with the number of property nodes as ψ increases. This also shows the scalability of our approach, as an increasing number of objects can be compensated by increasing the number of property nodes. This increase in property nodes can be achieved by a finer quantization of the sensory representation or by adding more sensory pathways. However, the quantization is limited by the algorithmic precision. The number of sensory pathways is also limited. Another possibility to reduce the collision probability is to reduce the number of competing objects by grouping objects with respect to co-occurrence, spatial context and temporal context. To implement this, the current memory architecture has to be hierarchically structured by adding further edge types.

Biological Plausibility

In the current system implementation the required features are measured sequentially. This to a certain degree interferes with findings in neurobiology, where the human brain is assumed to process data in a massively parallel manner. Even though the proposed dependency resolving mechanism is only necessary in technical systems, the actual resolving of the dependency tree can be parallelized by simultaneously resolving all subtrees and leaf nodes. Furthermore, the scheduling process of our system is not necessarily of sequential nature. Once the feature set which is required to solve the task is determined, the measurement of those features can be executed in parallel. If we furthermore assume that in the brain all features of the currently focused location are measured in parallel, we can interpret the suggested selection mechanism as an information filter. This information filter only selects the measurement for the required features and passes them to the memory system. Features that do not contribute in solving the given task are discarded. This view conforms to the findings in Corbetta et al. [10], where only task-relevant features influence the attentions process. The strong filtering results in a reduced reaction time, as the human vision system only needs to wait until all required information is measured to either reject or validate a hypothesis for the current location. In case the hypothesis is rejected, the system can saccade instantaneously to the next location. The retention time is thus reduced for locations containing objects not conforming to the current task. As a result, we here predict, that during a correlation between the number of discriminating target features and the search time will be measurable. An additional effect of attending only relevant features for a location is the reduction in required memory capacity. By only passing a few important features, information about more locations can be held in memory compared to the unfiltered case. According to [10] a low memory load is a prerequisite for a high search performance.

Summary

The goal of our work is to build-up a scene representation in a task-driven way utilizing the knowledge of the system about the world and the current scene. We propose the idea to only acquire those pieces of information about the current visual scene that are needed to solve the given task. Up to now, state-of-the-art systems do not consider this kind of selective processing in the feature space. Instead, all these systems rather focus on the selective processing of locations in the image. That is, once fixating a certain object candidate, they measure all information about this candidate regardless of the task. To selectively process information in both the feature and spatial domain as described before, flexible processing pathways are required. Therefore, we split the fixed saliency-segmentation-classification processing pipeline found in most systems into functional parts that can be flexibly combined. This is required to be able to selectively trigger visual routines for a selective acquisition of information. In turn this also requires a dynamical and flexible system architecture that allows for a demand-driven combination of processing modules. We proposed such an architecture in "System Architecture" of this paper along with highly specialized visual routines that only measure a single property of an object. To acquire more complex information, the system needs to combine those routines in a suitable way.

The combination of visual routines requires the knowledge of which routines have to be executed in order to extract certain information from the scene (e.g. segmentation before color measurement, etc). We propose to store procedural

[1] This estimation assumes an equal connection probability of an object node over all property nodes.

knowledge in the long-term memory in "Memory Architecture". This procedural knowledge is represented consistently as a graph along with the knowledge of the system about the world. We then present an algorithm that allows to parse the graph containing the procedural knowledge, enabling the system to combine visual routines in the correct temporal order to process information in a sensible way. Furthermore, the memory architecture stores a link between sensory nodes of the memory graph and the corresponding sensory representations. This anchors the nodes of the long-term memory in the sensor space, allowing for an easy biasing of visual routines in these spaces. This is also a major difference to classical AI approaches that only work on symbolic information. Beside the ability to order the visual routines temporally, the system needs to decide which properties it has to measure for solving the current task. We present an attention control mechanism in 4 that bases its decision on the long-term knowledge of the system. Here, a subset of measurements is selected that is sufficient to solve the given task. The experiments in "Results" involving visual search tasks of different complexity show that the application of the proposed algorithms leads to a reduction of both the computational and memory load compared to state-of-the art algorithms. We predict that when increasing the number of possible properties in the system, the savings will also increase. The resulting representation is task-related, as only the pieces of information to solve the task is acquired. Furthermore, the complexity of the given task in a certain scene determines the number of measurements and thus the computational load.

To summarize, the main contributions of this paper are the flexible system architecture, the graph-based memory architecture with anchored sensory nodes and the attention control mechanism. Those contributions allow to build a system that flexibly combines its processing pathways in a task- and demand-driven way. The experiments we conducted show that the selective processing of visual information beyond the spatial domain results in a considerable reduction of both computational (44%) and memory load (18%) while solving the given task. That way, a task-driven scene representation can be realized serving as a foundation for learning new objects and interpretation of the current scene.

References

1. Aloimonos Y. Active vision revisited, chapter introduction. Hillsdale: Lawrence Erlbaum Associates; 1993. p. 1–18.
2. Aloimonos Y, Weiss I, Bandyopadhyay A. Active vision. Int J Comput Vision. 1988;1(4):333–56.
3. Amari S. Dynamics of pattern formation in lateral-inhibition type neural fields. Biol Cybern. 1977;27:77–87.
4. Bajcsy R. Active perception vs. passive perception. In: Proceedings of the IEEE workshop on computer vision: representation and control. 1985. p. 55–62.
5. Ballance RA, Maccabe AB, Ottenstein KJ. The program dependence web: a representation supporting control-, data-, and demand-driven interpretation of imperative languages. In: Proceedings of the ACM SIGPLAN 90 conference on programming language design and implementation. 1990. vol. 25, p. 257–71.
6. Ballard DH. Animate vision. Artif Intell. 1991;48(1):57–86.
7. Ballard DH, Hayhoe MM, Pelz JB. Memory representations in natural tasks. Cogn Neurosci. 1995.
8. Boykov YY, Jolly MP. Interactive graph cuts for optimal boundary &amp; region segmentation of objects in n-d images. In: Proceedings of the 8th IEEE international conference on computer vision. 2001, vol. 1, p. 105–12.
9. Chun MM, Nakayama K. On the functional role of implicit visual memory for the adaptive deployment of attention across scenes. Visual Cogn. 2000;7(1):65–81.
10. Corbetta M, Patel G, Shulman GL. The reorienting system of the human brain: from environment to theory of mind. Neuron. 2008;58(3):306–24. pdf.
11. Eggert J, Rebhan S, Körner E (2007) First steps towards an intentional vision system. In: Proceedings of the 5th international conference on computer vision systems.
12. Ferrante J, Ottenstein K, Warren JD. The program dependence graph and its use in optimization. ACM Trans Program Lang Syst. 1987;9(3):319–49.
13. Frintrop S, Backer G, Rome E. Goal-directed search with a top–down modulated computational attention system. In: Proceedings of the 27th annual meeting of the German association for pattern recognition, vol. 3663 of *LNCS*. 2005. p. 117–24.
14. Frintrop S, Rome E, Christensen HI. Computational visual attention systems and their cognitive foundations: a survey. ACM Trans Appl Percep. 2010;7(1):1–46.
15. Hamker FH. Visuelle Aufmerksamkeit und lebenslanges Lernen im Wahrnehmungs-Handlungs-Zyklus. PhD thesis, Technical University of Ilmenau. 1998.
16. Hamker FH. The emergence of attention by population-based inference and its role in distributed processing and cognitive control of vision. Comput Vision Image Understand. 2005;100: 64–106.
17. Hayhoe M. Vision using routines: a functional account of vision. Visual Cogn. 2000;(7):43–64.
18. Hayhoe M, Bensinger D, Ballard D. Task constraints in visual working memory. Vision Res. 1998;38(1):125–37.
19. Henderson JM, Weeks PA, Hollingworth A. The effects of semantic consistency on eye movements during complex scene viewing. Exp Psychol Human Percep Perform. 1999;25(1): 210–28.
20. Itti L, Koch C. A saliency-based search mechanism for overt and covert shifts of visual attention. Vision Res. 2000;40(10–12): 1489–06.
21. Just MA, Carpenter PA. Eye fixations and cognitive processes. Cogn Psychol. 1976;8(4):441–80.
22. Langley P, Choi D, Rogers S. Acquisition of hierarchical reactive skills in a unified cognitive architecture. Cogn Syst Res. 2009.
23. Lloyd SP. Least squares quantization in pcm. IEEE Trans Inform Theory. 1982;28(2):129–37.
24. Michalke T, Fritsch J, Goerick C. Enhancing robustness of a saliency-based attention system for driver assistance. In: Proceedings of the 6th international conference on computer vision systems. 2008. p. 43–55.

25. Navalpakkam V, Itti L. Modeling the influence of task on attention. Vision Res. 2005;45(2):205–31.
26. Navalpakkam V, Itti L. Search goal tunes visual features optimally. Neuron. 2007;53:605–17.
27. Rebhan S, Röhrbein F, Eggert J, Körner E. Attention modulation using short- and long-term knowledge. In: Proceedings of the 6th international conference on computer vision systems, vol. 5008 of *LNCS*. 2008. p. 151–60.
28. Rensink RA. The dynamic representation of scenes. Visual Cogn. 2000;7(1–3):17–42.
29. Rensink RA, O'Regan JK, Clark JJ. To see or not to see: The need for attention to perceive changes in scenes. Psychol Sci. 1997;8(5):368–73.
30. Röhrbein F, Eggert J, Körner E. Prototypical relations for cortex-inspired semantic representations. In: Proceedings of the 8th international conference on cognitive modeling. 2007. p. 307–12.
31. Rothkopf CA, Ballard DH, Hayhoe MM. Task and context determine where you look. J Vision. 2007;7(14):1–20.
32. Sedgewick R. Algorithms. USA: Addison-Wesley; 1988, 2nd edn.
33. Sonka M, Hlavac V, Boyle R. Image processing, analysis and machine vision, Chap. 5. PWS Publishing; 1998, 2nd edn, p. 176–90.
34. Soto D, Hodsoll J, Rotshtein P, Humphreys GW. Automatic guidance of attention from working memory. Trends Cogn Sci. 2008;12(9):342–48.
35. Toussaint GT. Solving geometric problems with the rotating calipers. In: Proceedings of IEEE MELECON. 1983.
36. Triesch J, Ballard DH, Hayhoe MM, Sullivan BT. What you see is what you need. J Vision. 2003;3(1):86–94.
37. Tsotsos JK. On the complexity of active vs. passive visual search. Int J Comput Vision. 1992;7(2):127–41.
38. Ullman S. Visual routines. Cognition. 1984;18:97–159.
39. Weiler D, Eggert J. Multi-dimensional histogram-based image segmentation. In: Proceedings of the 14th international conference on neural information processing, vol. 4984 of *LNCS*. 2007. p. 963–72.
40. Yarbus AL. Eye movements and vision. New York: Plenum Press; 1967.