J Real-Time Image Proc (2015) 10:699–722
DOI 10.1007/s11554-013-0391-9

SPECIAL ISSUE PAPER

Embedded and real-time architecture for bio-inspired vision-based robot navigation

Laurent Fiack · Nicolas Cuperlier · Benoît Miramond

Received: 22 May 2013 / Accepted: 16 December 2013 / Published online: 22 January 2014

Abstract A recent trend in several robotics tasks is to consider vision as the primary sense to perceive the environment or to interact with humans. Therefore, vision processing becomes a central and challenging matter for the design of real-time control architectures. We follow in this paper a biological inspiration to propose a real-time and embedded control system relying on visual attention to learn specific actions in each place recognized by our robot. Faced with a performance challenge, the attentional model allows to reduce vision processing to a few regions of the visual field. However, the computational complexity of the visual chain remains an issue for a processing system embedded onto an indoor robot. That is why we propose as the first part of our system, a full-hardware architecture prototyped onto reconfigurable devices to detect salient features at the camera frequency. The second part learns continuously these features in order to implement specific robotics tasks. This neural control layer is implemented as embedded software making the robot fully autonomous from a computation point of view. The integration of such a system onto the robot enables not only to accelerate the frame rate of the visual processing, to relieve the control architecture but also to compress the dataflow at the output of the camera, thus reducing communication and energy consumption. We present in this paper the complete embedded sensorimotor architecture and the experimental setup. The presented results demonstrate its real-time behavior in vision-based navigation tasks.

Keywords Real-time vision · Feature detection · FPGA · Robot navigation · Sensation/action

L. Fiack · N. Cuperlier · B. Miramond (✉)
ETIS Lab UMR 8051 CNRS/ENSEA/UCP,
6 Avenue du Ponceau, 95014 Cergy-Pontoise Cedex, France
e-mail: miramond@ensea.fr

1 Introduction

Vision modality plays a center role in robotics for a number of tasks like self-localization, navigation, object recognition and manipulation, target tracking, social interaction between robot and human, imitation, etc. While computer vision systems are becoming increasingly powerful, real-time visual processing still represents a major challenge for autonomous robots. Indeed, visual systems require large computing capabilities which make them hard to embed. At the opposite, biological vision systems have developed through millions of years of evolution, efficient and robust solutions. Studying the complex neural structures involved in biological visual processes allows to elaborate better computational vision system by applying the findings. But to make this progress usable for autonomous robot, it is necessary that the hardware implementation of these algorithms also satisfies the real-time processing requirement, the power consumption limitation inherent to embedded systems as well as constraints regarding the weight and the size of the solution. Hence, smart cameras are proposed as an alternative to get visual low-level processing back into the robot.

Moreover, these visual systems are not considered isolated anymore but as part of an architecture integrated in its environment. They take into account several parameters related to the dynamic properties of the systems they belong to (see active vision [2]). These visual processing algorithms strongly depend on the dynamics of interactions between the system and its environment by continuous feedback regulating even the low-level visual stages such as the attentional mechanisms in biological systems.

The robotic missions considered in this paper consist of a subset of a complex cognitive system allowing a robot equipped with a digital camera to navigate and to perceive

objects. The global architecture in which the visual system is integrated is biologically inspired and based on the interactions between the processing of the visual flow and the robot movements (Per-Ac architecture [18]). The learning of the sensorimotor associations allows the system to regulate its dynamics [13, 32] and, therefore, navigate, recognize objects, or create a visual compass in real time.

The notion of real-time system is still an ambiguous term, with different definitions and interpretations according to the application domain or the design abstraction level. From a formal aspect the real-time constraint has been defined as the respect of the notion of deadline, that is to say the date at which a specific task must be executed. But one has then to define when a specific cognitive task is finished. In the context of robotics also, the design of a real-time system can have several definitions and consequences. Firstly, the different domains of application of these robots can specifically bring critical constraints according to the interaction between the robots and their environment (patrol in factory, cars assembly line, etc.) or worse with humans (medical robots, companion robot, autonomous vehicle, etc.). In this robotics case, the real-time constraint corresponds to determinism, proven and predictive behavior at every stage of the system: computing, mechanics, electronics, mechatronics, etc. In this context, the design of the computing part of a robot has no differences with other real-time systems. The problem is related to the method employed to specify the system and its constraints, to refine progressively this specification and finally to reach an implementation whose real-time behavior is proven [44].

It is specifically the case of the visual subsystem of a robot where the time required to process a frame must be predictable. First, vision is often used as the primary sense for robots to perceive their environment. Secondly, the richness of the information provided by vision has also as a consequence the need for excessive computing. Therefore, the rhythm at which the robot computes visual data becomes the rhythm at which he perceives the external world, at which he can react to a changing environment or at which he can interact with humans. Finally, vision processing becomes the internal rhythm of evolution of the entire robot. All the other cognitive parts of the robot (fusion with different modalities, recognition, tracking, navigation, interacting, etc.) can easily be limited by vision processing. They must then adapt their own rate to that of vision. The robot can basically meet internal limitations to explore the world before this one has changed.

We are specifically interested in this paper in the design and the use of a real-time and embedded vision system for robot navigation. In the application domain of robot patrolling, the robot is considered as fully autonomous and all the computing parts of the robot are designed as dedicated embedded systems. The robot is not considered as a remote set of sensors, but as a programmed and independent agent, taking its own decisions on the basis of learning. That is the reason why we propose in this paper a specific real-time and embedded vision system. The system, and our contribution, is composed of two main parts:

- the Vision layer: a smart camera extracting visual features according to a bio-inspired attentional model;
- the Neural control layer: an embedded framework implementing higher cognitive tasks such as those implicated in the navigation scenario proposed in this paper.

A particular optimization effort has been done on vision for the both reasons presented before: richness of this sense and its importance in the sensation–action loop.

Thus, we present a visual processing system in the form of an embedded platform targeting several robot missions. On the one hand, the platform consists of a specific hardware architecture that should provide intensive computation capabilities to deal with low-level image processing. The proposed architecture combines video sensing, image processing and communication into a System-on-Chip (SoC) embedded onto the robot. This vision system is designed as a full hardware architecture deployed onto a FPGA device. On the other hand, the platform also provides a programmable embedded processing part implementing the neural architecture responsible for multimodal fusion and learning. This complementarity brings the flexibility needed to allow a variety of vision-based navigation missions. Finally, the proposed system processes video frames in real time and reduces the amount of data communication between the visual and the control layer without loss of relevant information.

The paper presents the set of computing subsystems (from perception to learning and action) embedded onto the robot to realize vision-based navigation.

The rest of the paper is organized as follows.

Section 2 presents and analyzes the existing approaches in the context of homing behavior, attentional models and SoC for vision. An abundant literature deals with these different subjects but no approach has been proposed to jointly build and optimize a complete processing system from sensing to decision-making. Section 3 presents the multi-scale attentional approach proposed as support of the Vision layer of our platform. Section 4 describes the architecture of the vision system and the parameters that are used to configure the smart camera. Section 5 presents the performance results achievable with our smart camera. Section 6 describes the Neural control layer. We focus on the neural architecture responsible for place/action learning in this paper. But this embedded framework also implements several tasks used in other robot behaviors. Section 7

presents the experimental platform coupling Vision and Neural control layers. We also present the behavioral results obtained during homing-based robotic missions. Finally, we conclude and outline future works in Sect. 8.

2 State of the art

2.1 Context of the study

Among different perceptual modalities, vision is certainly the most important building block of a bio-inspired cognitive robot. Vision enables robots to perceive the external world in order to perform a large range of tasks such as navigation, object tracking and manipulation, and even interaction with humans. This broad range of tasks often relies on different models and architectures of artificial vision. According to a given task, the visual processing often uses only a subset of the available algorithms like optical flow, feature points extraction and recognition (over one or several scale spaces). For instance, navigation often relies on low resolution images to only capture the main regularities from the environment, whereas object recognition may need to characterize an object over several scales in order to take into account more details of the object.

In the context of this work, we aim to design the cognitive architecture of an autonomous robot patrolling in an unknown environment (coarse-grain vision) and coming back periodically to its *home* location (homing behavior). At that time, the robot has to locate its energy connection (fine-grain vision) and plug its battery. We thus need a visual algorithmic chain bringing either low or high-resolution data according to the robot's state and to the battery level. The present paper only focuses on a subset of this scenario: the homing navigation task. We describe a vision-based architecture for mobile robot navigation achieving a sensory–motor task (homing behavior) in real time in an unknown indoor environment.

In the following sections we study and discuss, the existing approaches in the domains of homing navigation and visual attention models. Next subsection deals with the available implementations of real-time vision systems in an embedded context.

2.2 Homing behavior

Homing is an efficient navigation strategy to reach a particular place in an open environment (such as a single room). Insects like ants, bees, and wasps are known to use visual information, in conjunction with a compass, to return to their nest [10, 17, 27, 41]. Similarly, birds, rats and primates can return to a place by using visual cues. Even if the implicated neural structures may be different, the sensory–motor learning used by insects to navigate can also, a priori, be used by mammals. Furthermore, in mammals like rats, a particular brain structure named hippocampus is involved during navigation tasks. In this brain structure, particular cells, named Place Cells (PC), have been discovered. These cells exhibit a firing pattern strongly correlated with a particular location in the environment [35]. Based on the spatial properties of these cells, several models of homing behavior have been proposed: homing can be based on directional place field[1] [46], on Place Cells encoding obstacle proximity [8, 9]) or Place Cells learning landmarks configuration [18, 20]. Our robot control architecture is directly inspired from [18, 20, 32].

2.3 Attention models

Bioinspired computer vision systems are a promising way towards building more adaptive and more robust computer vision solutions. Among the different possible solutions, we will focus on the visual attention paradigm which can help to overcome the complexity issue in computer vision [43]. Visual attention is the ability of a vision system, to only focus on the most relevant part of the image. Once detected, these salient parts become the starting points of the next visual processing stages (recognition, localization, etc.) thus drastically reducing the amount of information to process. Most of the psychophysical models of visual attention describe it as a two-stage mechanism: a preattentive stage that processes in parallel different kind of feature extractions on the image and a second stage that sequentially handles a subset of this information. For a review, see Heinke and Humphreys [24]. With 'A feature-integration theory of attention' [42], Treisman work has led to numerous computational models among which we can cite the saliency-based model of [26, 28]; the selective tuning model of [43]. These early works have inspired several models for object detection and recognition [36], robot localization [18, 19, 20, 22, 37, 40], and robot navigation [13, 15]. One can refer to [16] for a more detailed review.

Our model is used to implement a homing navigation strategy similar to [18, 19, 20]. At the opposite of [13, 15], it does not require to build a map to allow the robot to reach a particular (goal) place, since in this part of the scenario, the robot evolves in an open environment (a room).

2.4 System-on-chip for artificial vision

The real-time execution of certain complex and regular algorithms, such as those used in vision processing,

[1] Place field is the projection in the environment of the locations where a given PC fires.

sometimes need a joint optimization of the algorithm and the computing architecture. This optimization process is needed in the embedded context of autonomous robots where computational power is limited. For the validation of bio-inspired models, designing autonomous agents is the base of our validation methodology. But it also brings additional constraints regarding size, weight, energy consumption, computational power, communication bottleneck, etc. And the best solution to face these constraints is to design a specific hardware architecture implementing the complex and heavy parts of the embedded computation. The problem is then to identify into the algorithm the functional blocks to deploy in hardware and software in order to build a programmable and powerful system. The final architecture is then called a SoC since integrating both programmable and dedicated blocks in a single chip. This type of solution is very efficient in term of density of computation (millions of operations per second and per Watt) compared to embedded processors or Graphical Processing Units (GPU) [11]. Then this architecture can either be deployed into a specific (Application Specific Integrated Circuit, ASIC) or configurable device (Field Programmable Gate Array, FPGA). We propose in this paper an FPGA-based smart camera in order to provide some flexibility for future robot missions. We will now present existing works in the context of SoC architectures for real-time vision. Most of them are based on feature detection algorithms. The application domain of these architectures does not necessarily focus on robotics, but the same properties of the visual chain are targeted: invariant to scale, rotation, and change in illumination, stability of features. A real-time parallel SIFT detector has been proposed in [6]. The proposed architecture detects features up to 30 frames per second in 320x240 images. The system has been configured for three octaves with five scales in each one. The whole system is completely embedded on an FPGA. The proposed architecture utilizes a 5×5 image region as neighborhood for the keypoint computation. We observed in our experimentations that this restriction could be a hard limitation to detect robust keypoints. The system has been applied to simultaneous localization and mapping system for autonomous mobile robots, where robot navigation environment maps are built based on features extracted from images [5]. But authors do not provide details on the utilization of their architecture in this context.

FPGA-based architectures have also been proposed for Speeded Up Robust Features (SURF) algorithm. Several approaches [3, 7, 39] have been validated and compared on different applications such as object deformations measurement, vehicle tracking. The results show that efficient FPGA-based feature detection can be obtained by employing the SURF algorithm. But the matching step of the SURF method is an expensive stage that greatly limits the real-time behavior of the method. Indeed, it looks for correspondences (match-points) between keypoints of two different frames by means of the nearest neighbor distance. In [3], the presented results show performances from 5 to 340 ms per frame without and with the matching stage.

The architecture proposed in [4] is based on a Harris and Stephen detector and selects and sorts the features in real time. The adaptive method used for points thresholding is interesting. The variations in illumination imply a variations in the number of the detected interest points. Using an adaptive threshold is an efficient way to ensure the proper number of points at each frame, but is not necessarily convenient for keypoints recognition at different times. Authors also propose a hardware-sorting module. But the method used to sort the detected interest points requires the presence of all the points. For that reason, the sorting is done only after the image treatment. We will present in Sect. 4 a hardware method that sorts the points continuously at run-time following the flow of pixels coming from the camera. This method allows to reduce significantly the amount of memory since only most important keypoints are kept. Moreover, the sort is only based on the interest value of each point and does not consider an inhibition neighborhood that prevents near keypoints unadapted to place recognition. This interest points detection system is capable of processing the images at an average speeds of 156 fps (frame per second) when applied to 512×512 pixel images.

A different approach was proposed in [38]. This approach presents an FPGA-based stream processor for embedded real-time vision with Convolutional Networks. This programmable architecture starts from the observation that many recent visual recognition systems, such those presented above, can be seen as multiple layers of convolutional filter banks with various types of non-linearities. The system presented in this paper is a programmable ConvNet Processor (CNP), which can be thought of as a SIMD (Single Instruction, Multiple Data) processor. Implementing a particular ConvNet simply consists in reprogramming the software layer of the processor, but also requires to reconfigure the runtime connections, operators, and DMA modes in the FPGA.

The programmable approach is very interesting. The coupling between the hardware grid of programmable convolvers and their control unit brings a high degree of flexibility. But the sequential reuse of the convolver unit between image filtering and features learning needs constant access to external memory for the line buffers of the convolver. This limits the performances reachable compared to a hardwired architecture where line buffers can be stored in the embedded RAM Blocks of the FPGA. But the main limitation for our purpose concerns the utilization methodology itself. Prior to being run on the CNP, a

ConvNet must be defined and trained on a conventional machine. In the context of mobile robotics, their is no clear separation between the learning phase and the running one. The robot learns its environment during navigation through each new room he meets. During its first patrol the robot navigates and learns the place topography with the aid of a human supervisor. Then the robot navigates alone and realizes patrolling across the different rooms. Moreover it seems obvious that there must be no differences into the robot between the computing machine used for place learning and recognition. However, for their application, authors also built a custom board around a Xilinx Virtex-4 FPGA [14]. The system delivers around 10 frames per second at VGA resolution for typical filter banks. The platform proposed in [14] consumes 15 W in peak, camera included, and is capable of more than 4.10^9 multiply-accumulate operations per second in real vision application. The consumption reduction obtained with this type of realization is a confirmation for us to propose in a near future our own specific board.

Another approach for flexibility is proposed in [45]. It relies on a low-cost embedded system based on an architecture that integrates FPGA and DSP (Digital Signal Processor) and thus integrates the parallelism of FPGA and the flexibility of DSP. We are following in this paper an identical choice for the hardware acceleration of a Difference of Gaussians pyramid. Once a complete octave has been processed, both implementations down-sample the Gaussian-filtered image by taking every other pixels in each row and column, and treat this image as the first image at the next octave. But unlike Zhong etal who aim for flexibility, our main goal is compactness of the vision system into a single System-on-Chip.

3 A multiscale attentional architecture

The current visual system integrates a multi-scale approach to extract the visual primitives. In doing so, it also allows a wider range of applications. Roughly the visual system provides a local characterization of the key-points detected on the image flow of a digital camera. This local characterization feeds a neural network which can associate motor actions with visual information: this neural network can learn, for example, the direction of a displacement of the robot as a function of the scene recognition. The neural architecture is presented in Sect. 6.

The studied visual system can be divided into two main modules described in the following sections:

- a multi-scale mechanism for characteristic points extraction (key-points detection),
- a mechanism supplying a local feature of each key-point.

3.1 The multiscale keypoints detection

The multi-resolution approach is now well known in the vision community. A wide variety of key-points detectors based on multi-resolution mechanisms can be found in the literature. Among them are the Lindeberg interest point detector [30], the Lowe detector [31] based on local maxima of the image filtered by difference of Gaussians (DoGs) or the Mikolajczyk detector [34], where key-points correspond to those provided by the computation of a 2D Harris function and fit local maxima of the Laplacian over scales. The visual system described here is inspired by cognitive psychology. The used detector extracts points in the neighborhood of the key-points, which are sharp corners of the robots visual environment. More precisely, the key-points correspond to the local maxima of the gradient magnitude image filtered by difference of Gaussians (DoGs). The detector is characterized by a good stability (or equivariance [30]) over scales. Following the detector itself, the system provides a list of sorted local features by the way of competition between the corresponding key-points.

Key-points are detected in a sampled scale space based on an image pyramid. Pyramids are used in multi-resolution methods to avoid expensive computations due to filtering operations. The algorithm used to construct the pyramid is detailed and evaluated in [12]. The pyramid is based on successive image filtering with 2D Gaussian kernels normalized by a factor S.

These operations achieve successive smoothing of the images. Two successive smoothing are carried out by two Gaussian kernels with variance $\sigma^2 = 1$ and $\sigma^2 = 2$. The scale factor doubles (achievement of an octave) and thus the image is decimated by a factor of two without loss of information. The same Gaussian kernels can be reused to continue the pyramid construction. Interestingly, the kernel sizes remain small (half-width and half-height of 3 σ) allowing a fast computation of the pyramid. Finally, the images filtered by DoGs in the pyramid can be simply obtained by subtracting two consecutive images. A representation of the whole algorithm is given in Fig. 1.

Key-points detected on the images are the first N local maxima existing in each DoG image of the pyramid. Thus, the key-point search algorithm sorts the N first local maxima according to their intensities and extracts their coordinates. The shape of the neighborhood for the search of maxima is a circular region with a radius of R pixels. This value is one of the system parameters. The number N which parametrizes the algorithm corresponds to a maximal number of detections. Indeed, the robot may explore various visual environments (indoor versus outdoor) and particularly more or less cluttered scenes may be captured (e.g., walls with no salience versus complex

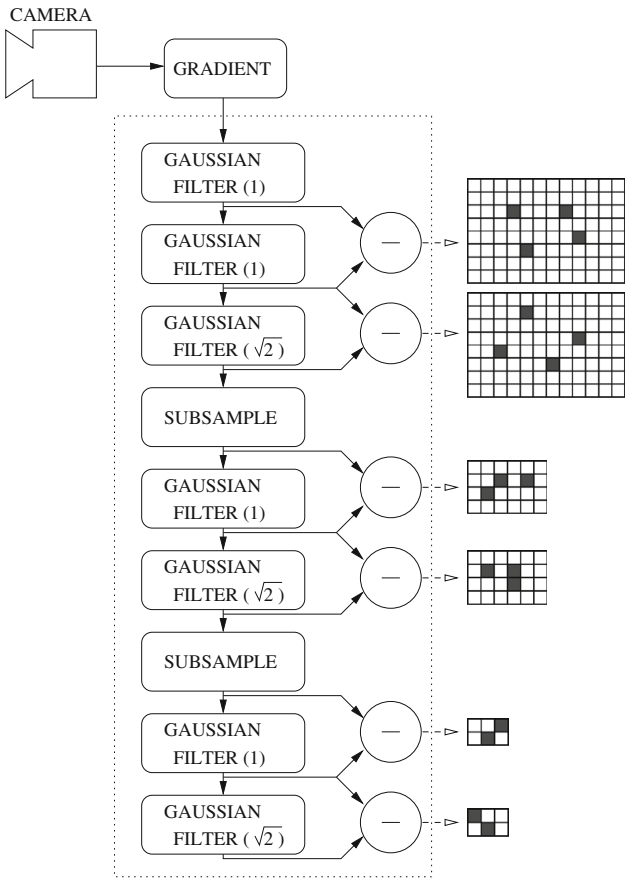


Fig. 1 Representation of the Gaussian pyramid

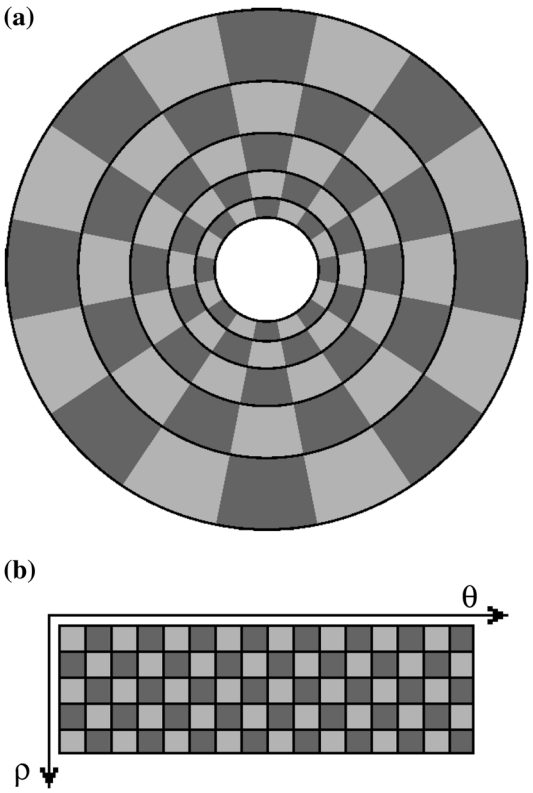


Fig. 2 Example of log-polar mapping. The image **a** represents the log-polar mapping with $\rho = 5$ and $\theta = 16$ and the image, **b** is the result of the remapping in the (ρ, θ) space

objects). A detection threshold (γ) is set to avoid non-salient key-points. This threshold is based on a minimal value of the local maxima detected. The presence of this threshold is even more important in the lowest resolutions since the information is very coarse at these resolutions. This particularity of the algorithm confers it a dynamical aspect. Precisely, the number of key-points (and consequently the number of local features) depends on the visual scene and is not known a priori. Furthermore, the threshold γ could be set dynamically through a context recognition feedback but discussing here this mechanism is not our current purpose. However, even if this threshold is considered as a constant value, the number of detected key-points varies dynamically according to the input visual scene. Consequently, the number of computations (neighborhood extractions) also depends on the input data.

For now we focus on the first scale of the detector, that is the first difference of Gaussians.

3.2 The local image feature extraction

At this stage, the neighborhood of each key-point has to be characterized in order to be learned by the neural network. Existing approaches to locally characterize key-points are numerous in the literature: local jets, scale invariant feature transform (SIFT) and its variants, steerable filters, and so forth. In the current application, we simply reuse a view-based characterization where key-point neighborhoods are represented in a log-polar space. This representation has good properties in terms of scale and rotation robustness. The local feature of each key-point is, therefore, a small image resulting from the log-polar transformation of the neighborhood (see Fig. 2).

The neighborhoods are extracted from the gradient magnitude image at the scale; the key-point was found by the detector. Each neighborhood extracted is a ring of radius (5, 16) pixels. Excluding the small interior disk avoids multiple representations of the central pixels in the log-polar coordinates. The angular and logarithmic radius scale of the log-polar mapping are sampled with, respectively, θ and ρ values. Each feature is thus an image of dimension $\theta \times \rho$ pixels. The sizes of the rings and feature images have been determined experimentally for an indoor object recognition. The given parameters represent a trade-off between stability and specificity of the features. Finally, the small log-polar images are normalized before their use by the rest of the neural architecture. By associating the

data provided by the visual system with actions, the global system allows the robot to behave coherently in its environment [32].

3.3 Computational complexity of the visual chain

A first study was performed to point out the limitations of the software implementation of the visual chain.

The following execution times correspond to the first scale of the algorithm. They are measured on a laptop equipped with an Intel T2300 @1.66GHz CPU, 2GB of DDR2 RAM. These data are given for a resolution of 192 × 144.

Function	Execution time (ms)
Gradient	1.09
Gauss 1	7.08
Gauss 2	7.02
DoG	0.44
Key-point search	3.29
Extraction	0.4
Total	19.32

One can easily see that for larger images this solution is unacceptable. For instance, we can estimate that on this type of platform the processing of VGA videos (640 × 480) will reach less than one frame per second.

The precise real-time constraint is hard to define since it depends on the robot's task. Exactly, the robot has to capture and perceive the important changes in its environment. The rate of these changes is different for navigation or object tracking. And for navigation tasks, the speed of the robot strongly depends on the rhythm at which he can perceive obstacles and moving peoples. In the context of patrolling robots in unknown and changing environment, presented in Sect. 2.1, we are proposing a dedicated hardware architecture for visual processing. The aim of this architecture is to meet the real-time challenge while remaining tightly coupled to the neural control layer.

4 Hardware architecture of the vision SoC

We describe the first scale of the detector presented in Sect. 3. Since the functional blocks are configurable in size and width, the rest of the visual chain corresponds to multiple instantiations of this first scale.

The organization of our architecture is depicted in Fig. 3. It is composed of a chain of custom Intellectual

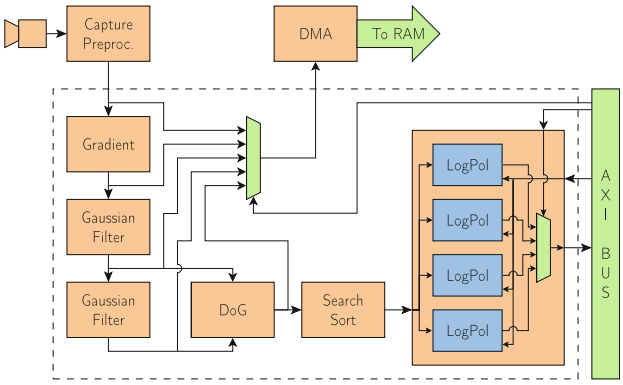


Fig. 3 Global view of the hardware architecture for one scale. The flow of pixel comes from the camera and goes to the CPU's memory through a DMA. An intermediate output can be selected thanks to a dedicated register. Another register allow to select which feature to read. Finally, the key-points can be read through the memory-mapped interface

Properties (IPs), written in VHDL, that takes its input from a camera thanks to a streaming interface (instantiated as an AXI streaming interface (see ARM [1]) into the prototype, see Sect. 5).

The IP can be configured by the CPU thanks to a memory-mapped interface.

The IP chain generates several results which can be read back by the software part:

- A DoG or an intermediate processed image, selectable thanks to a dedicated register;
- The list of key-points extracted and sorted by the IP at the different frequency bands;
- The list of log-polar features associated to each key-point.

The detected features are identified by their coordinates, scale and octave.

Since most of the IPs follow a data-flow model of computation and produce and consume pixels at inputs and outputs, a standardized interface has been developed to connect these IPs all together.

The coordinates of the pixels are mandatory for each IP to take care of the side effects. Moreover, the learning of the visual cells for the navigation needs to know the coordinates of the key-points. The coordinates of the pixels must then be transferred from one IP to one other. Across the pipeline, these data must be either delayed or re-generated by each processing IPs depending on its latency.

According to these constraints, the standardized interface is composed of the following signals:

- The pixel value,
- its coordinates (x, y),
- an enable signal.

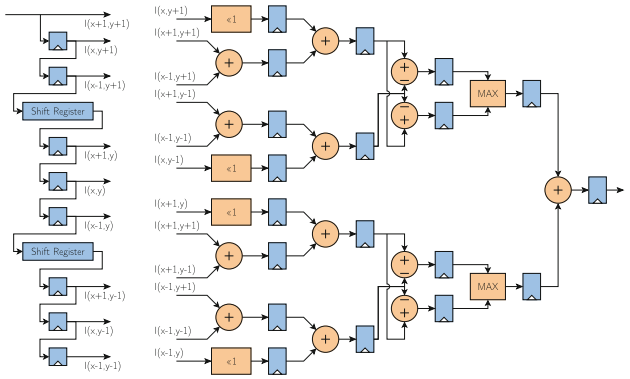


Fig. 4 Gradient magnitude IP. The management of the input pixels is shown on the *left*, the computation of the gradient magnitude is shown on the *right*

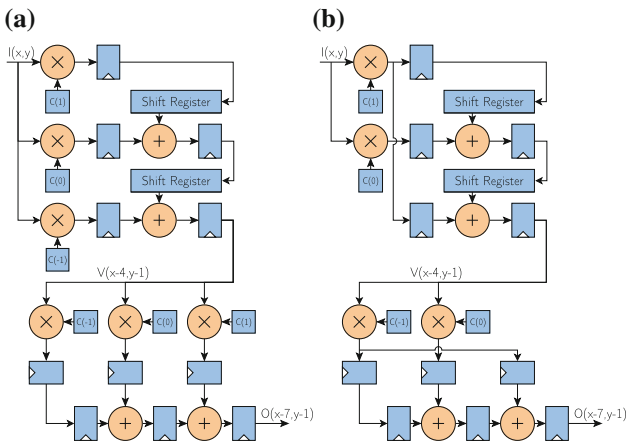


Fig. 5 Gaussian Filter IP. In **a**, the two-dimension convolution is separated into two one-dimension convolutions. In **b**, redundant multipliers are suppressed

4.1 Gradient

The gradient magnitude is computed by a simplified version of the Sobel operator.

If I is the input image, G_x and G_y represent, respectively, the horizontal and vertical derivatives. These images are computed thanks to a convolution by the classical Sobel kernels (Eq. 1).

The main difference compared with the classical Sobel operator is the calculation of the gradient magnitude image G. We avoided the usage of a square root operator by replacing it by a simple sum of absolute values (Eq. 2).

$$G_x = I * \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}, \quad G_y = I * \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{1}$$

$$G = \text{abs}(G_x) + \text{abs}(G_y) \tag{2}$$

The architecture used to compute the gradient magnitude is depicted in Fig. 4.

A structure composed of shift registers and regular registers is used to store the input pixels. Each clock cycle, this structure outputs the eight pixels needed by the processing part of the IP to compute the output pixel.

The computation part of the IP is built in a pipelined way, allowing the throughput to increase, with a few extra latency. The five extra cycles are not significant compared with the line and the pixel latency imposed by the input pixel memorization. Finally, the total latency of the IP is $img_{width} + 6$ clock cycles.

A side effect management avoids the border of the image to be corrupted. Here, we use a classical method that simply consists in replacing all *non-existing* pixels by zero in equation Eq. 1. The detection of these *non-existing* pixels is done thanks to the coordinates of the input pixel and the img_{width} and img_{height} generic parameters. This side effect management mechanism attenuates the gradient magnitude on the border of the images, but does not affect the other pixels.

4.2 Gaussian filter

The Gaussian filtering operation consists in the convolution of the image by a Gaussian function

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3}$$

In this formula, the term $\frac{1}{2\pi\sigma^2}$ ensures that the area under the Gaussian bell is equal to 1, that is to say the filter gain is equal to 1.

This function is pre-processed and can be stored in a 7×7 or 9×9 array. The value of the coefficients depends on the size of the buses. We can tune them so that the term $\frac{1}{2\pi\sigma^2}$ can be achieved by a division by a power of two. This allows us to replace it by a right shifter. In our case, for compatibility with the software, we chose a 16-bit bus width for the input pixels and for the coefficients and a 32-bit internal bus width.

The architecture of the IP, depicted in Fig. 5, is able to process a pixel at each clock cycle. This point is mandatory since the camera can send valid pixels at this rhythm.

In order to save some multiplier, the two-dimension convolution is separated into two one-dimension convolutions. One can see on the top of the Fig. 5a the vertical convolution. The horizontal convolution is achieved by the bottom of the IP.

As the Gaussian function is even, we know that some coefficients will be equal (for instance: $C(1) = C(-1)$). We can benefit from this property to *factorize* the redundant multipliers as depicted in Fig. 5b.

There are plenty of solutions to take care of the side effects. The first solution is to crop the output image in

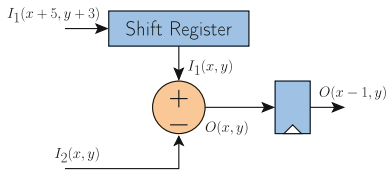


Fig. 6 Difference of Gaussian IP. A shift register is responsible for the synchronization of the two pixel flows. The difference is computed by a simple subtract module

order to work only with valid pixels. With this solution, the resolution will dramatically decrease, especially in the low resolutions.

The solution is then to reduce the corruption induced by the invalid pixels. Lets discuss the different approaches. The invalid pixels can be set at the maximal value, this will lead to bright borders. They can be set at the minimal value, leading obviously to dark borders. They also can be set at the median value, that will limit the impact. Another solution is to mirror the image at the borders, but it will make the IP heavier.

To take a decision, we need to take a look at our application. The first Gaussian filter takes its input from the gradient magnitude IP. The gradient magnitude is usually dark, unless the visual environment is full of salient points. We need to keep in mind that the salient points will be sorted by intensity, and that the lowest points will be discarded. A bright border will lead to fake key-points detection. The dark-border solution is better because it allows to detect only true key-points.

4.3 Difference of Gaussian

The difference of Gaussian is computed thanks to a simple subtract module. The difficulty of this IP is due to the fact that its inputs are taken from the input and the output of the same Gaussian filter IP.

One can then observe a delay between the input of the IP coming from the filter latency. A shift register allows to synchronize these two pixel flows, as shown in Fig. 6.

4.4 Key-point search

4.4.1 Classical approach

The key-point search algorithm consists in finding the local maximums in the DoG images. For each pixel, the IP searches in a disk of radius R if the pixel is greater than the others, to determine if it is the maximum in this area. The IP works in the manner of a classical convolution operator, except that it is circular. To respect the working rhythm, a pixel must be processed at each cycle.

A pixel must satisfy four criteria to be identified as a key-point:

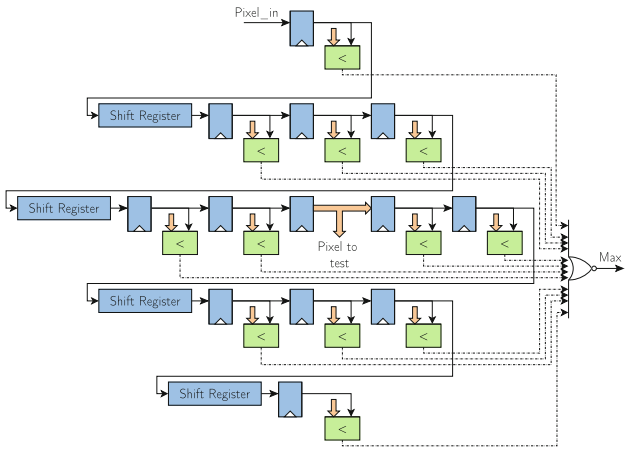


Fig. 7 Key-point search IP: local maximum detection

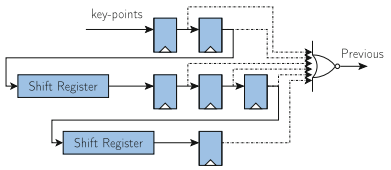


Fig. 8 Key-point search IP: key-point inhibition

- It must be a local maximum as described above (Fig. 7);
- There must be no other key-point in the detection disk. As the half-disk above the pixel to process is already tested, it is possible by storing the results to ensure that this constraint is respected (Fig. 8). This block allows to set different radius for the detection and the inhibition;
- The value of the pixel must be greater than a noise threshold γ. The key-points then cannot be detected in a monotonic area (Fig. 9);
- The pixel must be far enough from the border such that all the pixels in the disk are valid. The (x, y) coordinates of the pixel to process can be used again (Fig. 10).

4.4.2 Optimized IP

One can see on Sect. 5.1 that the classical design of this IP is an obstacle to scalability due to the disk of comparators.

To avoid the scalability limit, we propose a solution which considers a square window. First, for each input pixel, a local maximum is found in the first line of the window. This search algorithm is implemented as a tree of comparators (see Fig. 11a). This maximum is then stored in a shift register. The x coordinate of the maximum in the window must also be stored.

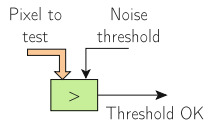


Fig. 9 Key-point search IP: threshold test

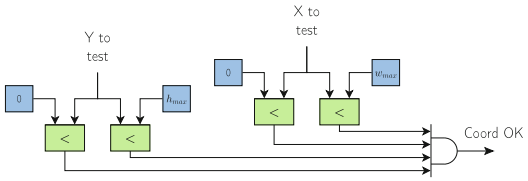


Fig. 10 Key-point search IP: coordinates test

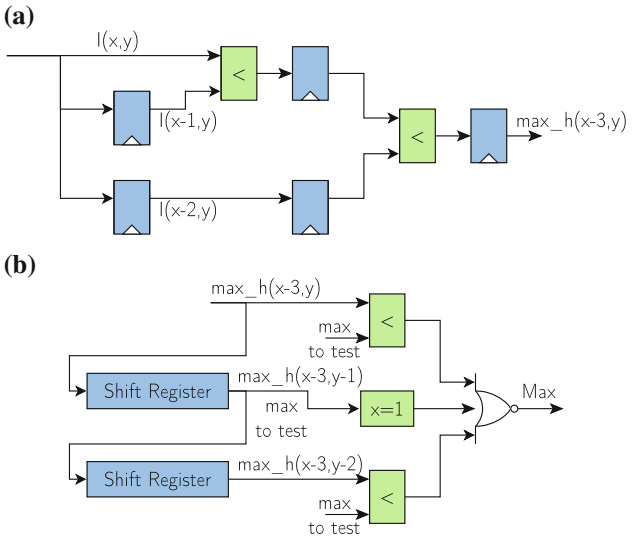


Fig. 11 Optimized local maximum search IP with a (3 × 3) window. In **a**, a horizontal maximum is found and sent, with its position, to (**b**). In **b**, the central maximum is checked. If it is the maximum and if its position corresponds to the center of the window, it is tagged as a local maximum

The second part of this solution consists in searching a vertical maximum among the horizontal maxima. If the maximum is detected in the center of the window, it is considered as a local maximum.

The knowledge of the vertical position of the maximum is not required, to avoid the latency of the comparator tree, the central vertical maximum is compared with the other vertical maxima. One can see in Fig. 11b that the comparators can then work in parallel.

Finally, we still need to assure that the local maximum is far enough from the sides of the image, and that the pixel is above the noise threshold. The architectures described above can be used again.

4.5 Key-point sort

The key-points are given to the sorting IP by the searching IP. They are represented by an {*x*, *y*, value} structure. An index representing which log-polar transform bloc will be used is added to this structure. The sorting IP sorts these structures in function of their value.

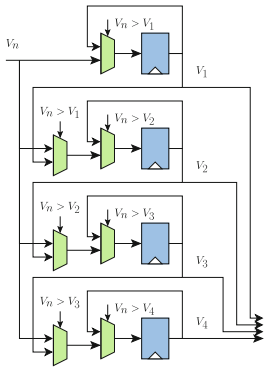


Fig. 12 Key-point sort IP. The {*x*, *y*, index, value} structures are sorted by their values

Once more, the IP must be able to respect the rhythm given by the sensor. This time, we can take advantage of the fact that at a given time only one point must be sorted.

Lets consider that at each time, the list of key-points—that is the structures described above and stored in registers—is sorted. When a new point inputs, its value is compared to the ones already sorted in the list. When its value is large enough to be inserted, the structure is stored in the right register. Every pixel below then shifts to the bottom.

The architecture of this IP is depicted in Fig. 12.

4.6 Log-polar transformation

Two IPs are responsible for the log-polar mapping, *Address Generator* and *Transform*.

The *Address Generator* (Fig. 13) converts the Cartesian address into the log-polar one. A test ensures that the incoming flow is in the neighborhood disk of the key-point, and then its coordinates are subtracted from the coordinates of the flow. These coordinates are the input of a look-up table (LUT) which stores the log-polar coordinates.

The *Transform* IP (Fig. 14) reads the incoming pixels, and put them into a memory at the address computed by the *Address Generator*. This IP is built as a double-buffer to compute an image while reading the previous one. The zones of the outer rings containing more than one pixels must be averaged. An accumulator is then build around the memory. As the address conversion is done by a LUT, the values for the average division can also be stored in a LUT.

One can see the global architecture of the Log-polar transformation block on Fig. 15. These IPs are duplicated *N* times, where *N* is the maximal number of key-points to detect. The key-points are dispatched to an *Address Generator* depending on the index from the *sort* IP. As we cannot shift down the content of the entire memory of a *Transform* bloc, we can only overwrite the memory

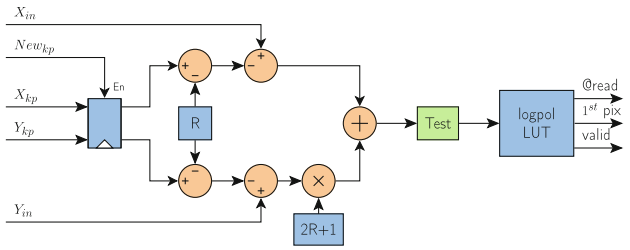


Fig. 13 Log-polar transformation IP: the address generation. The coordinates of the pixels coming from the gradient (X_{in}, Y_{in}) are re-arranged in function of the coordinates coming from the key-point sort IP (X_{kp}, Y_{kp}) and the extraction radius. The result is then sent to the LUT if the incoming pixel must be extracted

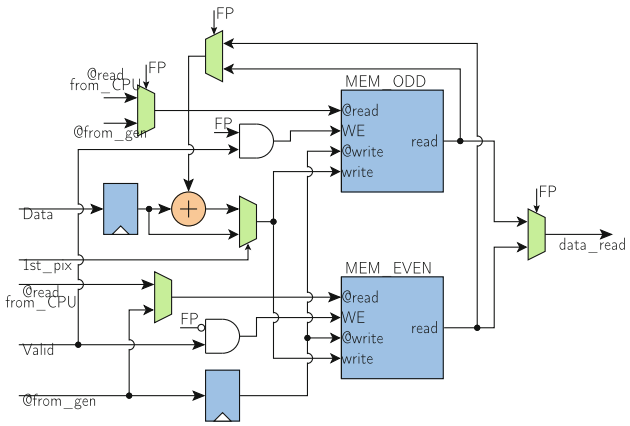


Fig. 14 Log-polar transformation IP: the transformation itself

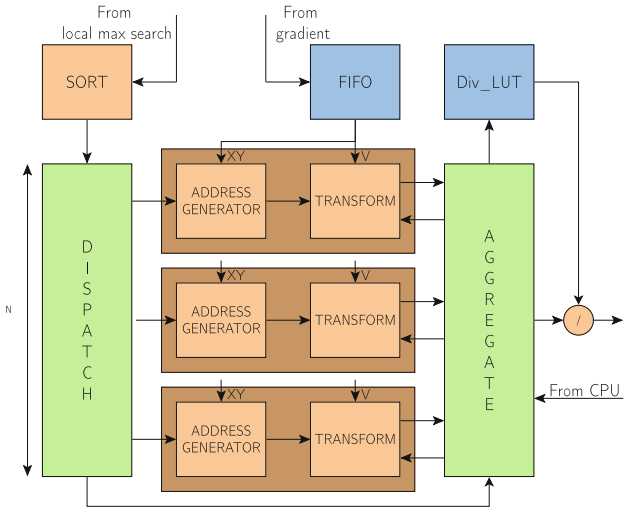


Fig. 15 Log-polar transformation IP. The *N* features can be extracted and remapped in parallel

corresponding to the lower key-point. That is why we need a connection between the rank of the key-point and its index.

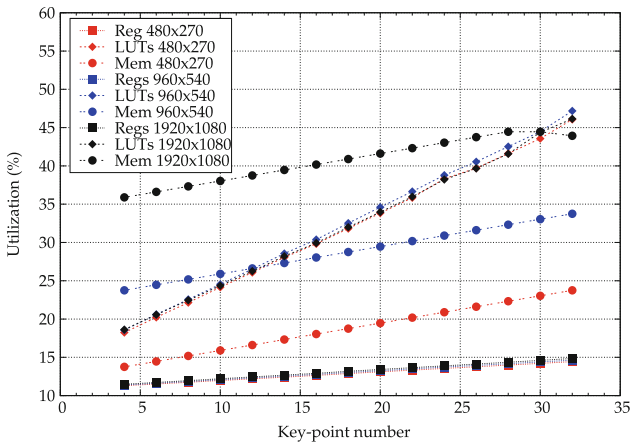


Fig. 16 Hardware consumption versus the number of key-points given in percent. The radius is fixed to 12

Finally, a CPU can read all the data by selecting an index. A memory (not represented) stores the value of the key-point, its coordinates, and the relation between its rank and its index, so that the CPU does not have to sort the values again. It can read the log-polar feature as a RAM memory.

5 Implementation results

5.1 Hardware resources consumption

We explore the impact of some parameters of the IP described earlier on the hardware resources consumption. This exploration will allow us to estimate the feasibility of a hardware implementation of the full multi-scale attentional architecture described in Sect. 3.

All figures represent a set of synthesis, before place and route. The rest of the logic, as for instance the preprocessing of the raw pixels or the AXI interconnect, do not appear in the following results.

This exploration was done on a Xilinx Zynq 7020 which is composed of 106*k* registers, 53*k* LUTs, 560 kB of memory and 220 DSPs.

For different resolutions—1,920 × 1,080 (fullHD), 960 × 540 (half), and 480 × 270 (quarter)—we vary the extraction radius and the maximal number of key-points to detect. We also compare the *disk* search algorithm (see Sect. 4.4) with the optimized one.

The impact of the number of key-points is depicted in Fig. 16 when using the classical algorithm. We choose an extraction radius of 12 as it appears to be the limit, when integrated in the whole architecture, with the classical key-point search algorithm.

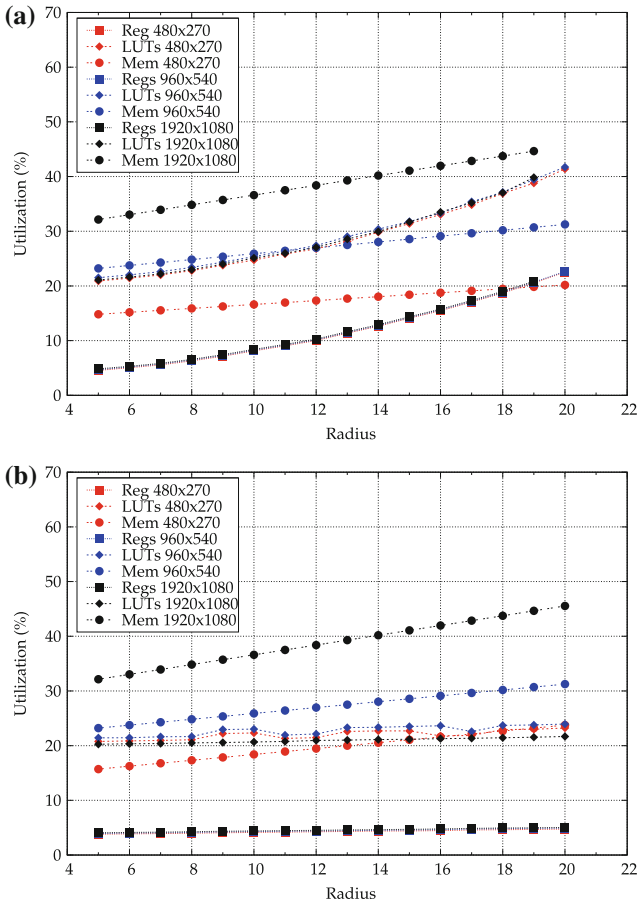


Fig. 17 Hardware consumption versus the extraction radius given in *percent*. The results for the classical approach are shown in (**a**) and the optimized one in (**b**). The number of key-points is fixed to 16

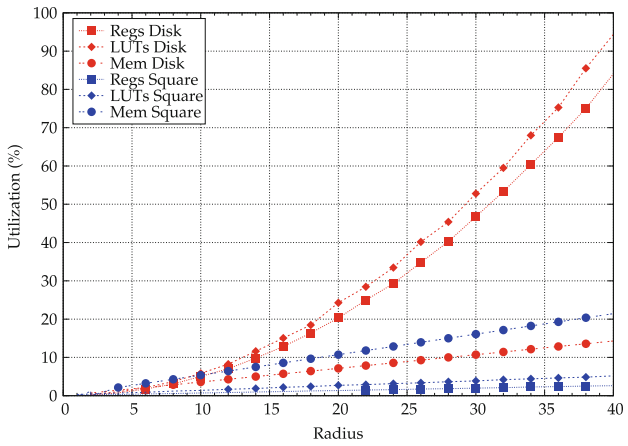


Fig. 18 Consumption of the local max search IP. Comparison between the classical (*disk*) and the optimized (*square*) approaches

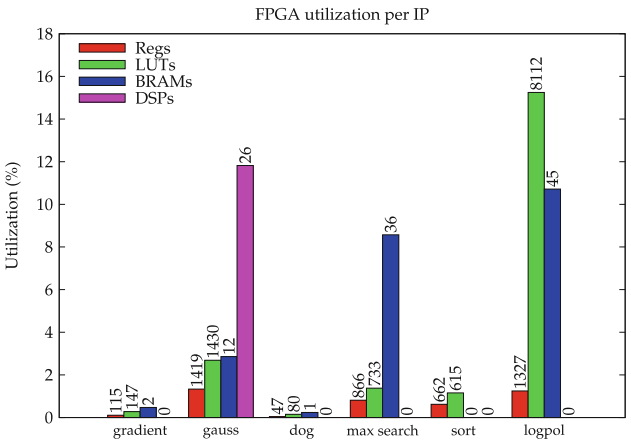


Fig. 19 Hardware consumption per IP given in *percent*. The consumption in element are displayed above each column

The impact of the radius for both classical and optimized approach is depicted in Fig. 17. We choose a maximum of 16 key-points.

One can see in Fig. 16 that the number of key-points has a linear influence on the consumption of all the resources even with the classical method. On the contrary, the extraction radius, with the classical approach, has a quadratic influence on the LUTs occupation (Fig. 17a). This is due to the increase of the area of the comparators disk, as shown in Sect. 4.4. With the optimized search IP, the influence of the extraction radius becomes linear (Fig. 17b).

The influence of the radius on both classical and optimized search IP is depicted in Fig. 18. This figure shows that the influence of the radius on the consumption of the optimized IP stays linear even with a higher radius.

The size of the image increases the needs of memory, simply because more pixels must be stored to compute the convolutions.

Finally, we plot the distribution of the consumptions of each bloc in Fig. 19. The parameters are the following:

- 480 × 270 images,
- a maximum of 16 key-points,
- an extraction radius of 12.

First of all, note that the *gauss* column corresponds to the sum of the two Gaussian filter blocs.

One can see again the benefits of the optimized maximum search IP over the classical one. The difference in memory usage seems to be due to the synthesizer which choose to use 36-bit wide memory instead of 18-bit wide.

Finally, we can see that the *logpol* is the IP which require the most part of the LUTs and memory. The memory needs are due to the storage of the log-polar features. The needs in LUTs are due to the fact that the IP is duplicated 16 times; this parallelism is required because some pixels can appear in various features.

5.2 Toward a multi-scale implementation

The hardware implementation of the low-resolution led to a working architecture. The architecture is able to process the images given by the sensor at always above 60 fps.

The hardware resources consumption have been plotted for different set of parameters. The distribution of the resources over the different sub-IPs has been observed too.

We then saw that two IPs could be an obstacle to a multi-scale implementation. The key-point search and the log-polar transformation are greedy in resource consumption.

The search IP has been optimized so as to reduce hardware resources consumption. This consumption increases now linearly in function of the search radius.

The log-polar transform IP can be reduced too if some hardware resources can be shared by several transform blocks. Therefore, this hypothesis requires that a given pixel cannot be in different features, that is the features do not overlap.

In conclusion, we can consider an implementation of the multi-scale architecture with two scales on our Zynq 7020. A larger FPGA may be required to integrate the three octaves presented in Sect. 3.

5.3 Compression rate

A 480 $\times$ 270 image with a 10-bit pixel width is composed of 1.296 Mbit. The one-scale architecture generates a maximum of 16 features. This set of 16 $\times$ 16 pixels with a 16-bit bus width is composed of 65,536 bits.

We can compute the compression rate τ which is

$$\tau = \frac{\text{features set size}}{\text{image size}} = 0.05 \quad (4)$$

It corresponds to 5 % of the original image. The amount of data communicated at the output of the smart camera is thus significantly reduced as the energy consumed to transmit them to the control layer. We can extrapolate the compression rate that can be achieved with a multi-scale architecture by multiplying this compression rate by the number of scales. For the full multi-scale pyramid, the compression rate of 30 % remains interesting considering the quality of visual data extracted by the architecture.

6 Neural architecture for environment learning

We describe in this section the neuronal architecture that controls the RobotSoC platform behavior. This neuronal architecture is based on PerAc building blocks for multimodal associations [18]. PerAc architecture models perception as a dynamic process linking sensation (ie: vision) with action (ie: movement). In this architecture, the robot perception and its behavior result from a tight coupling between visual input and motor action: a PerAc loop is a combination of a reflex pathway and a categorization pathway.

Following a constructivist approach, we have then proposed several neuromimetic control architectures of increasing complexity (involving several PerAc loops) for navigation of a mobile robot [32]. These models are all inspired from mechanisms used by some insects like honey bees and some mammals like rats or monkeys for self-localization and navigation. More precisely, they result from functional models of some cerebral structures and their interplay. These models are mainly driven by vision processes with such high computational cost that it is not possible to execute them onto the embedded computer of our previous robot platforms based on Robulab.[2] Hence, these architectures still need an external workstation wirelessly linked to the robot that can handle the heavy visual processing. This external link limits the autonomy of the robot. As a response to this limitation, the RoboSoc platform allows to embed all processes in the robot relying on a hardware solution to perform most of the visual processing and an onboard computer on which a neuronal simulator (Promethe, see Gaussier and Zrehen [18], Matthieu et al. [33]) runs the control model. To test the validity of our robotic platform, we chose to evaluate RobotSoC with our simplest PerAc architecture for navigation based on a single sensorimotor loop.

We first describe how neuronal networks are simulated in this work and after a brief overview of the neuronal control architecture, we will detail each part of it. We will end showing how this model can control a mobile robot engaged in a visual navigation task: a homing behavior.

6.1 Neuronal framework

We describe here how neural networks are simulated by our neuronal simulator *Promethe*.[3] All neurons modeled in this work are based on rate coding. Neuron activities are coded by floating point values normalized between zero and one. Neuronal networks are simulated with discrete time step t. At each simulation time step, activity of all neurons is calculated. Next in the same simulation time step, a learning phase occurs to modify synaptic weights of connections. In this paper, our architecture can involve several time scales. Time scales behave here like imbricated loops. So, several time steps of the deepest time scale have to be performed for each single time step of the upper scale.

[2] A robulab from the Robosoft company equipped with an additional computer based on a I5 processor.

[3] More information on how the neuronal simulator works can be found in [18, 33].

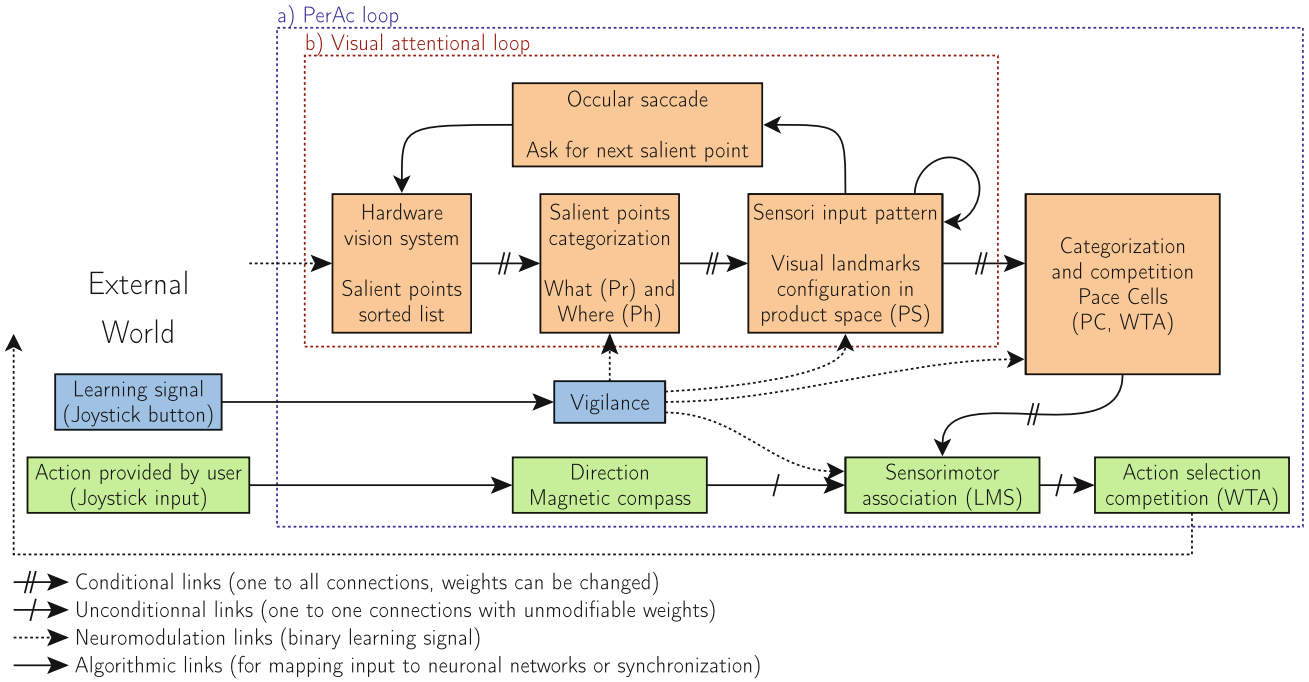


Fig. 20 A PerAc architecture to control RobotSoC. **a** PerAc loop to learn sensorimotor units. This architecture runs on a robot equipped with a pan camera that can take a full panorama of the visual scene. Each *block* represents one or more neural networks (specified in *parenthesis*) performing a specific treatment on the information flow. PerAc architecture aims to allow online learning of sensorimotor associations between a reflex motor pathway (in *green*) and a categorization pathway (in *red*). **b** Visual attentional loop. The spatial landmark configuration results from a visual attentional process performed at a finer time scale than the neural learning. This time scale corresponds to the frame rate of the images provided by the hardware visual system. It loops over all images corresponding to the same panorama. The attentional mechanism allows to categorize both the identity of the landmark (what) and its angular position related to the magnetic north (where). Each landmark focused during this process is accumulated into a product space matrix (PS) which codes the landmark configuration extracted from the visual scene. Activity of the product space neural network is reseted at the beginning of each visual scene exploration (new panorama)

In this neuronal architecture, learning is a one shot process driven by a specific signal named vigilance (see Fig. 20). This binary global signal is distributed on most of our neuronal networks (see Fig. 20). If this signal is equal to zero, the weight values of synaptic connections of all neurons remain unchanged. When set to one, it first triggers the recruitment of a new neuron dedicated to learning the current input and second enables the learning. Recruitment is done sequentially taking the next unused neurons in the network. Other neurons do not modify their synaptic connections weight. We use in the learning equation of each neuronal network the following binary term V_k^{NN} to model this filtering effect that controls which neuron should learn. For instance, in each neuronal network, when the global vigilance signal is equal to one, V_k^{NN} is also equal to one only if k is the index of the recruited neuron in the network NN and is equal to zero otherwise.

6.2 Neuronal robot control architecture overview

The presented architecture aims to allow a robot to perform a simple visual navigation (homing behavior). Visual input is provided by a pan mounted camera that can take a panorama of the environment surrounding the robot.

We develop our model following the PerAc concept (see Fig. 20a). PerAc architecture allows online learning of sensorimotor associations between a motor pathway (motor order) and a categorization pathway (localisation). These sensorimotor elements are the building blocks of the robot behavior. They code the action to perform in a given perceptual situation (location). The current location results from an active visual scene analysis (visual attentional mechanism, see Fig. 20b) which extracts and categorizes salient features (landmarks) of captured images. This learning step uses several neural networks along the categorization pathway of our model. A neural network (PS) merging detected landmarks and their azimuth in the scene is fed by this visual attentional mechanism. Activity of PS neural network formed a visual sensory input that is specific of a given robot location in the environment.

Categorization pathway of our model ends with a neural network (Place cell) that learns this visual sensory input. Thus, place cells neurons have an activity that is correlated with the spatial position of the robot in the environment.

Finally, the sensorimotor association block in Fig. 20 drives the robot behavior. This neural network codes the localisation information which will be bound with an expected motor action (direction) coming from the reflex pathway.

In this simple model, robot behavior learning is supervised by a user. Sensorimotor elements driving the robot behavior result from a classical conditioning operation, linking desired action (maintain a course) with a specific robot position. Hence the user, at first, has to place the robot at a chosen location. The robot is then stopped once it is in the required orientation (desired action). The reflex pathway unconditionally links the direction (orientation of the robot) to the sensorimotor neural network.

The user then sets a vigilance signal for one (neural) simulation time step allowing the model to learn the new sensory input pattern corresponding to the current location. A new neuron on the place cell neural network is recruited to categorize the spatial configuration of the visual landmarks detected in the entire panorama and preprocessed by the hardware visual system. Then, sensorimotor neurons learn to associate the recruited place cell neuron that predicts the current location, with the action provided by the user (current orientation of the robot). These operations are performed as many times as needed to define the desired robot behavior. For example, in our homing behavior test, only four of these sensorimotor units have to be learned.

In testing phase, no learning is allowed (no vigilance signal) and sensory input patterns directly activate previously learned Place Cells according to the current visual scene perceived. Contrary to the learning phase, panorama of the visual scene is taken while the robot is moving. The best recognized place cell, which has won the competition and thus best predicts the current location, triggers the previously learned activities on the sensorimotor neural network. Hence, the categorization pathway can take control of the robot's action and avoid the reflex pathway via the Winner Takes All (WTA) competition performed by the Action selection neural network.

6.2.1 *Visual attentional mechanism*

Categorization pathway of this PerAc architecture leads to the creation of a visual sensory inputs that is inspired by the mammal visual processing. Indeed, observations of mammal visual processing have led to the identification of two main pathways: the *What* and the *Where* [23]. The first allows identifying the characteristic points (landmarks) found in the retinal image and the second gives information on their locations in this image (azimuth). Based on these findings and following a PerAc approach, we have chosen in our model to define the visual sensory input that results from visual processing as the merge of these two kinds of information (What and Where).

As mentioned before this neuronal robot controller is driven by visual input coming from RobotSoC hardware architecture (RobotSoC vision system). RobotSoC vision system provides a sorted list of salient points (landmark candidates) to our neuronal model which processes each of these salient points one after the other via an attentional mechanism. This attentional mechanism allows to categorize both the identity of the landmark (what) and its angular position related to the magnetic north (where) (see Fig. 20b). Each landmark focused during this process is accumulated into a product space matrix (PS) which codes the landmark configuration extracted from the visual scene.

Thus, this attentional mechanism focuses on salient points extracted from the scene by the vision system. For each of these salient points, two processes then occur in parallel:

- a categorization allowing to code for the corresponding landmarks (pixels of a small local views: see Sect. 6.2.2).
- an angular position relative to the north, given by a compass, is computed for this point (see Sect. 6.2.3). This angle is coded on a neural population and a gaussian diffusion is used to allow generalization.

Fusion of these two streams of information allows to code the spatial landmark configuration via a constellation of landmarks with their azimuths (see Sect. 6.2.4). A simple feedback inhibition allows then to select the next salient point (ocular saccade). The whole process can thus be seen as a spotlight mechanism based on a visual attentional process.

This process must be performed at a finer time scale than the rest of the neural networks, since the attentional mechanism must loop over all images corresponding to the same panorama. This time scale corresponds to the frame rate of the images provided by the hardware visual system. At the end of this attentional process a maximum of (N) of the most salient landmarks are recruited. The number of visible landmarks needed is a trade-off between the robustness of the algorithm and the computational and memory cost of the process. In theory, if all landmarks in the visual scene are fully recognized (without perceptual aliasing), only three of them are needed to code a unique location (triangulation). As some of them may not be recognized in practice, for example in case of changing condition like occlusion, we need to take into account more landmarks to guarantee the robustness [18]. Activity of the product space neural network is reset at the beginning of each visual scene exploration (new panorama).

6.2.2 *What: landmark identity layer (Pr)*

The landmark layer is composed of neurons that each codes for a small local view around one salient point. This layer

models the "what" information that might be coded in the perirhinal cortex (Pr) (or in other areas of the ventral visual pathway of the temporal cortex [29]). Pr neurons are all connected to small local images provided by the hardware visual system via alterable one to all connections.[4] Weight modification of link coming from a neuron k of this layer are done according to the following rule:

$$\Delta W_{k,ij}^{\mathrm{Pr}} = I_{ij}(t).V_k^{\mathrm{Pr}} \tag{5}$$

$W_{k,ij}{}^{\mathrm{Pr}}(t)$ is the weight of the link from pixel i, j to the k^{i}th landmark neuron. $I_{ij}(t)$ is the intensity value of pixel (i, j) from the small local view I at time t. Initially $W_k{}^{\mathrm{NN}} = 0$. As stated before, the vigilance signal triggers the recruitment of a new neuron in the network. This neuron is the only one that can learn in this network, since $V_k{}^{\mathrm{Pr}} = 1$ only when k is the index of the recruited neuron, and $V_k{}^{\mathrm{Pr}} = 0$ otherwise.

Activity $X_k{}^{\mathrm{Pr}}(t)$ of the $k$ith landmark neuron at simulation time step t is computed according to the following equation:

$$X_k^{\mathrm{Pr}}(t) = f^{\mathrm{RT}}\left(1 - \frac{1}{N_I.M_I}\sum_{i,j=1}^{N_I,M_I} \|W_{k,ij}^{\mathrm{Pr}}(t) - I_{ij}(t)\|\right) \tag{6}$$

with N_I and M_I the number of pixels on X and Y of the corresponding small local view. $W_{k,ij}{}^{\mathrm{Pr}}(t)$ is the weight of the link from pixel i, j to the k^{i}th landmark unit. $I_{ij}(t)$ is the value of the ij^{i}th point of the small local view. $f^{\mathrm{RT}}(x) = \frac{1}{1-\mathrm{RT}}[x - \mathrm{RT}]^+$ is an activation function that extends the dynamical range of the output. RT is a Recognition Threshold. $[x]^+ = x$ if $x \geq 0$ and 0 if not. More details on the impact of this soft competition can be found in [22].

6.2.3 Where: landmark azimuth layer (Ph)

The parahippocampal layer (Ph) uses a neural population to code the azimuth of the focus point (absolute direction coming from the magnetic compass of our robot). No learning is performed on this layer since it directly maps analog compass values in a neural population. More precisely, the azimuth of a landmark $\theta(t)$ is calculated by applying the magnetic compass value to the center of the image and by shifting this value according to the $\times$ position of the landmark relative to the center in the image and the horizontal opening angle of the camera used (θ_c):

$$\theta(t) = \theta_{\mathrm{compass}}(t) - (X_{\max}/2 - x) * \theta_c/X_{\max} \tag{7}$$

with $X_{\max}$ the horizontal resolution of the image used.

Each of the N_{Ph} neurons of Ph (with activity $X_i{}^{\mathrm{Ph}}$) has a preferred direction for which it fires with maximal rate. Its firing rate monotonously decreases from one to zero with the angular distance between its preferred direction and the direction of the current focus point $\theta(t)$. Activity on Ph is given by:

$$X_i^{\mathrm{Ph}}(t) = f\left(\|2.\pi.\frac{i}{N_{\mathrm{Ph}}} - \theta(t)\|\right) \tag{8}$$

with f a lateral diffusion around the neuron X_k^{NN} such that activity decreases for near angles (f is a Gaussian function). Sigma value of this gaussian fixes the lateral diffusion size.

6.2.4 Visual sensory input: Product Space layer (PS)

The merging of "what" (landmark unit) and "where" (azimuth) information is performed in a sigma-pi neural network called Product Spacer (PS).[5] Neurons on PS remain active until all small local views around each salient point have been explored (accumulation). Neurons of this network are arranged in a matrix. The number of lines of this matrix is equal to N_{Pr} (hence they share the same index i), the number of neurons in Pr and the number of columns is set to five, meaning it can only be five different orientations for a given landmark (given by index l). Neurons in this matrix thus have an activity noted $X_{il}{}^{\mathrm{PS}}(t)$.

PS is connected to Pr via one to neighborhood connections with fixed weight values equal to one. Hence, an active neuron on the landmark layer pre-activates five neurons in PS (the five neurons corresponding to the five possible azimuth under which the robot could see this landmark). PS is connected to Ph via one to all connections.

Activity on PS is computed in three steps. First, it finds maximum activity coming from the $i$ith neuron of Pr (X_k^{NN}): $\max_{i\in N_{\mathrm{Pr}}} X_i^{\mathrm{Pr}}(t).W_{il,i}^{\mathrm{Pr-PS}}(t)$. Then, we determine the maximum of all activities coming from the $j$ith neuron of Ph (X_k^{NN}): $\max_{j\in N_{\mathrm{Ph}}} X_j^{\mathrm{Ph}}(t).W_{il,j}^{\mathrm{Ph-PS}}(t)$ where N_{Pr} is the number of neurons in Pr and N_{Ph} that of Ph.

In a second step, the product P_{il} of these two activities is computed by:

$$P_{il}(t) = \left(\max_{i\in N_{\mathrm{Pr}}} X_i^{\mathrm{Pr}}(t).W_{il,i}^{\mathrm{Pr-PS}}\right).\left(\max_{j\in N_{\mathrm{Ph}}} X_j^{\mathrm{Ph}}(t).W_{il,j}^{\mathrm{Ph-PS}}\right) \tag{9}$$

The last step, accumulates the P_{il} product with previous product terms coming from already processed landmarks in the same visual scene:

$$X_{il}^{\mathrm{PS}}(t+1) = \left[X_{il}^{\mathrm{PS}}(t) + P_{il}(t)\right]^+ \tag{10}$$

[4] A neuron of this layer is connected to all the pixels of a small local image.

[5] The merge may be performed in the superficial layer of the entorhinal cortex or in the postrhinal cortex.

A PS neuron learns to be activated when a landmark is recognized under a given angle. This activity gradually decreases for near angles. Activity of all PS neurons are forced to zero at the beginning of the attentional loop when a new panorama is processed.

Learning is only performed on the weights between Ph and PS. The weight is maximal (equal to one) for the angle under which the corresponding landmark was learned. Weights learning is the following:

$$W_{il,j}^{\text{Ph}-\text{PS}}(t) = \left(X_{il}^{\text{Pr}}(t)\right).\left(X_j^{\text{Ph}}(t)\right).V_{il}^{\text{PS}}(t)$$
$$i = \text{argmax}_{p\in N_{\text{Pr}}}\left(X_p^{\text{Pr}}(t)\right)$$
$$j = \text{argmax}_{q\in N_{\text{Ph}}}\left(X_q^{\text{Ph}}(t)\right) \quad (11)$$

Once again, learning on $W_{il,j}^{\text{Ph-PS}}$ and recruitment of a new neuron is only performed when the vigilance signal equals one ($V_k^{\text{NN}} = 1$). Initially $W_{il,j}^{\text{Ph–PS}} = 0$. For a given recognized landmark i a new neuron at index il is recruited sequentially along the index l of the PS matrix column.

The spatial landmarks constellation on PS, resulting from the visual input process, characterizes one location. We use a neural network modeling "Place Cells" (PC, see Sect. 6.3) to learn the activity pattern on PS.

6.3 Robot localization: place cell layer (PC)

In our model, a PC neuron learns to categorize a particular pattern of activity on the PS and hence a particular location. Each PC neuron is linked with all neurons of the PS via one to all connections that follow a Hebbian like learning rule:

$$\frac{\text{d}W_{k,il}^{\text{PS}-\text{PC}}(t)}{\text{d}t} = V_k^{\text{PC}}(t).X_{il}^{\text{PS}}(t).X_k^{\text{PC}}(t) \quad (12)$$

Recruitment of a new neuron for encoding a new location occurs when the neuromodulation vigilance input is set to one. For this new neuron recruited at time t its activity is set to the maximum value $X_k^{\text{PC}}(t) = 1$ and $V_k^{\text{PC}}(t)$ is set to one and zero otherwise.

The activity of the kth PC results from the computation of the distance between the learned and the current PS activity pattern and is expressed as follows:

$$X_k^{\text{PC}}(t) = \frac{1}{W_j}\left(\sum_{il}^{N_{\text{PS}}} W_{k,il}^{\text{PS}-\text{PC}}.X_{il}^{\text{PS}}(t)\right) \quad (13)$$

with $W_j = \sum_{il}^{N_{\text{PS}}} W_{j,il}^{\text{PS}-\text{PC}}$.

Neurons of this layer models Place Cells (PC) discovered in mammal brains. Place cells have a firing pattern strongly correlated with a particular location in the environment: they exhibit a high firing rate when the animal is at a given location and are more silent when the animal is somewhere else [35]. The PC layer of our model is then able to characterize and so recognize different places in the environment: if the robot is at the exact position where the PC has been learned, its activity is maximal. When the robot move from this position, the activity of this PC decreases according to the distance between the learned position and the current one. Hence a PC keeps a certain amount of activity around the learned position that corresponds to the place field of the PC. Place field is the projection in the environment of the locations where a given PC fires. Generalization of sensorimotor units is an interesting property of this model exploited by the homing navigation strategy. A competitive mechanism (Winner Takes All) is then used to select the place cell that best recognizes the current place.

6.4 Orientation

This layer codes the current direction the robot is pointing to. When the user moves the robot, the corresponding direction is updated. This orientation neural network is a simple mapping of the magnetic compass value expressed with the same neural population code (describing the absolute direction to follow at constant linear speed) as the sensorimotor layer.

6.5 Sensorimotor association layer (SM)

Following the PerAc sensorimotor architecture [18, 32] we define the robot perception of the goal place as a dynamical process linking the sensations (landmark configuration coded by PC) and their corresponding action. In our model, neurons of the sensorimotor association neural network learn to associate the winning place cell to an action (a direction to follow in our case). Sensorimotor elements thus defined can be used to perform a robust homing navigation behavior.

Each sensorimotor neuron is linked to both the PC network via learned connections and the orientation network trough constant connections. The weights of these last links are fixed to a lower value than those coming from the categorization neural networks (after learning), so that activities coming from this later network can override orientation network ones, after a sensorimotor association has been learned.

The activity $X_j^{\text{SM}}(t)$ of the jth neuron of this network at simulation time step t is computed by:

$$X_j^{\text{SM}}(t) = (1 - V_j^{\text{SM}}(t)).M_j^{\text{SM}}(t) + V_j^{\text{SM}}(t).S_j^{\text{SM}}(t)$$
$$S_j^{\text{SM}}(t) = X_k^{\text{PC}}(t).W_{j,k}^{PC-SM}$$
$$M_j^{\text{SM}}(t) = X_j^{\text{Orientation}}(t).W_{j,j}^{\text{Orientation}-\text{SM}} \quad (14)$$

Learning in the SM layer follows a Hebbian learning rule:

$$\frac{\text{d}W_{j,k}^{\text{PC}-\text{SM}}(t)}{\text{d}t} = V_j^{\text{SM}}.X_k^{\text{PC}}(t).X_j^{\text{SM}}(t) \quad (15)$$

In a homing behavior, the associated action is the direction to follow to reach the goal from the current place. Thus, learning four sensorimotor associations pointing to the goal is enough to allow the robot to navigate to this place. These four sensorimotor associations forms an attraction basin centered on the goal location (see 24 in "Appendix").

Even if this navigation strategy relies on very basic sensorimotor learning it nevertheless allows robust behavior in open indoor and outdoor environment [18, 2021, 22].

Note that when navigational tasks must be performed in more complex environments (ie: multi-rooms and corridor, mazes), the model can easily be extended to rely on sequence of sensory–motor units (linking PC and action see Giovannangeliet al. [22]) or on planification using a topological map linking sensory–motor units describing paths [13, 25]. Our approach of the perception relying on dynamical sensorimotor attraction basins has also been applied to object recognition [32].

6.6 Action selection layer (WTA)

This neural network is composed of a simple Winner Takes All that selects a single action from those proposed by the sensorimotor layer trough one to one links. Like in the SM network, each neuron represents an action which corresponds to the absolute orientation of the motor order the robot must perform at constant speed.

7 Behavioral results

After giving a short overview of the experimental setup, we then discuss the results.

7.1 Experimental setup for indoor navigation

In this work, we consider a robotic platform based on a Robulab from Robosoft with 8 IR sensors, that embeds a PC (Intel I5) used to implement the neural network that controls the robot, a smart camera (vision system based on FPGA) for the visual processing system, and a magnetic compass to get an absolute orientation information (see Fig. 21).

We choose to evaluate our control architecture on a simple homing behavior. Experiments are performed in two phases. First, the robot learns four Place Cells (PC) around the goal place. Each PC is spaced of 2.4m from its two neighbors and 1.70 m of the goal center. The robot is passively guided by the experimenter via a joystick to one of the location to learn and is then oriented toward the goal location. Learning of landmark, PrPh neurons, PC neuron and the associated sensory–motor unit (action) is supervised (a vigilance signal is triggered ($V_k = 1$) when the experimenter press a joystick button). When a new sensorimotor has to be learned ($V_k = 1$) a full panorama of the surrounding is performed ($N_{\text{pictures\_learning}}$ are taken over 270°). Second, the robot is placed at different locations in the environment and the joystick of the experimenter is disconnected.

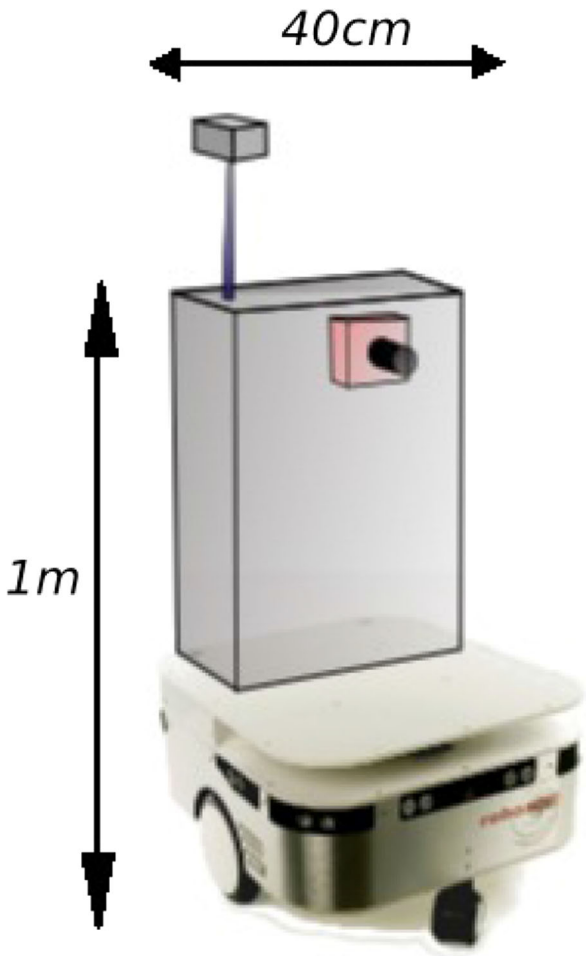


Fig. 21 Robotic setup. We use a Robulab robot from Robosoft with our hardware visual system, a magnetic compass and an embedded I5 PC. The robot is 40 cm large and 1 m height

Panorama of the surrounding is performed with less pictures (N pictures are taken over 270°) when the robot is in recognition phase. Hence, the robot vision system grabs several pictures of the surrounding and extracts log polar images of the 16 most salient points found in the images. This information, together with their absolute orientations, are sequentially processed and then integrated by the neural architecture that finally results in activations of neurons of the PC layer. Activity of these cells codes for their degree of recognition of the current scene. A competition mechanism allows to select the best recognized PC and trigger the corresponding (previously learned) movement. At this

Table 1 Parameters of the presented system

Parameter	Value
Image size	480 × 270
$N_{\text{extracted\_keypoints}}$	16
N_l	16
N_m	16
$N_{\text{pictures\_learning}}$	16
$N_{\text{pictures\_recognition}}$	8
DoG: σ_1	7
DoG: σ_2	2
DoG: size	16
RT	0.80
N_{Ph}	360
N_{Pr}	3,000
PS size	N_{Pr}*5
ϵ	0.2

time, the behavior of the robot (the direction it chooses to follow) is only the expression of the best-recognized sensory–motor unit.

In order to test the performance of the model, we choose to record the trajectory of the robot from eight starting points around the goal place.

First, we investigate, if the learned sensory–motor association was correct by placing the robot around 60 cm behind each of the four learned PC. Notice that the robot was not placed at the exact location where the PC were learned, but in the same alignment in respect to the goal.

Second, we test the generalization property of the sensory–motor units by placing the robot at 4 other starting points located between the place cells (at the frontiers of their place fields).

In all the experiments we use the following parameters: Table 1.

Position measurement precision is 10 cm.[6]

7.2 Results

Figure 22 shows the results of the first experiment. The sensory–motor associations are correctly learned and allow the robot to reach the goal when the robot is placed closed to one of the learned PC. Each trajectory (excepted from PC4) only implies the recognition of a single place cell. The trajectory starting from PC4 shows a brief recognition of PC3. This can be explained by the fact that PC4 was learned close from a desk and thus relies on proximal landmarks. Proximal landmarks imply strong angular variation while a relatively small displacement is performed. This effect impacts the generalization property of this PC. These four PC formed an attraction basin similar to the one of Fig. 24c in "Appendix".

Figure 23 shows the trajectory performed by the robot starting at four different places located near the frontiers of the four place fields of the PC learned (second experiment).

The trajectory is composed of different phases in which the robot follows the direction of the best recognized PC at each time. At a given point, best activated PC changes and so do the path followed by the robot. Trajectories performed by the robot are similar to the one of Fig. 24b. This behavior allows to underline changes in PC recognition but is not suitable for our final robot behavior. Obtained trajectories could be enhanced by summing vectorially the response of the three best recognized place cells (neighbor PC).

Figure 25 in "Appendix" shows the activity of each PC along the last four trajectories.

These experiments underline the interest of the generalization of these sensorimotor units as it is not necessary to visit all the possible locations to derive a possible action in the environment. In this experiment, each place field radius is around 1.2m. Only four sensory–motor associations have to be learned to cover an area greater than $16m^2$ allowing the robot to reach the goal. The size of the place field (from which directly depends the size of the attraction basin) depends on the distance of the landmarks. One can refer to [22] for a more detailed study of the generalization property of the model in indoor and outdoor environments.

8 Conclusion

Vision processing in the context of embedded robotics is a great challenge both for real-time exploration of the environment and for interactions with other agents or humans. This constraint on the system behavior affects the design of the different parts composing such robots. This matter is addressed in this paper as a joint method coupling local optimization of intensive computing and global design of the perception/control couple as a regulation loop. Meeting the embedded and real-time computation challenge needed by vision processing requires an optimized design that leads us to propose a specific smart camera. Thus the approach presented in this paper allows to embed all these tasks together into an autonomous robotic platform. That were the challenges addressed and solved in this paper. The three main contributions enabling these results are:

[6] The tracking system used to plot trajectories is subject to local errors represented in figures by small jumps and discontinuities. Some videos of the experiments are available at the following address: http://www-etis.ensea.fr/robotsoc

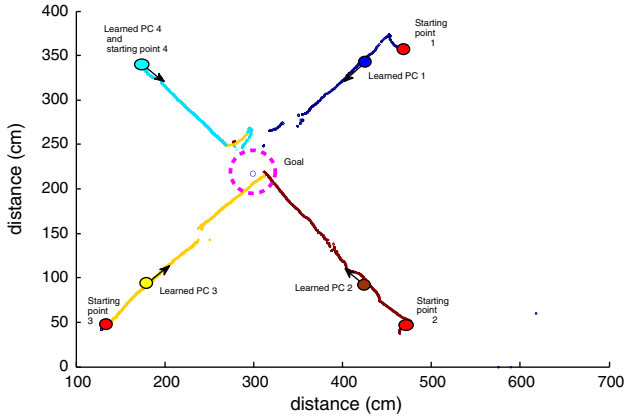


Fig. 22 Result of the homing from 4 starting places located behind the learned PC. Starting point of each outward trip is marked by a *red dot*, except for the 4th. Due to limitation on the environment size, the 4th starting point is the same as the location of the PC4 (since it is close of a desk). A *color circle* and an *arrow* show the 4 places where the PC were learned and their associated movement. *Color* of the corresponding recognized PC is superimposed to the robot trajectory. Trajectories from PC 1, 2 and 3 only rely on a single PC. The *dashed red circle* (50 cm diameter) symbolizes the goal place to reach. Note that jumps are only due to position error of the tracker used to plot the trajectories

- a globally coherent algorithmic method linking sensation to action via a visual attentional model,
- a hardware optimization of the vision attentional model into a single System-on-Chip reaching real-time exigencies. This hardware architecture is designed with its HD CMOS sensor. We are currently working on a specific board in order to build an optimized smart camera, with small form factor. The hardware architecture is already available as an open access IP for several FPGA evaluation boards.
- an embedded platform merging the proposed hardware smart camera and the software neural architecture. This part of the platform is not only responsible for sensation/action learning, but also for integration and fusion of vision with other sensors such as compass.

All these points together bring to the robot autonomous and reactive capabilities important for the generalization of this platform to other types of missions. For example, we are currently working on the application of this platform to missions alternating sequentially navigation and object tracking. Thus the exigencies and the level of details of the visual task become variable along time. We are working both on a dynamically configurable smart camera and on the associate control loop from the neural architecture according to the robot's state.

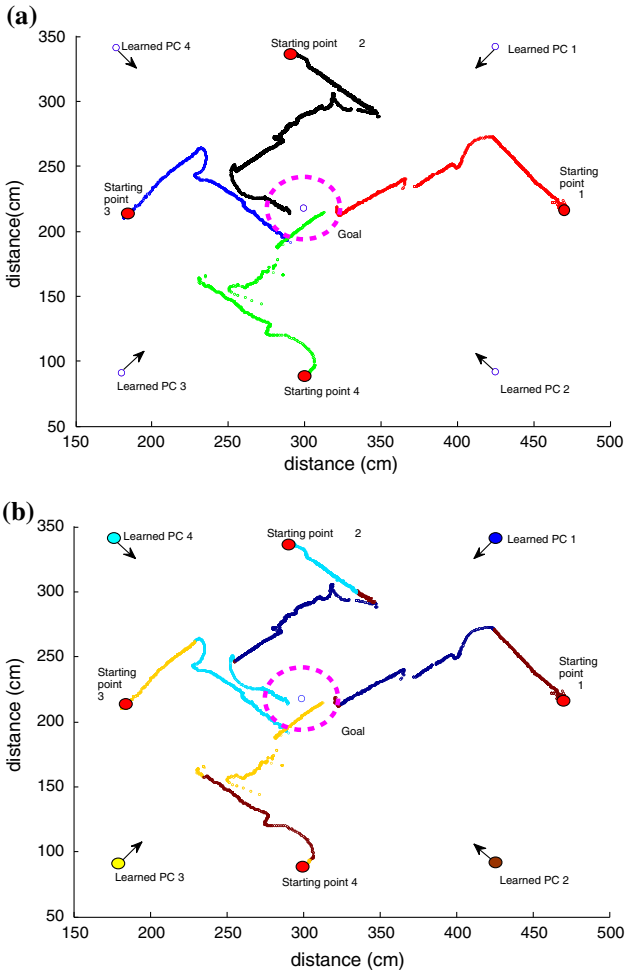


Fig. 23 Result of the homing from 4 starting places located near the frontiers of the four place fields of the PC learned. Starting point of each outward trip is marked by a *red dot*. **a** To avoid confusion, each trajectory is printed with a single *color*. **b** A *color circle* and an *arrow* show the 4 places where the PC were learned and the associated movement. *Color* of the corresponding recognized PC is superimposed to the robot trajectory. Each trajectory involves several PC recognition. The *dashed red circle* (50 cm diameter) symbolizes the goal place to reach. Due to limitation on the environment size, the 4th starting point is the same as the location of the PC4 since it is close of a wall. Note that jumps are only due to position error of the tracker used to plot the trajectories

Appendix

Open access design files

The FPGA-based vision architecture can be freely downloadable for several platforms[7]

- Altera DE2-115 board equipped with D5M camera,
- Xilinx Zynq ZC702 board equipped with the On-semi camera.

[7] http://www-etis.ensea.fr/robotsoc.

The design files contain:

- the configuration file of the target FPGA (respectively, Altera Cyclone IV 115kLE, and Zynq 7000),
- the flash image for the embedded processor (respectively, Nios-II and dual-core Cortex A9). This image contains the executable files that read back the features.

One can then send the extracted features through an Ethernet link to a distant computer or compute them locally.

Additional figures

See Figs. 24 and 25.

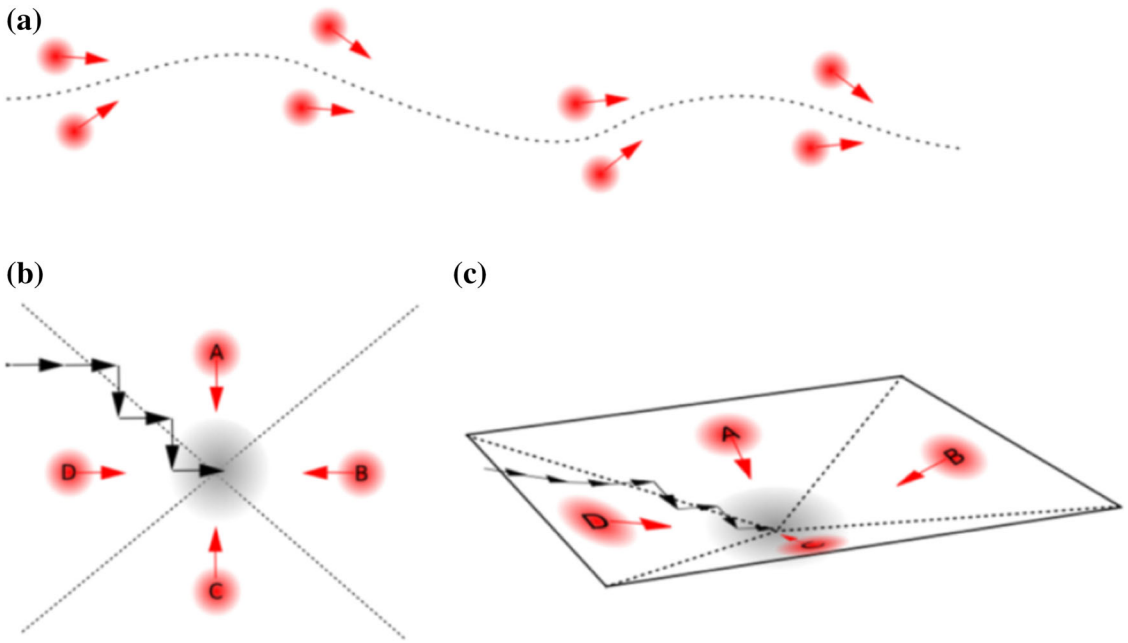
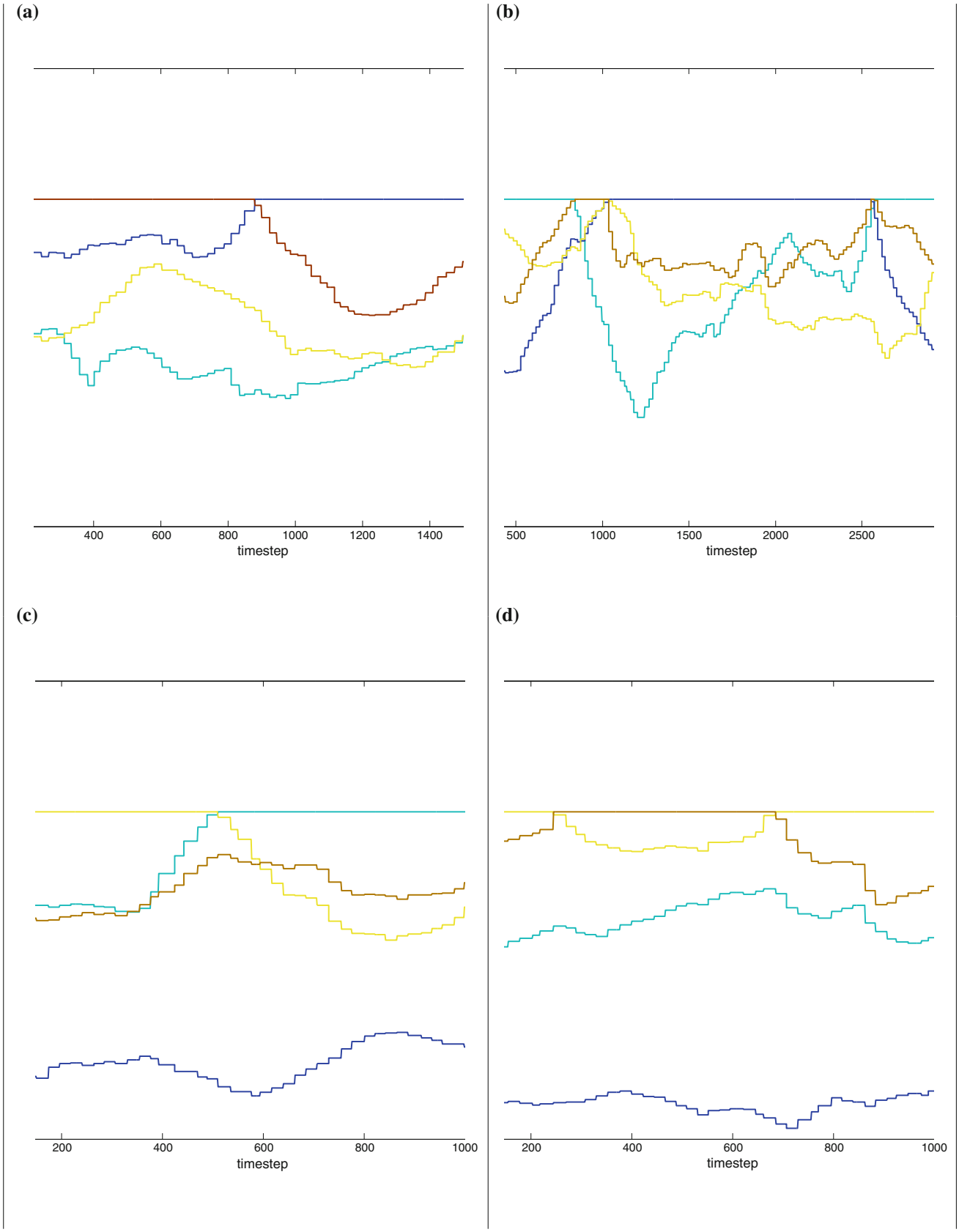


Fig. 24 Homing mechanism based on a simple sensorimotor loop relying on vision. A particular direction is associated with each winning PC. This association is learned by a least mean square algorithm. The system is then able to move in the learned direction when the associated place cell wins the recognition competition. This simple mechanism allows the system to exhibit robust behaviors by just a few sensorimotor associations (**a** path following, **b** homing task). An attraction basin (**c**) emerges from the sensorimotor dynamic


(a)
400
600
800
1000
1200
1400
timestep
(b)
500
1000
1500
2000
2500
timestep
(c)
200
400
600
800
1000
timestep
(d)
200
400
600
800
1000
timestep

◀**Fig. 25** Activity of the four PC along the four trajectories starting near the frontiers of the PC. The *color* code used is the same than in Fig. 23. Activity of the winning PC is forced to one. Activities of the other cells represent the similarity of the currently observed activity pattern on PrPh layer and the one when the corresponding PC were learned. **a** Activity of the PC layer on the trajectory from starting point 1. PC2 and then PC4 are successively recognized. **b** Activity of the PC layer on the trajectory from starting point 2. The sequence of activated PC is PC4, PC2 for a short period, then PC1 and PC4 again. **c** Activity of the PC layer on the trajectory from starting point 3. PC3 is first recognized then it is PC4. **d** Activity of the PC layer on the trajectory from starting point 4. PC2 and next PC4 are activated

References

1. ARM: Amba open specifications, the de facto standard for on-chip communication. (2013) http://www.arm.com/products/system-ip/amba/amba-open-specifications.php
2. Ballard, D.H.: Animate vision. Artif. Intell. **48**(1), 5786 (1991)
3. Battezzati, N., Colazzo, S., Maffione, M., Senepa, L.: Surf algorithm in fpga: a novel architecture for high demanding industrial applications. In: Rosenstiel, W, Thiele, L (eds.) DATE, IEEE, pp. 161–162 (2012)
4. Birem, M., Berry, F.: Fpga-based real time extraction of visual features. In: Conference (2010)
5. Bonato, V., Holanda, J., Marques, E.: An embedded multi-camera system for simultaneous localization and mapping. In: Proceedings of Applied Reconfigurable Computing, Lecture Notes on Computer Science (2006)
6. Bonato, V., Marques, E., Constantinides, G.A.: A parallel hardware architecture for scale and rotation invariant feature detection. IEEE Trans. Circ. Syst. Video Technol. **18**(12), 1703–1712 (2008). doi:10.1109/TCSVT.2008.2004936
7. Bouris, D., Nikitakis, A., Walters, J.: Fast and efficient fpga-based feature detection employing the surf algorithm. In: 2010 18th IEEE Annual International Symposium on, Field-Programmable Custom Computing Machines (FCCM), pp. 3–10 (2010). doi:10.1109/FCCM.2010.11
8. Burgess, N, O'Keefe, J.: Neuronal computations underlying the firing of place cells and their role in navigation. Hippocampus **7**, 749–762 (1996)
9. Burgess, N., Recce, M., O'Keefe, J.: A model of hippocampal function. Neural Netw. **7**(6/7), 1065–1081 (1994)
10. Cartwright, B.A., Collett, T.S.: Landmark learning in bees. J. Comp. Physiol. **151**, 521–543 (1983)
11. Cope, B.: Implementation of 2d Convolution on fpga, gpu and cpu. Tech. rep (2006)
12. Crowley, J.L., Riff, O.: Fast Computation of Scale Normalised Gaussian Receptive Fields. Springer Lecture Notes in Computer Science 2695 (2003)
13. Cuperlier, N., Quoy, M., Gaussier, P.: Neurobiologically inspired mobile robot navigation and planning. Fronti. NeuroRobot. **1**(1) (2007)
14. Farabet, C., Poulet, C., LeCun, Y.: An fpga-based stream processor for embedded real-time vision with convolutional networks. In: 2009 IEEE 12th International Conference on, Computer Vision Workshops (ICCV Workshops), pp. 878–885 (2009). doi:10.1109/ICCVW.2009.5457611
15. Frintrop, S., Jensfelt, P.: Attentional landmarks and active gaze control for visual SLAM. IEEE Trans. Robot. **24**(5), 1054–1065 (2008). doi:10.1109/tro.2008.2004977
16. Frintrop, S., Rome, E., Christensen, H.I.: Computational visual attention systems and their cognitive foundations: a survey. ACM Trans. Appl. Percept. **7**(1), 6:1–6:39 (2010). doi:10.1145/1658349.1658355
17. Gallistel, C.R.: The Organization of Learning. MIT Press, Cambridge (1993)
18. Gaussier, P., Zrehen, S.: Perac: a neural architecture to control artificial animals. Robot. Auton. Syst. **16**(24), 291320 (1995)
19. Gaussier, P., Joulain, C., Zrehen, S., Banquet, J.P., Revel, A.: Visual navigation in an open environment without map. In: International Conference On Intelligent Robots and Systems-IROS'97, pp. 545–550. IEEE/RSJ, Grenoble, France (1997)
20. Gaussier, P., Joulain, C., Banquet, J.P., Leprêtre, S,. Revel, A.: The visual homing problem: an example of robotic/biology cross fertilization. Robot. Auton. Syst. **30**, 115–180 (2000)
21. Gaussier, P., Revel, A., Banquet, J.P., Babeau, V.: From view cells and place cells to cognitive map learning: processing stages of the hippocampal system. Biol. Cybern. **86**, 15–28 (2002)
22. Giovannangeli, C, Gaussier, P., Banquet, J.P.: Robustness of visual place cells in dynamic indoor and outdoor environment. Int. J. Adv. Rob. Syst. **3**(2), 115–124 (2006)
23. Goodale, M.A., Milner, A.D.: Separate visual pathways for perception and action. Trends Neurosci.**15**(1), 20–25 (1992)
24. Heinke, D., Humphreys, G.: Computational models of visual selective attention: a review. Connect. Models Psychol. 273–312 (2005)
25. Hirel, J., Gaussier, P., Quoy, M.: Biologically inspired neural networks for spatio-temporal planning in robotic navigation tasks. In: 2011 IEEE International Conference on, Robotics and Biomimetics (ROBIO), pp. 1627–1632 (2011). doi:10.1109/ROBIO.2011.6181522
26. Itti, L., Koch, C., Niebur, E.: A model of saliency-based visual attention for rapid scene analysis. IEEE Trans. Pattern Anal. Mach. Intell. **20**(11), 1254–1259 (1998). doi:10.1109/34.730558
27. Judd, S.P.D., Collet, T.S.: Multiple stored views and landmark guidance in ants. Nature **392**, 710–712 (1998)
28. Koch, C., Ullman, S.: Shifts in selective visual attention: towards the underlying neural circuitry. Hum. Neurobiol **4**, 219–227 (1985)
29. Kolb, B., Tees, R.: The Cerebral Cortex of the Rat. MIT Press, Cambridge (1990)
30. Lindeberg, T.: Feature detection with automatic scale selection. Int. J. Comput. Vis. **30**(2), 79116 (1998)
31. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. Int. J. Comput. Vis. **60**(2), 91110 (2004)
32. Maillard, M., Gapenne, O., Hafemeister, L., Gaussier, P.: Perception as a dynamical sensori-motor attraction basin. In: Proceedings of the 8th European Conference on Advances in Artificial Life (ECAL 05), vol. 3630, p. 3746 (2005)
33. Matthieu, L., Pierre, A., Philippe, G.: Distributed real time neural networks in interactive complex systems. In: Proceedings of the 5th International Conference on Soft Computing as Transdisciplinary Science and Technology. ACM, New York, NY, USA, CSTST '08, pp. 95–100 (2008). doi:10.1145/1456223.1456247
34. Mikolajczyk, K., Schmid, C.: Scale & affine invariant interest point detectors. Int. J. Comput. Vis. **60**(1) (2004)
35. O'Keefe, J., Nadel, N.: The Hippocampus As a Cognitive Map. Clarenton Press, Oxford (1978)
36. Oliva, A., Torralba, A., Castelhano, M.S., Henderson, J.M. Top-down control of visual attention in object detection. In: Proceedings of the IEEE Int'l Conference on Image Processing (ICIP '03) (2003)
37. Ouerhani, N., Hügli, H.: Robot self-localization using visual attention. In: CIRA, IEEE, pp. 309–314 (2005)
38. Pham, P.H., Jelaca, D., Farabet, C., Martini, B., LeCun, Y., Culurciello, E.: Neuflow: dataflow vision processing system-on-a-chip. In: International Midwest Symposium on Circuits and Systems (MWSCAS'12) (2012)

39. Schaeferling, M.: Flex-surf: A flexible architecture for fpga based robust feature extraction for optical tracking systems. In: Conference on Reconfigurable Computing and FPGAs (2010)
40. Siagian, C., Itti, L.: Biologically inspired mobile robot vision localization. IEEE Trans. Robot. **25**(4), 861–873 (2009)
41. Tinbergen, N.: The Study of Instinct. Oxford University Press, London (1951)
42. Treisman, A.M., Gelade, G.: A feature-integration theory of attention. Cogn. Psychol. **12**(1), 97–136 (1980). doi:10.1016/0010-0285(80)90005-5
43. Tsotsos, J.: Analyzing vision at the complexity level. Behav. Brain Sci. **13**(3), 423–469 (1990)
44. Verdier, F., Miramond, B., Maillard, M., Huck, E., Levebvre, T.: Using high-level rtos models for hw/sw embedded architecture exploration: case study on mobile robotic vision. EURASIP J. Embed. Syst. (2008)
45. Zhong, S., Wang, J., Yan, L., Kang, L., Cao, Z.: A real-time embedded architecture for sift. J. Syst. Archit. **59**(1), 16–29 (2013). doi:10.1016/j.sysarc.2012.09.002
46. Zipser, D.: Biologically plausible models of place recognition and goal location. In: McClelland, JL., Rumelhart, D.E (eds.) Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol 2. MIT Press, Cambridge, MA. pp. 423–470 (1986)

Laurent Fiack received Master degree in Autonomous System Electronics from the University of Cergy-Pontoise and an Electrical Engineering, Computer Science and Telecommunications Degree from the Ecole Nationale Suprieure de l'Electronique et de ses Applications specialized in Electronics and Embedded Systems, both in 2012. He is currently a Ph.D. student working on a self-adaptive, reconfigurable robot controller at the University of Cergy-Pontoise. He is a member of the Architecture team at information processing and system Lab (ETIS). His research interests include hardware architecture, microprocessor systems, Algorithm Architecture Adequacy, machine vision and robotics.

Nicolas Cuperlier was born in 1979 in Reims, France. He received M.S. degree in Image and Signal Processing from Cergy-Pontoise University in 2003. In 2006, he received a Ph.D. degree in computer sciences from the same university for a work on autonomous mobile robot navigation inspired by mammals neural processing. In 2007, he conducted research in Neural Network modeling Attentional Vision mechanisms at the The Computer Sciences Laboratory for Mechanics and Engineering Sciences (LIMSI). He is now assistant professor at the Cergy-Pontoise University in the neurocybernetic team of the information processing and system Lab (ETIS) where he uses robots as tools to study, in collaboration with neurobiologists, different cognitive models from an ecological and developmental perspective. Currently, his work focuses on the modelization of the cognitive mechanisms involved in visual perception, motivated navigation, action selection and the role of emotion in behavioral metacontrol.

Benoît Miramond received M.S. degree in Architecture of Integrated Circuits and Micro-Electronics in 2000 from the Pierre et Marie Curie University. In 2003, he received a Ph.D. degree in computer sciences from University of Evry Val d'Essonne for his work on design space exploration for data-flow applications on dynamically reconfigurable architectures. He conducted research in real-time scheduling at the national institute for research in computer science and control (INRIA) in 2004. He is now assistant professor at the Cergy-Pontoise University in the information processing and system Lab (ETIS). He is scientific animator of the Architecture team of the ETIS Lab. Currently, his work focuses on bio-inspired hardware architectures, self-organization in grid of processors and smart cameras for artificial vision. Recently, he integrated these different research subjects in the approach called Embodied Computing at the frontier of digital architectures and neurosciences.