

Co-design of Fast Biologically-plausible Vision-based Systems for Controlling the Reactive Behaviors of Mobile Robots

R.Zapata, P.Lépinay, D. Roman Ocampo

LIRMM (UMR 5506 CNRS - UM2)
Université Montpellier II
161, rue Ada
34392 Montpellier Cedex 5
zapata@lirmm.fr

Abstract

This paper addresses the co-design of biologically-plausible vision-based algorithms applied to control the reactive behaviors of mobile robots and especially, the control of collision avoidance in unknown and dynamic worlds. Our main tools are Programable Logic Devices like FPGA (Field Programmable Gate Arrays) to implement real-time Perception/Actions schemes.

Contents

1	Introduction	4
2	Biologically-plausible reactive behaviors	6
2.1	Vision-based reactive systems	6
2.2	Collision avoidance and target pursuit	6
2.2.1	General principle	6
2.2.2	Mathematical basis	7
2.2.3	Sensor-based control algorithm	10
2.2.4	Links with vision	11
3	Vision-based control algorithms	11
3.1	3D reconstruction	11
3.1.1	Spatial stereovision	12
3.1.2	Temporal stereovision	13
3.1.3	The complete algorithm	13
3.2	Image segmentation	14
4	Simulation	16
4.1	Simulation of 3D reconstruction	16
4.1.1	Spatial stereovision	16
4.1.2	Temporal stereovision	16
4.2	Simulation of image segmentation	18
5	Implementation of fast vision-based control algorithms	19
5.1	Design flow	20
5.2	General architecture and co-design	20
5.3	Implementation	22
5.3.1	PLD structure for 3D reconstruction	23
5.3.2	VHDL implementation of a vertical gradient	24
5.3.3	VHDL implementation of a set of prototypes (<i>Theme</i>)	25
5.3.4	Morphological erosion and dilatation	26
6	Discussion and conclusion	28

List of Figures

1	DVZ principle in 2D and examples in 3D	7
2	Right and left images at time t	12
3	A typical neuron and a 3-Theme PNN	15
4	Flow of 3D reconstruction	17
5	Right images at time t and $t + 1$	18
6	Determination of ego and external motions	18
7	Segmentation process	19
8	Design flow	21
9	General architecture for co-design	22
10	Implementation of the vision system	23
11	PLD structure for 3D reconstruction	24
12	Principle of gradient computation	25
13	PLD architecture for the Probabilistic Neural Network	26
14	Original image, neural classification and yellow objects	27
15	Morphological operation MATLAB (left) vs PLD (right)	27

1 Introduction

High level sensory information such as visual images must be managed to make mobile robots move faster in unpredictable worlds. Unfortunately, the image processing requires space and time. High level perception systems are generally developed independently on visual workstations which provide sophisticated but often slow tools. The task is then to try to fit these "good" algorithms to the constraints of a real-time robot. The co-design approach involves developing the vision system with these constraints in mind and to allocate the problem between software and hardware resources.

Concerning the classical artificial vision approach, many researchers have addressed co-design methods. In [1] the authors studied a motion-based collision detection algorithm divided into 2 main subroutines that are allocated to firmware and hardware resources (DSP and FPGA). The algorithm has been tested and compared on a Sun ULTRA1 and Pentium pro133, and targeted to a DSP56002/XILINKS[®] FPGAs for real-time implementation.

In [2] the authors designed a real time vision embedded system for motion segmentation purposes, collision detection and object tracking. The hardware is based on a Pentium[®] III PC running a Linux light kernel at video rates up to 25 frames per second with medium size images (128×96 and 384×288).

A very good review of advances in stereovision can be found in [3]. The authors address three basic topics: correspondence methods (local and global), occlusion problems and real-time implementations. This paper lists more than thirteen references in this last field from 1993 to present. The technologies used range from single off-the-shelf processors to custom FPGAs [4],[5] and DSPs [6]. For all of these projects, the grayscale images were less than 350×250 in size and the frame rates ranged from 0.5 frames per second to 42 fps.

Note also that several industrial products that include FPGA technology for vision purposes are presently emerging. They offer CAD capabilities to develop co-design architectures (for instance [7],[8])

Biologically-plausible algorithms spring from the observed natural systems especially through analogies between artificial perception/actions systems and natural ones [9]. The idea is not only to imitate the natural functions but also to extract the main mechanisms [10]. Many papers deal with this subject and whole conferences are devoted to it (see for instance SAB proceedings [11]). Neural network theories are an important part of this research and much work has been done on neural aspects of artificial vision. For instance, two recent works are closely correlated with biological experiments [12] [13].

This paper relates our experience in the co-design of biologically-plausible vision-based algorithms applied to control the reactive behaviors of mobile robots. Our main tools are Programmable Logic Devices like FPGA (Field Programmable Gate Arrays) to implement Perception/Actions schemes. We use a 4-step co-design methodology:

- Develop and simulate a given vision-based algorithm
- Co-design of this algorithm
- Implementation on a prototype board
- Integrate the design on a specialized FPGA (or eventually in a specialized ASIC)

This paper addresses the first 3 issues as the fourth one is just an engineering process which terminates the co-design. The first point will be illustrated through a detailed example (3D reconstruction for collision avoidance) and through a less detailed example (image segmentation for localization and target pursuit). The second point is the bottleneck of the whole process since the co-design of a complex algorithm is case-dependant and there is no automatic procedure to solve it. This will be illustrated through the same examples. Finally, we will describe several aspects of the implementation of these 2 examples on a prototype board.

This paper is divided into four parts. Section 2 stresses the need for fast vision systems in mobile robotics especially for the control of reactive behaviors. The interlinkage of perception (vision) and action (control of motion) is demonstrated through an example: the control of motion during collision avoidance and target pursuit.

Section 3 describes two biologically-plausible vision-based control algorithms : a 3D reconstruction and an image segmentation algorithm.

Section 4 illustrates these two algorithms with some simulation examples. In our mind, "simulation" means an off-line implementation with real images, contrary to the implementation presented in Section 5 which provides technical answers to our issue.

We will conclude this paper with a discussion on the co-design approach for sensor-based mobile robotics and with some prospective work in this field.

2 Biologically-plausible reactive behaviors

2.1 Vision-based reactive systems

In many industrial, exploration or lab robotic tasks, a mobile robot mainly has to perform four subtasks:

- Collision avoidance
- Target pursuit
- Localization
- Path planning

The first 3 tasks obviously need perception capabilities, while the fourth one also does when the environment is subject to change (re-planning when a failure occurs). Vision has the great advantage of providing the richest information but also the major disadvantage of being difficult to process in real-time (the characteristic pass-band for collision avoidance can reach 100Hz). Therefore including vision systems in a control loop requires constraining the vision algorithms either by simplifying them or by allowing them to run in real time. Hardware implementation thus seems to be an effective solution.

2.2 Collision avoidance and target pursuit

As an example, let's consider the important case of obstacle avoidance and target pursuit in an unknown and dynamic world.

2.2.1 General principle

The reactive control algorithm we use is based on the definition of a protecting and deformable zone surrounding the robot. This DVZ (Deformable Virtual Zone) is parameterized by the motion variables of the moving robot and can deform in the presence of distance information in the robot workspace. When an obstacle enters the sensor space, it induces a deformation of the DVZ that will be compensated by the robot motion controller. Therefore, the algorithm is a kind of 2-player game: the first one, i.e. the environment, induces undesired deformations; the second one, i.e. the robot controller, tries to rebuild the DVZ.

This algorithm was first described in [14] for many applications and tested for robots moving in 2 dimensions, flying robots or autonomous submarines

and also mobile manipulators [15]. This algorithm which was initially designed for obstacle avoidance, has two main advantages. First, the environment does not need to be *a priori* known, and second, the controller can take into account other constraints such as target pursuit (see 2.2.3), altitude maintainance, course control and so on.

Figure 1 illustrates this general principle that will be described in paragraph 2.2.2

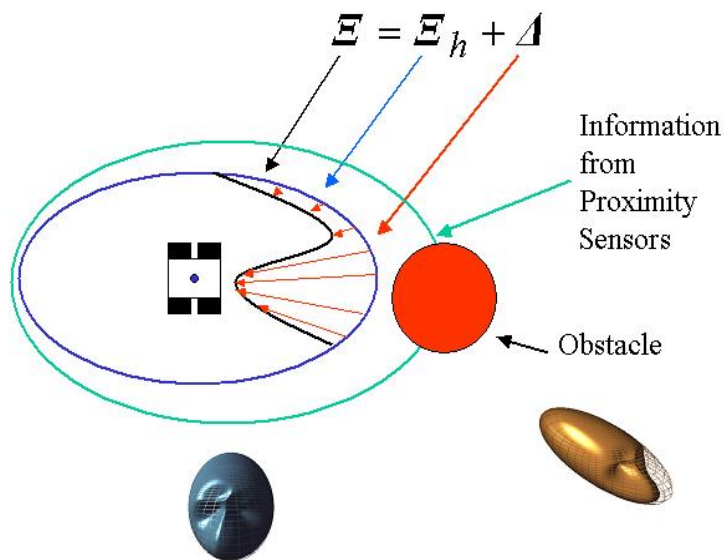


Figure 1: DVZ principle in 2D and examples in 3D

2.2.2 Mathematical basis

The general framework for formalizing this principle is the category \mathcal{D} of topologically equivalent sets of the $(n-1)$ -dimensional unitary sphere $S^{n-1}(0, 1)$ in \mathbb{R}^n . An object A of this category is related to the unitary sphere through an homeomorphism (imbedding of the sphere):

$$\delta_A : S^{n-1}(0, 1) \longrightarrow A \subset \mathbb{R}^n \quad (1)$$

The transformations between two objects A and B of \mathcal{D} are the deformations obtained by the combination of the two defining homeomorphisms:

$$\delta_B \circ \delta_A^{-1} : A \longrightarrow B \quad (2)$$

Let $R \subset \mathbb{R}^n$ be a convex rigid body, subset of the nD-space \mathbb{R}^n . The boundary ∂R of R can also be considered as the result of an imbedding of $S^{n-1}(0,1)$ in \mathbb{R}^n . We have $\partial R \in \mathcal{D}$ and:

$$\delta_R : S^{n-1}(0,1) \longrightarrow \partial R \subset \mathbb{R}^n \quad (3)$$

Reciprocally, if f is an imbedding of ∂R in \mathbb{R}^n with $E = f(\partial R)$, it is to say that:

$$f : \partial R \longrightarrow E \subset \mathbb{R}^n \quad (4)$$

then : $E \in \mathcal{D}$ by the choice: $\delta_f = f \circ \delta_R$.

Any object $A \in \mathcal{D}$ separates \mathbb{R}^n in two connected components, the interior $Int(A)$ of A and the exterior $Ext(A)$ of A . Therefore we have $\mathbb{R}^n = Int(A) \oplus Ext(A) \oplus A$. A partial order is induced on \mathcal{D} by the relation:

$$A \prec B \Leftrightarrow Int(A) \subset Int(B) \quad (5)$$

The rigid body R will represent a controlled robot moving among obstacles in \mathbb{R}^n . Any n-dimensional state vector characterizing the motion of R (e.g. generalized momenta= translational and rotational velocities) is denoted as a vector

$$\pi = [p_1 \ p_2 \ \dots \ p_n]^T \quad (6)$$

Axiom 1 *We assume that the robot R can be controlled by the derivative of this state vector. We note:*

$$\phi = \dot{\pi} \quad (7)$$

Definition 1 *We define a DVZ of R as any imbedding Ξ of ∂R in \mathbb{R}^n such that the relation $\partial R \prec \Xi(\partial R)$ holds. We have $\Xi(\partial R) \in \mathcal{D}$.*

Definition 2 *We define a controlled DVZ, Ξ_h , as a DVZ which depends on the state vector characterizing the motion of R :*

$$\Xi_h = \rho(\pi) \quad (8)$$

Definition 3 *Let $\mathcal{P} = (\Xi_h, \Xi)$ be a pair of two DVZ of R (the first one being a controlled DVZ) and such that $\Xi(\partial R) \prec \Xi_h(\partial R)$. We define the deformation Δ of the DVZ Ξ_h with respect to Ξ as the functional difference of Ξ and Ξ_h :*

$$\Delta = \Xi - \Xi_h \quad (9)$$

According to this definition, the deformation Δ is a one-one map that associates the vector $P - P_h$ to the point $M \in \partial R$, where $P = \Xi(M)$ and $P_h = \Xi_h(M)$. It can therefore be considered as a vector field defined on ∂R .

Axiom 2 *We assume that the robot can perceive distances in all directions of space. We also assume that the set of maximum distances that can be perceived by R and the set of actually perceived distances are two objects of the category \mathcal{D} , respectively named Sensor boundary and Information boundary (and respectively denoted Θ and Ψ), such that $\Psi \prec \Theta$. The deformation I of Θ with respect to Ψ is given by:*

$$I = \Psi - \Theta \quad (10)$$

The deformation I can also be considered as a vector field on ∂R .

Definition 4 *We define an uncontrolled DVZ, Ξ , as a DVZ which depends on the Sensor boundary deformation I :*

$$\Xi = \beta(I) \quad (11)$$

Let $\mathcal{P} = (\Xi_h, \Xi)$ be a pair composed of a controlled DVZ and an uncontrolled DVZ, the deformation Δ of the DVZ Ξ_h with respect to Ξ can be written :

$$\Delta = \Xi - \Xi_h = \beta(I) - \rho(\pi) \quad (12)$$

For a given point $M \in \partial R$ the deformation vector $\Delta(M)$ depends on the intrusion of proximity information $I(M)$, in the rigid body workspace, and on the controlled DVZ Ξ_h .

By differentiating equations (12) with respect to time, we get:

$$\dot{\Delta} = -\nabla_\pi[\rho] \phi + \nabla_I[\beta] \psi \quad (13)$$

where ∇_ξ is the derivation operator with respect to the vectorial variable ξ and $\psi = I$.

This equation can be rewritten as:

$$\dot{\Delta} = A\phi + B\psi \quad (14)$$

Variations in Δ are controlled by a 2-fold input vector $u = [\phi \ \psi]^T$. The first control vector ϕ , due to the robot controller, tends to minimize deformation of the DVZ. The second one, ψ , is unknown and induced by the environment itself (and could, at most, try to maximize these deformations).

Hereafter, this equation will be referred as the *main* equation of the problem.

Once the main equation obtained, its integration (i.e. the obtention of a "good" control vector ϕ) can be computed in 2 steps:

1. Choosing the desired variation of this deformation as a function of the real deformation and its derivative:

$$\dot{\Delta}_{des} = -K_{prop}\Delta - K_{der}\dot{\Delta} \quad (15)$$

where K_{prop} and K_{der} are heuristically chosen.

2. Computing the best control vector $\check{\phi}$ at time t obtained by inverting equation 14 after replacing the deformation derivative by its desired value $\dot{\Delta}_{des}$:

$$\check{\phi} = A^\dagger(\dot{\Delta}_{des} - B\hat{\psi}) \quad (16)$$

where A^\dagger is the inverse function (pseudo-inverse) of the linear function A and $B\hat{\psi}$ is an estimation of the second control vector ψ at time t obtained at time $t - 1$:

$$B\hat{\psi}(t) = \dot{\Delta}_{measured}(t - 1) - A\phi(t - 1) \quad (17)$$

Important remarks:

- The control law (Equation 16) tends to minimize the function $\left\| \dot{\Delta}_{des} - \dot{\Delta} \right\|$ in the Least Squares sense.
- The ∞ - *dimensional* functional equation 14 cannot, of course, be used directly. It is necessary to sample the sensor space in order to obtain an n - *dimensional* definition of the DVZ. This can be done by considering that the information vector has n dimensions (as many as the number of distance sensors). Equation 14 keeps its general form but all its entries are now matrices or vectors. For instance, let's consider the simple case of a car-like robot equipped with 5 distance sensors. In this case, the functions I , Δ , $\dot{\Delta}$, $\dot{\Delta}_{des}$, Ξ_h and Ξ are 5 - *dimensional* vectors. The vector π is a 2 - *dimensional* vector and function A is a (5×2) - *dimensional* matrix. The matrices K_{prop} and K_{der} are (5×5) square matrices.

2.2.3 Sensor-based control algorithm

In summary, *collision avoidance* is a 4-step process:

1. Measurement of the intrusion of information I as an n -dimensional vector (as many dimensions as the number of sensors)
2. Derivation of the deformation Δ and of its derivative $\dot{\Delta}$

3. Estimation of the uncontrolled control vector $\hat{\psi}$
4. Computation of the best control vector $\check{\phi}$

Regarding *the target pursuit* algorithm in the presence of obstacles, we have the same algorithm except that in this case the deformation vector includes lines of vector components for measuring the deformation between a desired state (robot on target for instance) and the real state (robot at a measured distance from the target): $\Delta = \begin{bmatrix} \Delta_{avoid} \\ \Delta_{target} \end{bmatrix}$

2.2.4 Links with vision

For these two algorithms, it is obvious that vision can be a helpful way for generating the necessary information.

For the collision avoidance scheme, the main input is the distance field in the robot front space. Stereovision allows this 3D reconstruction by measuring the disparity field in two images. By capturing images at successive step times, two or more cameras can also be used to detect motion in the robot workspace. Assuming that almost all points in the robot workspace are static, it is also possible to derive an expression of the ego-motion (self motion of the robot) and to fuse it with proprioceptive information.

For the target pursuit scheme, the visual recognition of the target can also involve a visual system.

In the next section, we will discuss two vision-based algorithms: 3D reconstruction and image segmentation, which will be the basis of the implementations proposed in the last section of the paper.

3 Vision-based control algorithms

3.1 3D reconstruction

The vision-based stereo-matching algorithm for distance estimation involves spatial stereo-matching of n characteristic points of two images (left and right) and on temporal stereo-matching on two pairs of images (pair of images at time t and at time $t + 1$). The first match provides a distance field in the robot front-space. The second one enables an identification of the robot motion (ego-motion) and of the motion of the mobile parts of the environment.

3.1.1 Spatial stereovision

From this first analysis, the field of distances in the robot front-space can be obtained by following the next steps (See figure 2):

- The right image $I_R(t)$ is divided into regular zones in order to cover the whole front space of the mobile robot.
- A particular zone of $I_R(t)$, called *fovea*, is subdivided into 16 smaller zones in order to increase the perception capabilities of the vision system in a given direction like in biological systems.
- A set $S_1(t)$ of interesting points (e.g. weighted gradient) is chosen in the right image $I_R(t)$. A subset of characteristic points $S_2(t)$ is derived from $S_1(t)$, containing one point per zone. Each of these points has the strongest gradient in the corresponding zone. Mathematically, this set $S_2(t)$ is characterized by the positions of these characteristic points in the image space (pixels).
- A stereo-matching algorithm, based on the evaluation of a distance between a (11x11) patch around the $S_2(t)$ points and a (11x11) patch moving in the left image $I_L(t)$, tries to find the correspondences between these 2 images. A counter correlation is then run to confirm the stereo-pairs. The disparity between the characteristic points provides a distance map to the robot main computer.

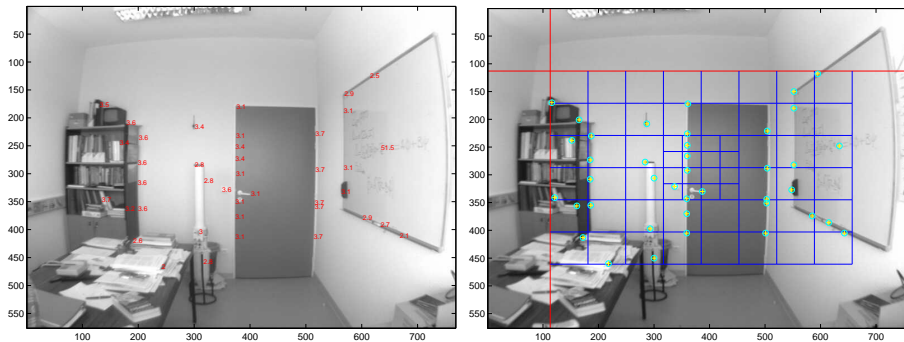


Figure 2: Right and left images at time t

3.1.2 Temporal stereovision

Between time t and time $t+1$, the robot and therefore the two cameras, move from their initial position P_t to their final position P_{t+1} . These two vectors are related by the matrix characterizing the small movement and noted ${}^tM_{t+1}$. This matrix is only approximately known and its approximation is noted ${}^t\widetilde{M}_{t+1}$. The characteristic points $S_2(t)$ in the image space obtained at time t are positioned in the real space and then reported on the right image at time $t+1$ using the approximation ${}^t\widetilde{M}_{t+1}$ of the matrix ${}^tM_{t+1}$. This set of reported points is noted $\widetilde{S}_2(t, t+1)$.

A stereo-matching algorithm, based on the evaluation of a distance between a (11x11) patch in the right image $I_R(t)$, centered on the $S_2(t)$ points and a (11x11) patch moving in the right image $I_R(t+1)$ around the points $\widetilde{S}_2(t, t+1)$, detects the real position $\widehat{S}_2(t, t+1)$ of these points in $I_R(t+1)$.

Assuming that almost all these characteristic points are static in the absolute frame, it is therefore possible to derive the best approximation ${}^t\widehat{M}_{t+1}$ of the matrix ${}^tM_{t+1}$ in the least squares sense by comparing the supposed position of each characteristic point at time $t+1$ and its real position in the image $I_R(t+1)$ taken at time $t+1$. Only the static points can help in determining the absolute motion of the robot.

In the seldom seen case of too many points moving in the absolute frame, the robot can only rely on the approximation ${}^t\widetilde{M}_{t+1}$, obtained, for example, by its proprioceptive sensors. However, this case can be easily detected by analyzing the variance of the difference between the supposed position of the characteristic points and their found position $\widehat{S}_2(t, t+1)$ in the image $I_R(t+1)$.

3.1.3 The complete algorithm

The complete algorithm is composed of 6 steps occurring at time t :

- Acquisition of the 2 images.
- Spatial stereo-matching for determining the map of distances in the robot front-space from the characteristic points $S_2(t)$.
- Position prediction of the set $S_2(t-1)$, transformed in $\widetilde{S}_2(t-1, t)$ for the image $I_R(t)$ by using the supposed motion ${}^{t-1}\widetilde{M}_t$ of the two cameras (obtained by the proprioceptive sensors).
- Temporal stereo-matching for determining the ego-motion ${}^{t-1}\widehat{M}_t$ by comparison of the supposed position $\widetilde{S}_2(t-1, t)$ of the characteristic points and their real position $\widehat{S}_2(t-1, t)$ in $I_R(t)$

- Computation of DVZ $\Xi(t) = \rho(\tilde{\pi}, t) + \Delta(t)$, the deformations (the map of distances) which trigger the collision reactive behavior. The computation of the controlled DVZ $\Xi_h(t) = \rho(\tilde{\pi}, t)$ requires determination of the ego-motion ${}^{t-1}\widehat{M}_t$.
- Foveatisation: The DVZ orientation induces re-computing of the position of the fovea in the right image for time $t + 1$.

3.2 Image segmentation

To recognize, or at least to follow, a given pattern in the robot space is a crucial issue in mobile robotics. For instance, in target pursuit tasks, the target has a very special role and its localization, and the measurement of its velocity or its acceleration are essential. Segmentation of a color image can provide this information. In the other ego-localization example (for path planning purposes), clues in the robot environment can also be detected by color segmentation. Even for obstacle detection, image segmentation can help in the determination of characteristic points before 3D reconstruction.

We have developed a segmentation algorithm based on two-layer Probabilistic Neural Networks (PNN) and on Mathematical Morphology tools. As summarized here, this algorithm is a 2-step process:

- *Segmentation*: Each neuron of the first layer of the PNN is dedicated to a given user-based colour. Its weight W is a point in the RGB space. Each color-prototype, called *Theme*, is therefore represented by several neurons in the network.

When a pixel P is read, its distance to W is computed and its probability of belonging to the neuron class is computed through a normalized Gaussian function (See figure 3). Then its probability of belonging to a given theme is computed by a combination of all neural probabilities (as many as the number of neurons of the first layer).

The next step carried out by the second PNN layer is the competition of all theme probabilities. At the end of the segmentation process, the original image has been replaced by a labelled image consisting of as many labels as there are different themes.

- *Morpho – Filtering*: Each theme can be transformed in a black and white image ($v=1$ for a pixel belonging to the Theme, $v=0$ if not). In order to clean this image, it is then eroded by an erosion process and dilated by a dilatation process. In a second step, the resulting image is labelled in order to detect all objects belonging to the theme.

Let us recall the definitions of the two main Mathematical Morphology operators.

The *dilatation process* of an image A by a structuring element B , which is a small matrix scanning image A , is a binary image A_d whose pixels are defined by:

$$[A_d(i, j) = 1] \iff (B \cap A \neq \phi) \quad (18)$$

where (i, j) is the position of B in image A

The *erosion process* of an image A by a structuring element B , is a binary image A_e whose pixels are defined by:

$$[A_e(i, j) = 1] \iff (B \subset A) \quad (19)$$

The dilatation process increases the white objects of A while the erosion process decreases them.

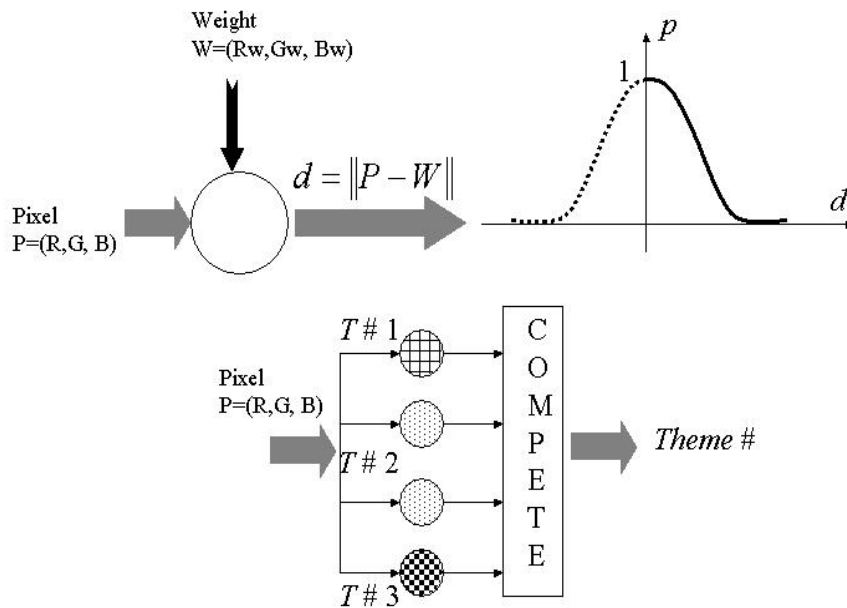


Figure 3: A typical neuron and a 3-Theme PNN

4 Simulation

Before implementing the two previous functions on prototype boards, we carried out some experiments with MATLAB to test the validity of these vision-based algorithms.

4.1 Simulation of 3D reconstruction

We tested the spatial stereo-matching algorithm on several pairs of images taken in unknown environments (rooms, corridors, outdoor).

The algorithm divides each image into 60 zones (44 for global vision +16 for the mobile fovea) and chooses one point of interest per zone (when there is one).

4.1.1 Spatial stereovision

Figure 4 shows an implementation of the stereovision algorithm in MATLAB from image acquisition to DVZ generation.

We carried out several experiments with different pairs of images and noticed that the number of characteristic points was not constant, due to the existence of uninteresting zones in the image (no significant gradient). For interesting points (with a strong gradient), if the stereo-matching algorithm reveals a good correlation between the left and right images, the distance of the point is computed (with a pin-hole camera model), and therefore its position in the real robot space is known. Otherwise, this distance is set at infinity. A visual check of the results indicated a success percentage from 90 to 100 percent.

4.1.2 Temporal stereovision

The next experiment was designed to determine the robot ego-motion and mobile obstacles in the robot space. Figure 5 shows the images at time t and $t + 1$. Between these two acquisitions, the robot has moved along with an object in the environment (marked POLE on the right part of the figure).

Between these two acquisitions, we assumed that the robot motion was given by the matrix

$${}^{t-1}\widetilde{M}_t = \begin{bmatrix} 0.9988 & -0.005 & 0 & 0 \\ 0.005 & 0.9988 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (20)$$

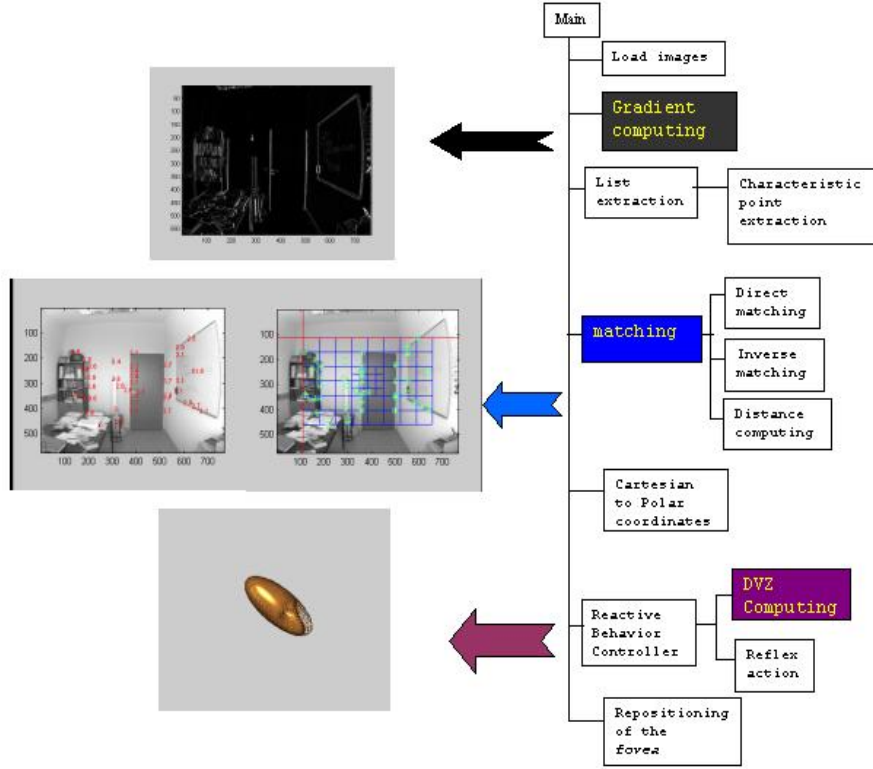


Figure 4: Flow of 3D reconstruction

which is a pure rotation around the vertical z axis with an azimuth angle of 0.05 rd.

During this motion, a study of the differences between $\tilde{S}_2(t-1, t)$ and their real position $\hat{S}_2(t-1, t)$ determines this absolute motion for four characteristic points. Figure 6 shows the distances in pixels between points of $\tilde{S}_2(t-1, t)$ and points of $\hat{S}_2(t-1, t)$ (left part of Figure 6). These distances are important (5 to 6 pixels) for the mobile characteristic points. These distances in the image space (the Cartesian position of these points were known) are transformed into their corresponding variations in meters in the real space (right part of Figure 6).

We can note the variation along the transversal y axis, characterizing an azimuthal rotation around the vertical z axis. The x axis is defined along the optical axis. The temporal stereo-matching provided a better evaluation of this matrix, given by:

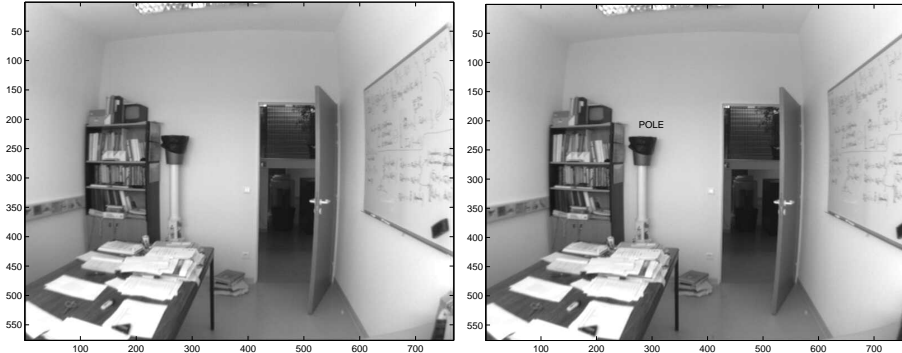


Figure 5: Right images at time t and $t + 1$

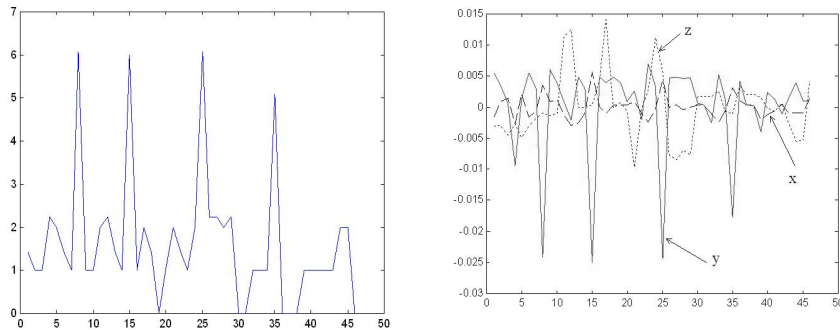


Figure 6: Determination of ego and external motions

$${}^{t-1}\hat{M}_t = \begin{bmatrix} 0.9998 & -0.0158 & -0.0005 & 0.0005 \\ 0.015 & 0.9994 & -0.0006 & 0.0005 \\ 0.0008 & -0.0004 & 0.9935 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (21)$$

4.2 Simulation of image segmentation

We also tested the image segmentation algorithm on several color images. A simulator was developed for designing different PNNs that are run on color images. The user decides how many themes are included in the network and how many prototypes (neurons) for each theme. This structure allows testing of several classifiers, depending on the chosen themes. For instance, it is possible to create a PNN to separate dark objects from light objects (2 themes), or to build one to detect a red object among others, and so on.

Figure 7 exemplifies the segmentation process for the detection of yellow segments in an image. Here, the PNN has 8 themes (Red, Blue, Green, Yellow, Magenta, Cyan, Dark and Light). At the end of the neural segmentation process, each original pixel has been transformed in a theme number. The whole image now looks like a pseudo-color image. Then, the morphological process can be applied to any of these themes.

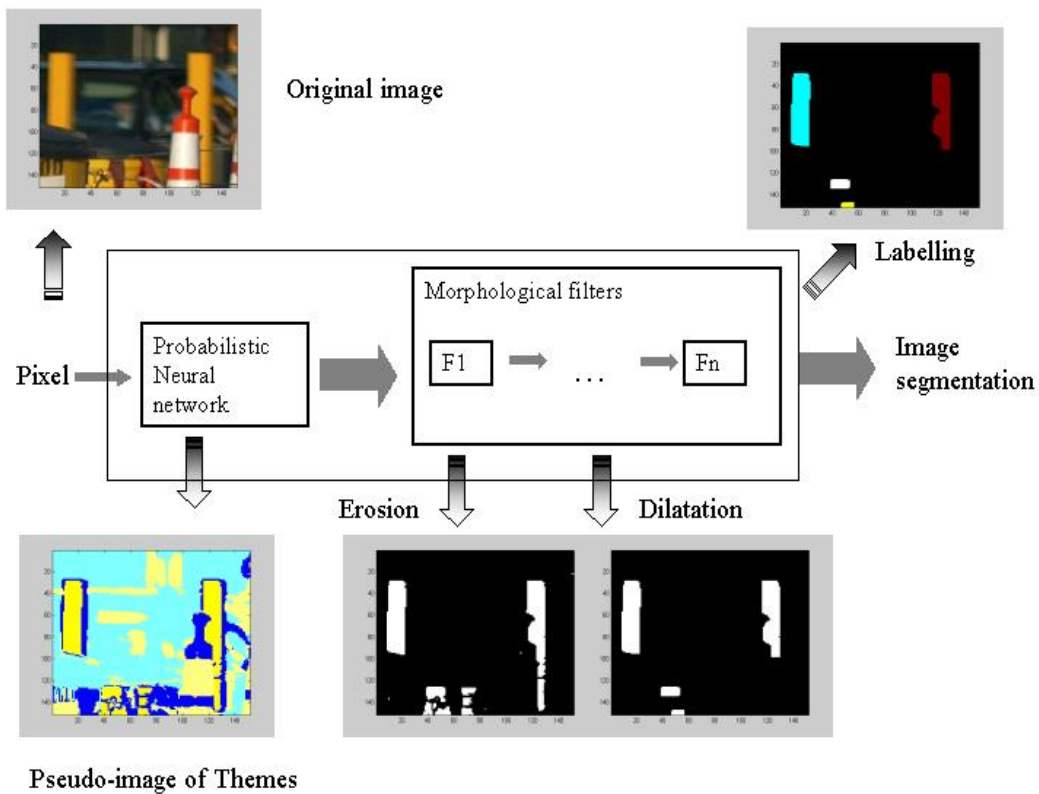


Figure 7: Segmentation process

5 Implementation of fast vision-based control algorithms

The architectural flexibility makes FPGAs the ideal technology to prototype complex designs. A system prototype running in real time addresses many of the shortfalls of using simulation alone for system validation. The re-

programmability of the hardware allows design iterations to be made quickly so the results of design changes can be quickly evaluated ([18], [19], [20], [21])

5.1 Design flow

Concurrent design of hardware/software systems uses prototyping to evaluate the performance of the final system and to facilitate hardware-software partitioning. Prototype simulation can be achieved in quasi real-time (the clock frequency is generally one order of magnitude smaller than that of the final system on silicone). Signal processing applications (like the image processing algorithm described above) require very long validation digital patterns that lead to a prohibitive time for model simulation. Prototyping a signal processing system on a programmable and configurable platform allows validation of the processing function in real time. Within this framework, the system is first designed at a high abstraction level as a dataflow, with potential links with MATLAB tools.

Figure 8 illustrates the design flow in which the integration phase has been omitted.

The first phase is the simulation of the vision-based algorithm as illustrated in the previous section.

The second stage is the co-design itself. In this phase, the user defines what parts of the simulated algorithm will be implemented with hardware or software resources.

The last stage involves implementation on a prototype board and testing of the algorithms with either the same images that served in the simulation phase or new images provided by the cameras.

5.2 General architecture and co-design

Programmable logic devices are standard user-configurable integrated circuits that allow the implementation of custom logic functions and offer high potential logic integration.

We co-designed vision systems, implemented and tested them on a EPXA10 development board from the Altera[®] company [22].

The ExcaliburTM development board is built around an EPXA10 device which integrates an ARM922T-based hard processor with 16 Mbytes of 16-bit-wide flash memory, SDRAM controller, UARTs, DMA, Ethernet and PIOs. This device can also host one or several NIOSTM embedded RISC soft processors with 32-bit data paths with the same kinds of interfaces. The device can also be interfaced (UART) with an onboard PC.

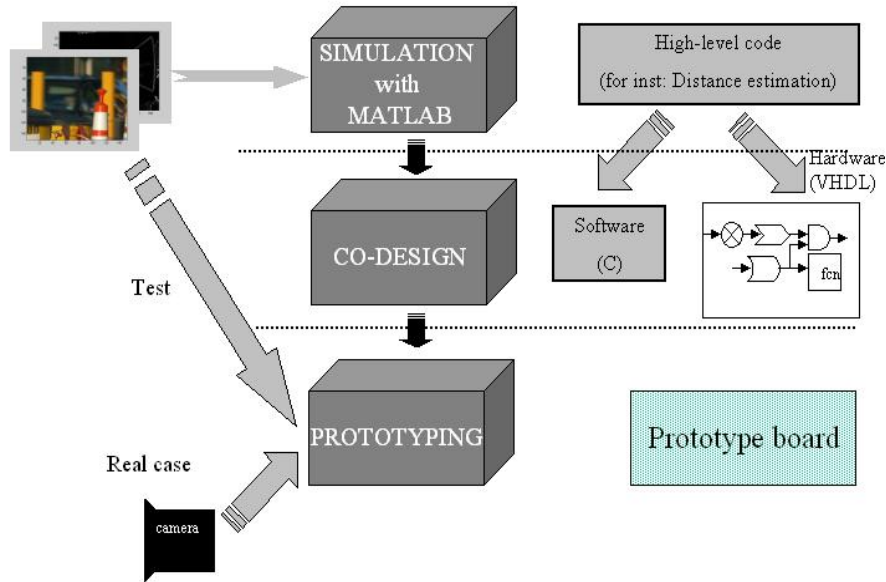


Figure 8: Design flow

This board comes with a development software kit (Quartus IITM and SOPCTM builder) which allows simulation and of software download. It is also possible to use dedicated IPs (Intellectual Property codes) and to develop dedicated home-made IPs.

The prototype board is connected through IOs to the different sensors and actuators. The circuit is divided into 2 zones: the processor zone which is hardware-implemented (in the case of the ARM-based board) or software-implemented (in the case of the NIOS-based board). The PLD zone which contains the different home-made IPs dedicated to the control of sensors, actuators and, of course, the vision system. The design involves defining what part of the whole process will be implemented with hardware resources and what part will be implemented with software resources.

There are 3 "programming" levels (see figure 9):

- A low level design (written in VHDL language), part of the "hardware resources", devoted to low level functions such as encoder reading, actuator control, image processing and image analysis.
- A medium level design (written in C language), part of the "software

resources”, which incorporates functions that are not easy to integrate at a hardware level (e.g. fix point operations).

- A high level design, implemented in C language on the onboard PC to manage communication, supervision and some mathematical issues (e.g. floating point operations).

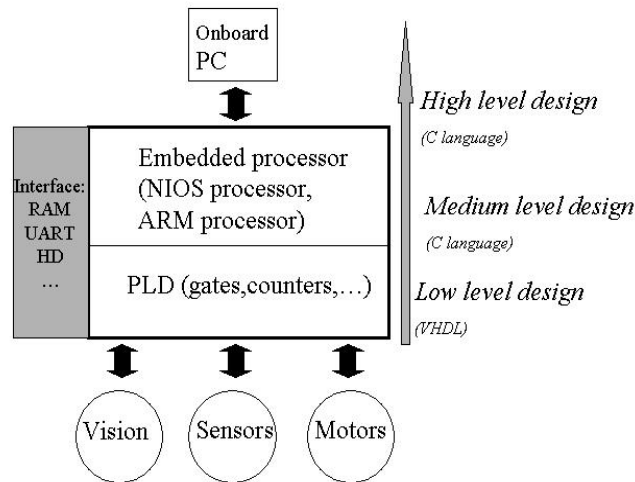


Figure 9: General architecture for co-design

The development software allows writing and testing of the VHDL code and of the C code that will be implemented on the imbedded processors (NIOS and/or ARM).

The vision system involves several home-made IPs (in our case, 3D reconstruction and color image segmentation) linked to the vision hardware (see figure 10), here, two 492x656 colour cameras with LVDS connections to the board.

5.3 Implementation

This paragraph describes several aspects of the real implementation of the vision-based algorithms both on an Excalibur NIOS embedded prototype board and on a Excalibur ARM-based prototype board. The prototyping was only tested here with test images obtained with digital cameras and

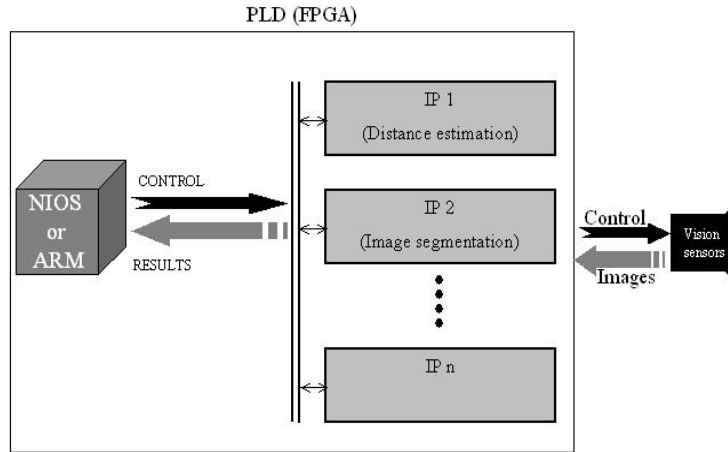


Figure 10: Implementation of the vision system

then downloaded to the embedded processor (See figure 8) Because of the numerous aspects of the complete integration, we decided to focus on 4 of them:

1. The PLD structure for 3D reconstruction
2. The VHDL implementation of a vertical gradient
3. The VHDL implementation of a set of prototypes (*Theme*)
4. The morphological dilatation operator

5.3.1 PLD structure for 3D reconstruction

The 3D reconstruction algorithm was developed for gray 256-level images. Figure 11 shows the structure that was chosen for the 3D reconstruction algorithm. It is composed of 7 main blocks:

- a RAM controller that stores the right and left images
- a zone operator that determines the zone to which the incoming pixel belongs (44 zones for global vision and 16 for the fovea. See 4.1)

- a gradient operator that computes the threshold of the 256-level image
- an operator that computes the best characteristic point for each of the 60 zones when there is one
- a matching operator that search the left image in order to minimize the SAD (sum of absolute differences) between an 11x11 patch around each characteristic right point and an 11x11 patch moving in the left image
- a sequencing machine that generates all the logical control signals
- a processor (NIOS or ARM) that reads images from a PC, transmits them to the PLD, gets the results and transmits them back to the PC

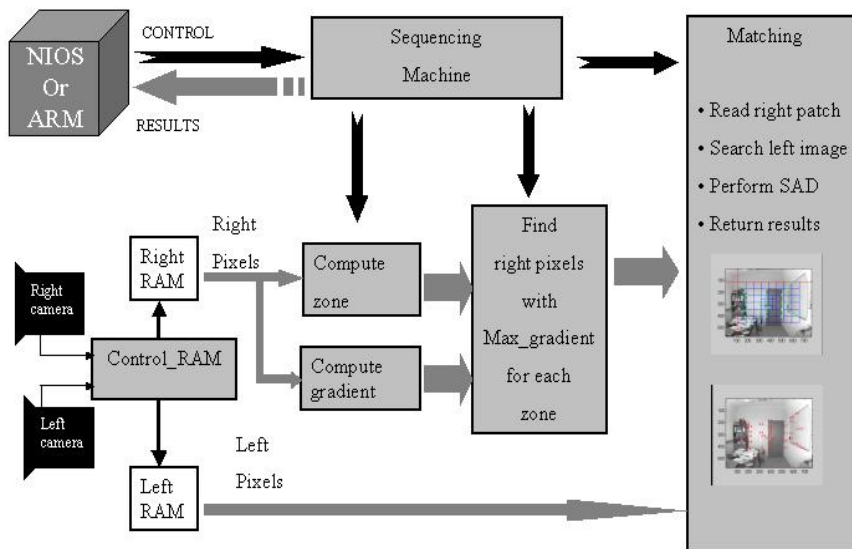


Figure 11: PLD structure for 3D reconstruction

5.3.2 VHDL implementation of a vertical gradient

VHDL implementation of a gradient operator is a very good example of the difference between a high level description language (C or MATLAB) and

a logical language (VHDL). In this case, the main difficulty is to provide the information needed to compute the convolution of a given image and a gradient mask (in general a 3×3 mask). The $m \times n$ image arrives sequentially and the incoming pixel allows computation of the gradient of the pixel up-and-left from it. The gradient logical function consists of storing 2 lines and 3 pixels, including the incoming pixel, and computing the gradient (see figure 12). These $2n + 3$ pixels are stored in a FIFO-type memory. At each top of the pixel clock, the whole memory is translated and a new gradient is computed.

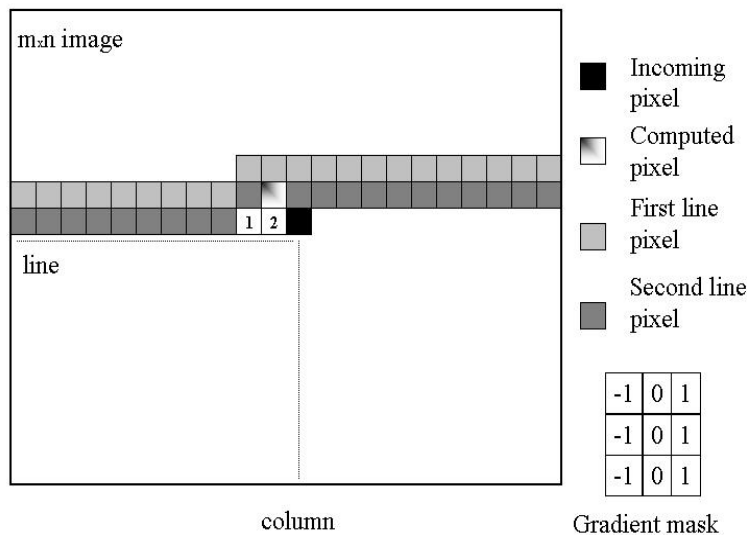


Figure 12: Principle of gradient computation

5.3.3 VHDL implementation of a set of prototypes (*Theme*)

Each theme is a VHDL object involving a maximum of N neurons (subroutines of the VHDL code) and can be configured in C by the NIOS during the programming phase. In this phase, the user chooses the number of themes to be programmed, the number of neurons per theme, and the weight of each neuron. At each top of the programming clock (Progclk), the processor provides the PLD with a validation signal and the weight of the i^{th} neuron to be programmed.

In the application phase, at each incoming pixel arriving at each top of the pixel clock (Pixclk), the theme gives the probability of the pixel belonging

to it.

All of these probabilities feed a compete layer that chooses the best theme. The left part of figure 13 shows this architecture. The Gaussian activation function was tabulated in a Look-Up-Table (LUT) whose inputs are a sampling of distances in the RGB space and whose outputs are the corresponding Gaussian probabilities normalized between 0 and 2047 (typically, a power of 2). The right part of figure 13 shows the VHDL code for the neuron and a small part of the LUT representing the sampled Gaussian.

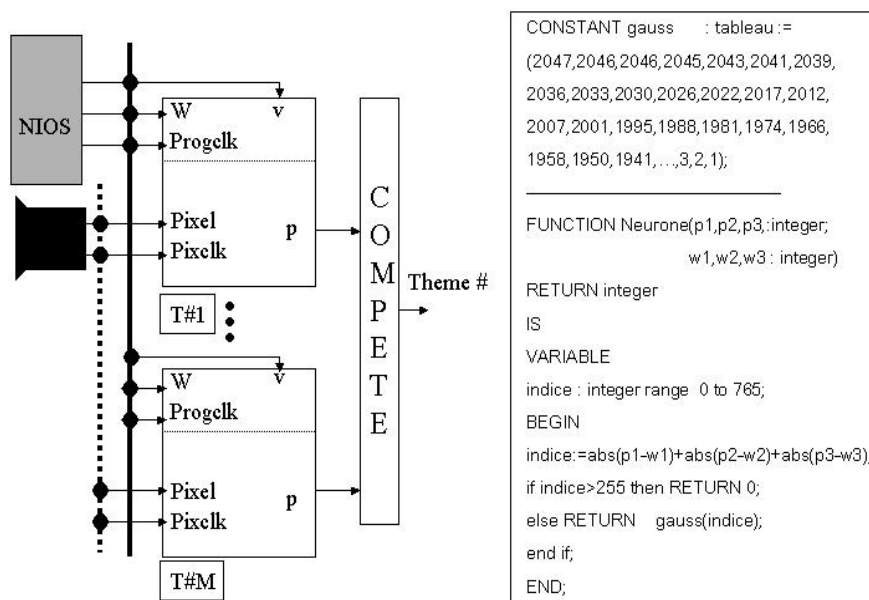


Figure 13: PLD architecture for the Probabilistic Neural Network

5.3.4 Morphological erosion and dilatation

The second step of image segmentation involves filtering using morphological operators. We implemented a sequence of two operators: an erosion followed by a dilatation. The main interest of this sequence is to clean the images, i.e. small objects or protuberances are erased while the size of bigger objects is maintained. In this experiment, we chose a classical 3×3 full matrix as the structural element operating on the image.

Figure 14 shows an original image (left), the pseudo-color image representing the different zones corresponding to the 8 themes previously defined in paragraph 4.2 (center) and the binary image representing the 6th theme, in this case the yellow objects (right).

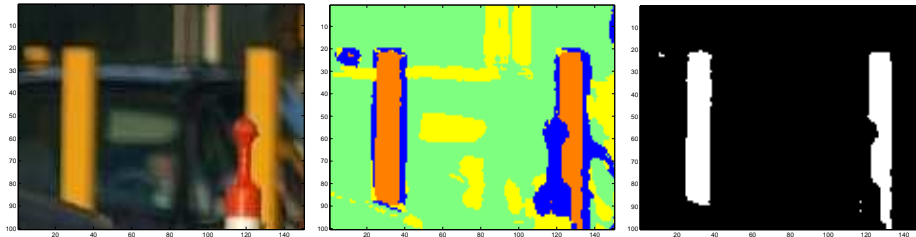


Figure 14: Original image, neural classification and yellow objects

Figure 15 compares the results for the MATLAB implementation and for the logical implementation.

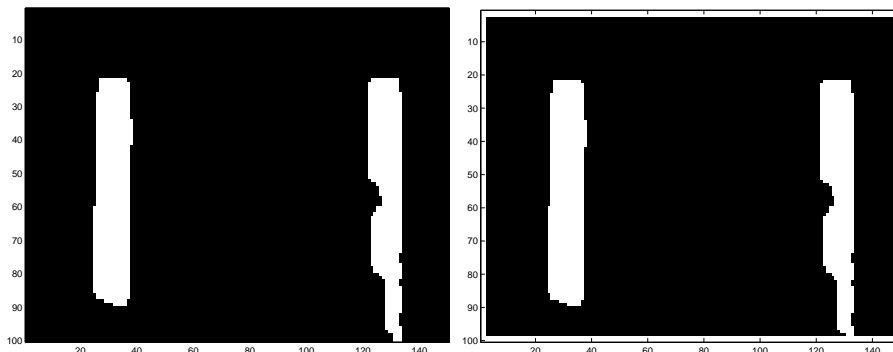


Figure 15: Morphological operation MATLAB (left) vs PLD (right)

A morphological filter is implemented according to the same principles that underlie the gradient computation. A 3×3 structuring element requires 2 lines and 3 pixels to be stored, including the incoming pixel. The last part of the implementation changes. When the incoming pixel arrives, the values A_d and A_e of the computed pixel (See equations 18 and 19) are given by:

$$A_d = \bigvee A(i, j) \wedge B(i, j) \quad (22)$$

and

$$A_e = \bigwedge [A(i, j) \wedge B(i, j)] \vee \overline{B(i, j)} \quad (23)$$

6 Discussion and conclusion

This paper tried to emphasize two main ideas:

- The needs for efficient vision-based systems seem obvious. Even if image processing often requires space and time, only high level sensory information provided by vision systems, can provide mobile robots with real autonomous capabilities.
- Looking for efficiency in vision-based systems can be done either by investigating fast natural vision-systems (natural systems) or by promoting co-designed hardware/software solutions. A mixed approach seems, of course, a good solution.

If simulation can bring some validation of the proposed approaches, real experimentation must complete the work. We are building a platform consisting of 3 mobile manipulators moving and cooperating in unknown environments and equipped with the vision system described above.

This paper addressed the development, the simulation and the co-design of biologically-plausible vision-based control algorithms in order to control the motions of mobile robots during two main tasks: collision avoidance and target pursuit.

It also analyzed 2 examples: 3D reconstruction for collision avoidance using a classical stereovision scheme and image segmentation for localization and target pursuit using Probabilistic Neural Networks. We also described several aspects of the implementation of these 2 examples on an EPLD prototype board.

The co-design process, consisting in sharing hardware and software resources, is the bottleneck of the overall process. However, we can imagine that the automatization of this co-design process will be realized in a next future. For instance, we can imagine bridges between high level functional languages (MATLAB, C++,...) and high level description languages like VHDL. Some industrial products already exist [7] and allow to expect this automatization.

References

- [1] S. Lodha, S.Gupta ,M. Balakrishnan, S. Banerjee: "*Real Time Collision Detection and Avoidance: A Case Study For Design Space Exploration in HW-SW Codesign*", 11th conference on VLSI Design, Chennai,India, January 4-7,1998

- [2] V. Haldar, G. Vardhan, A. Saxena, S. Banerjee, M. Balakrishnan: "*Design of Embedded Systems for Real-Time Vision*", Indian Conference on Computer, Vision, Graphics and Image Processing (ICVGIP2000), Bangalore, India, December 20-22, 2000
- [3] M. Brown, D. Burschka, G. Hager: "*Advances in Computational Stereo*", IEEE Trans. On Pattern Analysis and Machine Intelligence, Vol.25, No 8, August 2003
- [4] O. Faugeras and al: "*Real time correlation-based stereo: algorithm, implementations and applications*", Rapport de recherche, projet Robotvis, INRIA No 2013, August 1993
- [5] S. Kimura, T. Shinbo, H. Yamaguchi, E. Kawamura, K. Naka, "*A Convolver-based Real-Time Stereo Machine (SAZAN)*" Proc. Computer Vision and Pattern Recognition, vol. 1, pp.457-463, 1999
- [6] D. Beymer, K. Konolige, "*Real-Time Tracking of Multiple People Using Continuous Detection*", Proc. IEEE Frame Rate Workshop, 1999
- [7] WEB SITE <http://www.prodesigncad.de/>
- [8] WEB SITE <http://www.gvpp.org>
- [9] Alain Berthoz, Giselle Weiss, "*The Brain's Sense of Movement*", Perspectives in Cognitive Neuroscience, 2002
- [10] F. Mura, N. Franceschini, "*Visual Control of Altitude and Speed in a flying Agent*", From Animals to Animats III (Eds : D. Cliff, P. Husbands, J.A. Meyer, S. Wilson), MIT Press, 1994, pp. 91-99
- [11] SAB: Simulation of Adaptive Behaviors, From Animals to Animats, 1990 to 2002, every two years
- [12] R. Wurtz, T. Lourens, "*Corner detection in color images through a multiscale combination of end-stopped cortical cells*", Image and Vision Computing Vol. 18, Number 6-7, pp. 531-541, April 2000.
- [13] R. VanRullen, S. Thorpe "*Surfing a spike wave down the ventral stream*", Vision Research 42 (2002) pp. 2593-2615
- [14] R. Zapata, P. Lépinay, P. Thompson "*Reactive Behaviors of Fast Mobile Robots*" Journal of Robotic Systems Volume 1, 1994

- [15] A. Cacitti " *Comportamento Reattivo di Manipulatori mobili non-Olonomi basato sul metodo della Zona Virtuale Deformabile*", Tesi di Laurea, Facolta di Ingegneria, Universita degli Studi di Bologna, Italia, 2000
- [16] R. Zapata, P. Lépinay , " *Collision avoidance of a 3D simulated flying robot* " ISRA'98: International Symposium on Robotics and Automation, Saltillo, Coahuila, Mexico, December 12-14, 1998, pp. 113-120.
- [17] J. Banks, M.Bennamoun, P. Corke " *Fast and Robust Stereo Matching Algorithms for Mining Automation* " IAF'97 Adelaide, Australia, pp.139-149, November 97
- [18] S.Pillement, L.Torres, M. Robert, G.Cambon " *Fast Prototyping: A Case Study - The JPEG Compression Algorithm* " IEEE - RSP'99 (Rapid System Prototyping) ,Clearwater (Etats Unis), pp101-106, juin 99.
- [19] T.Benner and R.Ernst " *FPGA based prototyping for verification and evaluation in hardware/software cosynthesis* " 4th Workshop on FPL, Prague 1994, pp. 251-258.
- [20] D.Gajski, F.Vahid, S.Narayan and J .Goy " *Specification and Design of Embedded System*" Prentice-Hall Book, pp. 233-304.
- [21] Y.Kim, K.Kim, T.Ahn, K.Choi, " *An Integrated Cosimulation Environment for Heterogeneous Systems Prototyping*" Special Issue on Design Automation for Embedded Systems, Kluwer Academic Publishers, March 98, pp. 163-186.
- [22] WEB SITE <http://www.altera.com/>