Adding Vision to a Salamander/Snake-Robot

Benoit RAT, Master Student
SSC - IC - EPFL
benoit.rat@epfl.ch

Supervisor
Dr, Francois Fleuret, Visiting Research Associate
CVLAB - IC - EPFL
francois.fleuret@epfl.ch

Collaborators
Alessandro Crespi, PhD Student
Prof, Auke Jan Ijspeert, Assistant Professor
BIRG - IC - EPFL
alessandro.crespi@epfl.ch,auke.ijspeert@epfl.ch

February 16, 2007

Contents

1 Introduction

Over the last decades, in the fields of the robotics, algorithms and mechanics inspired by the real-world have started to be used. In order to obtain realistic gaits for different robots, research has been focused on how to handle multiple degrees of freedom, various type of redundancies in movement and smooth transitions between gaits. It has been demonstrated that bio-inspired robots only need to generate specific rhythms to produce a gait. However feedback and higher control center should also be implemented in order to strongly modulated these rhythms and optimize the gait.

The *Biologically Inspired Robotics Group*, BIRG at EPFL is currently designing two bio-inspired robots: The *salamander robot* and the *amphibious robot*. They can provide complex gaits under different stimuli. Simulating a good response to a stimulus is not an easy task due to the absence of feedback. In order to handle this lack of information, artificial vision can be implemented on these robots. Vision is very interesting in bio-inspired robotics since it can easily generate a multitude of distinct stimuli (avoiding obstacle, exploration, following objects,...). Moreover, when the robot changes its motion, it receives feedback, in the sense the sense that its field of vision in modified.

This project focus on primary level of artificial vision such as color segmentation. A robust algorithm for color based tracking supporting changes of illumination is implemented. By using this algorithm the robots are able to follow a pink ball in real-time generating specific stimuli. A more complex artificial vision process has also been developed during the resarch phase. This second algorithm finds the geometrical transformation between two consecutive frames and thus deduces the relative position of our robot.

2 Project Definition

As the project is named *Adding Vision to Salamander/Snake Robots* it is referenced in the further chapters with the acronym AV2SR.

2.1 Goals

Initially, the goals of the AV2SR project were to design algorithms for stimulus tracking and obstacle avoidance for the robot moving in cluttered environments. The algorithms needed to be tested on the real robot in engineered environments of increasing complexity.

As it is a wide subject we focus on these principal features:

- Capture videos and study the robot gaits.
- Color tracking: The salamander should be able to follow a color ball and change its trajectory to keep the ball in it vision field.
- Matching Patch algorithm: Detect the transformation between two frames in order to estimate its relative position.

2.2 Hardware

Adding vision on a robot is not an easy task. The vision algorithms should be able to work in real time and modify the gaits on the fly. For this purpose we used specific hardware:

- The salamander/snake-robot: We used during the whole development phase two robots developed by the BIRG (Ijspeert et al. 2005). The salamander robot which is able to walk and swim. And a amphibious robot (aka snake robot) which is able to crawl. The Section 3.2 describes in more detail how these robots work.
- Sony Photos Camera: A simple photo camera was used in video mode at the beginning of the project. It was a temporary solution in order to start working with offline videos. (Section 3.1)
- Radio-Camera: Later, a radio camera[1] is installed on the head of the salamander robot. It has CMOS color sensors with a minimum luminous intensity of 3 Lux.

[1]Color radio camera - Nr. 75 11 76, Conrad, www.conrad.com

Its resolution is 628×582 (PAL) and it transmits data by radio in a range of frequencies between 2.4-2.4845 Ghz. It was a simple solution to solve the problem of an embedded camera on the salamander-robot. (c.f. Section 3.1)

- The frame grabber:
  The GrabBeeX+deluxe[2] is a simple frame grabber that permits to convert a signal receive on a RCA composite and transmits it by USB 2.0 on a computer. It works with different video formats, included PAL (25fps), and can capture image in a resolution of 640×480 pixels.
  As the GrabBeeX do not have drivers that work properly on a Unix OS, our application should be run on a Windows/Mac platform.

2.3 Software

The development of AV2SR was done in three steps:

Firstly, Matlab[3] was used during the initial research phase. As Matlab is an easy script language, it is a convenient solution for rapid prototyping algorithms, and test them easily on a video. However its main disavantage, as all script languages, it is really slow in computation time.

To solve the time dependency problem. A library called c4m[4] was added to Matlab code. This library can be resumed as an interface between Matlab and C language using MexFile. Using C language permits to increase speed on the bottle neck parts of our algorithm while the flexibility of a script language is still used in other parts. As it is a proprietary library, the source code can not be join to this project. This is something that we are not expecting from an educative semester project.

Finally, the code was written in C++, to decrease the computation time and avoid the proprietary problem. OpenCV[5] library was mainly used to manipulate images and the STL to make a portable code.
The C++ code was first written in a Unix environment using Vim[6] editor. However, during the implementation of the online part of the color tracker, we have constated that the GrabBeeX+ frame grabber was not supported on Unix environment. Thus this algorithm was ported in Visual Studio C++ 6.0 and slightly modified to make it compatible with Windows.

[2]GrabBeeX+deluxe, Greada, www.greada.com
[3]Matlab R2006b, Mathwork, `www.mathworks.com/products/`
[4]C4M (C for Matlab) library, STMicroelectronics, `www.st.com/`
[5]OpenCV (Open Source Computer Vision), Intel, `www.intel.com/technology/computing/opencv/`
[6]Graphical Vim Editor (VI IMproved) 7.x, `www.vim.org`

3 Salamander-Robot Locomotion

3.1 Capture

We have initially started using a digital photo camera for making the first videos. They were not perfect due to the slow autofocus of the camera who lead to a motion blur on 40% of the frames. It was, at least, a starting point to study salamander movement and implement first algorithms.

The second session of videos capture was with the radio-camera (Section 2.2). It has a lot of advantage to use it with a robot: The camera is really light so it can be placed on the head of the robot without consequence. As both radio-camera and salamander/snake robot use wireless channel for communication with a computer, we can implement algorithms in real time that capture the vision field, process them and send back a response to the robot.

However the radio-camera generates a lot of problems that we need to take into account :

1. As it used radio frequencies it has a lot of problem to transmit a perfect frame to the receptor. It is worse when the robot is in movement near something with metal. In the worst case, a bad transmission can last 10 frames (about 0.5s).

2. This also implies, that a frame can be wrongly transmitted. This is worse than random noise because our algorithm can detect something that does not exist or that is not at the right place. The Figure 3.1 shows pink blots which can be considered as the pink ball. This demonstrates the need to implement algorithms which can reject false positive objects/frames or simply avoid that a false positive could bad influence our algorithm.

Figure 3.1: Bad frame with pink blots, it can be detect as the pink ball (False Positive) if we use an algorithm with only a color scheme

3. The receptor always receives an external signal periodically even if the camera is near it. We have presumed that this type of interference comes from the wireless network used in our faculty and should correspond to the SSSID Broadcast. (Figure 3.2)

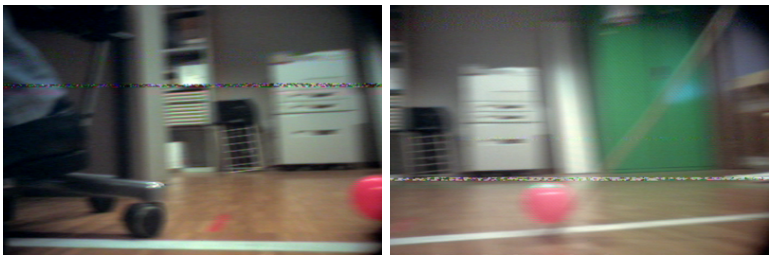


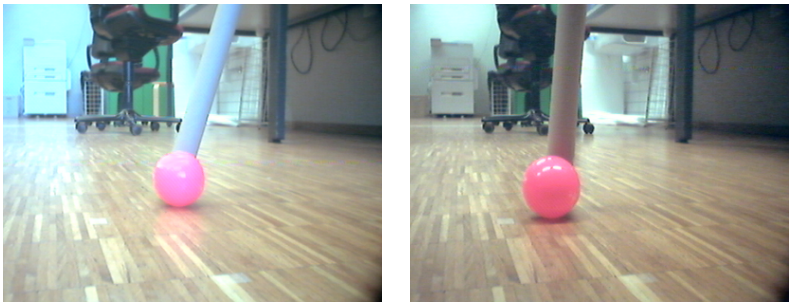
Figure 3.2: Periodical interference due to the wireless network

4. The problems due to radio transmission are not the only one. The camera itself generates bad inputs. A blurring effect can be observed during motion (Figure 3.3). The autofocus is not in cause this time because the camera does not have this option, it always focus on infinity . This may come from the CMOS captors which need a long exposure time to render correctly the environment.

(a) blurry frame (b) correct focus

Figure 3.3: Difference of focus between 2 frames due to motion

5. Quick changes of illumination between two frames can also be observed (Figure 3.4). This is an important thing to consider during our implementation of the color tracker and the matching by histograms patches. We should try to estimate the global illumination of the image and change parameters of our algorithms considering it.



(a) natural or cold color (b) warmer color

Figure 3.4: Difference of color between 2 frames due to a quick change of illumination

3.2 The robot CPG

The salamander and snake robot were studied due to their ability to deal with difficult environments (Ijspeert 2001), in which other types of robots often fail. They also have a neuronal system easier to model than mammals (Roth & Al., 1993, 1997).

We first started studying the salamander robot due to its interesting particularity of using two gaits of locomotion: swimming and walking. Both are completely different, but the salamander surprisingly manages to switch from one to the other as soon as it passes from one environment to the other (water to ground and ground to water). Since the radio-camera is not waterproof, it was impossible to test our algorithm when the salamander was swimming. Therefore, the snake robot (Crespi & Ijspeert 2006) was used in its crawling gaits, because it is similar to the swimming gait of the salamander.

To implement on these robots gaits inspirated from the real world, namely by trying to imitate the behavior of a real reptile, we use an on-board central pattern generator (CPG) :
A CPG is a system of coupled nonlinear oscillators that controls the locomotion (Figure 3.5). Remarkably those systems are able to convert a one dimensional signal such as a firing rate into a locomotion with corresponding speed/frequency. In other words, they can generate complex control patterns from simple tonic inputs. In the actual version of the robot, we are able to send these tonic inputs on the fly by using transmission between a computer and the robot over a wireless channel.

From an hardware point of view, the salamander robot consists of nine body segments of equal length:

- The first segment (the head of the salamander) is equipped with the radio-camera describe in Section 2.2.
- The second and sixth have each a leg on both sides. The legs are controlled by the CPG and are constructed to rotate continuously on a frequency related to the

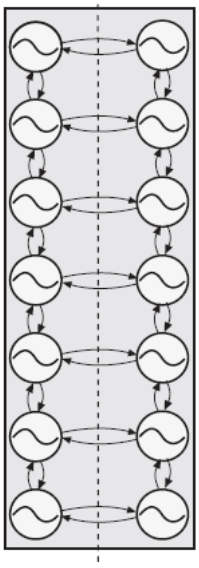


Figure 3.5: Structure of the CPG: A system of non-linear coupled oscillators that generates patterns (rhythms).

whole body. When the salamander is swimming the CPG do not send informations to the leg motors, they are maintained parallel to the body.

- Further motors are used to bend the robot, they can admit angles of up to 65° (during swimming) and are installed in between the segments two to five and six to nine.

The snake robot is similar to the salamander without the leg motors. Consequently it has more "bending" motors to control its gaits.

The CPG generates two types of signals during the different gaits:

1. The travelling waves: Corresponds to the signals generated by the snake-robot while crawling and the both robots while they are swimming. The Figure 3.6 illustrates the different signals generated by the oscillators in the CPG. We can clearly see that there is the same shifted phase between each signals. This Figure also show how the robots handle the *brutal changes* of parameters by making *smooth transitions*.

2. The standing waves: Corresponds to the signals generated by the salamander robot while walking. The legs are synchronized with the frequency of the body such that a foot touches the ground when on a bump and leaves it after the stride when it is in a dip (See figure 3.7). In order to obtain such a gait in the robot the first three bending motors are controlled with a time dependent sine and the last three with

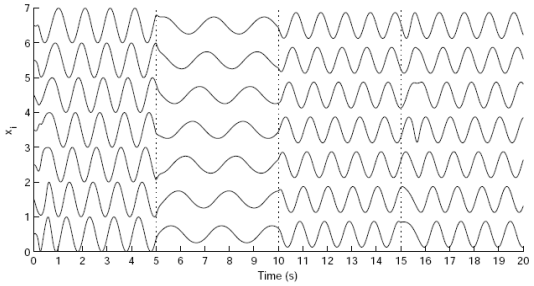


Figure 3.6: Travelling waves generate by the CPG. At $t = 5s$, The amplitude and frequency is reduced. At $t = 15s$, same signals than $[0, 5]$ with half amplitude. At $t = 15s$ the difference of phase is inverted

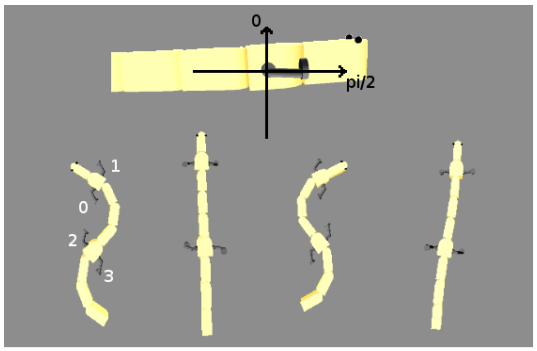


Figure 3.7: S-shape of the salamander and corresponding position of the legs

the same signal shifted by π. This leads to the desired oscillating S-shape. Then the legs are connected to the oscillators to generate their synchronized rotations. The standing wave also provides smooth transition while a parameter is modified.

The robots store in internal registers all the parameters to produce a desired gait. These internal registers correspond to the tonic inputs see previously. In our application we focus on two of them:

- The *turn* that changes the center of oscillation in order to make the robot goes on the left or right.
- The *drive* which interpolates different amplitudes and frequencies knowing the best couples. The drive is then scaled between 0 and 5 which correspond to the different "speeds" that the robot can have during its locomotion.

4 Color Tracking

Color tracking has been mainly used as initial pre-processing for faces detection and tracking (Anisetti et al. 2006). To resolve the problem of skin color segmentation several approaches has been developed. Comparative studies has been done by Vezhnevets V. (2005), Terrillon et al. (2000) and Storring (2000).

This part of the AV2SR project is focusing on the implementation of a simple and quick algorithm to make our salamander/snake-robot able to follow a pink ball. Since the AIBO[1] robot is delivered with a pink ball and a pink bone that it is able to follow, we have used the same ball in order to develop this algorithm for our robot.

4.1 Color Space

A color model is an abstract mathematical model describing the way colors can be represented as tuples of numbers, typically as three or four values or color components. The color space is the function which maps these values to a referenced system according to its color gamut. The gamut correspond to the subset of color that an output device can display. As the gamut of papers, LCD monitors, TV is not the same, they can not show/render exactly the same color to the human eyes. Therefore different color spaces need to be used depending on the type of applications. The most common color space is *RGB*, and mainly use to store color values in digital data.

However, *RGB* is a very correlated color space. This means that it is difficult to make a color brighter without changing at least the values of two dimensions. Moreover *RGB* has its color values represented in three dimensions and this can make the computation cost heavier due to the fact that each channel has to be processed.

Several color coding spaces were compared in (Vezhnevets V. 2005, Storring 2000) to find the one with the best skin color segmentation. As these publications are skin color specific we try to focus on other papers to select the color space corresponding to our application.

The most important set of color coding spaces is the set of chrominance spaces developed using knowledge about human color perception (Terrillon et al. 2000): The human visual system forms an achromatic channel and two chromatic color-difference channels

[1] AIBO Robot, Sony, `www.sony.net/Products/aibo/`

in the retina. It has considerably less spatial acuity for color information than for brightness. Therefore a color image can be coded into a wide-band component representative of brightness, and two narrow-band color components, where each color component has less resolution than brightness.

This propriety was used for video coding, and YC_rC_b, a typical chrominance color space, is commonly used by European television studios and for image compression. Color is represented by luma (which is luminance, computed from nonlinear RGB), constructed as a weighted sum of the RGB values, and two color difference values C_r and C_b that are formed by subtracting luma from RGB red and blue components.

$$Y = 0.299R + 0.587G + 0.114B \tag{4.1}$$
$$C_r = R - Y \tag{4.2}$$
$$C_b = B - Y \tag{4.3}$$

The transformation simplicity and explicit separation of luminance and chrominance components makes this colorspace attractive for our segmentation problem. Moreover as explained above, YC_rC_b is mainly use in video applications, which means that the camera used on the salamander could have YC_rC_b as native color space encoding. In our case, the radio-camera does not permit this option, therefore each frames are converted from RGB into YC_rC_b.

For a simplification of notation YC_rC_b will be denoted YCC.

4.2 Adaptation on Illuminant

Once YCC has been selected as the color space of our application, the illumination of the scene need to be studied in order to make the algorithm working independly from the source light. The channel Y also called luma or luminance do not represent the same quantification than the illuminant. The luminance corresponds to a quantitative photometric measure of the density of the luminous intensity in a given direction whereas the illuminant corresponds to the total luminous flux incident on a surface.

In this part, we focus on the term illumination which can be referred informally to the color temperature. The color temperature of a light source is determined by comparing its hue with a theoretical, heated black-body radiator. The Kelvin temperature at which the heated black-body radiator matches the hue of the light source is that source's color temperature. In other words, taking a picture with a day-light illumination will not render the same as taking the same picture neither with an indoor incandescent light nor a fluorescent light. From Figure 4.1 we can see that an object under incandescent light (color temperature of 3800K) will seem more reddish (red is more reflect) than the same one under a daylight illuminant.

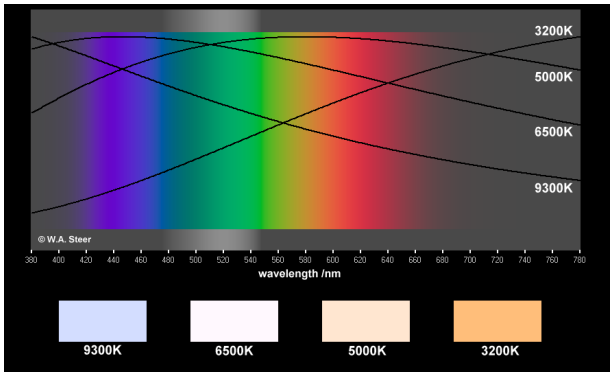


Figure 4.1: Black-body radiation curves for different color temperatures

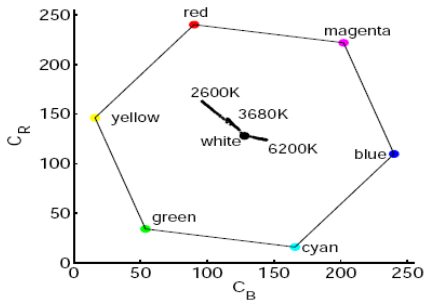


Figure 4.2: Reference white point for different color temperature in C_rC_b space

Human vision deals with these changes of illumination to make objects appear the same color under different color temperatures. A banana always appears yellow, whether viewed at night or during the day. This feature of the visual system is called chromatic adaptation, or color constancy. Different techniques have been introduce to simulate this effect (Finlayson et al. 1995). The white balancing is one of the most famous, and it consists to balance all the colors depending a *reference white*. The Figure 4.2 shows the white references in the C_rC_b space under different illuminations.

To make skin recognition independent from illumination, Hsu et al. (2002) introduce a technique that estimates "reference white" to normalize the color appearance. They regard pixels with top 5 percent of the luma (nonlinear gamma-corrected luminance) values as the reference white if the number of these reference-white pixels is larger than 100. Finally they introduce a non-linear transformation on YCC images to obtain a luma-independent chromatic colors space, and thus process their skin segmentation easier. Phung et al. (2002) decided to take into account the luma channel Y to process their skin detection algorithm approximating the luminance by 3 levels: low, medium, high.

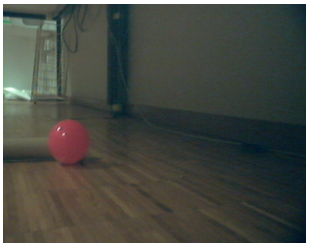
As a bad pre-white balancing is include in the native mode of our radio-camera and impossible to remove, the algorithm used by Hsu et al. (2002) is difficult to implement.

So, we have decided to use an approach similar to Phung et al. (2002) by approximating the luminance in 3 levels.

We compute the distribution in C_r and C_b channels of our pink ball, and the total luminance of our image under 3 different illuminations. The computation of the distribution was done by setting a handmade mask on the ball for different frames under each type of illumination. The Figure 4.3 shows the different type of illuminations and their corresponding masks.

- Low: Artificial illumination of the laboratory light with the robot and the pink ball under a table.
- Medium: Artificial illumination of the laboratory light with the robot directly under the source light.
- High: Day-light illumination with the robot and the ball near a window.

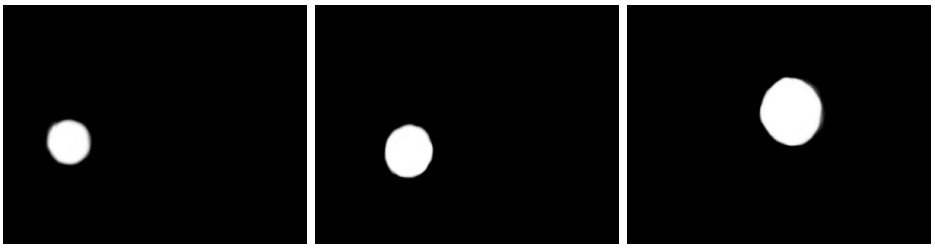
Finally we obtain 3 slightly different distributions depending on the luma value. The distribution under medium illumination is shown in next Section with Figure 4.5.



(a) *low illuminant*

(b) *medium illuminant*

(c) *high illuminant*



(d) The corresponding handmade masks

Figure 4.3: Images and their corresponding handmade masks under 3 different illuminants

An handmade mask has been draw for 5 different frames under the 3 types of illuminations, then all parameters have been computed for each illumination. The results of this experience are displayed in two tables. Table 4.1 show all the parameters to model the Gaussian density of the pink ball and Table 4.2 shows the parameters of the relative luma (Y channel) under the 3 different illuminations.

	Compute with the pink ball mask						
illumination type	μ_{Cr}	μ_{Cb}	σ_{Cr}	σ_{Cb}	ρ (CrCb)	μ_Y	σ_Y
low	161.57	120.73	4.67	1.86	-0.28	87.04	12.97
medium	174.06	124.37	13.60	5.63	-0.47	164.12	20.11
high	178.75	125.93	14.92	7.49	-0.52	182.06	21.13

Table 4.1: Parameters to model the Gaussian density of the pink ball under different illuminations

illumination type	μ_Y	σ_Y
low	94.16	29.59
medium	130.02	43.19
high	152.25	47.78

Table 4.2: Mean and standard deviation of the Y channel under different illuminations

with μ: the mean, σ: the standard deviation, ρ: the coefficient of covariance between C_rC_b.

Selecting the different parameters to model the pink ball distribution is done in function of the luma: The mean and variance of the whole Y channel (luma channel) are computed and then compared to their references under different types of illumination. A simple Gaussian function is used to determine which type of illumination generates the highest score.

Figure 4.4 illustrates the Table 4.1. It also demonstrates that it is easier to make color tracking when the illuminant is *low* due to the fact that standard deviation is very small. A *high illuminant* is the worst case due to the fact that the surface of the plastic pink ball is relatively specular and can behave like a mirror.

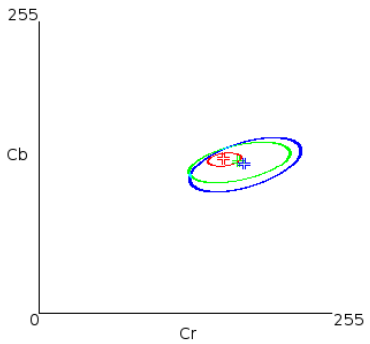


Figure 4.4: Representation of the Gaussian density parameters by an ellipse for each illuminant (Red ⇔ Low, Green ⇔ Medium, Blue ⇔ High)

4.3 Color density modelling

Before processing this algorithm we first blur our image in the C_r and C_b channel to remove undesired noise. (A mean blurring with a kernel of 3×3 pixels is applied to remove high frequency on color.)

Once we have converted our images in a 2D color space, we need to compare them with the different models obtained in Table 4.1. Probalistic density functions can be used to resolve this issue.

4.3.1 Gaussian density function

As the ball has a uniform color we can model it by a mono-modal distribution. Therefore the use of a single Gaussian density function is legitimate. Moreover Gaussians are fast to compute.

A multidimensional Gaussian in $\Re^p$ can be express as follows:

$$f(\vec{x}, (\vec{\mu}, \Sigma)) = \frac{1}{\sqrt{(2\pi)^p det(\Sigma)}} \exp\Big(-\tfrac{1}{2}(\vec{x} - \vec{\mu})^t \Sigma^{-1} (\vec{x} - \vec{\mu})) \tag{4.4}$$

with the covariance Matrix : $\Sigma_{i,j} = COV(X_i, X_j)$.

In the C_rC_b space we have (Bidimensionals space):

$$\Sigma = \begin{vmatrix} \sigma_{cr}^2 & \rho\sigma_{cr}\sigma_{cb} \\ \rho\sigma_{cr}\sigma_{cb} & \sigma_{cb}^2 \end{vmatrix} \tag{4.5}$$

As explained in Section 4.2, the different parameters of the Gaussian distribution are taken from Table 4.1 according to the luma values of the whole image. For more clearness, in all figures presented further we used parameters coming from a *medium illumination*.

In a perfect color space we should have both chromacities completely independent, which is not our case. So we compute our model with and without the coefficient of covariance ρ. In figure 4.5 we can see that taking into account the covariance we have the Gaussian distribution rotated. This mean that when the color change on the red chromacity, it also modifies a little bit the blue chromacity. We can also see in this example that we have much more variation in the C_r channel due to the fact that we set-up our detector on a pink ball.

For the further explanation we expand the expression in the exponential to obtain:

$$((\vec{x} - \vec{\mu})^t \Sigma^{-1} (\vec{x} - \vec{\mu})) = \frac{1}{det(\Sigma)}((\Delta_{cr}\sigma_{cb})^2 + (\Delta_{cb}\sigma_{cr})^2 - 2\rho\sigma_{cr}\sigma_{cb}\Delta_{cr}\Delta_{cb}) \tag{4.6}$$

with: $\Delta_{Cr} = (X_{Cr} - \mu_{Cr})$ and $\Delta_{cb} = (X_{cb} - \mu_{cb})$

The Gaussian function returns a number between 0 and 1. It will be mapped on a "byte" image (values $\in [0, 255]$).

4.3.2 Speed up

To accelerate our algorithm we want to discard the computation of the exponential when the probability of a pixel to be in our selected zone is really small (to have the smallest pixel with value 1 we need to have $\varepsilon = \frac{1}{255}$):

$$\exp(-\frac{1}{2}(\vec{x} - \vec{\mu})^t \Sigma^{-1}(\vec{x} - \vec{\mu})) \leq \varepsilon \tag{4.7}$$

$$(\Delta_{cr}\sigma_{cb})^2 + (\Delta_{cb}\sigma_{cr})^2 - 2\rho\sigma_{cr}\sigma_{cb}\Delta_{cr}\Delta_{cb} \geq -2det(\Sigma)log(\varepsilon) \tag{4.8}$$

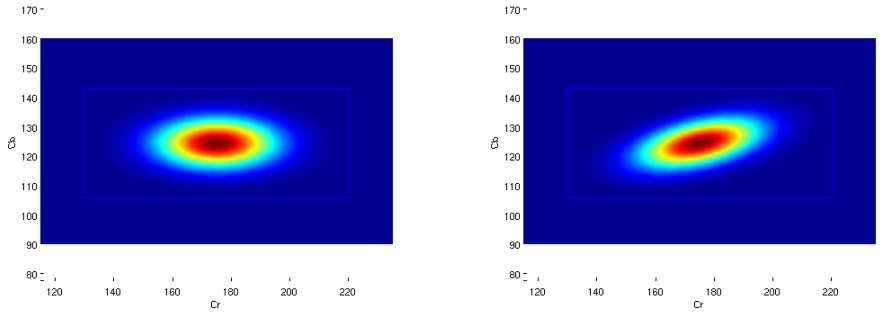
The *box* that have all the values upper than ε inside is defined by $boundary(\Delta_{Cr})$ and $boundary(\Delta_{cb})$ which correspond respectively to half of its width and height. To find this box we remove the terms of covariance $\rho = 0$. Intuitively, the covariance do not enter in this calculation due to the fact that only the two variances limit the gaussian density in spreading. This give us an expression that can be evaluated quickly in order to avoid computing the exponential and the square powers:

$$|\Delta_{cr}| > boundary(\Delta_{cr}) = \sqrt{-2\sigma_{cr}^2 log(\varepsilon)} \quad or, \tag{4.9}$$

$$|\Delta_{cb}| > boundary(\Delta_{cb}) = \sqrt{-2\sigma_{cb}^2 log(\varepsilon)} \tag{4.10}$$

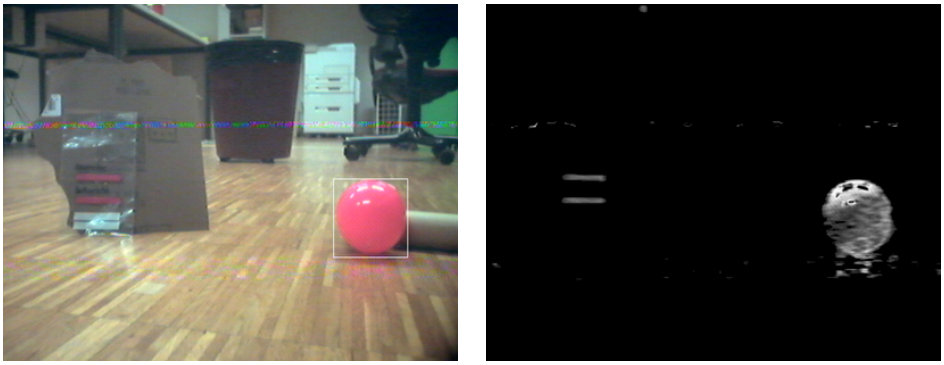
The representation of these two boundaries is done in Figure 4.5 by a blue *box*.

At the end of this algorithm, we obtain a probability map image (values $\in [0, 255]$) that represent the response of the gaussian density function. The probability map of Figure 4.6(a) is displayed in Figure 4.6(b).



(a) without covariance ($\rho = 0$) (b) with covariance

Figure 4.5: 2D Gaussian distribution and its *box* for the the pink ball's color under *medium illumination.*



(a) Original frame (b) Probability Map

Figure 4.6: Applying the gaussian density function with parameters computed under *medium illumination*

In Section 4.2 an offline illuminant adaptation algorithm has been presented. We have also tested a different process that try to adapt the gaussian parameters online:

We use a distribution of the pink ball color with a large variance to process the 2D gaussian density function. Then we obtain a probability map on a bigger zone than the ball. From this temporary map we select a new mean for C_r, and C_b (taking in account that they can not be far from the default values). With these new means we run another time the algorithm with a smaller variance in order to select only the pink ball this time. This algorithm works well if the illumation makes smooth changes which is not the case when the robot moves. It's why we prefer to use only the offline illuminant adaptation.

4.4 Finding the mask

4.4.1 Morphological operations

The probability map is then thresholded to create a mask. A threshold of 50 is applied which means that all pixels inferior to this limit are equal to 0 and the others to 255. To accelerate the computation, the formula to test the Gaussian *box* (Equation 4.9) could be used with $\varepsilon = \frac{50}{255}$, but we have preferred to keep the probability map in its original version for future processing (as online illuminant adaptation).

Once we obtain a mask of the pink ball, we use morphological operations in order to find a shape more similar to the pink ball. We process 2 times an ERODE operation in order to remove the noise. Then we process 4 times a DILLATE operation to try to fill the holes. These operations refer also to an OPENING operation with a kernel size for both operators equal to 3×3 pixels.

The Figure 4.7 represents each step of these operations and corresponds to the frame in Figure 4.6(a).

(a) Threshold operation (b) Eroded operation (c) Dilate operation

Figure 4.7: Simple operations applied to remove noise

This new mask that seems to correspond to the pink ball, have an area and a position that we need to communicate to our robot. In order to find these parameters we have developed different algorithms:

4.4.2 Mean position of the mask

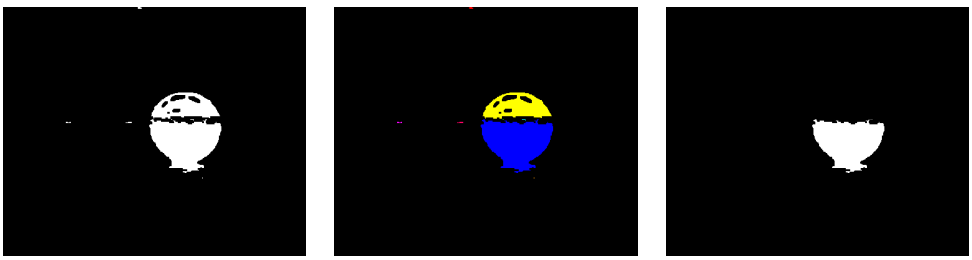
We first compute the mean and variance of the position of all pixels equal to one in our mask. This simple and fast algorithm work well if we don't have a lot of noises in the mask or if the pink ball is the only pink element in all the images. For example in Figure 4.8 there is two small pink objects on the left of the image and our ball on the right. This algorithm find the mean position in the center of the image and the variance on x coordinate extremely high.

Figure 4.8: Square is given by $center = (\mu_x, \mu_y)$ and $[width, height] = 4 * [\sigma_x, \sigma_y]$

4.4.3 Blob algorithm

A blob algorithm has been tested to solve this problem. It sets the same label to all pixels that are contiguous. Once this process is done we look for the biggest component connex (the most used label), and then compute the mean and variance of all pixels position in this blob.

However this algorithm has two drawbacks: Firstly, its cost in computation for finding the blob. Secondly the fact that is not readily adaptable when the pink ball color mask is separated. For example when the camera receiver transmits interferences, noisy-lines can be observed on our videos. Figure 4.9 illustrates how these kind of lines typically cut the mask in two parts and the problems generated for further processing.



(a) Original Mask (b) Blob labeling (c) Selected Blob

Figure 4.9: Problems in blob selection when interferences divide the mask

4.4.4 Finding the best square

Another fast algorithm has been applied to beat this problem using the fact that we are looking for a circle shape. We have define a simple rule where the score of a square corresponds to:

$$score(x, y, d) = \sum_{\substack{\|x' - x\| \leq d \\ \|y' - y\| \leq d}} mask(x', y') - \kappa d^2 \tag{4.11}$$

This equation mean that we are looking if a square have the highest mask area (number of white pixel) knowing that in the corners there is no ball (black pixels). The κ represents the black pixel in the corners and it permits to neither take the biggest square where we have all white pixels ($\kappa = 0$), nor a small one full of white to not introduce error ($\kappa = 1$).

In a formal way, κ derives from the circle and square areas. We have computed the formula bellow knowing that our mask never has a perfect circle shape.

$$\kappa = 1 - \frac{\pi r^2}{d^2} = 1 - \frac{\pi r^2}{4r^2} = 1 - \frac{\pi}{4} \simeq 0.2 \tag{4.12}$$

with $d = 2r$ = square size, r = radius

Now that this rule is defined we need to compute this operation on all possible squares in the image in order to find the highest score. Doing this operation in real time using a brutal method is impossible.

We first reduce the number of squares size, starting with $d = 30$ and then incrementing it by a factor of 1.2 ($d_{next} = 1.2 \times d$). Taking the first square with a size $d = 30$ avoids computing a lot of operations and also removes frames with only small spots coming from noises.

Although we have reduced the set of squares to test, computing at each different scales and positions the sum of all pixels in the patch is still to slow. To deal with this issue we use integral images.

4.5 Integral Image

Integral image was first used in computer graphic for texture mapping (Crow 1984) and then in image processing (Simard et al. 1999). A famous face detection algorithm (Viola & Jones 2001) illustrates their efficiency.

The integral image at location x, y contains the sum of the pixels above and to the left of x, y inclusive:

$$iim(x,y) = \sum_{y'=0}^{y} \sum_{x'=0}^{x} im(x',y'), \tag{4.13}$$

where $iim(x,y)$ is the integral image and $im(x,y)$ is the original image (see Figure 4.10).

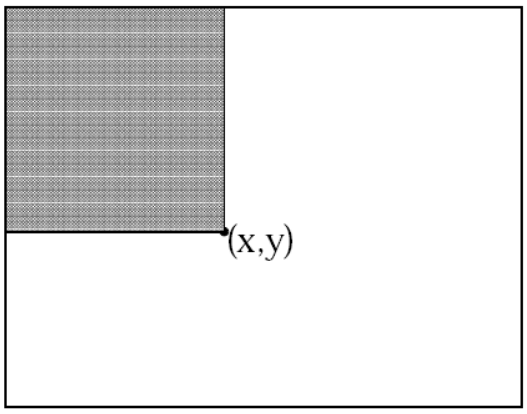


Figure 4.10: The value of the integral image at point x, y is the sum of all the pixels above and to the left.

We can use the following relation of recurrence to compute more efficiently the integral image:

$$s(x,y) = s(x,y-1) + im(x,y) \tag{4.14}$$
$$iim(x,y) = iim(x-1,y) + s(x,y) \tag{4.15}$$

(where $s(x,y)$ is the cumulative row sum, s(x,-1)=0, and iim(-1,y) =0).
The integral image can be computed in one pass over the original image.

Now using the integral image we can compute the sum of pixels over any rectangle with only four array references (see Figure 4.11).

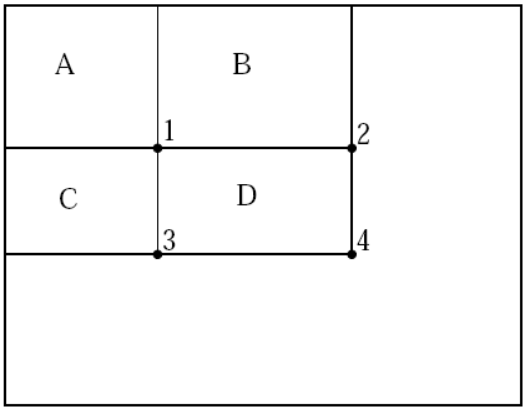


Figure 4.11: The sum of the pixels within rectangle D can be computed with four array references. The value of the integral image at location 1 is the sum of the pixels in rectangle A. The value at location 2 is $A+B$ at location 3 is $A+C$, and at location 4 is $A + B + C + D$. The sum within D can be computed as $4 + 1 - (2 + 3) = A + B + C + D + A - (A + C + A + B) = D$

Once the integral image of the mask is computed, we apply the 4 array references scheme (Figure 4.11) and the formula 4.11 on different square position and size in order to find the best square. Figure 4.12 shows how the last algorithm can handle the problem of small pink object in a frame which is not the case of simple mean and var of position in Figure 4.8.

Figure 4.12: Square found using integral image and Formula 4.11.

4.6 Tracking Color Online

In this last part we implement our algorithm to make it work online with the robot. A class in *C++* was developed to provide communications between our program and the server connected to the robot by wireless. In this report, we will not extend on the implementation of this class. More informations can be find in the *C++* code.

The output of the color tracking algorithm return only two parameters: the position and the size of the square. Consequently, we play with only with two registers which respectively correspond to the turn and the drive of the robot (c.f. Section 3.2).

To make the motion of the robot realistic, a simple heuristic has been used. We compute the mean of the square size and its position respected to the center over $\frac{1}{5}s = 0.20 \times 25fps = 5$ frames and $2s = 2s \times 25fps = 50$ frames. We have represented this model using two different queue sizes for the two input parameters. The first one represent quick changes while the second represent changes more stable.

Then we use these two new outputs in order to make the robot:

- Accelerating to reach the ball when it is far away.
- Moving slowly or stopping when the ball is near.
- Looking for the ball making more rotation when no pink ball is founded.

These two outputs drastically change in some cases but the CPG in our robots integrates these parameters smoothly (See Figure 3.6 in Section 3.2). Besides, all these kind of robot attitudes under different stimulus has been encoded ad-hoc for the snake robot.

They should be improved in the future using more intelligent algorithm like the ones developed in machine learning. We could also used the variance and the derivate of the square size and position over different number of frames in order to collect more information and make the robot responds in a more realistic way.

5 Matching Patch Tracking

The second part of AV2SR project is focused on how to estimate the relative position of the robot. This position can be deduced by knowing the sequence of the transformations between each consecutive frames. Ideally this algorithm could also rectify the video stream for a tele-operator.

This algorithm can find a geometrical transformation by matching small patches from a frame to the next one. A patch represent a small part (20×20 pixels) of a frame on which image processing has been done. The matching is done looking at their grey-level histogram. As grey-level channel, we decide to use the first channel of the YCC image compute in Section 4.1 which represent the luminance of an image. Then we try to find the "best" transformation between these two frames using RANSAC.

5.1 Histograms matching

5.1.1 Classical histograms building

The first part of this algorithm is about building an histogram for each patch. We start selecting K patches at random in our images. Then we divide the grey level set in N bins. In our case we use $N = 16$, because it's a multiple of the cardinality of the grey-level set (256 levels).
Now that we have divide our set in 16 bins of width $\frac{256}{16} = 16$, for each bin we count the number of pixels in the patch belonging to this bin.
In another words, we compute iteratively :

- for bin#1 number of pixels in patch between [0,16[
- for bin#2 number of pixels in patch between [16,32[
- ...

This is the classical way of computing histograms, however this method can generate some problems when a pixel is at the border of a bin:
For example, in $frame_{(i)}$ a patch has all its pixels equal to 32, consequently it will vote 400 (20x20) times in bin#3. On next frame, the image did not move but the light has change a little bit. The same patch has now half of its pixels equal to 32, and the other half equal to 31. It will vote 200 times in bin#3 and 200 times in bin#2.

Therefore the matching patch algorithm will have difficulties to detect that it's the same patch with few changes due to illumination.

5.1.2 Histograms with votes in two bins

To avoid this kind of "border error" we can use an algorithm that makes a value vote in two bins of the histogram. To implement this, we use a linear rule and the fact that the sum of the vote of a pixel in two bins should be normalized (in our case 1):
If a pixel's value is exactly at the middle of bin we vote 1 in this bin and 0 in the others, then if this value is at a border between two bin we vote 0.5 in both. Then, this rule is linearly followed in the other cases (See Figure 5.1).

We look the two nearest bin centers (C_n, C_{n+1}) for a pixel p in the patch :
if $val_p \in [C_n, C_{n+1}]$:
we compute $bin_n = bin_n + \frac{\Delta-(val_p-C_n)}{\Delta}$ and $bin_{n+1} = bin_{n+1} + \frac{\Delta-(C_{n+1}-val_p)}{\Delta}$
with Δ = width of each bins, C_n=center of bin#n.

5.1.3 The layered Integral Bin Image (LIBI)

In order to find the corresponding matching of each patch we need to structure the frame for making the search faster.

First, we create a 3D matrix H, which number of layers is equal to the number of bins. Then each pixel of the original frame set a value at position x, y in each layer with the vote's result for the corresponding bin.
In our case we have a matrix (640x480x16). Then we apply the 2 bins votes histogram building method for each pixels in the images.

For pixel p(x,y) with $val_p \in [C_n, C_{n+1}]$ we write in our 3D matrix:

$$H(x,y,n) = 255\ \frac{\Delta-(val_p-C_n)}{\Delta}\ and, \tag{5.1}$$

$$H(x,y,n+1) = 255\ \frac{\Delta-(C_{n+1}-val_p)}{\Delta} \tag{5.2}$$

$$H(x,y,j) = 0 \quad (\forall j \neq n, n+1) \tag{5.3}$$

(In our example we have multiply by 255 our results because we have our 3D matrix with an unsigned char type.)

The figure 5.1 illustrate this construction with two points:

- with $p_1 = (x_1, y_1) \in [C_1 = 8, C_2 = 24]$ the layered bin images (H) have:

- $H(x_1, y_1, 1) = 255 * \frac{16-(15-8)}{16} \approx 255 * 0.56$
- $H(x_1, y_1, 2) = 255 * \frac{16-(24-15)}{16} \approx 255 * 0.44$
- $H(x_1, y_1, 3) = ... = H(x_1, y_1, 16) = 0$

- with $p_2 = (x_2, y_2) \in [C_2, C_3]$ the result on each layers is :

- $H(x_2, y_2, 1) = 0$
- $H(x_2, y_2, 2) = 255 * \frac{16-(39-24)}{16} \approx 255 * 0.06$
- $H(x_2, y_2, 3) = 255 * 0.94$
- $H(x_2, y_2, 4) = ... = H(x_2, y_2, 16) = 0$

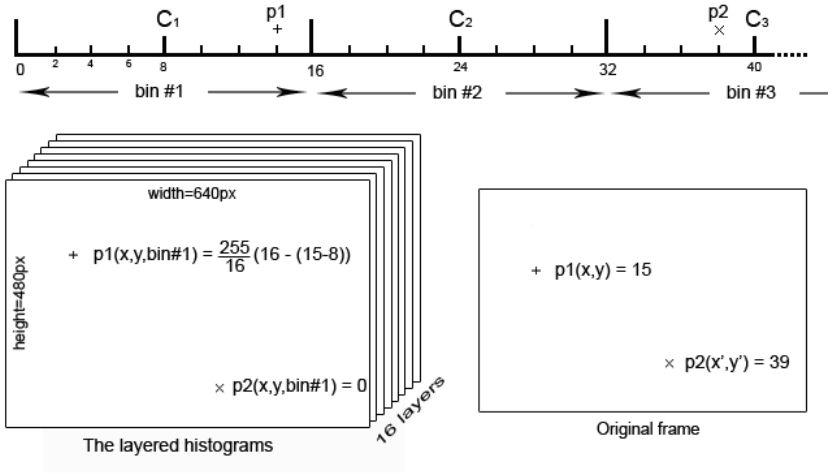


Figure 5.1: top: A schema of the histogram building for p_1 & p_2. right: Value of two pixel (p_1 & p_2) in a frame. left: The layered bin images with the result of pixel p_1 & p_2 show on $bin\#1$.

Now that we have our layered bin images (H) we want to browse them really fast and for each patch find the ones which are the most similar. In order to realize such operation with a low computation cost, we use integral image as describe in Section 4.5.

Thus, we compute for each layer of our layered bin image ($N = 16$ layers corresponding each one to a bin) an integral images. Therefore it will be easier to obtain the histogram of a patch using only 4 references per bin to have the sum of all the pixels in the patch. In the following part we call this new 3D matrix the LIBI (Layered Integral Bin Image) because each layer correspond to the integral image of the results of a bin.

In a recurrent scheme, the LIBI of $frame_{(i+1)}$ can be used to compute the K randomly

selected patches when we want to find their corresponding patches on $frame_{(i+2)}$. This shows that the LIBI is also used to build the histogram of each K randomly selected patches.

5.2 Matching patch algorithm

At this step of the algorithm we have a set of K patches belonging to the $frame_{(i)}$, and a LIBI (Layered Integral Bin Image) corresponding to $frame_{(i+1)}$. The purpose of this algorithm is to find in the $frame_{(i+1)}$ correspondences to the K patches of $frame_{(i)}$.

5.2.1 Selecting patch

The K patches selected randomly may have histograms that are really similar and in consequence difficult to identify. A solution to avoid this kind of problem and find a subset that contains only interesting/unique patches need to be implemented.

To solve this problem we first thought about using the entropy of an histograms. The principle of the entropy is used due to the fact that an histogram which have a lot of informations should be more interesting than one without divercity. To compute the entropy of an histogram we use the following formula derived from *Shannon entropy* (Balian 2003) :

$$entropy = \sum_{n=1}^{N} bin_n \times log2(bin_n) \tag{5.4}$$

Then the process that select the best patch do the following:

- First, select K random patches (with $K = 400$) and compute their corresponding histograms.
- Then compute the entropy of each patches.
- Finally select the $K' = 50$ that have the smallest entropy.

Unfortunately, the hardwood (parquet floor) contains a lot of informations due to the fact that its texture is particular. Therefore the selection of patches using the entropy was not helpfull in our case because all patches were selected on the hardwood on which the salamander moves. As this part represent one third of a frame, a patch can find a lot of corresponding matching patches which is not interesting for the matching patch algorithm.

Further improvement should be done to select better patches. The use of an algorithm that compare the Euclidian distance between histograms and select the ones which have the highest distance between themself should be implemented.

5.2.2 Find patch to match

Once we have compute the $LIBI$ (Layered Integral Bin Image) and selected K patches with their corresponding histograms, the research of matching patches can start. For each patch P_k in the selected list P ($\forall k \in [1, K]$) we call algorithm 1, which browse each position x,y of the $LIBI$, and for each layer $n \in [1, N]$ compute the difference between the corresponding bin of the P_k's histogram and the function $SumPatch(x, y, LIBI_n)$.

$SumPatch(x, y, LIBI_n)$ is a function which return the sum of all pixels in a patch starting at x, y on the layer n of $LIBI$ ($LIBI_n$). It's similar to the 4 array references describe in figure 4.11 taking $A = (x, y)$ and calculating B, C, D with the size of a patch (20×20) on an layer $LIBI_n$.

During the algorithm a special structure called $BMPS_k$ (Best Matching Patch Structure) is used. It permits to find M different matching patches for a corresponding patch P_k. $BMPS$ is composed as followed:

- $BMPS.error$: a vector of length M, $error = (error_1, ..., error_M)$
- $BMPS.coord$: a vector of length M, $coord = (coord_1, ...coord_M)$

(In this project setting $M = 3$ is enough)

The algorithm 2 (PushInMinBMPSList) permit us to conserve the $BMPS$ ordered and to operate easy insertion of new elements. The $BPMS$ have its minimum error at position $m = M$, so the best matching patch for the patch P_k have the coordinate $BMPS_k.coord_M$.

Algorithm 1 Matching patch algorithm (for one patch)

Input: k index of the patch selected.
P_k a patch of the selected list P.
$LIBI$ a 3D Matrix (width×height×N)
Output: $BMPS_k$ that have found the best matching with P_k

1: $width \leftarrow$ the first dimension of $LIBI$
2: $height \leftarrow$ the second dimension of $LIBI$
3: $N \leftarrow$ the third dimension of $LIBI$ *//the number of layers\bins*
4: **for** $y = 0$ to $height$ **do**
5: **for** $x = 0$ to $width$ **do**
6: $error \leftarrow 0$
7: **for** $n = 0$ to N **do**
8: $error \leftarrow error + (P_k[n] - SumPatch(x, y, LIBI_n))^2$
9: **end for**
10: $BMPS_k \leftarrow$ PushInMinBMPSList($BMPS_k, coord(x, y), error$)
11: **end for**
12: **end for**
13: **return** $BMPS_k$

Algorithm 2 PushInMinBMPSList

Input: $coord(x, y)$: coordinate of a pixel.
$error$: the error of the matching between P_k and $LIBI$ *at* (x, y)
$BMPS_k$: a structure describe in §5.2.2
Output: A $BMPS_k$ with error ordered (descending)

1: **if** $error > BMPS_k.error_1$ **then**
2: **return** $BMPS_k$
3: **end if**
4: $BMPS_k.error_1 \leftarrow error$
5: $BMPS_k.coord_1 \leftarrow coord(x, y)$
6: $m \leftarrow 1$
7: **while** $BMPS_k.error_m < BMPS_k.error_{m+1}$ and $m \leq M$ **do**
8: $swap(BMPS_k.error_m, BMPS_k.error_{m+1})$
9: $swap(BMPS_k.coord_m, BMPS_k.coord_{m+1})$
10: $m \leftarrow m + 1$
11: **end while**
12: **return** $BMPS_k$

5.2.3 Increase speed

The first implementation of the algorithm was really slow (about $7s$/frames), and so impossible to implement it on the salamander robot that need algorithm that can work in real time. We use some "tricks" to make it faster.

1. We change the line 7 of the algorithm 1 and break the loop if the actual error was bigger than the biggest error in $BPMS_k.error_1$. This simply avoid to continue looping on the next layers when the error with the first ones is already too high. With a low cost operation we reduce seriously the computation time (about $5.4s$/frames).

2. Then we decide to compare the mean and the variance of a patch P_k and the one starting at position x, y in $frame_{(i+1)}$. This operation is done at line 5 before making the test on the N layers of the histograms.

To compute the mean of a patch we use an integral image on the original frame (which is called IIm). We first compute the sum of all pixel in a patch by calling the SumPatch algorithm, And then we divide by the number of elements in the patch:

$$SumPatch(x, y, IIm) \Leftrightarrow \sum_{(x,y)\in P_k} pixel_{(x,y)},\ and\ so \tag{5.5}$$

$$E[P_k] = \frac{1}{20 \times 20} SumPatch(x, y, IIm) \tag{5.6}$$

For the variance we also use integral image, but this time we compute a square integral image (also called $ISqIm$) by computing the integral image on the original frame where each pixel is multiplied by itself. We easily obtain the second moment of the patch, and with a small operation with the mean we can have the variance:

$$SumPatch(x, y, ISqIm) \Leftrightarrow \sum_{(x,y)\in P_k} [pixel_{(x,y)}]^2,\ and\ so \tag{5.7}$$

$$E[P_k^2] = \frac{1}{20 \times 20} SumPatch(x, y, ISqIm) \tag{5.8}$$

Then we have,

$$Var[P_k] = E[P_k^2] - (E[P_k])^2 \tag{5.9}$$

The problem with this method is that we need to know how to thresholds the mean and the standard deviation in order to discard bad candidate to the matching histograms part. To fix these thresholds we build an experience set where:

- We have process the algorithm on 201 frames. (We use a pair $frame_{(i)}$ and $frame_{(i+1)}$ with $i \in [0, 199]$).
- We take K different P_k with $k \in [1, K = 50]$

- We use the $BPMS$ with $M = 3$ (3 Best Matching Patch). Thus we will get $M \times K$ (150) BMP per $frame_{(i+1)}$.
- And, we create 100 random "matching" patches on $frame_{(i+1)}$ for a corresponding patch P_k (We have $100 \times K(5000)$ random patches per $frame_{(i+1)}$).

For each pair of frames ($frame_{(i)}$ & $frame_{(i+1)}$), we compute the difference of mean and variance between all P_k and their 3 corresponding BMPs , then we do the same with their 100 corresponding RMPs.

1. We first look at the difference of the mean. The result of this experience is show on figure 5.2. We needed to plot this figure with a log scale due to the fact that more than 80% of the BMP have a difference/error less than 1 with the mean of their corresponding P_k. For errors smaller than 30 we have almost all BMPs (99.83%) and only one third of the RMPs (32.42%). This threshold make a good separation between the two set without rejecting a lots of BMPs
   With this threshold we reduce our computation by a factor of 3 (about $1.2s$/frame), due to the fact that the cost of computing the IIm and make a pre-checking is less than browsing all the N layers of the $LIBI$.

2. Then we do the same operation for the second moment, and we observe that we can't use the difference of second moment as a simple threshold because it doesn't give us more information than the difference of mean.
   If we look at figure 5.3 we see that if we threshold the mean difference ($E[MP_k] - E[P_k]$) with a maximum of 30. Adding a threshold on the second moment ($E[MP_k^2] - E[P_k^2]$) set at 12000 doesn't reduce our set of RPMs. (BMPs=99.83% and RMPs are only reduce from 32.42% $\rightarrow$ 32.35%).

3. We try the same operation using the variance, but it doesn't give us more informations than the second moment. Looking at figure 5.4, we set our threshold for the maximum difference of variance at 110 and we obtain with our training set a separation of 32.27% of RMPs and 99.83% of the BMPs.

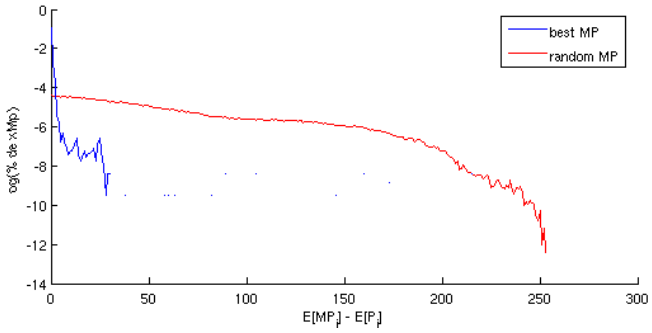


Figure 5.2: Log of the percentage of best/random matching patch that have a difference of mean with their corresponding P_k equal to $x \in [0, 1, ..., 255]$

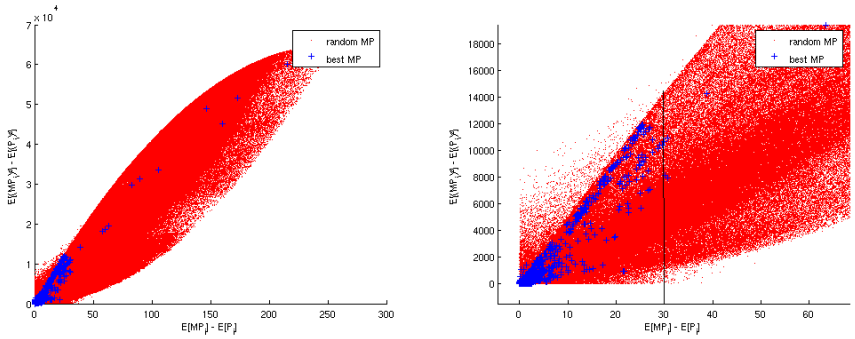


Figure 5.3: The distribution of the difference of mean (x axis) and second moment (y axis) between Best (Blue) or Random (Red) Matching Patch (MP) and their corresponding P_k

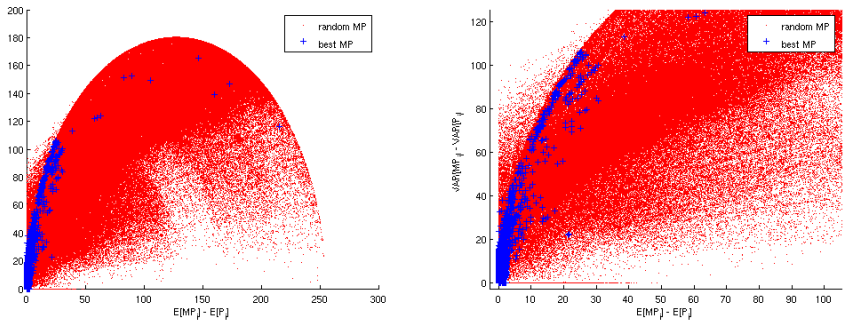


Figure 5.4: The distribution of the difference of mean (x axis) and variance (y axis) between Best (Blue) or Random (Red) Matching Patch (MP) and their corresponding P_k

In conclusion we don't use any threshold on the standard deviation or variance due to the fact that we loose more time computing the $ISqIm$ and checking the threshold than entering in the loop on all the layers. (line 7)

The fact is that this algorithm is still to slow, and that the checking of the mean is really good but not sufficient. Further improvement should be implemented to solve the problem of time dependency.

- The problem is maybe due to violations of caches during the browsing on these n layer, because we have our LIBI matrix which is of dimensions (width×height×N) and instead of browsing it starting by the bin layer, then the position y and finally the position x, we browse the bin layers on the last step (line 7 of algorithm 1) and this cause violation of cache. We should used instead of a layered image, an interleaved integral images where N consecutive values represent the integral histograms of all pixels before this one. The matching patch algorithm would be faster but maybe the time to construct this new interleaved integral bin images will made this algorithm difficult to use for real time purpose.
- Another thing is that we use integral image and 4 array references functions to make the computation faster, but as the size of the patch is fixed we should construct another type of "integral" image where we can get with one array reference the sum of pixels in the patch. We still have to test if we loose more time constructing this type of structure than we save browsing it.

5.3 R.A.N.S.A.C

The Random Sample Consensus (RANSAC) paradigm is an extremely powerful selection method invented by (Fischler & Bolles 1981). RANSAC is used to improve matching techniques that occur in many pattern matching applications of computer vision and computational geometry. One of the key difficulties of matching two point sets, is to detect those which match from those that do not. The matching points are called the inliers, and the remaining points are called outliers.

This standard definition (Nielson 2005) of RANSAC is slightly modified in our version. This adaptation comes from the fact that we have a priori knowledge of the correspondence between P_k, and 3 BMP_k in our set.

In our version we use pairs of points to find a similitude. Using only two points make RANSAC converging faster, and as the robot moves smoothly between two frames we do not need to find neither an homography nor an affine transformation. Referring to figure 5.5, the coordinates of a pair of patches are denoted by $P_k.coord = f_1, P_{k'}.coord = f_2$ and their corresponding BMPs by $BMP_{(k,m)}.coord = g_1, BMP_{(k',m')}.coord = g_2$.
In order to compute the similitude we need to compute the transformation matrix T given these two matching pairs $f_1 \leftrightarrow g_1$ and $f_2 \leftrightarrow g_2$.

- The first operation is a translation computing the midpoints of line segments $[g_1, g_2]$ and $[f_1, f_2]$: $\tau = \frac{g_1 - g_2}{2}$, $\tau' = \frac{f_1 - f_2}{2}$.
- Then a scaling factor is applied $\kappa = \frac{\|g_1 g_2\|}{\|f_1 f_2\|}$,
- and finally the rotation θ is calculated by using a simple arctan illustrated in figure 5.5 ($\theta = atan2(g_2.y - g_1.y, g_2.x - g_1.x) - atan2(f_2.y - f_1.y, f_2.x - f_1.x)$).

Finally, the transformation matrix T is obtained by the following transformation pipeline (with associated matrix concatenation):

$$T = \begin{vmatrix} I & \tau \\ 0^t & 1 \end{vmatrix} \begin{vmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} \kappa & 0 & 0 \\ 0 & \kappa & 0 \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} I & \tau' \\ 0^t & 1 \end{vmatrix} \tag{5.10}$$

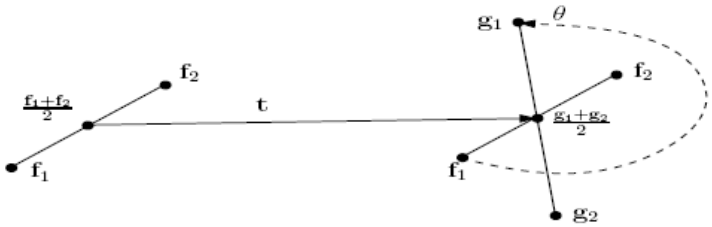


Figure 5.5: Similitude transformation is defined by a pair of matching features.

Once, the computation of the similitude is well defined we can start to use our version of RANSAC inspired from (Kazhdan 2005). In Algorithm 3, we have adapted RANSAC in the way that we need to compute only 9 different transformations for one pair of randomly selected $P_k, P_{k'}$ (line 3). Moreover the computation of the transformation's error is done in a special way:

- Firstly, the distance error generated by the transformed point $T(P_j)$, is not considered if it is outside the $frame_{(i+1)}$ (Line 10). In other words we do not expect to find a matching point that can not exist in the $frame_{(i+1)}$.

  Checking this condition, RANSAC may take a transformation that put all the points outside except the two used for computing the transformation, it's why we check at line 17 that we have compute this error with at least 20% of patches in the set P.

- At line 11, we have a special function that compute the distance between the transformed patch P_j and the nearest matching patch BMP. It return a linear normalized distance before a threshold and the maximum normalized error after. This function represented by the figure 5.6 avoid that one pair of points separated by a distance bigger than the threshold can be considered as better matching than another pair that also has a distance bigger than the threshold. In other words

when the distance between two points is bigger than the threshold we consider them as "no-matched" points.

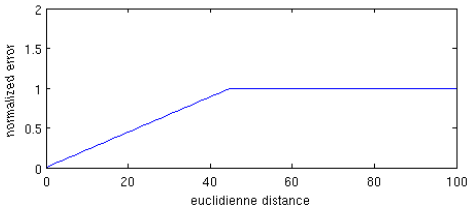


Figure 5.6: Error of matching between P_j and a BMP giving the Euclidian distance of the two points (Threshold=45px).

Algorithm 3 RANSAC (similitude)

Input: $maxround$ The maximum number of rounds that RANSAC can do
P: a list of src patches from $frame_{(i)}$
BMP: A list of best matching patches from $frame_{(i+1)}$
Output: T' a similitude transformation

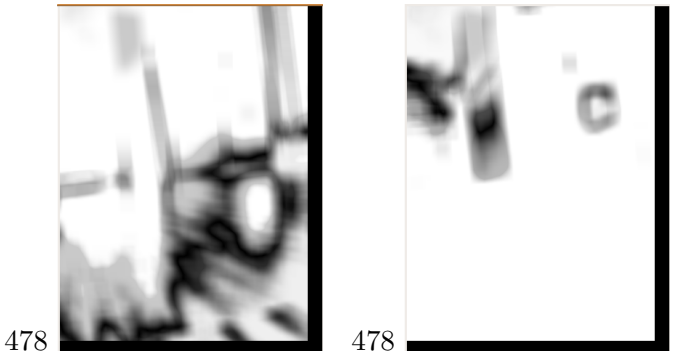
$round \leftarrow 0$
while $round < maxround$ **do**
  pick a random k and k' with $k, k \in [1, K' = 50]$
  for all $m, m' \in [1, M = 3]$ **do**
    $T \leftarrow GetSimilitude([P_k, P_{k'}], [BMP_{k,m}, BMP_{k',m'}])$
    $error \leftarrow 0$
    $nofpts \leftarrow 0$
    for all $j \in [1, K']$ such that $j \neq k \neq k'$ **do**
      $T(P_j) \leftarrow ApplySimilitude(P_j, T)$
      if $T(P_j) \in ImageSize$ **then**
        $dist \leftarrow$ smallest distance between $T(P_j)$ and a BMP
        $error \leftarrow error + dist$
        $nofpts \leftarrow nofpts + 1$
      end if
    end for
    $error \leftarrow \frac{error}{nofpts}$
    if $error < minerror$ & $nofpts > 20\%$ **then**
      $T' \leftarrow T$
      $minerror \leftarrow error$
    end if
  end for
end while
return T'

5.4 Experiment Results

This part shows how the matching patch algorithm work on video's frames with totally random selected patches. These experiments has been divided in two set:

5.4.1 Matching patch algorithm results

First, Figures 5.7(a) & 5.7(b) show the error of the matching patch algorithm for a $P_k,\ k \in [0, 24]$. The error is normalized between $[0, 1]$ (black pixel $\simeq 0$ and white pixel $\simeq 1$).



(a) k correspond to patch 13 in Figure 5.8(a)

(b) k correspond to patch 16 in Figure 5.8(b)

Figure 5.7: Normalized error of matching for P_k.

Then Figures 5.8(a) and 5.8(b) show 25 patches P_k selected from $frame_{(i)}$ and their corresponding 3 BMP_k compute on $frame_{(i+1)}$. The same pair of frames is used on the two figures, only the random distribution of the patches P_k have changed.

Figure 5.7(a) represents the matching histogram error for the patch 13 in the left part of figure 5.8(a). As patch number 13 is composed by a piece of the hardwood floor, a blurry black spot representing the minimum error, has been draw on Figure 5.7(a). The black spot is situated at the same place as the hardwood floor is (in Figure 5.8(a)(right)) excepting where the light is strongly reflected by the floor. Therefore this indicates that the algorithm has properly worked and wants to match the patches on a zone similar in grey-level distribution. Furthermore the result of the Best Matching Patches (BMP) selection is shown in the right part of Figures 5.8(a). For each P_k drawn in the left part, 3 BMP are placed in the right part. The one which has the smallest error is drawn in red, the two others best are drawn in blue. As the patch 13 is not really specific (the hardwood floor correspond to one third of $frame_{(i+1)}$), the area of the black spot in Figure 5.7(a) is big and in consequence the 3 selected BMP do not really correspond to the patch P_{13}.

Similarly, Figure 5.7(b) is associated to patch 16 selected from the left part of Figure 5.8(a). The minimum error (black spot) represent a smaller area than the one in Figure 5.7(a) due to the fact that patch 16 is more characteristic. Accordingly the 3 selected BMP are better situated by the matching patch algorithm.

The two Figures 5.7(a) & 5.7(b) have a bottom and right black border due to the fact that one pixel correspond to the top-left corner of a patch $P_{(x,y)}$, and as a patch can not be at postion $x \in [frame_{width} - patch_{width}, frame_{width}[$ and $y =\in [frame_{height} - patch_{height}, frame_{height}[$ a black pixel is set as a default value value.

(a) A random distribution of the P_k (corresponding to Figure 5.7(a))

(b) Another random distribution of the P_k (corresponding to Figure 5.7(b))

Figure 5.8: left: $frame_{(i)}$ ($n = 8$) and the 25 randoms P_k. right: $frame_{(i+1)}$ and the 3 corresponding BMP_k. (the 1^{st} best matching patch is in red and the 2^{nd} and 3^{th} are in blue)

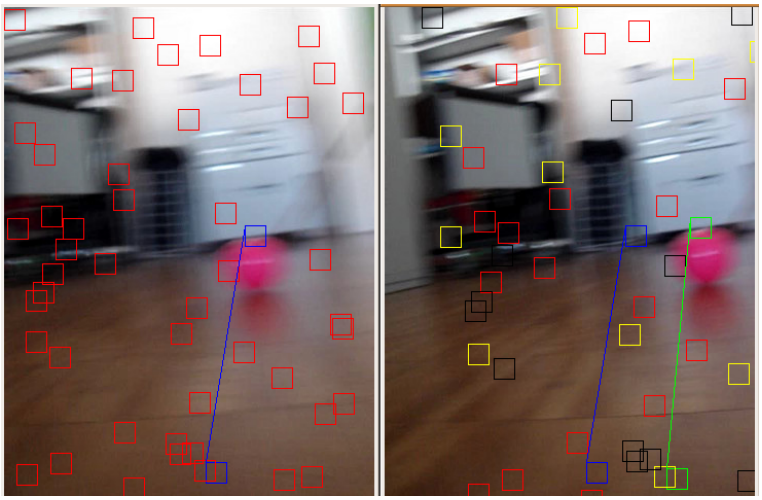
5.4.2 R.A.N.S.A.C results

Figures 5.9(a) & 5.9(b) illustrate the processing of RANSAC. To ease the comprehension, the previous notations of Section 5.3 are used : T the similitude transformation, P_j a selected patch, $T(P_j)$ the transformation's result of P_j.

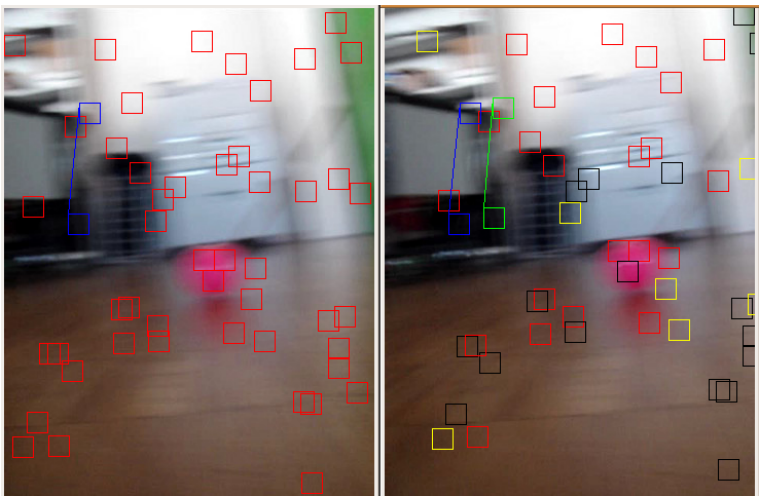
All the K patches used to compute RANSAC are drawn in the left part of Figures 5.9(a) & 5.9(b). The two patches selected randomly by RANSAC (Algorithm 3, line 3) are drawn in blue.

The transformation T is then used in order to place all $T(P_j)$ in the right part of Figures 5.9(a) & 5.9(b). The two pairs used to find the transformation T are also drawn (the original pair of patches is drawn in blue while their corresponding BMP are in green).

The color of the patches in the right part of Figures 5.9(a) & 5.9(b) is selected depending on the distance between the $T(P_j)$ and the nearest BMPs. The computation of this distance is explained in Section 5.3 with Figure 5.6. A red color means that a $T(P_j)$ has matched with a BMP, whereas a yellow color means that the distance between the $T(P_j)$ and the nearest BMP is short. When there is no possible matching we draw $T(P_j)$ in black.



(a) Frames 18 and 19



(b) Frames 52 and 53

Figure 5.9: left: $frame_{(i)}$ all the patches P_j. right: $frame_{(i+1)}$ with the corresponding transformation of patches $T(P_j)$.

Finally, the result of all the algorithm is illustrated with Figures 5.10(a) & 5.10(b). This two images represent the inverse transformation (T^{-1}) apply on $frame_{(i+1)}$. If it has worked properly this transformed image should be similar to $frame_{(i)}$.

(a) Correspond to Figure 5.9(a) (b) Correspond to Figure 5.9(b)

Figure 5.10: Inverse transformation of $frame_{(i+1)}$ illustrated in Figures 5.9(a) & 5.9(b), the result looks like the corresponding $frame_{(i)}$.

In conclusion, Figure 5.10(b) demonstrate the strength of RANSAC and its ability to deal with a lot of bad matching pairs. With only 44% of good matching patches (in red), and 14% of nearly matched patches: $T^{-1}(frame_{(i+1)})$ looks like $frame_{(i)}$.

6 Project Status

6.1 Conclusion

In the AVS2R project we have developed two main algorithms:

6.1.1 Color tracking

The fast color based tracking allow our robots to follow a pink ball It is a robust algorithm that can handle different illuminations. It also leads to a better understanding of the integration and feedback for the different stimulus (position and distance of the pink ball). Moreover, it proves that if the color tracking generates bad inputs in some frames, the CPG of the robot handles them, and the movement of the robot still looks realistic.

6.1.2 Matching patch algorithm

Trying to find the sequence of the different transformations in order to estimate the relative position of the robot is difficult task. The use of histogram patches for matching is not common, in this way we can say that our second algorithm has been developed in a research purpose. It can not be applied online due to its heavy computation cost (it can only process about $1.4fps$). However, interesting thing has been demonstrated during its development:

It illustrates the strength of RANSAC even with a great number of bad matching pairs. It also shows how histograms can be used for detection.

6.2 Future Improvements

In addition to the current capabilities mentioned above, the following future improvements could be done at each levels of the AVS2R projects

6.2.1 Hardware improvements

The robot should benefit a more bio-inspired vision and gaits more in relation with its artificial vision:

- Two embedded cameras should be added in order to obtain stereovision (Section 2.2).

- The robot should have a travelling wave with increasing amplitude from the head to the tail in order to avoid big movements of the head and loose our goal. Similarly, we could also let the head looking for the pink ball and then transmits its relative angle to the rest of the body. (Section 3.2).

- Fish-eye lenses could also be use to obtain a larger vision field.

6.2.2 Color tracking

- Improvements should be done to obtain a better chromacity adaptation under various illumant (Section 4.2) . We should merge offline and online adaptation in order to obtain color constancy. The ball should always appear pink to the robots due to the fact that it has a previous knowledge of the ball color and it should also conform to the illumination of the actual environment.

- The color tracking algorithm should deal in a better way with the presence of several pink objects (Section 4.4). This can be done by knowing the previous position of the pink ball that it was following, and stay focused only on this object.

- The input stimulus send on the fly should be used with more heuristics in order to generate a more realistic gaits (Section 4.6). Keeping in memory the last positions of the ball, we should create better rules with machine learning algorithm to generate a correct feedback. This feedback could make the robot behave in several ways following its different goals (accelerating to reach the ball, stopping when it is near,...).

6.2.3 Matching patch tracking

- Improvement can be done in selecting the patches, increasing the dispersity of patches for instance. Image processing like colors, edges, etc should also be used to find the best key patches (Section 5.2.1).

- A faster computation can also be achieved using interleaved bins image instead of layered bins image. A specific integral image can also be generated (Section 5.2.3).

- A sequence of transformations should be memorized to estimate the relative motion of the robot. This estimation should consider the fact that bad transformation can be introduced in some frames. Then it should detect them using the fact that the gaits of our robot is generated by some redundancies.

Finally, we have to consider that different algorithms could also perform the same tasks. Using bio-inspired algorithm such as stereo-vision and coarse coding (e.g. Eurich et al. 1997, Ijspeert & Arbib 2000), or saccadic search with log-polor vision field and gabor filters (e.g. Lim et al. 1996, Rao et al. 1997, Smeraldi et al. 2000), could be the best way of implementing artificial vision keeping the philosophy of a bio-inspired robot.

Bibliography

Anisetti, M., Bellandi, V., Damiani, E., Beverina, F., Arnone, L. & Rat, B. (2006), A3fd: Accurate 3d face detection, *in* 'Proceeding of IEEE International Conference on Signal-Image Technology and Internet Based Systems IEEE SITIS'06,Hammamet, Tunisia'.

Balian, R. (2003), Entropy, protean concep, *in* 'Poincare Seminar 2', pp. 119–145.

Crespi, A. & Ijspeert, A. (2006), AmphiBot II: An amphibious snake robot that crawls and swims using a central pattern generator, *in* 'Proceedings of the 9th International Conference on Climbing and Walking Robots (CLAWAR 2006)', pp. 19–27.

Crow (1984), Summed-area tables for texture mapping, *in* 'SIGGRAPH', Vol. 18 of *3*, pp. 207–212.

Eurich, C. W., Schwegler, H. & Woesler, R. (1997), 'Coarse coding: applications to the visual system of salamanders,', *Biological Cybernetics* **77**, 41–47.

Finlayson, G., Funt, B. & Barnard, K. (1995), 'Color constancy under varying illumination', *iccv* **00**, 720.

Fischler, M. A. & Bolles, R. C. (1981), 'Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography', *Commun. ACM* **24**(6), 381–395.

Hsu, R.-L., Senior, A., , Mottaleb, M. A. & Jain, A. K. (2002), 'Face detection in color images', *IEEE Trans. Pattern Anal. Mach. Intell.* **24**(5), 696–706.

Ijspeert, A. (2001), 'A connectionist central pattern generator for the aquatic and terrestrial gaits of a simulated salamander', *Biological Cybernetics* **84**(5), 331–348.

Ijspeert, A. & Arbib, M. (2000), Visual tracking in simulated salamander locomotion, *in* J. Meyer, A. Berthoz, D. Floreano, H. Roitblat & S. Wilson, eds, 'Proceedings of the Sixth International Conference of The Society for Adaptive Behavior (SAB2000)', MIT Press, pp. 88–97.

Ijspeert, A., Buchli, J., Crespi, A., Righetti, L. & Bourquin, Y. (2005), 'Institute presentation: Biologically inspired robotics group at EPFL', *International Journal of Advanced Robotics Systems* **2**(2), 175–199.

Kazhdan, M. (2005), 'Shape matching for model alignment', ICCV 2005 Short Course. Johns Hopkins University.

Lim, F.-L., Venkatesh, S. & West, G. A. (1996), Human saccadic eye movements and tracking by active foveation in log polar space, *in* B. E. Rogowitz & J. P. Allebach, eds, 'Proc. SPIE Vol. 2657, p. 338-349, Human Vision and Electronic Imaging, Bernice E. Rogowitz; Jan P. Allebach; Eds.', pp. 338–349.

Nielson, F. (2005), *Visual Computing: Geometry, Graphics, And Vision*, Charles River Media/Thomson Delmar Learning, chapter 7, pp. 400–405. RANSAC Algorithm.

Phung, S., Bouzerdoum, A. & Chai, D. (2002), A novel skin color model in ycbcr color space and its application to human face detection, *in* 'ICIP02', pp. I: 289–292.

Rao, R. P. N., Zelinsky, G. J., Hayhoe, M. M. & Ballard, D. H. (1997), Eye movements in visual cognition., Technical Report NRL97.1, National Resource Laboratory for the Study of Brain and Behavior, Department of Computer Science, University of Rochester.

Simard, P., Bottou, L., Haffner, P. & Cun, Y. L. (1999), Boxlets: a fast convolution algorithm for signal processing and neural networks, *in* 'Advances in Neural Information Processing Systems', Vol. 11, pp. 571–577.

Smeraldi, F., Carmona, O. & Bigun, J. (2000), 'Saccadic search with gabor features applied to eye detection and real-time head tracking', *Image and Vision Computing*.

Storring, M. (2000), Computer Vision and Human Skin Colour, PhD thesis, Faculty of Engineering and Science, Aalborg University, E-mail: mst@cvmt.dk URL: http://www.cvmt.dk/mst.

Terrillon, J.-C., Fukamachi, H., Akamatsu, S. & Shirazi, M. N. (2000), Comparative performance of different skin chrominance models and chrominance spaces for the automatic detection of human faces in color images, *in* 'FG '00: Proceedings of the Fourth IEEE International Conference on Automatic Face and Gesture Recognition 2000', IEEE Computer Society, Washington, DC, USA, p. 54.

Vezhnevets V., A. A. (2005), A survey on pixel-based skin color detection techniques, Technical report, Moscow, Russia, Graphics and Media Laboratory. GML Computer Vision Group, 2005.

Viola, P. & Jones, M. (2001), 'Robust real-time face detection', *Int J Comput Vision* **57**(2), 137–154.