Master Project

Robotics applications of vision-based action selection

Author:
Matteo De Giacomi

Supervisor:
Prof. Auke Ijspeert

26th July 2007

Acknowledgements

First I would like to thank Elia Palme for realizing the stereo algorithm and for helping me. We had a very enjoyable and motivating collaboration, and got along extremely well.

Alessandro Crespi, for realizing the two robots I worked on, and for being able to repair every circut, fix any bug and cheer everybody up with his neverending good mood and his sometimes crazy creations.

Professor Auke Ijspeert, for giving me the opportunity of working on such an interesting project.

Professor Jean-Marie Cabelguen for an interesting discussion about real salamanders. It will take a while before *Salamandra* will be able to do the same things, but I did my best.

All the lab members for their great team spirit and friendly attitude. Good luck to you, guys!

Last, but definitely not least, I would like to thank family for supporting me from primary school until now, and in particular my beloved grandmother. She always supported me in every moment of my life, and has always been proud of me as I was proud of her. Grazie nonna.

Contents

Chapter 1

Project Objectives

The conception of an autonomous robot is a challenge involving the application of knowledges provided by various research fields.
Bio-inspired robotics is a relatively recent paradigm according to which biology and robotics can be mutually helpful.

In the scope of bio-inspired robotics, the modular robots *Amphibot II* and *Salamandra Robotica* have recently been developed. Their look and locomotion system are concieved to mimick the ones of a snake and a salamander respectively. Results show that complex behaviors needing tight synchronisation and adaptability to the everchanging environmental conditions can be easily obtained by inspiring the control system to the way an animal Brain Stem generates the various locomotion patterns.

This project aims at the development of a reactive terrestrial navigation system for such robots.
In detail, we want our system to control the locomotion system already implemented such that the robot is able to perform several simple behaviors (obstacle avoidance, fleeing from a predator, prey hunting) and manage correctely the priorities between them.
Inputs are provided by two cameras generating a stereo information and by motor feedback.

The implementation phase of our control system went through several steps. Initially it has been tested in simulation, afterwards it has been implemented on the two modular robots cited above.

Chapter 2

Amphibot II and *Salamandra Robotica*

Through evolution, the physical structure of vertebrates is being modified, though locomotion still relies on the same principle: a synchoronous muscle activation is produced by signals generated in the brain stem and propagated to the CPG along the spinal chord. No high level cognitive processes are needed to trigger this muscular activity, though they can influence it to produce an high level behavior.
One of the most simple representants of vertebrates still living on earth is the lamprey. This animal is able to produce an efficient aquatic locomotion by activating in a synchronized way the muscles along its back so that a travelling weave is produced.
Another interesting case study for understanding the evolution of vertebrates is represented by salamanders. In the last 150 millions of years their physical structure has been practically unchanged, thus providing us with a good example of how the look of an ancient sensory-motor system was. If we couple this information to the fact that about 10% of the *Batracians* are salamanders [1], we can conclude that this animal is both extremely simple but also extremely effective.

2.1 The Robots

Amphibot II and *Salamandra Robotica* are amphibious modular robots developed in the scope of the Phd Thesis of Alessandro Crespi [10], and are concieved to mimick both the body structure and the brain stem of a lamprey and salamander respectively. Both *Amphibot* and *Salamandra* are realized

[1] from a discussion with prof. J-M. Cabelguen

with similar building blocks: the first is composed of 8 blocks called *body elements*, whereas the latter is composed of 6 body elements and 2 *legs elements*. Body elements contain a motor allowing a lateral oscillation, whereas body elements present 2 rotating legs connected to its sides.

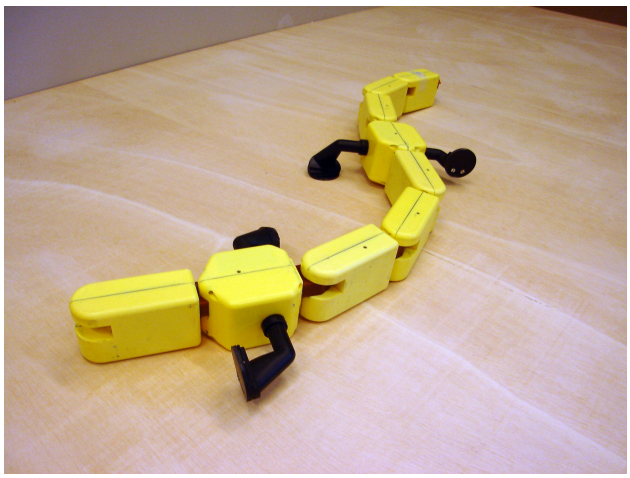


Figure 2.1: *The* Salamadra Robotica *robot.*

Elements are connected together and controlled by a set of coupled differential equations. These equations, similar for both robots, can be easily controlled through a unique signal called *Drive*. This system is presented in chapter 2.2 and 2.3
It has to be pointed out that recently it has been shown that *Salamandra Robotica* can actually be compared to a real animal [14]. This clearly shows the goodness of this architecture in modeling real biological systems.

2.2 CPG controller

We define a perturbed dynamical system as

$$\dot{\vec{q}} = \vec{F}(q) + p \tag{2.1}$$

Where q is a vector representing the state variables of the dynamical system, and p a perturbation vector. If the system, with $p = 0$, generates a stable set of solutions around a circle, we call the set of solution limit cycle, and call the dynamical system an oscillator.
Salamandra and *Amphibot* are controlled by a phase oscillator, which is a system of non-linear differential equations of the following form:

$$\begin{aligned} \dot{\theta} &= 2\pi\nu + p_\theta \\ \dot{\dot{r}} &= a\left(\tfrac{a}{4}(R - \dot{r})\right) \\ \dot{x_i} &= r(1 + cos(\theta)) \end{aligned} \tag{2.2}$$

where r and θ are the state variables of the system (polar coordinates), ν is the desired frequency, R the desired amplitude, a the convergence speed and p_θ is a perturbation vector on the θ state variable.
The introduction of a perturbation in the oscillator may produce different behaviors; an interesting solution may be obtained by perturbing it with the state variables of another oscillator.
The coupling of two or more oscillators may produce a phase locked state, which means that the difference between the phase of the perturbed oscillator and the phases of the perturbing ones is approximatively constant.
Coupling in a phase oscillator can be easily obtained in this way:

$$\begin{aligned} \dot{\theta}_i &= 2\pi\nu_i + \textstyle\sum_j r_j w_{ij} sin(\theta_j - \theta_i - \phi_{ij}) \\ \dot{\dot{r}}_i &= a_i\left(\tfrac{a_i}{4}(R_i - \dot{r}_i)\right) \\ \dot{x_i} &= r_i(1 + cos(\theta_i)) \end{aligned} \tag{2.3}$$

where the new term ϕ_{ij} defines the desired phase between the oscillator i and the oscillator j, and w_{ij} the coupling strength between them.

This architecture allows the realisation of a controller for a robot producing a serpentine locomotion, such as *Amphibot II*.
For each servo, two oscillators bidirectionnally coupled with a phase of $\phi_{ij} = \pi$ model the behavior of two antagonist muscles in the animal's back. The oscillation of a spine servo is defined by the difference between the values of the two oscillators corresponding to its body element. Oscillators located on the robot's spine have been coupled so that a travelling wave, corresponding to a serpentine locomotion, is spontaneously produced. This can be achieved by imposing a certain phase in the coupling of the different sets of antagonist oscillators.

As said in the previous chapter, salamanders may be considered as an evolution of lampreys. On a spinal chord able to generate a travelling weave four legs are connected, allowing the generation of a walking gait. To model the spinal chord of a salamander, the schema shown in Figure 2.2 is used.

Walking locomotion is produced by adding to the previously introduced system four oscillators descibing the movement of the four legs, unidirectionnally coupled to the spine with a w_{ij} bigger than the one coupling the spine

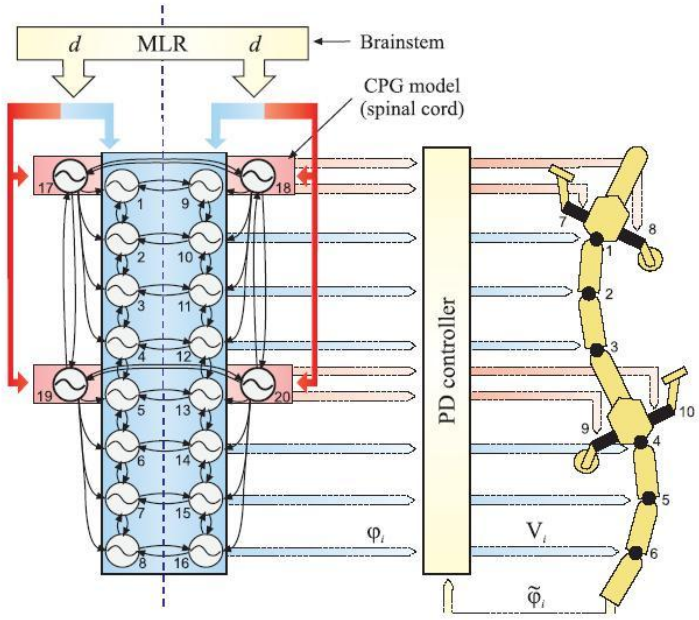


Figure 2.2: *Oscillator's coupling schema and relation with the robot's servos in* Salamandra Robotica. *Four legs are unidirectionnally coupled to the spinal chord.*

elements. While in swimming locomotion the legs' servos are motionless (amplitude and frequency equal to zero), in walking gait they move. Since their coupling strength to the spine oscillators is greater than the coupling between the latters, spine oscillators are forced to produce a standing wave.

2.3 Control of speed and direction

The CPG described in Chapter 2.2 is embedded in the robot. Trough a wireless communication it's possible to provide legs and body's amplitude and frequency, gain and phase difference between head and tail (first and last body element). These values are sufficient to define precisely the robot's speed and direction.

Robot's speed can be controlled by varying its amplitude and frequency of oscillation. A good relation between those two parameters allows the robot to produce a performing locomotion. In other words this means that the robot is able to advance without slipping on the soil.
To obtain such a relation a Drive signal is introduced. This signal, that

can be interpreted as a tonic input in the brain stem, is converted to the amplitudes and frequencies provided to the motors according to the following linear equations:

$$\begin{array}{ll} a = k_2 \cdot d & if\ d_{min} < d < d_{max} \\ f = k_1 \cdot d & if\ d_{min} < d < d_{max} \\ a = f = 0 & else \end{array} \tag{2.4}$$

where d is the Drive signal, a the amplitude, f the frequency and d_{min} and d_{max} boudaries where the robot is able to move correctly (values high enough to produce movement, but not high enough to compromise the motor performance). The values of k_1, k_2, d_{min} and d_{max} are critical and, specially in *Amphibot*, they depend on the kind of surface the robot is walking on. Used values will be introduced in chapter 5.3.2.

Direction is controlled by a Turn signal. Its effect is to impose a difference between the amplitude of oscillators on the left (a_L) and right (a_R) side of the spine:

$$Turn = \frac{a_L - a_R}{a_L + a_R} \tag{2.5}$$

Turn varies therefore between -1 (turn left) and 1 (turn right).

Chapter 3

High Level Behavior

Obtaining a high level behavior in a mobile robot is one of the most important challenges in robotics. Several approaches have been developed in the last 20 years, providing different features to the control system.

In planning-based architectures each decision is taken according to a set of objectives the robot has to satisfy according to a previously built knowledge base about the environment.
This method is in general not adapted to dynamic environments, for that reason some architectures (such as *THEO* [30] or *SOAR* [18] developed in the domain of artificial intelligence) are concieved to allow replanning. Given the large quantity of memory needed and the difficulty of performing replanning in real-time we don't cosider this approach as useful for a mobile robot having a limited quantity of memory and computationnal power.

In contrast with these architectures, reactive systems are just concerned by the next action to take, and tend to be more adapted to react in real-time to environmental stimuli.
Artificial Neural Networks may be considered as reactive systems. They can provide quick reactions and take a limited amount of memory. However they need to be trained with input examples and, in the context of our project, this phase may become extremely tricky.

The Potential Field approach, originally proposed by Andrews and Hogan followed by Kathib [19], interpretes robot behavior as a reaction of imaginary forces coming from the environment. Robot is therefore attracted thoward a specific goal while repulsed by obstacles and other dangers. This approach has a good success among roboticists, though few criticisms have been for-

mulated [32]. In fact, drawbacks such as trap situations due to local minima, no passage between closely spaced obstacles and oscillations in presence of obstacles have been shown.

The steering technique proposed by Craig Reynolds [26] is based on a similar approach. This architecture relies on the idea that complex behaviors can be decomposed in a set of simple actions simultaneously producing an output according to the environmentals conditions, and that emergent behavior may be obtained by summing up all their contributions. Every behavior produces a certain force (attraction), and at every instant the direction selected by the mobile robot is determined by the vectorial sum of all the actions'outputs. This approach is however more adapted to simulations since it relies on the quality of sensing, which is a critical issue in real robots.

Arkin proposes a Schema-based approach [5]. A Schema is defined as an independant process able to activate determined perceptual and motionnal tasks in order to modify the environment. The system is based on an action-perception cycle: perception phase modifies an internal representation of the environment, and activates a set motionnal patterns that will modify the environment. The various patterns act indepentently one to each other, so that the final output is obtaind by a simple vectorial sum of all their contributions.
A schema-based architecture supported by ethological observations about mantis was implemented on the hexapod robot Miguel [2].

Rodney Brooks's Subsumption Architecture [7] relies on a behavior-based approach. This system is based on the idea, also accepted by etologists, that complex behaviors emerge from a set of simple indipendant behaviors, and that at each instant just one behavior is selected. Unlike the Schema-based approach, the Subsumption Architecture does not need an internal representation of the environment. Behaviors, organized in a layered augmented finite state machine according to predefined priority, are tightly connected to actions and hardware. This structure allows a quick reaction time, is robust to perturbations and may show emergent behavior given a sufficiently complex structure.

Since we want our control system to be able to face a dynamic unpredictable environment in real-time using just limited sensing capaticities, a Subsumption-like Architecture seems well adapted. We'll have however to accept its drawback: this technique is extremely limited in terms of planning.

3.1 Control Architecture

In the scope of this project we are interested in the implementation of three basic behaviors: Obstacle Avoidance, Predator Fleeing and Prey Hunting. The process of action selection is summarized in Figure 3.1 and based on a classic subsumption architecture.
In our architecture, possible actions are sorted according to a fixed priority level; when a certain action is selected, all the ones having lower priority are bypassed. Priorities have been selected according to the importance for robot survival represented by the different actions.

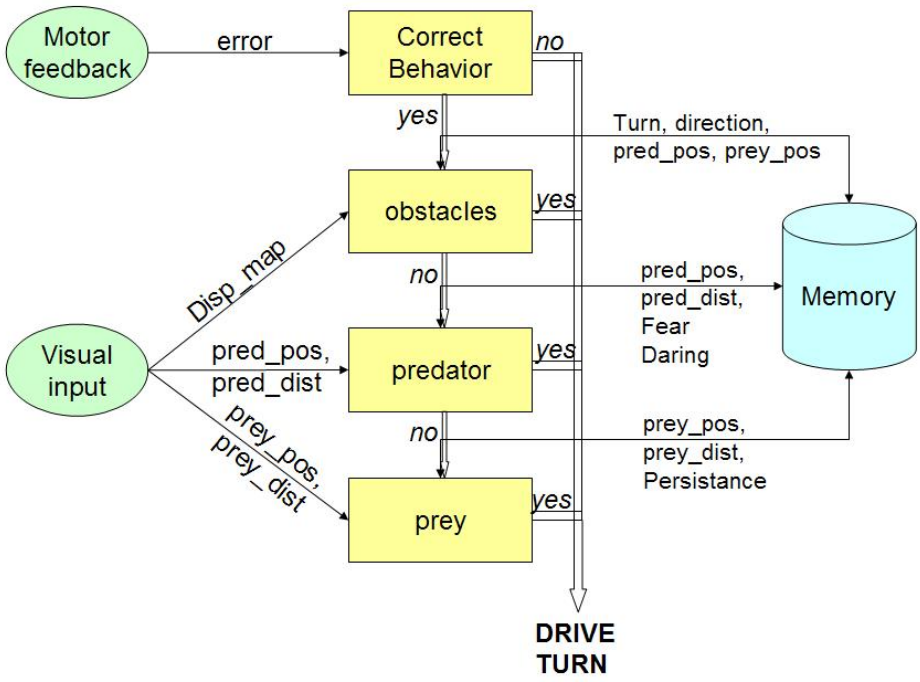


Figure 3.1: *At every time step action selection is performed according to this architecture. Outlined in yellow, the decision tree shows that maximal precedence is given to robot integrity, followed by obstacle avoidance, fleeing from predator and finally by prey hunting. A short term memory stores data about robot motion status (direction, turn), prey and predator position ($pred_{dist}$, $pred_{pos}$, $prey_{dist}$, $prey_{pos}$) and emotional state ($fear$, $persistance$, $daring$ see section 3.3). Action selection takes in account both the environmental inputs and the memory of the robot.*

Unlike the "basic" subsumpition architecture, our architecture also takes into account that the state of low priority action may still influence the output of an higher priority action. We consider this feature as desirable since it may allow our robot to produce more performing and realistic behaviors. For instance when an obstacle is detected while hunting a prey, the robot will try to avoid it in a direction favorable to its hunting (i.e. will try, if possible, to select the direction selected by the prey). Therefore, in this use case the hunting performance is increased.
The implementation of this feedback system enriches the robot performances by still preserving the independance of the different behaviors (in principle their addition and removal remains easy). This interpretation is related to cognitive scientist Marvin Minsky's idea that mental activity is divided in independant agents, expressing intelligent behaviors of varius complexity by communicating within each others (*Society of Mind* [21]).

To avoid continuous switch between an action and another, our action selection system is set so that a switch from an action to another is allowed only if a minimal time t_r has passed since the last action change. In the following of this text we will call it *Reaction Time* t_r.

3.2 Obstacle avoidance

3.2.1 Relation Between Robot's Gait and Drive Signal

Salamandra basic control through a drive signal has been presented in chapter 2.3. We observe that this controller does not include backward locomotion. This gait can be considered as a desirable feature to help *Salamandra* to avoid an obstacle who came too close to be avoided with a simple steering movement. This may happen in case of a moving object suddently appearing in front of the robot or in case the salamander enters in a dead-end.

Several studies about backward locomotion have been realized for both biped and quadrupeds. The conclusion is that the same spinal mechanism controls both locomotions. In mammals, all the small differences in body posture are believed to be produced by cortical system [29][17]. In the case of salamander it seems however that a slightly different motion pattern is used [22].

In our CPG the most effective way to produce a backward locomotion is to send a negative frequency value to the legs servos.

Since amplitude and frequency are related to a drive signal in a remote computer, to provide a negative frequency to the robot we have to extend the drive signal to negative values.
To increase backward speed we have then to increase legs and spine amplitude exactely as it is done in forward locomotion and use an increasingly negative value for legs and spine frequency.
We extend then the equation 2.4 describing the relation between drive, frequency and amplitude in the following way:

$$\begin{array}{ll} a_i = k_2 \cdot d & if\ d_{min} < d < d_{max}\ or\ -d_{max} < d < -d_{min} \\ f_i = k_1 \cdot abs(d) & if\ d_{min} < d < d_{max}\ or\ -d_{max} < d < -d_{min} \\ a_i = f_i = 0 & else \end{array} \tag{3.1}$$

Where d is the drive value, a_i the amplitude of the ith oscillator and f_i its frequency (see equation 2.3)
The behavior of oscillators with this extended signal is shown in figure 3.2.1: to obtain a forward locomotion we now just have to set a positive drive signal, whereas a negative drive signal produces a backward walk.
The resulting gait is not perfectly consistent with [22] because the phase relationships within the four legs are the same used for forward locomotion, but is still biologically plausible.

Transition from forward to backward locomotion is performed when an object is closer than a certain threshold t_1, the inverse transitions is done at a threshold $t_2 > t_1$. The use of two different thresholds is necessary to ensure that the robot does not dither between the two walking directions.
The drawback of the technique presented here is that an abrupt change in the drive sign produces an abrupt change in salamander movement as well. Since an abrupt change does not look natural, we program the drive signal so that it varies smoothly. We introduce then the following differential equation:

$$\dot{d} = q(D - d) \tag{3.2}$$

Where d is the current drive value, D the desired target value and q a convergence coefficient. Figure 3.2.2 shows the behavior of the drive signal using different values for q while switch from forward to backward walk.

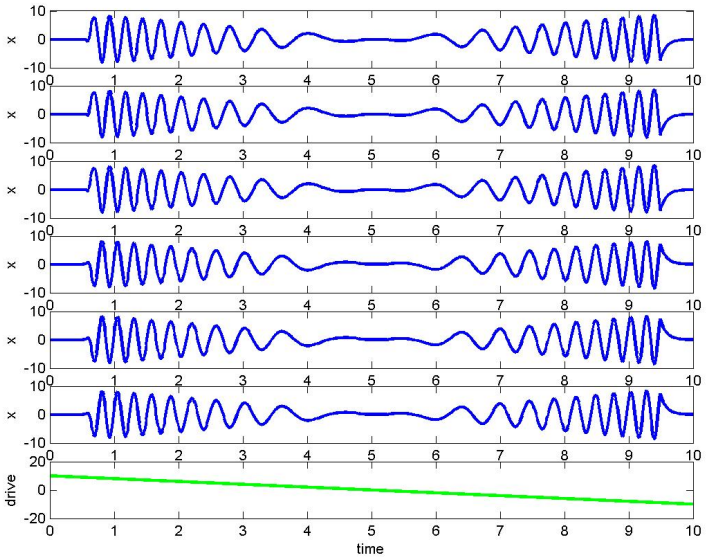


Figure 3.2: *Relation between the drive signal (in green) and robot's spine amplitude and frequency (blue) in walking gait. From 0 to 2 seconds frequency and amplitude are saturated (set to 0). Between 2 and 6 seconds the robot walks forward with decreasing speed, then completely stops and starts again with backward walking at 8 seconds. Backward walking speed increases until saturation at 12 seconds.*

3.2.2 Relation between Drive Signal and Obstacle Distance

The value of target drive D is affected by the distance of the obstacle. As said we want it to be negative when the obstacle is closer than t_1. Moreover, we want it to decrease when an obstacle approaches (at a distance bigger than t_1). Far obstacles should have little or no relevance, whereas the more the obstacle an gets close, the more it has to be considered as dangerous. We decide therefore to use the following sigmoid function to relate the robot speed with the obstacle distance:

$$D = \frac{2m}{1 + s^{t_1 - k}} - m; \tag{3.3}$$

where m is the maximal drive value we want to reach (may be interpreted as robot's *haste*), s the steepness of the sigmoid function (the bigger it gets, the more the sigmoid looks like a step function) and k the measured obstacle distance.

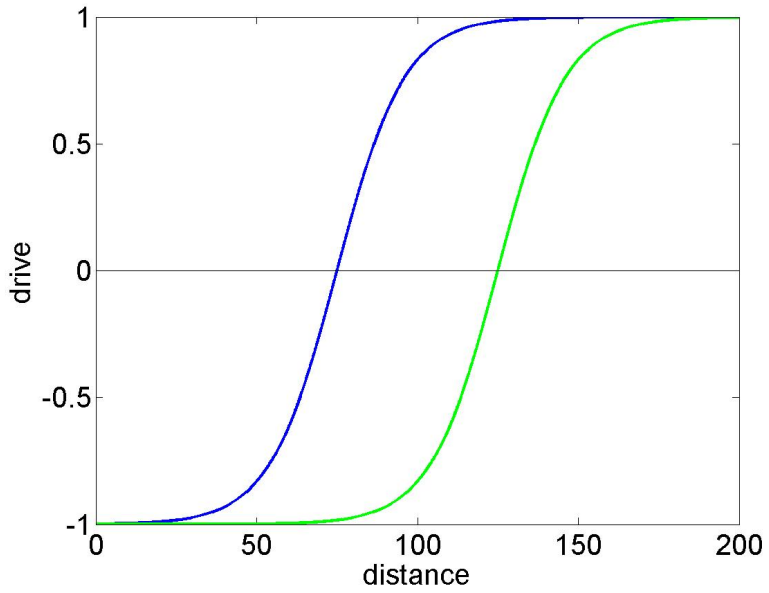


Figure 3.3: *Relation between obstacle distance and target drive signal D. D may vary between -1 (backward locomotion) and 1 forward locomotion, whereas obstacle distance is measured in centimeters. t_1 (equal to 75 cm in the graph) determines the interception point between the function and the x axis, and therefore the inversion from forward to backward locomotion, whereas t_2 (equal to 125 cm) determines the inversion from backward to forward locomotion. Steepness s is set in this graph to 1.1 for both courves to better show the shape of the function, higher values will be used in implementation.*

In conclusion, the parameters controlling the drive signal in our obstacle avoidance system are t_1 (inversion forward/backward), t_2 (inversion backward/forward), s (sigmoid steepness), m (max drive) and q (drive convergence). In chapter 5 their value will be defined.

Finally, we have to point out that the original controller for *Salamandra* presented in [14] produces a stepping locomotion when the drive signal is small, whereas over a certain threshold a swimming locomotion is obtained.
Since our system saturates the drive signal to the maximal walking speed achievable by the robot, one may argue that our new controller is not compatible with the original one. In other words, we add a feature by removing another one.
A simple solution to achieve swimming locomotion with our controller is to

add a humidity sensor on the robot itself: when this sensor is activated the drive signal is increased of a certain value higher that m, thus making it go over the swimming threshold. In this way, not only both locomotions can be achieved, but both can be saturated to a maximal speed.
We have to remark that with this technique it would be difficult to obtain a performing backward swimming. However this can not be considered as a real problem since real salamanders are just able to walk backward, but not to do that by swimming.
Being obstacle avoidance while swimming out of the scope of this project, the solution presented above has not been tested. We believe that this could be an interesting research topic for further studies.

3.2.3 The Obstacle Avoidance Controller

We divide the robot's behavior in two subsets corresponding to two distinct gaits: Forward Walk (FW) and Backward Walk (BW).
As input, robot's visual field is divided in sectors. For each sector an estimation of obstacle distance and information safety is provided (see Chapter 4.2 for further details).

In *Forward Walk* robot selects the most obstacle-free direction. Drive signal is defined according to the distance of the object detected in front of the robot. Turn signal is defined by selecting the most safe direction within the visual field sectors in terms of obstacle distance and information safety.
Having selected the desired sector, We can impose an offset to motor's oscillation corresponding to the designated sector's angle with $Turn = \frac{SectorAngle}{amplitude}$.

Backward Walk is needed to escape dead-ends and avoid moving obstacles suddenly appearing in front of the robot. In both cases, we want *Salamandra* to go backward until a good option for escaping these situations (using forward locomotion) is found.
Since we want *Salamandra* to scan its sides looking for a good escape option, we set its Turn signal to a constant value.
To define the input for the drive signal in the two escape conditions described above, we have to observe that:

- when avoiding a moving object the robot has just to establish a sufficient distance between it and the obstacle in order to be able to turn. In this case the drive signal can be easily controlled by the distance to the obstacle in front of the robot.

- In a dead-end the robot has to go back until an escape is found on its sides. Therefore in this case the drive signal can not just be controlled by the front distance, because this would produce an oscillating movement (backward/forward) in front of the dead-end. In this case controlling the Drive signal with the distance from the furtherst lateral obstacle seems more adapted: when enough space is detected on the sides the robot can stop and try to turn.

Those two behaviors can be summarized in the following expression:

$$drive\ \alpha\ min(x_{center}, max(X|x_{center})) \tag{3.4}$$

Where X represents the set of obtained distance measures, and x_{center} the obstacle distance in front of the robot.
Schema 3.2.3 sinthetises the ideas exposed above.

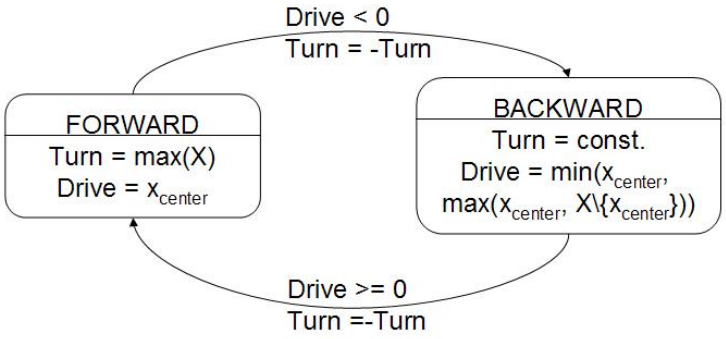


Figure 3.4: *The Obstacle Avoidance controller distinguishes two main gaits, forward and backward locomotion, controlled by different inputs. It has to be noticed that an active control of the Turn value is only possible when walking forward.*

If the robot is in FW and Turn value is equal to zero we can conclude that no obstacle is detected. The control can therefore be passed to the predator detector.

The obstacle avoidance behavior should be triggered once a detected obstacle is considered as "too close". We define therefore a threshold for obstacle detection T: when the obstacle distance measured in front of the robot becomes smaller than T, the obstacle avoidance behavior is triggered. The value of

T is adapted according to obstacle density. We decided to introduce this feature having observed that when the environment is almost obstacle-free one tends to initiate avoidance when an obstacle is still far away. Conversely, in a clutted environment one tends to approach an obstacle more in order to be able to move.
A dynamical T can be implemented by adding in our system a *decision buffer*, storing the decisions taken in the last decision steps. The following euristic has been used: if in the last 10 seconds more than $\frac{2}{3}$ of the decisions were about obstacle avoidance, T is decreased, whereas if it's less than $\frac{1}{3}$ t is increased. T may vary t_1 and t_2 (see chapter 3.2.1).

3.3 Predator Avoidance and Prey Chasing

Both Predator Avoidance and Prey Chasing behavior are inspired by Reynold's Steering Behavior [26]. In case a predator is detected, the robot turns the opposite way, performing an escape manouver. Drive is set to m, Turn is set to 1 if predator is detected on the left, whereas it is set to -1 if it's found on the right side.
When this is done the predator goes rapidly out of view, which obviously does not forcely imply that it's gone away. Since no more information about predator position can be obtained, the only solution is to force the fleeing behavior for a certain time. We will call *Fear* the time the robot continues escaping before feeling safe. We can refer to this feature as Object Permanence. Humans usually develop this feature around the 9th month.

When a prey is detected the robot turns in such a way that the prey is placed in front of it, and when it gets sufficiently close its speed scaled until a complete stop in front of the object is performed (Arrival Behavior).
Modulation of the Drive signal necessary to produce such behavior is performed according to the following equation:

$$\begin{array}{ll} drive = m & if\ distance > s_1 \\ drive = m \cdot \frac{distance}{s_1} & if\ distance < s_1\ and\ distance > s_2 \\ drive = 0 & if\ distance < s_2 \end{array} \tag{3.5}$$

where m is robot's maximal speed (same value as chapter 3.2.2), s_1 is the distance at which we want to initiate the decrease of robot's speed, s_2 is the distance from the prey at wich we want the robot to stop.
Turn value is modulated so that the prey is kept as much as possible centered in front of the robot: $Turn = \frac{SectorAngle}{amplitude}$, where Sector Angle is the angle

correponding to the sector where the prey center has been located.

If the prey is lost (fast prey, troubles in detection) the robot insists for a certain time with the *turn* and *drive* it was using the last time the prey was detected. We define this time as *perseverance*.
We should point out that a prey being too close to the robot may be considered as an obstacle by the Obstacle Avoidance behavior. This situation shows the importance of communication between behaviors. The prey position detected by the Prey Hunting system are sent to the Obstacle Avoidance behavior. When evaluating obstacle distance, the latter will skip the distance measure obtained by the sector containing the prey.

In a situation where both prey and predator are detected, the predator fleeing behavior should take control given its higher priority. In a case where prey is very close and predator very far one may however consider an advantage to "take a risk" and approach the prey instead of simply fleeing. Therefore we implement a feedback between the Prey Chasing and Predator Fleeing system, and define as *Daring* the minimal acceptable distance between a predator and a prey at which the robot preferes to ignore the danger.

Chapter 4

Visual System

A stereo system has been selected for providing input for action selection in our robots. By running a specific algorithm distance from obstacles can be estimated, and an appropriate decision can be taken (see chapter 3).

In this context, obstacle avoidance algorithms have been particularly studied. Algorithms such as [11] (implemented on robot *Speedy*) and [20] (implemented in Sony's humanoid robot Qrio) are based on plane extraction. The advantage of such algorithms is that they can easily determine both positive (walls) and negative (holes) obstacles. They however perform well if cameras can be placed ad a certaing height over the ground, and possibly not perfectly parallel to it. Given our robot's architecture, our cameras will be located extremely low, and parallel to the ground. For this reason we considered this approach not suited to our case.

Zanela and Taraglio propose an application of cellular neural networks (CNN) to stereo vision [4]. CNN are analog circuits featuring characteristics of both cellular automata and neural networks, capable of processing signals in real-time. Interestingly the proposed system is able to produce a sufficiently precise depth map even in case of discrete camera misaligments, contrast differences and noise addition [3]. System seems however extemely sensitive to light changes. Results have been presented as software simulations, though this approach is clearly harware-oriented.

Our obstacle avoidance system is inspired by the Borenstein's Vector Field Histogram [16][6] (VFH). This algorithm, based on previous work of Carnegie-Mellon University [12], has been tested on robots endowed with sonar sensors. Since this algorithm is adapted to interprete noisy sonar input, we consided

that it can also be adapted to stereo vision.
Perceived word is divided in a Histogram Grid, every cell containg certainity information about obstacle presence. Based on this data a polar histogram (array of cells representing obstacle distance in equally sized sectors of the visual field) is generated, and its information used to determine where to steer. Robot tries to always keep the maximal speed, unless an abstacle gets too close.

The main difference in our algorithm is that a polar information is directly generated. Also, information certainity value is produced in a radically different way. Finally, as shown in section 3.2, our decision process allows a smoother control of speed using a non-linear function defining repulsion from obstacles. This function also permits backward locomotion, which is not the case in VFH.

Compared to most of the existing robots, *Amphibot* and *Salamandra* present an additionnal difficulty: since locomotion is obtained through body oscillations, camera position respect to trajectory is not fixed. Moreover, due to possible body drifting and absence of an external supervisor, robot direction and speed can not be correctly estimated.
In the following sections we show how to provide our reactive decision making process with consistent input information. In particular, we will show how obstacle position can be detected and correctly mapped in the robot visual field. Finally a way to profit of stereo information to create an effective prey and predator tracking system will be presented.

4.1 Camera Field and Visual Field

4.1.1 Introduction

As said in chapter 3.1, Our control architecture do not rely on the creation of a precise map of the environment. Just a minimal representation of the world (basically just the a knowledge of the prey and predator characteristics is introduced) is needed: action selection relies mostly on a very short memory of perceived inputs.

Our task in to map the Camera Visual Field into the real Visual Field so that visual inputs are interprested correctly (see Figure 4.1.1).
First, we decompose both the Camera Field and the visual field is sectors of equal size. Visual Field data consists in a polar representation of the

environment in front of the robot, and is constantly updated by readings from Camera Field spanning over it.
Information safety is identified by adding an information age field *AgeMax* to the Visual Field. When a sector is not updated for a certain time, its value is declared as not valid.

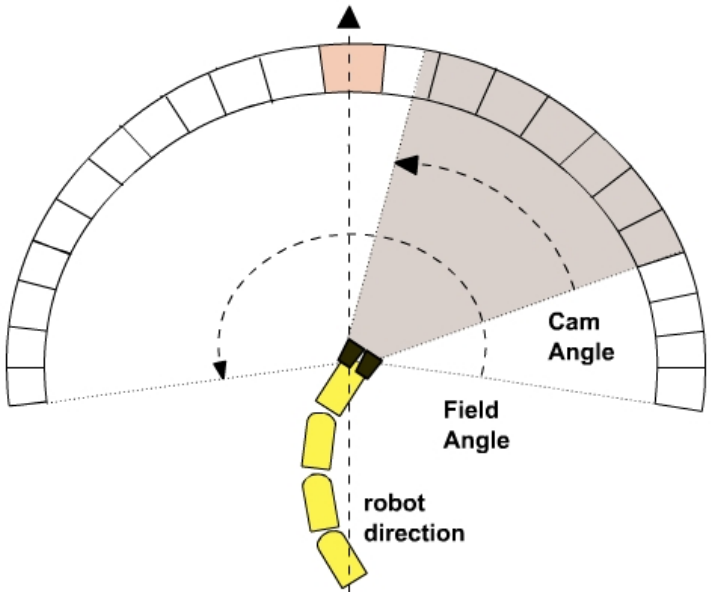


Figure 4.1: *Because of head movement, robot visual field is bigger than the field covered by its cameras. The cell highlighted in rose is the one corresponding to the robot trajectory.*

To map correctily cameras position in the real vision field we decompose the robot movement in two components: the sinusoidal oscillation of the body and the angular speed caused by a *Turn* value different from zero.

4.1.2 Head Angle and Angular Velocity Estimation

The effect of a *Turn* value different from zero is to introduce an angular velocity component on the robot. By fixing the referential on the robot, the effect of a turn in one direction is is to "rotate the world" of the same amout in the opposite direction. This corresponds to a shift in the array representing the sectors of the Visual Field.
As a result the cell containing data about ostacles located exactely in front of the robot is always in the same position in the array (rose cell in Figure 4.1.1).

Estimation of head angle is not trival: to do it we should know which part of its robot's spine is parallel to its trajectory. If the robot is producing a standing weave (such as *Salamandra* walking on the ground), such positions are located at 1/4 and 3/4 body length. Ideally we should sum the infinitesimal angles from head to 1/4 of body length to obtain a perfect evaluation, but since just two motors are available, we will take them as a good approximation of head angle.
For a robot producing a travelling weave (such as *Amphibot* and *Salamandra* while swimming) of exactely one body length, such position is variable, and located between the head and 1/4 of body length. For *Amphibot* this corresponds to one of the first four motors. The better approximation is obtained by summing the angles of all the motors until the following stopping condition is verified:

$$abs(x_{n-1}) - abs(x_n) < abs(x_{n-2}) - abs(x_{n-1}) \tag{4.1}$$

where x_n are the motor angles and $n = 1..4$.

Finally, we should consider that the measured motor positions include the effect of the Turn parameter (which imposes a certain offset to the motors). Since turning effect is already taken into account in the World Rotation phase, we have to substract it from the setpoint positions of all the motors we take into account. Head oscillation effect is therefore calculated according to the following equation:

$$HeadAngle = \sum_{i=0}^{n} x_i - n \cdot 2 \cdot Turn \cdot a \tag{4.2}$$

Where x_i is i-th motor angle, a is robot's amplitude and n respects the stopping condition above.

Field Angle size can be estimated by finding the maxima of Head Angle, and adding the Camera Angle to it.

$$FieldAngle = max(\sum_{i=0}^{3} x_i(t)) + CameraAngle \tag{4.3}$$

with $0 < t < \pi$ and $x_i(t)$ the i-th oscillator's value at time t.

A correct representation of the world is therefore obtained by shifting the array representing the Visual Field according to the Drive and Turn value (rotating the world), and then updating part of it according to Head Angle.

4.2 Obstacle Detection

Robot Stereo System hardware and Software for Measurement of Obstacle distance have been developed by Elia Palme [23].

Stereo Vision is based on the concept of *disparity*: an object filmed by two different point of views will be located in two slightly different positions on the x-axis of the two obtained images (see Figure 4.2). Disparity provides and evaluation of object distance, since the more its value is high, the more the object is close to the stereo system and viceversa.
Our stereo system is based on two *Firefly* cameras, using lenses with an opening of about 60° (that's the value we will use for the variable *Camera Field* presented in previous chapter).

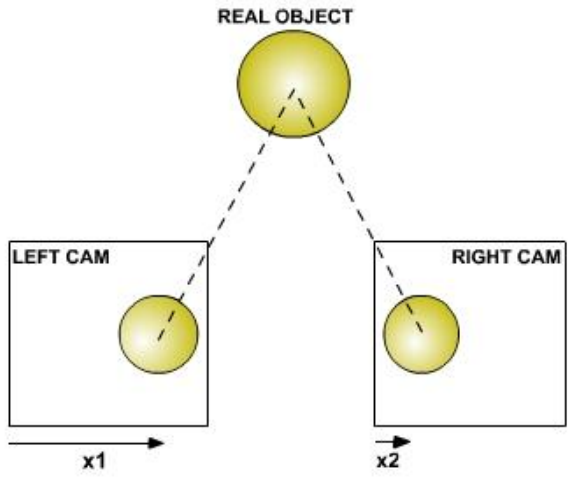


Figure 4.2: *Stereo Vision allows to determine the distance of a specific object by evaluating its disparity on the cameras. Disparity $x_1 - x_2$ is the position difference of the same object on the x-axis of the two input image. The higher the disparity, the closer the object. For further insides about the stereo vision system of our robots, see [23].*

To determine its value one has however to be able to define a correspondence between the pixels on the two images.
This problem is not trivial, and goes through several steps. First both input images are undistorted (to remove fish-eye effects) and rectified (to align scan lines).

Assuming that cameras are perfectly aligned, one should be able to find the correspondence to a pixel on a specific line in an image on the same line in the other image.
Unfortunately the only criteria for finding a possible match is a similarity in pixel intensity. Since usually more than one pixel may match a certain intensity, a technique based on sliding windows is applied. A fixed window size has been set, and thanks to a correlation function it is possible to compute the correspondence between two windows in the two images.

Algorithm outputs a distance map obtained by converting disparity values in centimeters. Input is divided in a nxm grid, and for cell a distance estimation is provided.

4.3 Prey and Predator Detection

Prey and Predator Detection on *Amphibot II* was already explored by Benoit Rat [25], where tracking was based on color detection. Robot was able to track and follow a rose sphere. Our project can not unfortunately reproduce the same results, since colors are poorly reproduced by our cameras. We decided therefore to track shape instead of color.
Prey an Predator will be represented with two spheres of different sizes, and tracked using information provided by a *Circular Hough Transform* run on a stereo input [28]

Circular Hough Transform is an algorithm concieved to identify circles into an grayscale image. To do this, pixels being part of an object edge have first to be identified by running a Canny edge detector [8]. Edges are recognisable since their intensity gradient is high, therefore by fixing an appropriate threshold is possible to identify most of them. Canny edge detector sets all edge pixels to white, whereas the others are set to black.

Afterwards, white pixels udergo the *voting* phase into a parameter space. For circles, parameters space is three-dimentionnal, since circle equation is:

$$(x - x_i)^2 + (y - y_i)^2 = r_i^2 \tag{4.4}$$

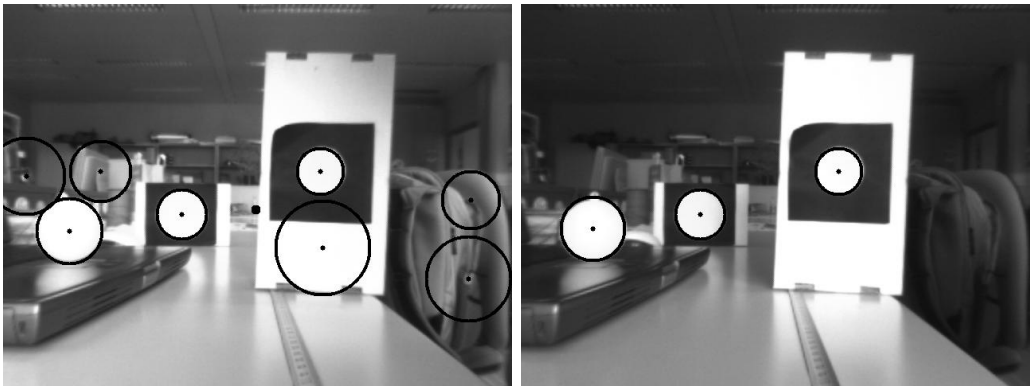
Where (x_i, y_i) is the coordinate of the circle's center, and r_i its radius.
Every white pixel gives a vote (increase a counter) to all coordinate in the paramenter space representing a circle it may belong to. Points in the parameter space receiving a high quantity of votes are more likely to represents

the (x_i, y_i, r_i) values of a circle into the image. Thus another threshold is necessary to define the minimal quantity of votes needed to consider a given coordinate in the paramenter space as a circle.
The output of Circular Hough Transform is a list of the parameters of all the circles found into the image.
We have to point out that this techninque is applicable to every sort of shape one can parametrize. Circles have been selected over other shapes since just three parameter are sufficient to describe a circle, whereas more complicated shapes would need a higher number of parameters, thus making the voting phase more computationnally expensive.
Circular Hough Transform is run on both camera inputs. Since aliases may be detected, a left-right check is performed. Circle found on both cameras are coupled if their measured radius and vertical position (y-axis) are similar. As Figure 4.2 shows, this removes most of the aliases.

The disparity (difference of position on the x-axis) of the position of the circle center in the two images gives us an indication of its distance from the cameras. Knowing the relation between circle distance and its apparent radius, it's therefore possible to recognize a circle of a specific size.
To accept a determined circle as being a prey or predator, it's sufficient to compare the expected circle radius according to disparity value with the radius found through the Hough Transform.
This supposes that a knowledge about the designated targed is introduced in the system. In section 5.2 measures relating object distance with apparent radius for three targets of different size are shown.

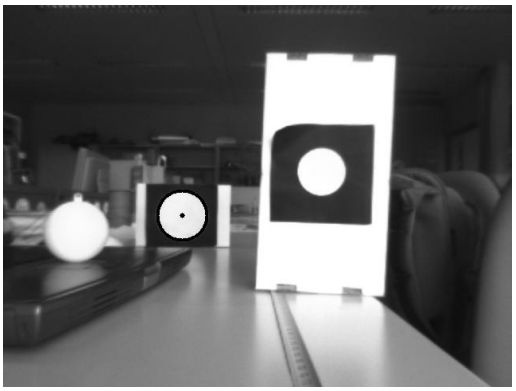
Further optimisations may include a color check in the neighborhood of the center of a circle identified as a target by the algorithm presented above. Color reproduction of our cameras is indeed poor and can not be used as main feature for prey selection, but may be used for confirming a selection, using for instance color uniformity information. Also, it may be possible to distinguish two speres of same size according to their color, provided that their respective intensity is extremely different (e.g. black and white).

Once a prey or predator are recognised, estimated distance is stored and their x position relative to camera is converted in absolute position (mapping in robot's Visual Field) in the same way obstacles are mapped.



(a) All circles found by the Circular Hough Transform. Aliases are also detected.

(b) Circles detected after left-right size and vertical position check. Aliases are removed.



(c) Prey detected after consistency check with data about prey's disparity-radius relation.

Figure 4.3: *Phases of the prey and predator tracking algorithm. Being insensitive to perspective effects, the algorithm is able to detect the target having the correct size.*

Chapter 5

Implementation

Initially our algorithm has been implemented on a virtual robot in a 3D environment using the *Webots* [31] software. The use of such an environment has been precious to observe the behavior of the robot in an ideal condition (i.e. in an environment where obstacles, predators and preys can be easily detected).

Afterwards, implementation on the real robot has been tested using the robot *Amphibot II*. The choice of this robot for a first implementation, though forced since *Salamandra* was broken, has been precious since in allowed us to test our algorithm on a robot presenting less computer-vision-related problems (mainly, the robot's cameras do not oscillate sidewise). Implementation on *Salamandra* was also performed, but tests were mainly performed on *Amphibot*.
In collaboration with Elia Palme a unique software endowed with a graphic interface able to control both robots has been developed.

Tests on real robots were performed in a uniform closed 2.5 x 4 meters arena. A 1.7 meters turning arm has been placed in the middle of the arena's long side, and is used to suspend all the cables and the trigger over the robot. Since friction during arm rotation is extremely low, the robot was able to drag it by itself aroud the arena, minimizing human intervention.
Finally, since the vision algorithm recognises obstacle distance only if the obstacle itself is covered by a texture, we covered arena's walls with black and white wood pictures.

Figure 5.1: *The testing Arena.*

5.1 3D Simulation

5.1.1 Environment

The controller presented in chapter 3 has been implemented on a Webots 3D model of *Salamandra* adapted from a previous version developed by Yvan Burquin.
This model respects the constraints of the real robot in term of servo movement, and allows the user to exploit the input provided by two virtual cameras having an opening of 60° each.
Since the objective of this project is to study the robot's behavior and not to develop its stereo vision algorithm, we tailored a virtual environment in which obstacles, predators and preys could be easily detected (see Figure 5.1.1).
Ground and sky have been set to a white color whereas obstacles, represented by several cubes of identical sizes, have been defined as black. In this way the obstacle distance at a certain turning angle could be easily estimated by calculating the average intensity of pixel in a determined sector of the image produces by the merging of the inputs of the two virtual cameras.
A predator and a prey randomy moving in the environment have also been introduced. Predator was represented by a red spot, whereas a green spot represented a prey. Detection of those objects could be obtained by simply

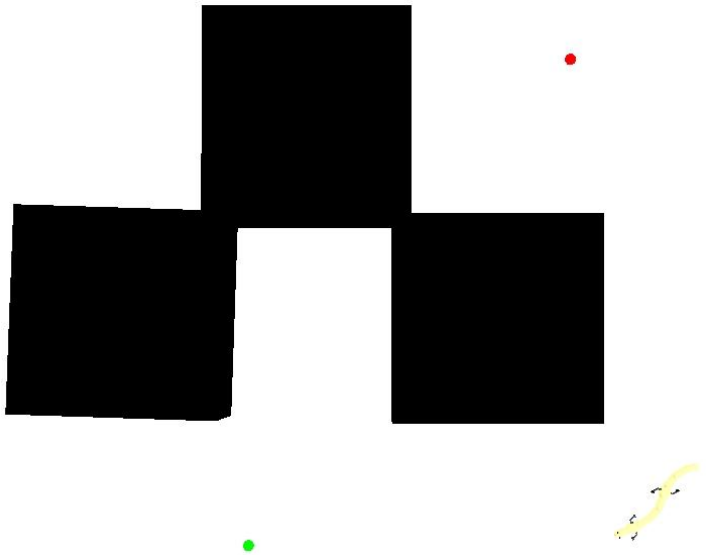


Figure 5.2: *Screenshot of the Webots environment. In black an obtacle representing a dead-end. The green spot represents a prey, the red spot a predator.*

looking for red or green pixels in the input image.

Through this environment an potential problem of our architecture could be avoided: if a prey comes too close to *Salamandra*, it should be considered as an obstacle, thus triggering the action.

5.1.2 Robot Locomotion

Relation between Drive signal, amplitude and frequency was defined according to [14].
According to the following biological observations, head motion was however modified compared to the original model.
In general, animals always turn the the head in the direction of the most interesting object, before the rest of the body eventually realignes itself to it. Experiments on humans and rats [24] [27] [13] have pointed out that a turning movement is actually always preceded by head rotation. This seem to be a general feature for all animals.
We therefore try to stabilize the head so that it looks always in the desired direction. To counterbalance the oscillations of a body producing a standing wave we must take into account that the head angle is shifted by the combined action of all motors between the head and the half length of the robot

(true only if a phase of 2π is defined between head and tail). The head will have to move in the opposed way with an angle corresponding to the sum of all motor positions. With this solution the robot head moves laterally, but always looks forward.

To obtain a head reacting faster than the body during turn initiation, we increase the convergence speed a_0 of the oscillators controlling the head. This ensures that when a turn is imposed to the salamander, the head will be the first to converge to the new desired axis of oscillation.
Empyrical observations lead us to the conclusion that $a_0 = 4 * a_{1..6}$ produces the most realistic behavior.

5.1.3 Robot Behavior

Our simulated robot's position has been tracked in several simple situations of obstacle avoidance, prey chasing and predator fleeing in order to show the effect and relevance of the parameters controlling it.
Figure 5.1.3 shows robot's behavior in a situation where both a prey and a predator are detected, with different values of Fear and Daring (see section 3.3). Behavior may vary from completely ignoring the predator to fleeing from it as fast as possible.
Interstingly a value of Fear too high may lead to a panicking-like behavior: once a predator is detected a sharp turn is performed, but since Fear is high this turning status may continue until about a 360° turn is completed. At that moment predator is detected again, causing the Predator Fleeing behavior to be triggerend again.

Figure 5.1.3 show how Obstacle Avoidance behavior is affected by the value of Reactivity (i.e. the minimal time between the selection of two different behaviors).
If reactivity is set to zero robot reacts instantaneously to every sort of input. While this may be considered as positive in case of sudden obstacles appearing in front of the robot, this causes the robot to move in a very irregular way.
We can see that low values of Reactivity still allow obstacle avoidance, but behavior does not look natural since the Obstacle Avoidance controller is continuosly triggered. This phenomena is due to robot's body oscillations: when cameras are pointing forward obstacle is detected and a turning manouver is launched, but as soon that cameras point sidewise, free space is detected and Obstacle Avoidance behavior is turned off.

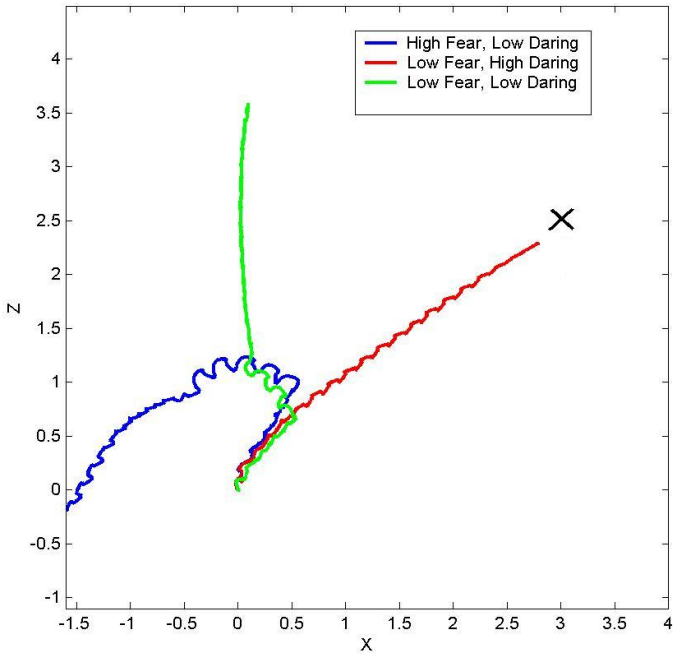
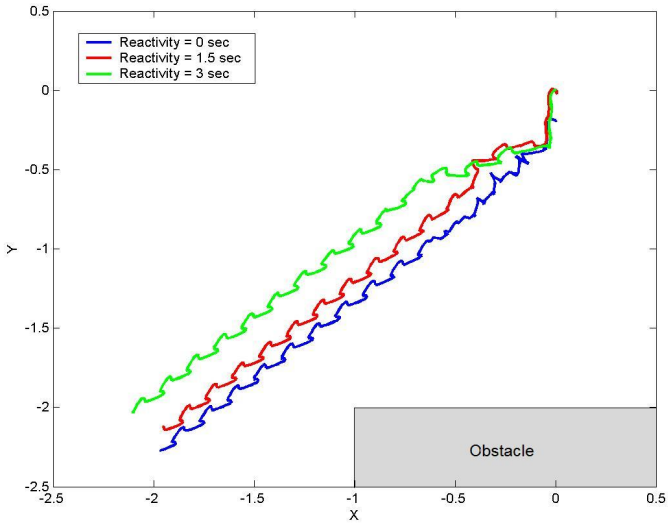
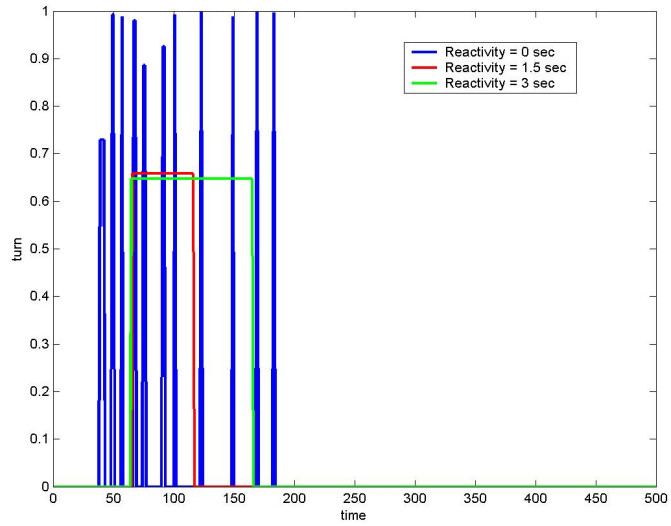


Figure 5.3: *Top view of trajectories performed by the our simulated salamandra (referential fixed on its center of gravity). In position* $(3, 2.5)$ *a prey is located (cross symbol), whereas in position* $(6, 2.5)$ *(out of the plot) a predator has been placed. Both entities are motionless. With high Daring (in red) the robot decides to ignore the predator and approaches the prey. With low Daring and low Fear (in green), prey is ignored and a discrete predator avoidance manouver is performed. Finally, with low Daring and high Fear (in blue) a complete fleeing manouver is performed as soon as predator is detected.*

Conversely, if reactivity is set to a too high value, sudden obstacles can't be detected on time. Also, the robot tends to turn away excessively from static obstacles as the one presented in Figure 5.1.3.



(a) Top view of robot's trajectory



(b) selected Turn value along time

Figure 5.4: *Obstacle avoidance behavior using different values for Reactivity (0, 1.5 and 3 seconds). Without reactivity (set to 0 seconds) the robot is still able to avoid the obstacle, but as subfigure (b) shows, the obstacle avoidance behavior is switched on and off continuously, producing an hesitating and unnatural gait. With values bigger than zero turning behavior is more constant. The more Reactivity gets big, the more the robot will go far from obstacle.*

5.2 Prey Tracker Calibration

We selected as prey a 6 cm sphere, painted with a fluorecent yellow paint. To evaluate the relation of its size with the disparity and apparent radius obtained through the Circular Hough Transform (see chapter ref), we measured those values between 20 and 110 cm in from of the stereo system.
In figure 5.2 we show the results for our prey (in green) and for two other targets, one bigger (10 cm, in red) and one smaller (4.5 cm, in blue).
It turns out that, in an ideal case, our system is supposed to be able to recognise two different balls having a difference of 1.5 cm in size at one meter of distance.

Measure for our designated target are hard-coded in the robot controller. Figure 5.2 shows us how the difference between apparent radiuses becomes significant, the more the target gets close to the cameras.
This difference will allow the robot to distinguish the prey within other circles in other positions, according to its size.

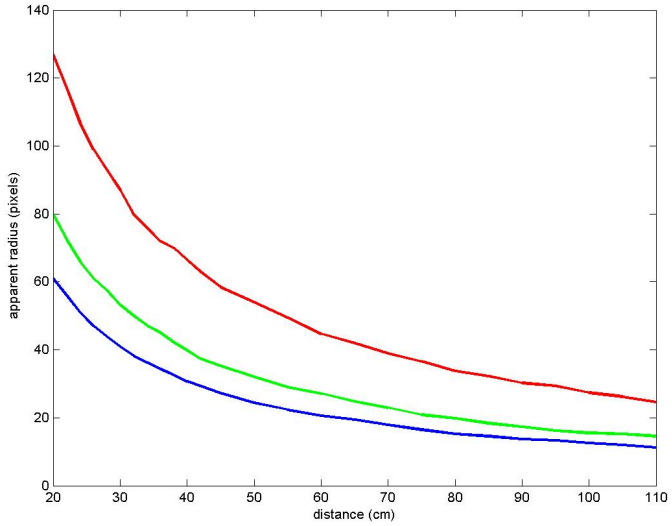
Arrival behavior parameter (see chapter 3.3) s_1 is set to the maximal distance the prey is detetected. Depending to illumination condition this distance varies between about 80 and 110 centimenters for the smallest target (the biggest one can be detected until abot 3 meters), we set therefore $s_1 = 80$. s_2 is set to the closest distance the Tracker is able to detect the prey. Once again, depending on illumination, this distance may vary between 10 and 20 cm (for all targets). We set therefore $s_2 = 20$.

5.3 Amphibot II

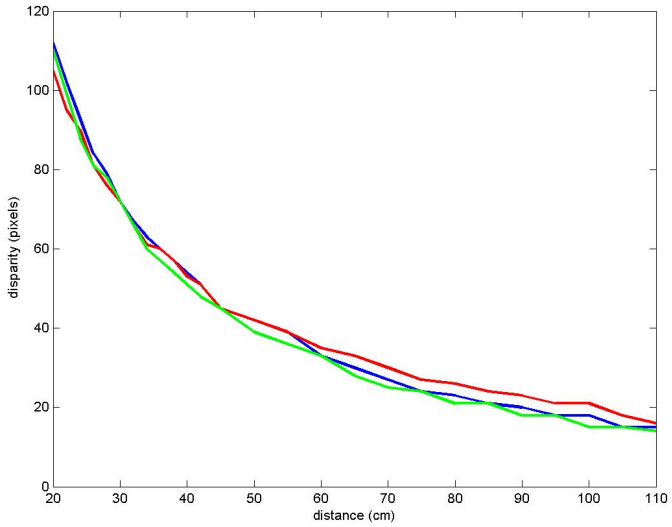
5.3.1 Implementation

Amphibot II is a snake robot realized with 8 body elements mounted on two passive wheels each, and is concieved to move producing a traveling wave along its body (lamprey-like locomotion).

Our control algorithm has been implemented on a computer communicating with the robot trough a wireless signal. The only values which can be provided to the robot are amplitude and frequency of oscillation, turning radius (those values are identical for all body elements), oscillation gain and number of travelling waves per body length.
We decided to fix the values of the last two parameters. Gain is a parameter



(a) Target Distance vs. Disparity



(b) Target Distance vs. Target apparent Radius

Figure 5.5: *In Blue the measures performed with a 4.5 cm ball, in red measures with a 10 cm one. Being disparity measure (obviously) about the same for the three targets, we should have measured it just once. We decided to measure it for every object anyway to detect eventual systematic error.*

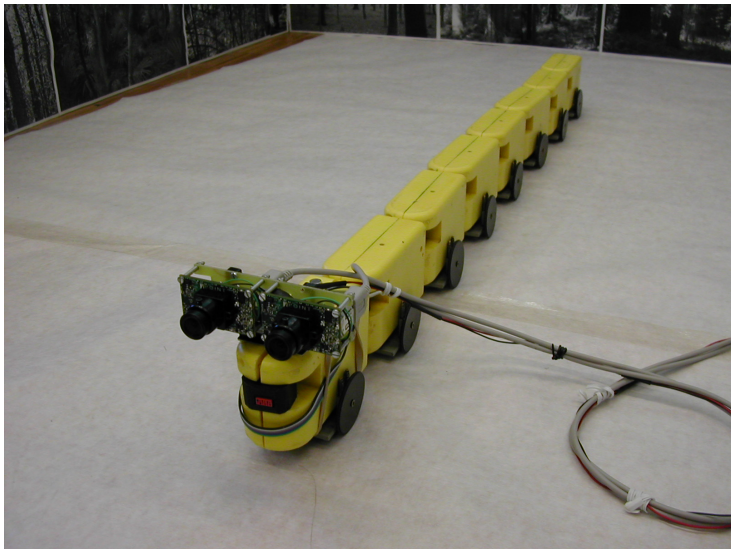


Figure 5.6: Amphibot II *equipped with the stereo system.*

relevant during serpentine swimming (higher speeds can be reached by amplifying the oscillation of the robot's tail) but on terrestrial locomotion has almost no influence. Therefore we set the Gain to 1 for all body elements. Amplitude, Frequency, Gain (related to the Drive signal) and Turn are parameters were defined by our controller.

To increase the quality of captured images we decided to reduce the amplitude of oscillation of the first element, reducing therefore the motion blur effect. To do this we multiplied the calculated values for this block by a gain of 0.5. This reduces the amplitude by half, therefore reducing motion blur and producing better representation of the represent more precisely the enviroment present in the front of the robot. This solution becomes precious to increase image analysis performance by adding redundancy.

In preliminary tests it turned out that an optimal relation between amplitude and frequency through a drive signal was essential to allow *Amphibot* to produce a smooth serpentine locomotion. In fact, it turned out that a "bad" Drive tended to produce lateral drifting in robot's tail. We performed therefore a set of systematic tests. Setup and results are presented in chapter **??**.

5.3.2 Drive Signal and Visual Field Size

To assess the performance of the robot give different values of frequency and amplitude we performed a systematic test. Such a test was already performed by Alessandro Crespi [9]. The need to perform it is due to the fact that robot's wheels have been changed, thus making its friction with the lab's soil different and thus its original drive signal no more optimal.

We made the robot move on a straight line during 5 seconds and measured the covered distance by varying its amplitude and frequency. Each measure was obtained by averaging three indipendent measures. We decided to vary amplitude value between 20 and 60 degrees with a step of 5 degrees. This range was selected after observing that with angles smaller than 20 degrees the robot never moves, whereas angles above 60 may compromise motor's integrity. Frequency was varied between 0 and 1.2 Hz with a step of 0.2 Hz. Once again, the upper bound was fixed after observing that higher frequencies may be dangerous for *Amphibot*'s motors.

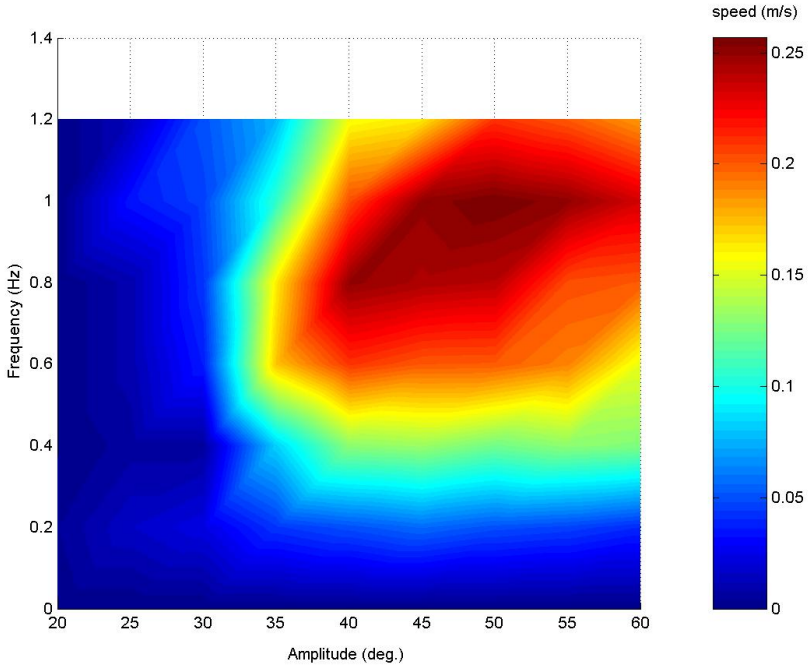


Figure 5.7: *Robot's speed in terms of amplitude and frequency, with phase and gain fixed to* 1. *A total of 63 measures have been performed (9 amplitudes, 7 frequencies).*

We have to point out that the space covered in the three measures performed for each set of parameters never differed of more than 0.05 m. We can conclude that robot's locomotion is extremely robust.
According to the measured data the following relation between drive signal and amplitude and frequency has been extracted:

$$\begin{array}{ll} a = abs(30 \cdot drive + 20) & if\ 0.1 < d < 1\ or\ -1 < d < -0.1 \\ f = d & if\ 0.1 < d < 1\ or\ -1 < d < -0.1 \\ a = f = 0 & else \end{array} \tag{5.1}$$

Drive d varies between -1 and 1, representing the maximal achievable speeds in forward and backward locomotion respectively.
Amplitude a and frequency f are forced to 0 for a drive between -0.1 and 0.1 since during measures we observed that, in any case, the robot is unable to move with such a small drive. The obtained locomotion is very effective, though small drifting on the tail elements is still noticeable when *Amphibot* performs sharp turns at low speeds.

It turns out that the maximal amplitude achievable by the robot is 60 degrees. Knowing that used cameras have an opening (Camera Field) of 60 degrees, using equation 4.3 we obtain Visual Field= 216.78.

5.3.3 Angular Speed estimation

To assess the angular speed of the robot we performed some systematic test by varying the Drive and Turn signals. Unfortunately it turned out that angular speed is greatly affected by the charge status of the batteries connected to each motor.

Figure 5.3.3 shows this effect. We loaded completely *Amphibot*'s batteries, then let it move with Drive = 0.6 and $Turn = 0.5$ and assessed its angular speed approximatively every 2 minutes. Angular speed assessment was performed by measuring the time needed to complete a loop. Values obtained are shown in figure 5.3.3.

Since battery charge directely affects the motor torque, we observe that a drifting on tail elements emerges after about 18 seconds of running time. This means that after about 18 minutes the values selected for the Drive signal are no more optimal.
Given the importance of battery charge on robot performance, we should

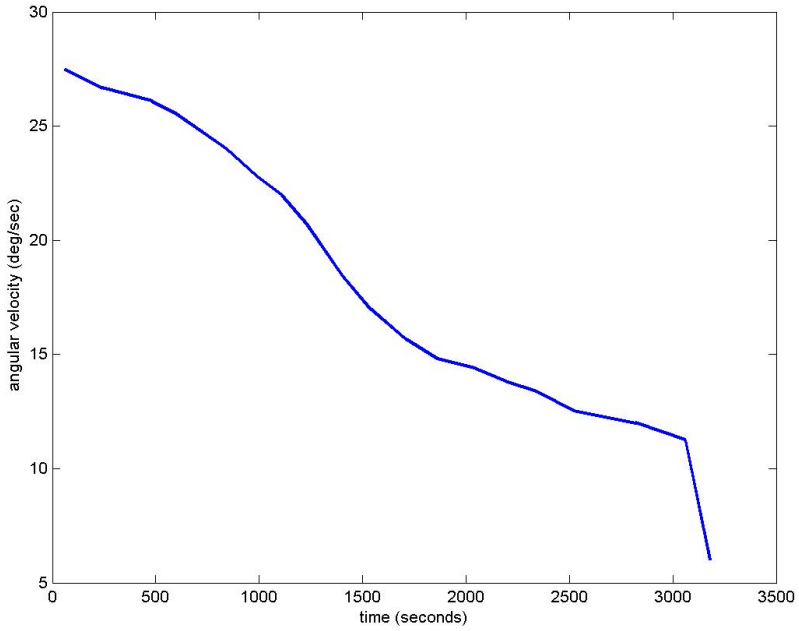


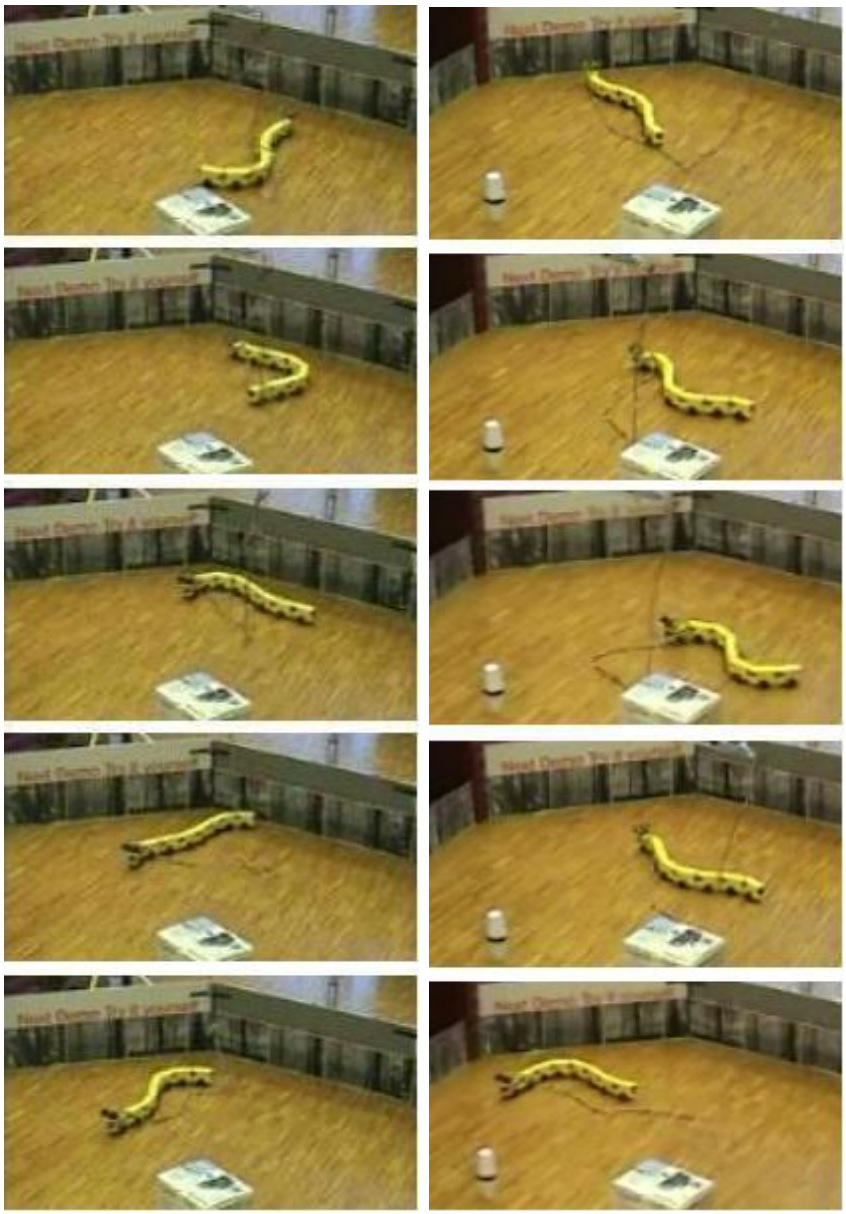
Figure 5.8: Amphibot*'s angular speed with drive = 0.6 and turn = 0.5. Battery charge status clearly influences motor performance. After about 50 minutes of uninterrupted work performance suddently decays. Just one measure after this time exists (at 3180 seconds), since afterwards robot was just unable to turn in the designates space.*

include its value into the parameters controlling the drive signal. Unfortunately at the moment there is no way to evaluate the *Amphibot* batteries's charge status, thus a correct assessment of angular speed is not possible.

The second phase of the system mapping Camera and Visual Field (world rotation) can not therefore be entirely implemented. The first phase is however sufficient to produce a consistent field of view, since its values are constantly updated by head oscillation, and old values are removed.
For all our experiments we will assume that the relation between drive signal, amplitude and frequency is based on a set of measures taken with sufficiently charged batteries.

5.3.4 Obstacle avoidance

As said in chapter 4.2, the stereo vision algorithm is able to detect obstacle at a maximum of 1.5 meters. This represents therefore an upper bound for obstacle detection T. From [15] we know moreover that the minimal turning radius for *Amphibot* is 0.25 meters, which represents a lower bound for obstacle detection. Obstacles closer than this distance should be avoided using backward locomotion.
Finally we know from chapter that the dynamic threshold for obstacle avoidance T should be smaller than t_1 (switch from forward to backward locomotion) and bigger than t_2 (switch from backward to forward locomotion). Therefore, the values selected for the obstacle avoidance behavior have been $t_1 = 0.3$, $t_2 = 1.25$ and $0.5 < T < 1$.
Figure 5.3.4 shows two obstacle avoidance situations.



(a) Avoding a wall

(b) Avoidance using backward locomotion

Figure 5.9: *From top to bottom,* Amphibot II *avoiding obstacles. In figure (a) simple wall avoidance is shown (one image per second). Image (b) shows a special case in wich the robot is too close to an obstacle to avoid it (distance smaller than threshold* t_1*, see chapter 3.2.1). Backward locomotion is performend until an escape is detected (one image every two seconds).*

Chapter 6

Conclusion

We have shown how a simple control system based on a subsumption architecture is able to control in real-time a robot using visual inputs.

In particular we have tackeled the problem of how to perform obstacle avoidance in an upredictable environment through minimal path planning provided possibly noisy stereo information (chapter 3).

What distinguishes *Amphibot II* and *Salamandra Robotica* from most of the other existing robots is that their locomotion is based on body oscillation.
We overcame the problem of correctly mapping the stereo vision input into the real field of view by computing an approximation of cameras' angle. Further improvements in the hardware may improve the quality of our algorithm, in particular allowing the implementation of the world rotation phase proposed in section 4.1.2.

Concerning the prey and predator detection, we proposed a fast and effective technique to track different types of circular targets and distinguish them according to their size and color. Performed tests were limited to size distinction, we believe that the addition of a color distinction phase may improve the quality of the algorithm.

Finally, as already shown in previous works, we have confirmed that an optimal relation between amplitude and frequency of oscillation in *Amphibot* is essential to produce an effective locomotion (section 5.3.2).
Furthermore we have proved how the robot performance may be largely affected by its battery charge (section 5.3.3). Scanning battery charge status is at the moment non possible on the robots. Adding this feature may lead

to an improvement in their locomotion. An even better solution would be however to add a step up to motor controller, so that performance of the latter could become independent to battery charge.

Implementation part of this project revealed to be particularly tricky due to several physical constraints represented by the robot and by the neecessity of sharing resources with other studends. On this point, I would like to thank again Elia Palme for sharing his stereo system with me.

Documentation and movies about this project can be found in [1].

Bibliography

[1] http://birg.epfl.ch/page63116.html.

[2] K. Ali and R. Arkin. Implementing schema-theoretic models of animal behavior in robotic systems, 1998.

[3] Sergio Taraglio Andrea Zanela. A robustness study of a cnn based stereo vision algorithm. *Fifth IEEE International Workshop on Cellular Neural Networks and their Applications*, pages 145–168, april 1998.

[4] Sergio Taraglio Andrea Zanela. Sensing the third dimention in stereo vision systems: a cellular neural networks approach. *Engineering Applications of Artificial Intelligence*, (11), 1998.

[5] Ronald C. Arkin. Motor schema-based mobile robot navigation. *Proceedings of the IEEE International Conference on Robotics and Automation*, April 1987.

[6] J. Borenstein and Y. Koren. The vector field histogram - fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3):278–288, 1991.

[7] Rodney A. Brooks. How to build complete creatures rather than isolated cognitive simulators. *Architectures for Intelligence*, 1991.

[8] John F. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(1), november 1986.

[9] A. Crespi and A.J. Ijspeert. 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)*, pages 19–27, 2006.

[10] Alessandro Crespi. Design and control of amphibious robots with multiple degrees of freedom. *EPFL, Phd Thesis*, 2007.

[11] Gregory Hager Darius Burschka, Stephen Lee. Stereo-based obstacle avoidance in indoor environments with active sensor re-calibration. may 2002.

[12] A. Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, RA-3(3), 1987.

[13] Giuseppe Amato Giuseppe Crescimanno, Adriana Emmi. Effects of intraaccumbens microinjections of quinpirole on head turning and circling movement in the rat. *Pharmacology Biochemistry and Behavior*, 60(4), 1998.

[14] A. Ijspeert, A. Crespi, D. Ryczko, and J.M. Cabelguen. From swimming to walking with a salamander robot driven by a spinal cord model. *Science*, 315(5817):1416–1420, 2007.

[15] A.J. Ijspeert and A. Crespi. Online trajectory generation in an amphibious snake robot using a lamprey-like central pattern generator model. In *Proceedings of the 2007 IEEE International Conference on Robotics and Automation (ICRA 2007)*, pages 262–268, 2007.

[16] Y. Koren J. Borenstein. Real-time obstacle avoidance for fast mobile robots in cluttered environments. may 1990.

[17] Jennifer A. Cook Joel A. Vilensky. Do quadrupeds require a change in trunk posture to walk backward? *Journal of Biomechanics*, (33), 2000.

[18] Allen Newell John Laird and Paul Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, (33), 1987.

[19] O. Kathib. Real-time obstacle avoidance for manipulators and mobile robots. *IEEE International Conference on Robotics and Automation*, pages 145–168, may 1985.

[20] Jens-Steffen Gutmann Takeshi Ohashi Kenta Kawamoto Takayuki Yoshigahara Kohtaro Sabe, Masaki Fukuchi. Obstacle avoidance and path planning for humanoid robots using stereo vision. april 2004.

[21] Marvin L. Minsky. *Society of Mind*. Simon & Schuster, March 1986.

[22] George V. Lauder Miriam A. Ashley-Ross. Motor patterns and kinematics during backward walking in the pacific giant salamander: Evidence for novel motor output. (78), 1997.

[23] Elia Palme. Stereo vision library for obstacle avoidance applications. *University of Fribourg, Master Project*, 2007.

[24] Diega Russo Giuseppe Crescimanno Arcangelo Benigno Pier Vincenzo Piazza, Michele Ferdico and Giuseppe Amato. Circling behavior: ethological analysis and functional considerations. (31), 1989.

[25] Benoit Rat. Adding vision to a salamander/snake-robot. *EPFL, Semester Project*, 2007.

[26] Craig W. Reynolds. Steering behaviors for autonomous characters. *Games Developers Conference*, 2002.

[27] Brian L. Day Richard C. Fitzpatrick, Jane E. Butler. Resolving head rotation for human bipedalism. *Current Biology*, (16), August 2006.

[28] Peter E. Hart Richard O. Duda. Use of the hough transformation to detect lines and courves in pictures. *Comm.ACM*, 15(1), january 1972.

[29] Jean M. McCrory Erez Morag Robert W.M. van Deursen, Timothy W. Flynn. Does a single control mechanism exist for both forward and backward walking? *Gait & Posture*, (7), 1998.

[30] P. Chalasani J. Cheng O. Etzioni M. Ringuette T.M. Mitchell, J. Allen and J.C. Schlimmer. Theo: A framework for self-improving systems. *Architectures for Intelligence*, 1991.

[31] Webots. http://www.cyberbotics.com. Commercial Mobile Robot Simulation Software.

[32] J. Borenstein Y. Koren. Potential field methods and their inherent limitations for mobile robot navigation. *Proceedings of the IEEE Conference on Robotics and Automation*, pages 1398–1404, April 1991.

Appendix A

Amphibot II used Parameters

name	*value*	*description*
t_1	0.3 m	inversion forward/backward
t_2	1.25 m	inversion backward/forward
T	$[0.5, 1]$	Threshold for obstacle avoidance (*caution*)
q	1	drive convergence
s	e	sigmoid steepness
m	0.75	max drive value (*haste*)
t_r	1 s	*reactivity*
persistence	3	time before considering a prey as lost (in seconds)
fear	3	time before considering a predator sufficiently far (in seconds)
daring	1	min distance from a predator when a prey is also detected
n x m	$5x5$	Disparity map grid size
maxAge	3	time before considering a distance measure too old (in seconds)
s_1	0.8 m	distance from prey where deceleration is initiates
s_2	0.2 m	halting distance inf front of a prey

Appendix B

Data CD Organisation

`Controller` : C++ source codes. Most recent controller is contained in `Robot Controller`. `CircularHoughTest` contains a demonstration of the Prey tracker. `snakeOptimisation` and `salamanderOptimisation` contains controllers for *Amphibot* and *Salamandra* respectively.

`Data` : performed measures in .csv format. Can be provided to Matlab for visual output.

`Matlab` : Scripts used to produce the images presented in the report or to demonstrate some features of our controlelr

`Movies` : videos about Amphibot, the prey tracker and the Webots simulation

`Papers` : cited papers

`Presentations` : mid-term and final presentation

`Report` : this report LaTeX source code

`Webots` : Webots worlds used for simulations