25

Impossibles: A Fully Autonomous Four-legged Robot Soccer Team

Hamid Reza Vaezi Joze, Jafar Habibi and Nima Asadi
Sharif University of Technology
Iran

1. Introduction

The goal of the international RoboCup soccer initiative is to develop a team of humanoid robots that is able win against the official human World Soccer Champion team until 2050. Currently, there exist a number of different RoboCup soccer leagues that focus on different aspects of this challenge. The Four-Legged League is one of them. In the league teams consisting of four Sony Aibo robots each play on a field of 6 m x 4 m. The robots operate fully autonomously, i.e. there is no external control, neither by humans nor by computers.

In this chapter we are going to present the Impossibles main architecture and its modules to create a fully autonomous team of 4-legged robots (Sony Aibo) for playing soccer according to RoboCup 4-legged Soccer League's rule. This architecture includes different modules such as World Model, Vision, Decision Making, Motion Controller, Communication, and Localization. This chapter presents the integration of our researches in different fields which came together to create fully autonomous robots for specific purpose that is playing soccer as humans do. And this could be a primitive attempt to develop intelligent robots.

In the first section we briefly describe the specifications of Aibo Robots, the history and rules of RoboCup 4-legged Soccer. In Section 2 we discuss previous works which consists of our team's architecture. Fig. 1 demonstrates our team (*Impossibles)* architecture. In next sections each module will be discussed separately. It includes the tasks of the modules, the corresponding task in other teams and the scientific methods which were used before or those that are presented by us.

The Vision module consists of chromatic distortion, color classification, line and object detection which is mainly concerned on colored image processing. Our Localization module applies piecewise linear probabilistic localization method which is based on Markov localization. In the case of distributed agents (against centralized agents) these raw data should be shared between all teammates via wireless communication. Decision making module is responsible for high-level decision and it consists of soccer strategies besides fuzzy rules. Motion skills, parameters of walking and movement estimation are the main parts of Motion module. In the Tools & Debugging Section we concentrate on debugging tools which simplify the process of debugging on the robots and also an interpreter which is used for facilitation of determining team strategies and player's roles.

Source: Robotic Soccer, Book edited by: Pedro Lima, ISBN 978-3-902613-21-9,
pp. 598, December 2007, Itech Education and Publishing, Vienna, Austria

1.1 Why Four-Legged Robot Soccer?

As an internationally recognized team game, soccer is a perfect standard project for studying how multi-robots perform and react autonomously in uncertain, team-oriented scenarios. The sport also provides some entertainment value adding good spirits and public interest to a concrete test bed event for researchers. These reasons could be the motivation of existence of RoboCup Soccer Competitions since 1996.
After years of competition in soccer with the Small Size and Middle Size robots, which moves with wheels, besides Simulation Leagues, 4-Legged Soccer League was added to RoboCup. The ambition of 4-Legged soccer league is to simulate the way human beings play soccer, with the use of legs. One of the other advantages of 4-Legged robots is that the platform is common among all the teams and this establishes a fair test-bed for Artificial Intelligence achievements.

1.2 Aibo Robot Specifications

AIBO robots' first generation was developed by SONY Corporation in 1999 for entertainment purposes besides its use in research laboratories. The Sony Aibo robot is currently a very interesting platform to conduct research in Robotics and Artificial Intelligence. Aside the numerous captors and actuators, the most important element is that Aibo is programmable. The Aibo programming language, built on top of C++, is provided by Sony as the OPEN-R SDK. The latest product of Sony Aibos is its third generation Aibo robot, ERS-7, with a great tool for wireless communication.
These robots have 20 degrees of freedom and are equipped with a 576 MHz processor and 16 MB of RAM. The most important sensor of this robot is a CMOS camera with 56.9° horizontal and 45.2° vertical vision angles.

2. Impossibles Architecture

Our previous experience in Multi-Agent System (MAS) architecture design in Simulation league environment led us to World Model Based Architecture (WMBA). We employ it as our basic designed architecture for concurrently-running objects of Open-R SDK. WMBA contains four major tasks which are done independently in following subsystems:

1. Sensing Subsystem
2. Communication Subsystem
3. Action Subsystem
4. Debugging Subsystem

These subsystems are to run repeatedly with different frequencies. They are managed in such a way that objectives are achieved and constraints are convinced. The main constraint of the AIBO robots is the limited resources such as CPU. The last subsystem which is new to previous architecture is in charge of gathering appropriate information from other subsystems and sending them to Aibo Controller software which will be described later.
Fig. 1 demonstrates the World Model based architecture. Subsystems are denoted by dotted rectangles, data flow is shown as arrows, and processes are shown by circles.
To make Decision Making module completely separate from other parts, we use a non Open-R abstract class called *AbstractPlayer*. There are two pure virtual functions in this

abstract class. The first one is *sense* which is called every time robot gets some new information. The second virtual function is *decisionMaking* which is called in a specific period of time to decide about new actions. The result of this function could be any of the actions such as walking, shooting, looking, etc. Some actions such as standing back to normal position in the case the robot falls down and also Blocking Skills (which will be discussed later in detail) are done completely without interrupts or preemptions, so *decisionMaking* is not called during these skills.

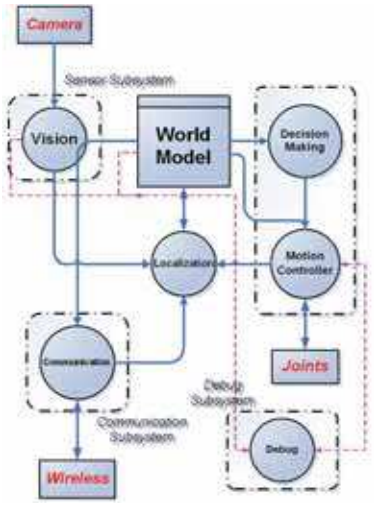


Fig. 1. Impossibles World Model base Architecture

3. Vision

Undoubtedly, vision is the most powerful sense of human being providing a great deal of information for interaction with environment without any physical contact. In this section, we concentrate on providing a method for real-time vision in a robot with low computational power and limited memory. Real-time vision means processing image frames with the speed of robot's camera. The most important sensor of this Aibo robot is a CMOS camera with $56.9°$ horizontal and $45.2°$ vertical vision angle.

Vision problem for these robots which refer to recognize type and location of surrounding objects is described as follows:

- Input: (i) The output of robot camera in the form of 30 pictures per second, with the size f 208x160 pixels and in the YCrCb color space consisting color distortion and noise. (ii) The value of robot's joints through which the value of camera's pan, tilt and roll can be obtained.
- Output: (i) Robot's distances and angles in relation to the ground's fix location by which the robot estimates its position in the field. (ii) Robot's distances and angles in relation to moving objects which determine their position relative to the robot.

The first work carried out on the image received from robot's camera is the correction of distortion existed in the color of image pixel far from the center of image. Then the color of

each pixel in the corrected image is attributed to one of the predefined color class. Then, existing blobs of each color are formed for the color of existing objects in the image, and based on the color, size and density of existing objects in the image is recognized. Also, by using a method provided for obtaining the three dimensional coordinates of each pixel in the image in the outside space relative to itself, position of each object relative to the robot is calculated. Specially, ground's lines are among of important objects in the soccer field which are distinguished in a separate manner with seeking green-white edges. Finally, having determined the position of objects relative to the robot, the position of robot in the field is calculated through the objects having a fixed place in the environment.

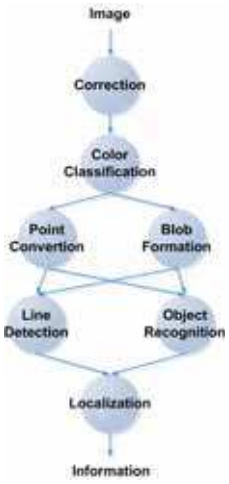


Fig. 2. Architecture of real-time vision system of robot

Fig. 2 displays the architecture of robot's real-time vision system (Mokhtarian et al., 2007). Also this system require offline processing regulating some provided algorithms' parameters for various setting prior to the application of software on the robot.

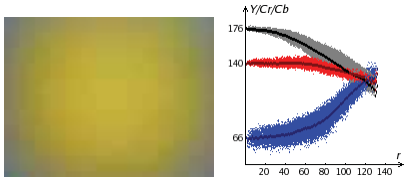
3.1 Correction of Chromatic Distortion

Aibo robot's camera causes a considerable chromatic distortion in the color of pixels at the corners of images. Fig. 3 (a) displays an image taken by such camera from a uniformly colored yellow page. Variations of the three color components (*Y*, *Cr*, and *Cb*) of this image's pixels are shown in Fig. 3 (b) in terms of pixels' distance from image's center (*r*).

The thick and fading curves of Fig. 3(a) illustrate variations of the observed values of the *Cr* component (channel) of pixels, in terms of *r*, in a number of images taken from uniformly colored pages. We define the *actual value* of each component for these colors, as that component's value in the pixels around the center of the corresponding image, where there is negligible distortion.

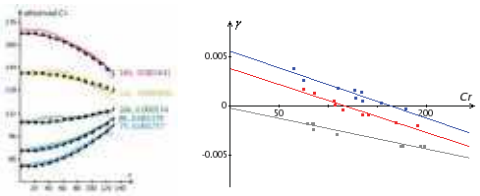
By shifting the horizontal axis to the width of *actual value* of the *Cr* component, for each curve (e.g. 141 for yellow) in Fig. 4(b), the diagram of $Cr_{observed} - Cr_{actual}$ is obtained for each color, which we approximate by a curve of second degree in the form of $\gamma \times r^2$. These

curves are shown in Fig. 4(a) by thin continuous curves. Beside each curve, the *actual value* of the *Cr* component and the corresponding γ value are displayed as well. The coefficient γ for each curve depends on the *actual value* related to the curve, as shown in Fig. 4 (b).



(a) (b)

Fig. 3. (a) Image taken from a uniformly colored yellow page. (b) Values of color components of Figure 2-a's pixels in terms of distance from the center (*Y*, *Cr*, and *Cb* are shown by gray, red, and blue colors respectively)



(a) (b)

Fig. 4. (a) Variations of the component Cr in terms of r in a number of uniformly colored pages. (b) γ values in terms of the actual value of Cr

We approximate the values of γ by a linear equation in terms of Cr_{actual} in the form of $\gamma' = \alpha \times Cr_{actual} + \beta$. Coefficients α and β for each color component, are parameters independent of images' colors and are determined for the camera with specified parameters (shutter speed, white balance, and gain).

Curves approximating $Cr_{observed} - Cr_{actual}$ using the new coefficient (γ' which is on turn obtained by a linear approximation from Cr_{actual}) are displayed in Fig. 4(a) as thick dotted curves. The maximum error of this two-level approximation in our experiments on uniformly colored images –Fig. 4(a) depicts variations of the *Cr* component of a number of them– has been acquired as an inaccuracy of at most 10 units for a component, which seems appropriate with respect to the interval of components' values (0..255). After determining coefficients α and β for the component *Cr*, Equation (1) holds for each pixel of an image.

$$Cr_{observed} - Cr_{actual} = (\alpha \times Cr_{actual} + \beta) \times r^2 \Rightarrow Cr_{actual} = \frac{Cr_{observed} - \beta \times r^2}{\alpha \times r^2 + 1} \quad (1)$$

Therefore, having $Cr_{observed}$ for each pixel, pixel's distance from the center of the image, and coefficients α and β, the actual value of the *Cr* component of each pixel's color is obtained using Equation (1). Similarly, the actual values of two other components of pixels' colors are

obtained and thus the chromatic distortion of the image is corrected. Fig. 5(a) and (b) depict two sample images captured by robot's camera and Fig. 5(c) and (d) depict the result of their correction. For speeding up the process of chromatic distortion correction of images in real-time visioning, we use pre-calculated tables for each process.

(a) (b) (c) (d)

Fig. 5. Two samples images taken by the robot's camera, before ((a) and (b)) and after ((c) and (d)) correction of chromatic distortion

3.2 Color Classification

In order to recognize objects existing in an image, firstly it should be segmented into color regions with specified colors and the rest of processes are carried out on the color-classified image. We use a three-dimensional $(128 \times 128 \times 128)$ color classification lookup table mapping points of the YCrCb color space to corresponding color classes. The number of colors that should be distinguished from one another (n) is 8. Moreover, we consider an additional class named *unknown* for colors similar to more than one of our specified classes.

In order to construct this table, we take a large number of images from the environment and objects with which the robot is dealing, and after correction of chromatic distortion, we specify relevant color segments in each image for the learning tool.

We conduct color learning and color classification based on the HSL color space since it resulted in best outcomes in our experiments regarding colors existing in our environment and its lightning conditions. Therefore, having collected the samples of each color class from captured images, the averages of *H*, *S*, and *L* components of each color class (C_i) is designated the standard point (h_i, s_i, l_i) for that color class in the HSL space.

In order to determine the class of each cell of the three-dimensional color classification table, first we obtain its corresponding point in the HSL space, and then calculate its similarity to each color class using a heuristic function in the form of $h : \{c_i \mid 1 \le i \le n\} \longrightarrow R$. The value of this function for a (h,s,l) point is calculated using Equation (2).

$$h(c_i) = \frac{w_i}{d((h,s,l),(h_i,s_i,l_i))} \tag{2}$$

In Equation (2), function d stands for the Euclidean distance between two points in the three-dimensional representation of the HSL color space, and w_i is the weight of each color class which somehow indicates its dispersion in the color space. Therefore, the standard deviation of positions of each class's sample points in the HSL space can be thought as an appropriate statistical criterion for the weight of that class. In addition, this criterion can be used as an initial state for obtaining the optimum set of weights which results in the best

outcome (i.e. minimum difference between automatically and manually color-classified images) using manual tuning tools or intelligent searching algorithms.
Having determined point in the HSL space corresponding to each cell of our lookup table, and calculated its similarity to each of our color classes, the most similar class is designated the color class of that cell. In addition, those points of the HSL space which are almost equally similar to more than one color class (i.e. the difference of their similarity to the most and the second most similar color classes is lower than a certain value) are placed in the *unknown* color class. Fig. 6 illustrates the color classification table obtained for our Aibo laboratory environment. This figure is a cube of side 256 from whose front a cube of side 192 is taken out. Gray areas of this figure stand for the unknown color class.

Fig. 6. Color classes in the three-dimensional space HSL

A sample of color classification (after correction of chromatic distortion) on the image shown in Fig. 7(a) is depicted in Fig. 7(b).

(a) (b)

Fig. 7. (a) Image taken by the robot's camera. (b) The result of color classification of (a)

3.3 Transforming a pixel in an image into a point in the space

A pixel in an image can simply be represented by (m,n), its coordinates in the image, where the coordinates axes of image are chosen as Fig. 8(a) for facilitating the calculations. On the other hand, a point in the space can be represented by (x,y,z), where the coordinates axes of the actual space are relative to the robot as shown in Fig. 8(a) (the floor is the plane $z=0$).

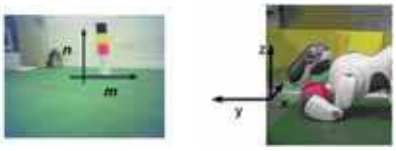


Fig. 8. coordinates axes in the image and in the actual space

Required parameters of the problem are:

- α : Camera's tilt (Angle made by camera and the horizon).

- β : Camera's pan (The angle of camera's deviation to left or right).
- γ : The angle between the oblique image and its horizontal representation.
- h_c : Height of the camera from the floor.
- ψ_{hor} , ψ_{ver} : Camera's horizontal and vertical angle of view.
- $width_{image}$, $height_{image}$: Image's width and height.

These parameters for Aibo robots are functions of the positions of rear and front legs and the three neck angles.

As a case in point, at each distance from the robot's camera, distances between objects in an image are assumed constant, i.e. if we know two objects are at distance d from the camera and at distance l pixels from one another in the image, then the actual distance between them in the space is regardless of whether they are seen at the center or at the corner of the image. If the given pixel has a value of m in the horizontal axis, then the actual point is located on a plane in the space crossing the point $(0,0,0)$, thus Equation (3) holds.

$$x / y = 2m / width_{image} \times \tan(\psi_{hor} / 2) \tag{3}$$

$$width_{image} \times x + 2m \times \tan(\psi_{hor} / 2) \times y = 0 \tag{4}$$

Therefore, the point is located on the plane represented by Equation (4). Similarly, if the pixel has a value of n in the vertical axis, then this point is located on a plane in the actual space represented by Equation (5).

$$height_{image} \times z + 2n \times \tan(\psi_{ver} / 2) \times y = 0 \tag{5}$$

The resulted line of crossing these two planes is a locus whose all points are seen at the pixel (m,n) in the image. Since the robot mainly interacts with objects located on the floor, we can posit the relevant point on the floor, thus the desired point is obtained by crossing the line mentioned above and the floor plane. After shift and rotation of coordinates axes relative to camera to axes relative to the robot itself using α , β , and γ , the results can be calculated using Equations (6) and (7).

$$\theta_x = a\tan(\tan(\psi_{hor} / 2) \times \frac{m}{width_{image}}) \qquad \theta_y = a\tan(\tan(\psi_{ver} / 2) \times \frac{n}{height_{image}}) + \alpha$$

$$x = -\frac{\cos(\theta_x)\sin(\theta_y)\sin(\gamma) + \cos(\theta_y)\sin(\theta_x)\cos(\gamma)}{\cos(\theta_y)\cos(\theta_x)\sin(\gamma) - \cos(\theta_x)\cos(\theta_y)\cos(\gamma)} \times h_c \tag{6}$$

$$y = \frac{\cos(\theta_y)\sin(\gamma) \times x + \cos(\theta_y)\cos(\gamma)}{\sin(\theta_y)} \times h_c \tag{7}$$

The above calculations have been performed assuming that $\beta = 0$, otherwise, the actual point $(x, y, 0)$ should be rotated by an amount of β around the origin.

3.4 Object Recognition

Object recognition in robot's vision consists of detecting objects existed in an image, assigning actual objects of the environment to them, and locating the recognized object regarding the coordinates axes relative to the robot.

The robot's working environment can be assumed closed, i.e. there is a set of specified objects to which no new one will be added. However, it is in general possible for Aibo footballer robots to see unspecified objects, i.e. other than known objects of the robot's environment (ball, goals, players, and landmarks). In this subsection, recognition of known objects based on blob formation is described first, and then recognition of unspecified objects.

A) Blob Formation

In our robot's real-time visioning sub-system, specified objects are recognized based on their color, size, and density and position of their corresponding blob in the image. Therefore, the next task after color-classification of an image is to form blobs existing in it.

In order to form such blobs in a color-classified image, connected pieces of relevant colors are obtained by a scan on the image. The density of a blob is equal to the number of points of the connected piece divided by the surface of the rectangle circumscribing that blob. Obviously, small blobs and those having low densities are ignored as noise. Fig. 9 shows a color-classified image and its relevant blobs.

Fig. 9. A color-classified image and its relevant blobs

B) Specific Object Recognition

The noteworthy characteristic of specified objects is that their shape and size are known for us. Therefore, types of these objects can be recognized knowing positions of relevant blobs, and their locations can be determined by geometrical calculations depending on their shape. Ball recognition is presented here as a sample of specified objects recognition.

In order to recognize the ball, which is an orange sphere, the relevant orange blob is considered the blob candidate to be the ball. Two parameters, circle's radius R, and coordinates of the circle's center (m_c, n_c) in the image, should be extracted from this blob. They can be calculated by averaging the center and the radius obtained for each three arbitrary border pixels of the observed ball. Regarding the parameters and coordinates, the ball's actual (x, y) relative to the robot can be calculated using Equations (9) and (8).

$$y = (\frac{33}{2R} \times 50 + 8.14) \times \cos(\beta) \tag{8}$$

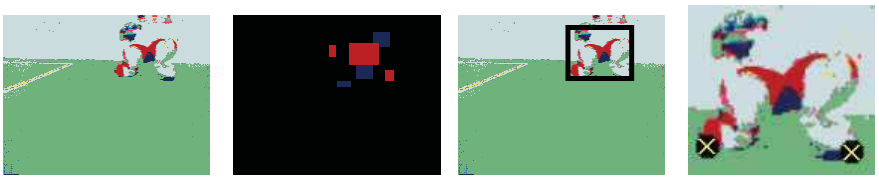
$$x = -R_{Ball}\, sign(\gamma) \times (n_c - \frac{m_c}{\tan(\gamma)}) \Big/ R\sqrt{1 + \frac{1}{\tan(\gamma)^2}} \tag{9}$$

White lines in the green fields are benefical information especially for determining the position of robot. For more information about our Line Detection methods refer to (Vaezi et al., 2007).

C) Recognition of Unspecific Objects

Unspecified objects' position can not be determined using their geometrical properties (i.e. shape). We consider unspecified objects just as some obstacles. Since an Aibo robot has a complicated shape whose recognition is not practical in our robot's real-time vision, the players in the field are viewed as unspecified objects by our robots.

Our algorithm is that existing blobs in the image which do not constitute any specified object and are located on the floor, are considered unspecified (unknown) objects. The assumption that they are placed on the floor allows us to locate their points on the floor using their blobs' lowest pixels and the method presented in Section 3.3 for transforming image's pixels into points in the actual space. At this location, there is merely an obstacle, and nothing about this object is determined but this obstacle's front edge placed on the ground. Stages of conducting this procedure for an unspecified object are shown in Fig. 10.



(a) (b) (c) (d)

Fig. 10. Stages of recognition of an unspecified object

The color-classified image and its relevant blobs are depicted in Fig. 10(a) and (b) respectively. Having checked these blobs and understood that they do not belong to any known object, the composition of these blobs, which is depicted in Fig. 10(c), is recognized as an unspecified object. Two points located at the bottom of this object (bottom-right and bottom-left corners of the object) leading us to calculation of this object's whereabouts are shown in Fig. 10(d).

3.5 Experimental Result

Since the presented methods and algorithms are to be used for Aibo robots' real-time visioning, the running time and the accuracy of them are two main parameters for assessment of these methods' appropriateness. Table 1 displays the running time of our visioning stages (accuracies are noted in relevant subsections). According to methods described above, the time required for correction of chromatic distortion and color classification is constant for all images. The time needed to form colored blobs, which is indicated in the third row of the table, is calculated for an almost crowded image including all colored objects existing in the environment of Aibos football field. Objects recognition

consists of a few calculations and its running time is negligible in comparison with other visioning algorithms.

Table 1. Running time of different process in Impossibles Vision subsystem

Visioning Stage	Running Time
Correction of chromatic distortion	4.8 *ms*
Color classification	4.1 *ms*
Blob formation	3.9 *ms*
Object recognition	Less than 1 *ms*
Line detection	3.8 *ms*
Total	16.7 *ms*

The total amount of time consumed by the visioning process is about 17 *ms* per image, thus almost the half of the time interval between two frames is left available for the rest of processes (e.g. decision making, communication, etc.).

4. Communication

Sony AIBO ERS-7 model have a wireless LAN module (wi-fi certificated) which made it able to communicate with other teammates to share useful information perceived from the environment to improve the quality of each agent's world model eventually resulting in more accurate localization and object detection.
Furthermore, Due to RoboCup 4-legged rules, each team has an upper band limit of 500 Kbps for communication among the agents which includes also game manager commands but every team has some special UDP port to broadcast. So ideally we can count on about 100 Kbps bandwidth for each AIBO robot.
In what follows we will discuss various aspects of communication and will explain the way that our communication module is working.
Impossibles AIBO communication module, as an independent module, works in parallel with other modules such as vision, and decision making. It is also in charge of sharing essential data amongst all players. For instance, knowing accurate positions of the players are only possible by having each player report his information such as its own position to the others.
The communication module is to be reliable and eventually be aware of the packet loss if any exists. It will repeatedly choose entities from World Model (WM) objects based on their last report time, their reliability measure and also importance of data for other teammates.

4.1 Information Level

There are two general strategies for communication in Multi-Agent Systems (MAS) that a team can employ depending on system's general architecture and also on what kind of data the agents intend to share.
High Level Commands: This strategy is best applicable in centralized system architecture where a center commands its agents; therefore, in this method, all critical and high level processes and decision makings are made in center. Consequently, only high level commands are sent to agents in order to make them aware of their behavior.

Low Level Commands: In this method communication system is trying to share all raw data that each agent have and every thing could and should be distributed.

4.2 Centralized vs. Distributed Architecture

Generally, we consider the communications amongst players distributed, but due to the large amount of transmitted data and hence time-consuming processes, agents themselves accomplish their own jobs and broadcast the results, i.e. processes data.
If there wasn't any broadcast feature in our access media, having centralized communication may also reduce number of messages which are needed to share all information among agents.

m = number of messages needed to have all information shared between agents

- With broadcast message:
  - Centralized approach $m = n + 1$
  - Distributed approach $m = n$
- Without broadcast message:
  - Centralized approach $m = n + n = 2n$
  - Distributed approach $m = n \times (n - 1)$

When we are considering our access media properties including its broadcast ability and limited bandwidth and also the fact that defining an agent as center might be unreliable we decide to use distributed communication by broadcasting messages.
The messages contain low level data sensed and acquired by agents from the surroundings such as ball, teammates, and opponent players which are used in localization and updating word model in with each agents self awareness.

5. Localization

Mobile robots must know where there are to operate their tasks properly and this is the first step to having autonomous mobile robot**Error! Reference source not found.** (Kortenkamp et al., 1998). Mobile robot Localization is the process of determining and tracking the location of a mobile robot in global coordinate frame. Localization problem is occasionally referred to as the most fundamental problem to providing a mobile robot with autonomous capabilities. A number of techniques have been used for this, including grid-based approaches or sample-based approaches such as Monde-Carlo. Grid-based approaches require computation of the probability even in the area where the probability is negligible and in the vast environments grids are either too many or big. The former makes communication expensive and the latter makes the results low-resolution. On the other hand, sample based approaches which are mostly based on sampling-importance re-sampling (SIR) algorithms (Rubin, et al., 1988) require the computation of a significant number of samples to support high-performance especially for large areas.
We use a localization algorithm to localize soccer player robots in the field which is called piecewise Linear Probability Distribution Localization (PLPDL) (Vaezi Joze, et al., 2007) that

besides designing for low-performance and low-memory machine can localize robots in the uncertain and dynamic situation. The approach is based on Markov localization (Fox, 1998) which localize robot probabilistically. Our approach inherits from Markov localization the ability to localize a robot under global uncertainty. Using piecewise linear functions to approximate probability distribution functions of these random variables would help our approach to be fast and inexpensive which is suitable for real-time processing of our robots.
The key idea of Markov localization is to compute a probability distribution over all possible position in the environment. Let $l = (x, y, \theta)$ denote a position in the state space of the robot, which x and y are the robot's coordination in the soccer field frame, and θ is the robot's orientation. The distribution $Bel(l)$ expresses the robot's belief for being at position l. Markov localization applied two different probabilistic models to update belief function, an action model to incorporate movement of the robot and a perception model to update the belief upon sensory input.

5.1 Separate Probability Density Functions

If we suppose that probability of $x = x_0$ is independent from probability of $y = y_0$ and also other independencies for other coordinate variables of robot location, we could conclude from independency rule of probability theory that:

$$\begin{aligned} &Bel(l_0 = (x_0, y_0, \theta_0)) = P(x = x_0 \wedge y = y_0 \wedge \theta = \theta_0) \\ &= P(x = x_0) P(y = y_0) P(\theta = \theta_0) \end{aligned} \tag{10}$$

This could convince us to use separate probability for each of the coordinates. In contrast with Markov Localization assumption or other grid-base localization coordinate variables such as x are continuous in real environment, so we should consider them as continuous random variables and their density function could be define in the following formula: (notice that we assume these random variables are independent, otherwise we have a three variable density function)

$$P(a < x < b) = \int_a^b f_X(x)dx \tag{11}$$

Equation (11) concludes $f_X(x)$ is non-negative function and $\int_{-\infty}^{+\infty} f_X(x)dx = 1$.

Using Equation (10) and definition of density function for coordinate random variables:

$$\begin{aligned} &Bel(l_1 = (x_1, y_1, \theta_1) < l < l_2 = (x_2, y_2, \theta_2)) = P(x_1 < x < x_2 \wedge y_1 < y < y_2 \wedge \theta_1 < \theta < \theta_2) = \\ &P(x_1 < x < x_2) P(y_1 < y < y_2) P(\theta_1 < \theta < \theta_2) = \int_{x_1}^{x_2} f_X(x)dx \int_{y_1}^{y_2} f_Y(y)dy \int_{\theta_1}^{\theta_2} f_\theta(\theta)d\theta \end{aligned} \tag{12}$$

The product of these functions obtains belief of robot to be at this position, namely $Bel(l)$. So our new localization method considers a separate probability density function for each variable (such as x, y and θ for mobile robot in 2 dimension environments). In Markov Localization0, for each position in the area $l = (x_0, y_0, \theta_0)$ there is $Bel(l)$ which means the robot's belief for being at position l. In contrast, in Probability Distribution Localization (PDL), we have three probability density functions to express our belief for location of robot. This model could be used for current location, new observation and also differential motion, i.e. $(\Delta x, \Delta y, \Delta\theta)$. We need to update current location of robot in the case of motion and reading sensors. In PDL, Motion Update is corresponding to robot motion of Markov Localization and Sensor Update is corresponding to sensor reading in Markov Localization.

- ***Movement Update:*** We consider X a random variable and its probability density related to x position of robot and ΔX as a random variable of movement of robot in x dimension and its probability density. So the new value for X will be $X + \Delta X$. In this way the corresponding density function for X is obtained. We know from probability theory that if X,Y,Z are random variables and Z=X+Y so probability density of Z could be conclude by convolving probability density of X and Y. And also other random variables will be updated independently using motion data that should be in the form of different probability density function for each random variable.
- ***Sensor Update:*** Sensor is usually return to the vision module in robot. However it could be any other sensor for localizing mobile robot. If we suppose sensor data return a probability density function for each variable such as X and p that is the belief of correctness of these data. Now we should create a new probability density for X by the following formula using previous density of X and sensor data in the form of new density function and our belief of its correctness:

$$Fx_{New} = Fx_{old} \times (1 - p) + Fx_{Sensor} \times p \tag{13}$$

Our probability density functions may become worthless after too many movements or sensor updates with small p. So we use the idea of "Sensor Resetting Localization" (Lenser & Veloso, 2000) that considers a threshold for average of p. Some new sensor updates must replace when it becomes lesser than the assigned threshold. This could be translated to threshold for distribution of our density functions. In such cases, more sensor data must be fed by sensor module. The case should happened when the result density function is so distributed (a precise parameter needed for determining threshold that could be variance)
As explained before, Probabilistic Distribution Localization (PDL)'s main output is a probability density function and not a crisp value, but PDL is required to provide more suitable results for other modules such as motion module. So a kind of clustering algorithm can be employed to prepare crisp data as output if it is needed.

5.2 Piecewise Linear Probabilistic Distribution Localization

In this section we are going to change PDL approach in such a way that it becomes simple and suitable for real-time applications for mobile robots. In order to simplify the PDL

process, we employ piecewise linear probability densities in order to make our calculation and storage system much simpler. In piecewise linear probability densities, functions are limited to be made by linear pieces. For storing these functions it is enough to store points which are disjunction of linear pieces. For instance, function shown in Fig. 11, could be determined by set of following points: { (0,0) , (1,.5) , (2,.5) , (3,0) }

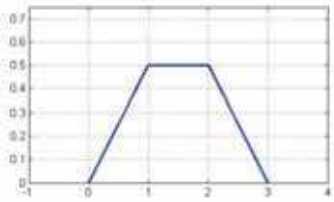


Fig. 11. A sample piecewise linear probability density function

Using piecewise linear function for expressing probability density of location parameter in PDL approximate the main probability density to obtain speed and decrease in memory usage. Storing set of points instead of storing complicated function could help to decrease memory which used for localization purpose. On the other hand applying linear limitation over probability density function could imply appearance of faster algorithms for Movement update and sensor update that are express bellow.

- ***Movement Update:*** With us considering both probability density of random variables X and Y piecewise linear, probability density X+Y is paranoid-segment function. To simplify the process, we approximate such functions to be piecewise linear function (we should also suggest such a converter algorithm). It could be done using convolution of these function as it is discuss before. As an example Fig. 12(a) is probability density function of a random variable before movement update. Fig. 12 (b) shows probability density of movement and Fig. 12 (c) is the final result of movement update. Fig. 12 (d) shows linear approximation of result via PLDL algorithm.

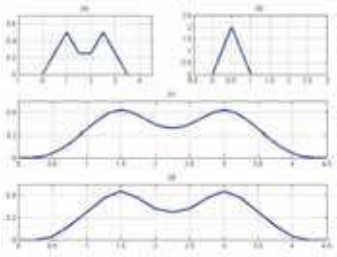


Fig. 12. An example of Movement Update process

- ***Sensor Update:*** This step is straightforward in piecewise linear density function. Using Equation 5-4 we should compute weighted sum of two density function such as Fig. 13 (a) and (b). These functions have stored by set of points so for calculation result we should construct result set using union of x points of both sets with corresponding y that could be calculated by sum of corresponding y. Fig. 13(c) illustrates result of Sensor

Update with p=0.7. Fig. 13(a) is previous density function for a distinct variable and Fig. 13 (b) is sensor data of that distinct variable.

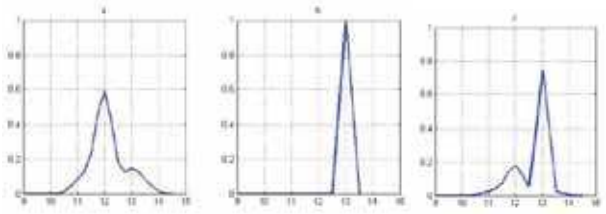


Fig. 13. Sensor Update example. (a) Previous density function (b) sensor data (c) result by p=.7

Also to decrease required memory for storing set of points for probability density functions we omit points that express small amount of information. Strictly speaking, each three consecutive point construct a triangle, we omit central point when the area of this triangle is smaller than a threshold. Additionally, the function is scaled in a way that integral of the function becomes one. We could also describe a parameter as maximum number of points in the set so we could control the computational time needed for PLPDL.

5.3 PLPDL Application

The mentioned algorithm used in the software of controlling Aibo robots for playing soccer as a self-localization sub-system. Vision used as a sensor and movement update support from Motion Controller sub-system. Fig. 14 demonstrates Self-Localization flowchart using PLPDL method. We explained sensor data which is supplied by vision in Section 3 and movement data will discussed in Section 7. As it presents before probability density function for each coordinate variable is stored by set of points. The next sub module is *PDF Filtering* which filters probability density functions in order to omit small values and also non-reliable ones. Then, it is determined if some extra samples are required from Vision sub-System. This decision is made using a threshold over probability density functions' variance after clustering them.

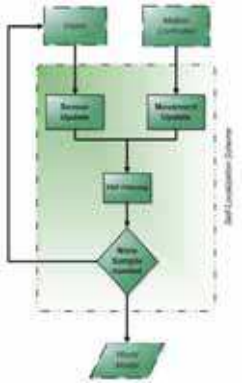


Fig. 14. Self-Localization Flowchart

When it needs more sensor data it sends a signal to vision module to provide localization module with some extra sensor data. This signal also contains information about the accuracy of such a data that may cause executing time-consuming routine in Vision sub-system for more accurate data. Although we do not use information about positions of Aibo robots from external source such as other Aibos using wireless communication, it can be employed as extra sensor data with smaller belief.

6. Decision Making

As explained in our architecture (Section 2), Decision Making (DM) module plays the key role in logical decisions made by agents in a multi-agent environment such as AIBO soccer. DM is done in a completely distributed manner in *Impossibles* AIBO robots; however, communication is employed in order to propagate the information obtained from vision, sensors, and communication modules. Consequently, information is propagated by communication and decisions are made by agents themselves. This section is organized as follows: The intra-DM architecture is explained in details in subsection 1. Team behavior is given in subsection 2. Subsection 3 is devoted to individual behaviors.

6.1 Architecture

Intra-DM module in *Impossibles* AIBO robots have a hierarchical layered architecture (shown in Fig. 15). In fact, DM module consists of two major layers. Team Behavior (TB), i.e. tactics, layer is the highest one which determines the tactics of the soccer team. Secondly, Individual Behavior (IB) layer is the techniques employed by individual players. As demonstrated in Fig. 15, Decision Making (DM) module gets its input from the system's world model including opponent players', teammates', and ball's locations accompanied by some degrees of belief which is due to existence of uncertainty in real system environment such as soccer.
Having gotten the inputs from its world model, the AIBO robot will analyze the input in a two-step procedure. Team behavior (TB) sub-module gets DM inputs from world-model and then resolves the whole team behavior, e.g. tactics stored in a database. Finally, TB passes the team behavior and world model information to the lower layer that is Individual Behavior (IB).

Fig. 15. Decision Making Module Architecture

As the second step, the Individual behavior (IB) sub-module obtains the whole team behavior and world model information form upper layer sub module (TB); then, analyzing its inputs, IB sub-module decides one of the possible actions to do. As a matter of fact, these actions are the outputs of the IB sub-module and hence the outputs of the whole Decision Making (DM) module. This actions set includes (1) shooting in a specified direction with a particular power, (2) blocking the way in a special direction, (3) walking through a path determined by an array of points, (4) looking in one direction, and (5) grabbing the ball.

6.2 Team Behavior

As explained above, *Impossibles* AIBO team tactics is resolved in Team Behavior (TB) sub-module. The final tactics of the team will be selected from tactics database.

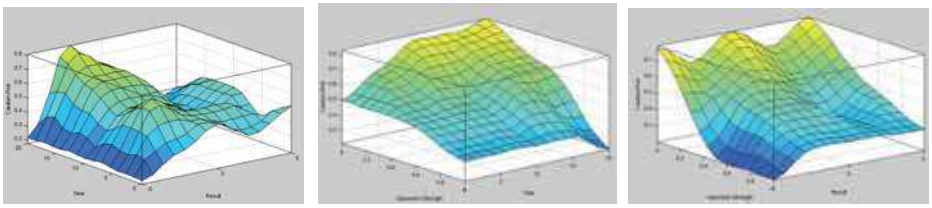
Determining Factors

Tactics selection step needs two parameters. First, fuzzy membership degree in offense set is to be determined, i.e. Defense-Offense (DO). Teammates' and Players' sites is defined to be the second parameter

A) Defense vs. Offense

Tactics of the team is defined by a fuzzy membership degree, i.e. DO, in offense set. Between complete defense condition, i.e. 0, and complete offense condition, i.e. 1. The following three parameters are calculated and then employed to obtain the membership degree.

1. *Caution and Risk:* The first parameter which contributes to obtain DO is Caution-Risk (CR) fuzzy membership degree. We have employed a fuzzy logic controller which outputs CR degree. This fuzzy logic controller gets three inputs: The result of the game, time, and opponent's strength.



(a) (b) (c)

Fig. 16. Caution and Risk which is the output of fuzzy controller for (a) a fixed result (b) a fixed Opponent-Strength (c) a fixed Time

2. *Ball Ownership:* Ball ownership is a critical factor which contributes in producing the final selected team strategy of an AIBO soccer team. In real world soccer environment we can define ball ownership as a crisp value which at least last long enough to determine team strategy. In fact it can be represented via digital magnitudes such as a Boolean variable. Ball ownership in AIBO cannot be defined in such a way. Because in AIBO soccer game robots intermittent lose the ball; therefore, selected team strategies will be changed so irregularly that it becomes impossible for a team either to defend or attack. Here we define a fuzzy membership degree in a Ball Ownership (BO) set. In

Impossibles robots, the following formula is employed to calculate the BO of the team in order to evaluate the team membership degree in the complete ownership set.

$$BO(Team_i) = \sum_{m_j \in Team_i} \frac{1}{d_j^{\alpha}} \tag{14}$$

The final Ball Ownership (BO) factor is obtained by division of our team's BO and opponent team's BO.

3. *Hyperbolic Danger Safety Degree:* As explained above, Ball Ownership (BO) is a factor due to players' rational locations to the ball; however, players' absolute locations are also important. In order to accomplish the job a Hyperbolic Factor (HF) is defined. A hyperbola (Gray, 1997) is a conic section defined as the locus of all points P in the plane the difference of whose distances $r_1 = F_1P$ and $r_2 = F_2P$ from two fixed points (the foci F_1 and F_2) separated by a distance $2c$ is a given positive constant k, $r_2 - r_1 = k$. Letting P fall on the left x-intercept requires that $k = (c+a)-(c-a) = 2a$. So the constant is given by $k = 2a$, i.e., twice the distance between the x-intercepts (left figure below).

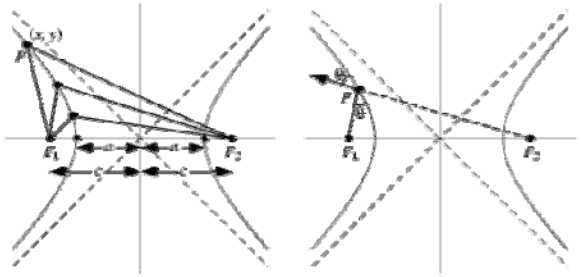


Fig. 17. Hyperbola used to calculate Hyperbolic Factor (HF)

We think of the AIBO soccer field to be locus of hyperbolas with variable positive 'a' and goals to be the focuses of theses hyperbolas; therefore, 'c' is defined to be $0.5 \times Field\_Length$, i.e. distance of the goals from the center of the soccer field. Each point in the field is defined to have a danger degree (DD) if an opponent team member is located in this point. On the other hand, Safety Degree (SD) is defined if one of our team members is located in that point.

$$DD(P_i) = \frac{\text{distance}(P_i, \text{opponent Goal}) - \text{distance}(P_i, \text{our Goal})}{Field\_Length} \tag{15}$$

$$SD(P_i) = \frac{\text{distance}(P_i, \text{our Goal}) - \text{distance}(P_i, \text{opponent Goal})}{Field\_Length} \tag{16}$$

According to above equations, points on a hyperbolic locus with a constant 'a' will have equal Danger Degrees (DD) in the case of having an opponent player in the point. Similarly, the points have equal Safety Degree (SD) if one of our team members is situated in one of those points. The final Hyperbolic Danger-Safety Degree (HDSD) factor is calculated employing players individual Danger Degree (DD), or Safety Degree (SD).

$$HDSD = \frac{f_1(SD(\text{our team members}))}{f_2(DD(\text{opponent team players}))} \tag{17}$$

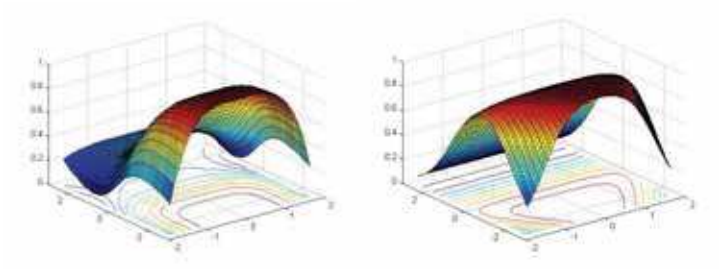
As a significant factor, $f(x)$ is selected according to coach basic idea of either defensive or offensive strategies. We have employed '*Averaging*' function. Therefore, *HDSD* factor is evaluated:

$$HDSD = \frac{\sum_{m_i \in \text{our team}} SD(m_i)}{\sum_{m_i \in \text{opponent team}} DD(m_i)} \tag{18}$$

B) Team Fear-Relax Emotional Degree
As explained before, team behavior is determined according to Defense-Offense Degree (DOD) which lies in the range of 0 to 1. In above subsections, three determining factors were defined: Caution-Risk (CR), Ball Ownership (BO), and Hyperbolic Safety Danger Degree (HDSD). Now these three parameters are to be combined to represent the final Team Behavior Defense Offense Degree.

Team Strategy Database
In order to avoid having CPU over usage problems, we save predefined team strategies in a Team Strategy Database (TSD). In each moment of the game, a linear combination of the proper strategies is computed based on TBDOD and players' locations. Selecting from TSD offers two priorities over computing dynamic team strategies. First it supports to have a more flexible team behavior, because further team strategies can be added to TSD later. Second, this approach helps to decrease the CPU usage.
Generally, team strategies are categorized into three groups. Defensive strategies, midfield strategies, and offensive strategies are the mentioned groups. Two independent defensive team strategies (DTS) of *Impossibles* AIBO robots are presented. **Error! Reference source not found.** demonstrates the first DTS in which our players try to defend opponent's forward players reaching the goal along a line from the center of the field to our goal. It is the most defensive strategy of the team employed in critical circumstances.
Note that the following surfaces represent the values of points on the soccer field according to the team strategy. For instance, the dark red points in the diagrams indicate the most important regions of the field. On the other hand, the blues ones denote the regions which are not considered as significant regions. To clarify the problem it may be useful to declare that (0, 2.7) is the center of our goal. Fig. 18(b) demonstrates the general defensive (GD)

strategy of the *Impossibles* AIBO robots employed to some objectives such as preventing the game result being changed.



(a) (b)

Fig. 18. (a) the maximum defensive team strategy. (b) General Defensive (GD) team strategy

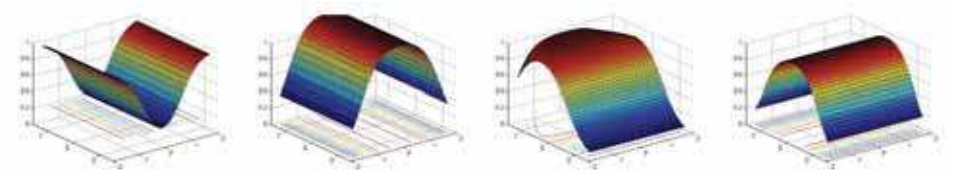


Fig. 19. Four basic strategies

Midfield and offensive team strategies of *Impossibles* are in fact generated easily as a combination of the following basic strategies (Fig. 19).
In Fig. 20 denote midfield and offensive strategies of the team. These strategies are generated using the above basic strategies. Here we have employed the simplest operation, i.e. multiplication, to produce midfield and offensive team behaviors.

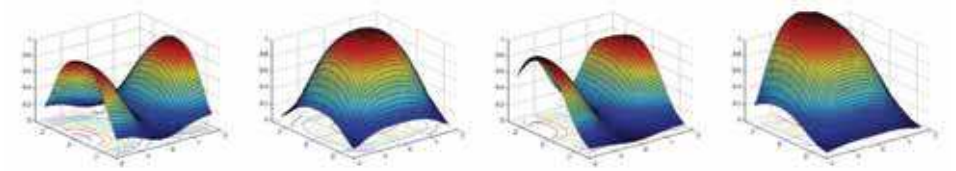


Fig. 20. Four strategies which generated using combination basic strategic

Emotional Tactics Selection
One of the most important aspects of human decision making is the role of emotions in its behavior and reactions. Humans choose their actions and make their decisions due to several internal variables called emotions. Emotional decision making is employed by humankind to avoid time-consuming and computationally-expensive approaches, e.g. using mathematical equations in decision making, to optimize the final result in critical circumstances such as danger (Locus et al., 2002).
In order to reduce the computation burden of the team behavior generation, we exploit Emotional Tactics Selection (ETS) approach. As presented in, agents' Emotional Decision Making (EDM) is based on their different states, called emotions. Up to now the robots have

calculated the fuzzy emotional Fear-Relax (FR) degree of the whole team. Given this degree, robots are to compute the team strategy, i.e. a linear combination of Team Strategy Database (TSD) elements. Fundamentally, in, transitions from one emotional state to the other are thought of being a gradual change, i.e. not suddenly, as it is done in real world animals and human. Therefore, we avoid quantization of the given FR degree. In contrast, based on the given FR degree, a linear combination of the strategies in TSD is computed as the whole team behavior, i.e. team strategy.

6.2 Individual Behaviors (Techniques)

Impossibles AIBO robots make use of a priority-based selection approach to choose the most proper action in various situations. In other words, after calculating some mathematical equations, each possible action is assigned to have a score; then, the most appropriate action is chosen. For instance, a player, who has the ball ownership, can select one of the following actions: (1) Moving with ball, (2) Passing to a teammate player, (3) Shooting toward the opponent team's goal, or (4) Looking around. In this section, different individual behaviors are explained logically. A lower level design is provided in Motion Controller (MC) section.

Predefined Dynamic Assigned Regions

Generally, in multi-agent systems, decision making can be accomplished using one of these four solutions (Habibi & Nayeri, 2006): (1) No Sharing Decision Making, (2) Information Sharing Decision Making, (3) Centralized Decision Making, (4) Fully Centralized Decision making. *Impossibles* AIBO robots use the second approach. In this method, communication is employed just for transporting the information; therefore, neither commands nor decisions made by center are transmitted.

With us using Information Sharing decision making solution, the most critical problem was similar behaviors of the robots in the same situations. So robots are assigned predefined roles as in real world soccer.

Players are assumed to have tendency toward their dynamically assigned regions. This tendency is represented by a simple spring; hence, there will be a linear dependency ($\alpha = 1$) between the player's tendency and the distance from its current location to its assigned region.

$$\text{Tendency}(P_i) \;=\; K \times \text{distance}\big(\text{Location}(P_i),\ \text{Assigned\_Region}(P_i)\big)^{\alpha} \tag{19}$$

Where 'K' is a positive constant which can be learned. Experiences show that setting 'α' to be 1.3 results in the best known outcome.

Outputs as Decision Making-Motion Engine Interface

Last of all, having logically produced Individual Behaviors (IB) of the players, Decision Making (DM) module passes IBs to the lower level module, i.e. Motion Controller (MC); therefore, IBs are considered to the interfaces between the DM and MC modules. In this section the employed Individual Behaviors are explained logically.

1. Looking: The objects stored in World Model (WM) own a saved parameter called Update Time (UT). It denotes the last time when a particular object has been seen by an

agent. So it may be necessary for agents to refresh their knowledge about their surroundings limited to their vision capability, i.e. approximately 1.5 meters.
2. Running: Walking and Running as two basic categorizations of motion should be implemented efficiently because of their importance. A lot of learning algorithms on AIBOs have been presented during the past few years (Kohl & Stone, 2004).
3. Ball Grabbing: In order to get the ball ownership, Decision Making (DM) module of a player passes 'GRAB' to the lower level module, i.e. Motion Controller (MC).
4. Shooting & Passing
5. Blocking

7. Motion

Different approaches can be considered as the way to make AIBOs move in a proper order, all accompanied by number of advantages and disadvantages. Enhancing robot's movement by restricting it to mathematical formulas and geometrical models, dividing motion skills into some predefined and atomic consecutive series of actions, using simulation results for realistic environment, eliciting models for each of the skills based on experimental data and learning algorithms, and so on, are instances of how to obtain methods for robot's motion. Studies conducted on all approaches, led us to a hybrid, innovative method with the most consistency to other modules, chiefly Decision Making.
We divided motion skills into two categories based on their usage: Blocking Skills and Non-Blocking Skills. While performing a non-blocking skill, Decision Making (DM) module can make new decisions if necessary, and send an interrupt to Motion Controller module. Thus, Motion Controller will preempt or halt the previous action and start performing new commands. Walk and rotate are basic actions included in non-blocking skills. On the other hand, Blocking Skills are smallest consecutive segments done atomically. Different kinds of shoots as well as ball blocking actions are examples of blocking skills.
We mapped the movement of robot's joints, while running, to a geometrical space so to have a complete set of parameters for Machine Learning techniques. The geometrical shape to which we modeled robot's movement was an ellipse. As a result, the main task of this module is done through some offline processes to improve ellipse's parameters in a way to reach a better speed and increase the accuracy of walks.

7.1 Architecture

This module, Motion Controller (MC), provides an interface of high level commands for DM module, such as *walk, look, localSearch, ballGrab, defend, block,* etc. When these high-level commands are received from DM module, a planning algorithm is used to break them into a series of low-level commands to satisfy the robot's conditions and team strategy. Finally, all generated data are converted to physical joint values inside the Motion Maker layer. Based on the attributes of each skill, the final data will be put in Motion Queues with the special characteristics of Blocking or Non-Blocking actions. Data inside the queue are considered as segments.
Segments are treated as atomic actions. These include some points in a 3D coordinating system, for example the coordinate of paws, or the camera in relation to body's center. So, in order to convert these points to joint values, we used Inverse Kinematics functions for both head and legs.

Based on aforementioned architecture, design and implementation of the MC in layers can avoid complexity and will increase simplicity of defining new skills. Fig. 21 shows the architecture of MC module in brief.

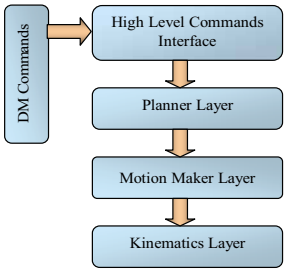


Fig. 21 Architecture of Motion Controller Module

7.2 Blocking Action

Blocking Skills are smallest consecutive segments which can not be preempted or halted while being processed. So Decision Making should use them in suitable situations. These actions contain consecutive value of joints which can be utilized from a static look up table that does not change during the game. We use our structure to store these date in files; therefore we have separate configuration files for each blocking skill. These actions contain all types of shoots, blockings by goalie and stopping the ball.

7.3 Non-Blocking Action

None-Blocking skills are those who accept interruptions. While Aibo is processing and running non-blocking commands, DM module can decide new actions and send them to the robot. In this case, the robot will halt the current job and start new task. These actions are necessary due to uncertain conditions of a soccer game. Walking and ball grabbing are simple non-blocking actions.

7.4 Movement Parameter

We modeled the movement of a robot to the movement of AIBO's paws on the perimeter of an ellipse. Although the Aibo leg's degrees of freedom are less than natural animals, this model resembles the natural walk of them more.

To form the ellipse based on which the robot plans to move, walk needs number of parameters. These parameters are as follows:

- Semi-major axis of the ellipse, with the symbolic name of *a*, for both front and rear legs
- Semi-minor axis of the ellipse, with the symbolic name of *b*, for both front and rear legs
- Coordination (x_0, y_0, z_0) of the center of the ellipses for all legs in relation to body's coordinating system.
- Angles alpha, beta and theta which gives the ellipse all degrees of freedom. So we can rotate the ellipse around each of the 3D coordinating system's axis.

Formulating movements of joints has additional advantages affecting the performance of other modules, especially Localization. By the use of geometrical models an increase in

accuracy of estimations is gained, although most of estimations are based on experimental data. This estimation helps Localization module to localize the robots without the use of camera or other sensors in an average period.
Computation of new walk parameters and constructing new motion skills by combining the previous motion skills is also another advantage of formulation. Based on some predefined parameters for basic skills in addition to mathematical models for skill combinations, there would be a chance to gain new motions. For instance, by merging the parameters for a simple forward walk and anticlockwise rotation, a new motion can be constructed to circulate around a circle with specific radius; this skill, with the radius as an input parameter, can be used in ball grabbling action.

7.5 Estimation

Since repetition of intricate geometrical solutions based on image processing algorithms may appear expensive in many cases, motion approximations for movements may be used to increase the accuracy of total estimations, and to lessen the cost of localization methods. The mechanism for motion estimations is based on robot's movement, in any of the directions, rotations and mixture of all couples of skills.
For each of the basic skills, X movement (d_x), Y movement (d_y) and angular movement (d_t) are estimated and written as attitudes for their own parameters. Whenever a skill is being performed by the robot, these movements will be calculated for each time slice first and finally will be calculated for robot's total walk. A simple way of calculating the estimations based on d_x, d_y, d_t is shown in Equation (20). Then all these factors are reported to Localization module to estimate the location of the robot.

$$D_x(S) = d_x(S) * t_p(S) \ , \ D_y(S) = d_y(S) * t_p(S) \ , \ D_t(S) = d_t(S) * t_p(S) \qquad (20)$$

These estimations cause some problems in some cases. Disadvantages of this method appears when each skill is performed for short time slices –a result of quick changes in decisions- or when there is a great change –unsmooth field for example- in the soccer field. Therefore deciding when to use these estimations can be critical and they can cause failures or inaccurate localization. So, to prevent these problems, we can avoid estimating quick actions (performed less than a specific amount of time) and to make robots learn the parameters of the skills when the condition is changed.
Estimation, in the case of great reliability, can be considered as a helpful component for localization and it is worthwhile working on motion estimations in the future.

8. Tools & Debugging

Running compiled code on AIBO is time-consuming and inefficient in many cases. Thus, lack of debugging software and simulator for testing and debugging purposes is felt and the need is obvious.
Therefore, Impossibles spent time on writing tools which let us debug the codes running on Aibo platform and simulate some geometrical codes without having robots available. These tools consist of AIBO Controller and AIBO Geometrical Simulator.

8.1 Aibo Controller

This software is divided into two units cooperating with each other. A *DEBUG* module which is written into memory stick (which can be accompanied by other soccer modules and run in parallel with them) and a client application which runs on the PC.
DEBUG module sends collected data to the client program. Data consist of all internal states of other soccer or non-soccer modules (provided for other challenges) in addition to other internal representations of the robot (including joint data, images, body sensor data, world states, perceptions, etc.)
On the other side, the client application receives the data and makes it possible to visualize and analyze them via different internal implemented algorithms and other user defined ones. This will give us opportunity to run vast of algorithms and methods in different fields (Image Processing, Motion, etc.) based on data collected.
On the other direction, since reaching to the state which revealed bugs in a nondeterministic and real environment is somehow impossible, setting parameters to return back to the desired state, can be found in our controller application.
Channel between *DEBUG* module and the client program is a wireless connection. Results from experimental statistical analysis of protocols in a wireless communication system led us to choose packet sizes and protocols in a way to achieve higher throughput.
To conclude, a list of features provided by *AIBO Controller* is followed:

- *Visualization* and *analysis* of data, especially intermediate representations of other modules running in the robot.
- Turning *on* or *off* different algorithms and parts of the code for debugging purposes.
- *Modification* of robot's state and parameters to algorithms.
- Run different algorithms solely on the collected data on the client side instead of the robot itself.
- Designing new blocking skills by reaching to desired and discrete states.

8.2 Aibo Geometrical Simulation

Writing codes to memory stick after each change and waiting for the robot to load modules, all occur for several times and all slow down the process of code development. Thus, a simulator was developed by *Impossibles* to simulate some special purpose codes on the PC instead of the Aibo itself.
One of the most time-consuming processes is the geometrical challenges of robot. Geometrical Simulator makes it possible to simulate motion's geometrical space on the client side instead of the robot itself. This will give us features to develop forward and inverse kinematics methods for any kind of robots, simulating and collecting the results of any kind of movements, and, designing in companion with testing the movement of robot's joints during blocking actions.

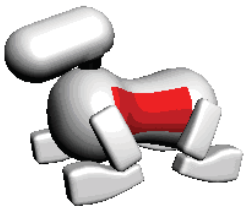


Fig. 22. Aibo model in Geometrical Simulator

This software is fed by the codes generating motion steps as inputs and the output will be the graphical movements of our Aibo model's legs and head. The Aibo model in our simulator is shown in Fig. 22.

8.3 Code Interpreter

Most of the codes in a soccer module are dependant on the condition in which the robots are playing. These include game strategies, player's roles and some complex decision making commands. Therefore in order to make the code flexible and prevent changes in the hardcode each time one of the above alters, converting hardcode into scripts which can be interpreted by the AIBO is an essence.

Scripts facilitate programming Aibos when accompanied by a high-level interpreter module, which runs inside the robots. When the fundamental parts of a soccer software is developed, the codes for decision making methods, determination of player's roles and game strategies are written in a simple scripting language and all are stored in memory stick as a raw text file. An interpreter module is in charge of translating the scripts into standard codes for Aibos and running them logically and consecutively.

These scripts use a high level interface provided by the main modules in our architecture, as mentioned earlier in this chapter, to fulfill the need of a potent access to hardcode. In addition to the interface of the main modules, our interpreter supports structural statements such as loops and conditional statements. There are some data structures provided for better manipulation of collected data from camera, joints and other sensors.

Players change roles dynamically, so putting them in hardcode would be bothersome. Some predefined functions in our interpreter make it feasible to distribute roles among players dynamically and prevent any sort of conflictions. This way even if the team strategy is changing dynamically in a specific game, responsibilities are assigned to players in a correct manner. Taking absence of players into account –when penalized- is another specification of those predefined functions.

Complex decisions can be implemented based on a State Diagram Machine (SDM) in our script. All decisions are first converted to states for simplicity, and then implemented by the features available in our scripts. SDM makes it possible to easily change decision steps and its contents.

9. Conclusion

In this chapter we presented the Impossibles main architecture and its modules to create a fully autonomous team of 4-legged robots for playing soccer. This architecture includes different modules such as World Model, Vision, Decision Making, Motion Controller, Communication, and Localization which are all independent of the robot platform. This chapter presented the integration of our researches in different fields which came together to create fully autonomous robots for specific purpose that is playing soccer as humans do. And this could be a primitive attempt to developing intelligent robots.

Participated in two years of RoboCup competitions in Bremen and Atlanta in RoboCup 2006 and 2007, the team gained some valuable experiences which led to designs of low cost algorithms. The most probable restriction in this soccer module is the limited resources such as CPU and memory, therefore developing optimized algorithms is the main target of team achievements.

10. References

Gonzalez, R. C. & Woods, R. E. (2001). *Digital Image Processing,* Prentice-Hall, 2nd edition.

Gray, A. (1997) Modern Differential Geometry of Curves and Surfaces with Mathematica, Boca Raton, CRC Press, 1997.

Fox, D. (1998). *Markov Localization: A Probabilistic Framework for Mobile Robot Localization and Navigation.* Institute of Computer Science III, University of Bonn, Germany, Doctoral Thesis, December 1998.

Habibi, J. & Vaezi Joze, H. R. (2005). A new Architecture for Multi Agent System in Rescue Simulation Environment. *the CSI Journal on Computer Science and Engineering,* Vol. 3, No. 2&4, pp. 1-7.

Habibi, J.; Vaezi Joze, H. R.; Aliari Zounoz, S.; Rahbar, S.; Valipour, M. & Fathi, A. (2006) Impossibles Aibo Four-Legged Team Description Paper RoboCup 2006, *RoboCup 2006*, Bremen, June 2006.

Habibi, J. & Nayeri P. (2006) Centralized vs. Non-Centralized Decision-Making in Multi-Agent Environments, *11th CSI computer Conference*, January, 2006.

Jain, R.; Kasturi, R. & Schunck, B. G. (1995). *Machine Vision*, McGraw-Hill.

Kohl, N. & Stone, S. (2004) Machine Learning for Fast Quadrupedal Locomotion, *In Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI)*, San Jose, CA, July 2004.

Kortenkamp, D.; Bonasso, R. P. & Murphy, R. (1998). *AI-based Mobile Robots: Case studies of successful robot systems*, MIT Press, Cambridge.

Lenser, S. & Veloso., M. (2000). Sensor resetting localization for poorly modelled mobile robots. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, San Francisco, 2000.

Locus, C.; Shahmirzadi, D. & Sheikholeslami, N. (2002), Introducing a Controller Based on Brain Emotional Learning Algorithm: BELBIC (Brain Emotional Learning Based Intelligent Controller), *International Journal of Intelligent Automation and Soft Computing (AutoSoft)*, USA, August 2002.

Mokhtarian, K.; Vaezi Joze, H. R. & Habibi, J. (2007). An Inexpensive Approach for Real-Time Vision on Four-legged Footballer Robots. *Proceedings of 12th International CSI Computer Conference*, pp. 1856-1861, February 2007.

Rubin, D.B.; DeGroot, K. M.; Lindley, D. V. & Smith, A. F. M. (1988). Using the SIR algorithm to simulate posterior distributions. In M.H. Bernardo, *Bayesian Statistics 3.* Oxford University Press, Oxford, UK.

Sridharan, M. & Stone, P. (2005) Real-Time Vision on a Mobile Robot Platform. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2148-2153, August 2005.

Vaezi Joze, H. R.; Habibi, J. & Rahbar S. (2007). Piecewise Linear Probability Distribution Localization: Fast and Inexpensive Approach for Mobile Robot Localization. *Proceedings of 4th TAROS Conference*, wells, September 2007.

Vaezi Joze, H. R.; Mokhtarian, K.; Asadi, N.; Kamali, A.; Zolghadr, N. & Kaffash, S. H. (2007). Impossibles Aibo Four-Legged Team Description Paper RoboCup 2007, *RoboCup 2007*, Atlanta, July 2007.

Robotic Soccer

Edited by Pedro Lima

ISBN 978-3-902613-21-9

Hard cover, 598 pages

Publisher I-Tech Education and Publishing

Published online 01, December, 2007

Published in print edition December, 2007

Many papers in the book concern advanced research on (multi-)robot subsystems, naturally motivated by the challenges posed by robot soccer, but certainly applicable to other domains: reasoning, multi-criteria decision-making, behavior and team coordination, cooperative perception, localization, mobility systems (namely omni-directional wheeled motion, as well as quadruped and biped locomotion, all strongly developed within RoboCup), and even a couple of papers on a topic apparently solved before Soccer Robotics - color segmentation - but for which several new algorithms were introduced since the mid-nineties by researchers on the field, to solve dynamic illumination and fast color segmentation problems, among others. This book is certainly a small sample of the research activity on Soccer Robotics going on around the globe as you read it, but it surely covers a good deal of what has been done in the field recently, and as such it works as a valuable source for researchers interested in the involved subjects, whether they are currently "soccer roboticists" or not.

InTech Europe

University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China

Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821