



Sharif University of Technology

Computer Science Department

Artificial Intelligence & Robotics Laboratory (AIRL)



Supervisor:

Jafar Habibi

Members:

Hamid Reza Vaezi Joze

Kianoosh Mokhtarian

Nima Asadi

Ali Kamali

Mahdi Safarnejad

Navid Zolghadr

Hossein Kaffash

<http://ce.sharif.edu/~impossibles>

Contents

1	Introduction.....	7
2	Architecture.....	8
3	World Model.....	11
4	Vision.....	13
4.1	Correction of Chromatic Distortion.....	14
4.2	Color Classification.....	16
4.3	Transforming a pixel in an image into a point in the space.....	18
4.4	Object Recognition.....	20
4.4.1	Blob Formation.....	20
4.4.2	Specified Objects Recognition.....	21
4.4.3	Recognition of Unspecified Objects.....	21
4.4.4	Line detection.....	22
4.5	Experimental Results.....	23
5	Localization.....	25
5.1	Markov Localization.....	25
5.2	Separate Probability Density Functions.....	26
5.3	Piecewise Linear Probabilistic Distribution Localization.....	28
5.4	PLPDL Application.....	30
6	Motion.....	31
6.1	Non-Blocking Skills.....	31
6.1.1	Walk.....	31
6.1.2	Instantaneous Rotation.....	33
6.1.3	Other special movement.....	33
6.2	Blocking Skills.....	34
6.3	Kinematics.....	34
6.3.1	Forward Kinematics.....	35
6.3.2	Inverse Kinematics.....	35
6.4	Head Movement.....	36
7	Decision Making.....	38
7.1	Architecture.....	38
7.2	Determining overall team strategy.....	39
7.3	Team Strategy Database.....	40
7.4	Individual Behaviors.....	40
8	Communication.....	42

- 8.1 Connection status42
- 8.2 Dead Connection Recognition42
- 9 Tools44
 - 9.1 AIBO Controller44
 - 9.1.1 DEBUG Module44
 - 9.1.2 Client Application45
 - 9.1.3 TCP vs. UDP.....46
 - 9.1.4 Conclusion46
 - 9.2 AIBO Simulator46
 - 9.2.1 Controller Unit48
 - 9.2.2 Kernels and Subsystems49
 - 9.2.2.1 Simulator Kernel50
 - 9.2.2.2 Vision Subsystem.....51
 - 9.2.2.3 Communication Subsystem51
 - 9.2.2.4 Motion & Localization Subsystem51
 - 9.2.3 Graphical User Interface52
 - 9.2.4 Future Works on Simulator.....52
- 10 References.....54

Table of Figures

Figure 2-1: <i>Impossibles</i> World Model Based Architecture (WMBA).....	9
Figure 2-2: Definition of <i>AbstractPlayer</i> class	10
Figure 3-1: Object diagrams of player's world model	11
Figure 4-1: architecture of real-time vision system of robot	14
Figure 4-2: (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).....	14
Figure 4-3: (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	15
Figure 4-4: Two samples images taken by the robot's camera, before ((a) and (b)) and after ((c) and (d)) correction of chromatic distortion.....	16
Figure 4-5: Color classes in the three-dimensional space HSL	18
Figure 4-6: (a) Image taken by the robot's camera. (b) The result of color classification of (a).....	18
Figure 4-7: coordinates axes in the image and in the actual space	18
Figure 4-8: (a) A color-classified image. (b) Relevant blobs	20
Figure 4-9: Stages of recognition of an unspecified object.	22
Figure 4-10: (a) The original image. (b) Detecting lines of Figure 10-a after size reduction (this figure is depicted at the doubled size).	23
Figure 5-1: A sample piecewise linear probability density function	28
Figure 5-2: An example of Movement Update process.	29
Figure 5-3: Sensor Update example. (a) Previous density function (b) sensor data (c) result by $p=.7$	29
Figure 5-4: Self-Localization Flowchart.....	30
Figure 6-1: Coordination system which is used in walk.....	32
Figure 6-2: Goalie block the ball.	34
Figure 6-3: Aibo ready to kick.....	34
Figure 6-4: Reference Frame of the Aibo Robot	35
Figure 6-5: Inverse Kinematics function of Aibo Legs	36
Figure 6-6: Determining head joint to look at (x,y).....	37
Figure 7-1: architecture of DM module	38
Figure 7-2: Team strategy fuzzy rules	39
Figure 9-1: AIBO Controller User Interface.....	45
Figure 9-2: AIBO Simulator snapshot	47
Figure 9-3: Structure of AIBO Simulator	47

Figure 9-4: Controller Unit structure	48
Figure 9-5: Kernel & Subsystems structure.....	49
Figure 9-6: Internal view of Kernel	50
Figure 9-7: Aibo model in our simulator	50
Figure 9-8: Aibo simulation in another environment	52
Figure 9-9: Simulator's Structure in the future	53

1 Introduction

As a research group, *Impossibles* team has been set up in Artificial Intelligence and Robotics Laboratory (AIRL) of Computer Science and Engineering Department at Sharif University of Technology since March 2004. Research Areas of *Impossibles* were categorized into three groups including Artificial Intelligence (Machine Learning, Multi-Agent Systems, and Reasoning), Theoretical Computer Science (Algorithms, and Data Structures), Soft Computing (Fuzzy Theory, and Genetic Algorithms).

RoboCup's interesting features attracted us to begin implementation of our ideas in Rescue Simulation Environment (RSE) to participate in RoboCup2005 in Osaka. So it was our first participation in such international competitions. Having coded from scratch, we applied our new ideas. Consequently, *Impossibles* got world championship in Rescue Simulation League in Osaka 2005.

As Second RoboCup experience we start working on 4-legged Soccer which have more serious robotic and multi agent environment. Participating in RoboCup2006 in Bremen in the first year of working on Sony Dogs (Aibo), the result is not as we want and we dropped in the first round. However defeat is a bridge to win and we learnt many things from Bremen. The most important one is the necessity of out of robot software to adjust parameters of algorithms in new environment and also developing suitable debug routine to reduce debug time.

In order to overcome these problems we start developing two softwares: (i) *Aibo Controller* which enables us to do most of the process in a computer which connects to the Aibo by wireless connection. These processes could be Color Segmentation, Object Detection and Localization, (ii) *Aibo Simulator* which is a high level simulator to test strategies and high-level decision making.

In this report we are going to discuss about different modules of our 4legged Soccer software for Aibo robots which was developed using Open-R SDK. This contains our approach for team which participated in RoboCup 2006 in Bremen and our new scientific approach for RoboCup 2007 competition which are either implemented or ongoing. Some of our scientific results have been published in [10], [11], [12].

2 Architecture

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

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

These subsystems are run repeatedly with different frequencies. They are also 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 from previous architecture is in charge of gathering appropriate information from other subsystems to communicate Aibo Controller software which will be described in Section **Error! Reference source not found.** in debug mode of robots.

Figure 2-1 demonstrates the World Model Based Architecture. As in DFD diagrams, subsystems are denoted by dotted rectangles, data flow is shown as arrows, and processes are shown via circles.

Sensing subsystem is responsible for perception via vision and other sensors. Additionally, communication subsystem is employed to transmit information among Aibo robots. Furthermore, action subsystem is in charge of determining what the Aibo robots decide and perform. Decision Making (DM) is responsible for high level decision makings, whereas in Motion Controller (MC) low level skills are implemented. Last of all, Localization is considered as an input gate to World Model (WM). Localization's main task is updating World Model (WM) using the data received from the adjacent subsystems, i.e. Motion Controller (MC), World Model (WM), Communication, and Vision. These subsystems will be described in the following of this report.

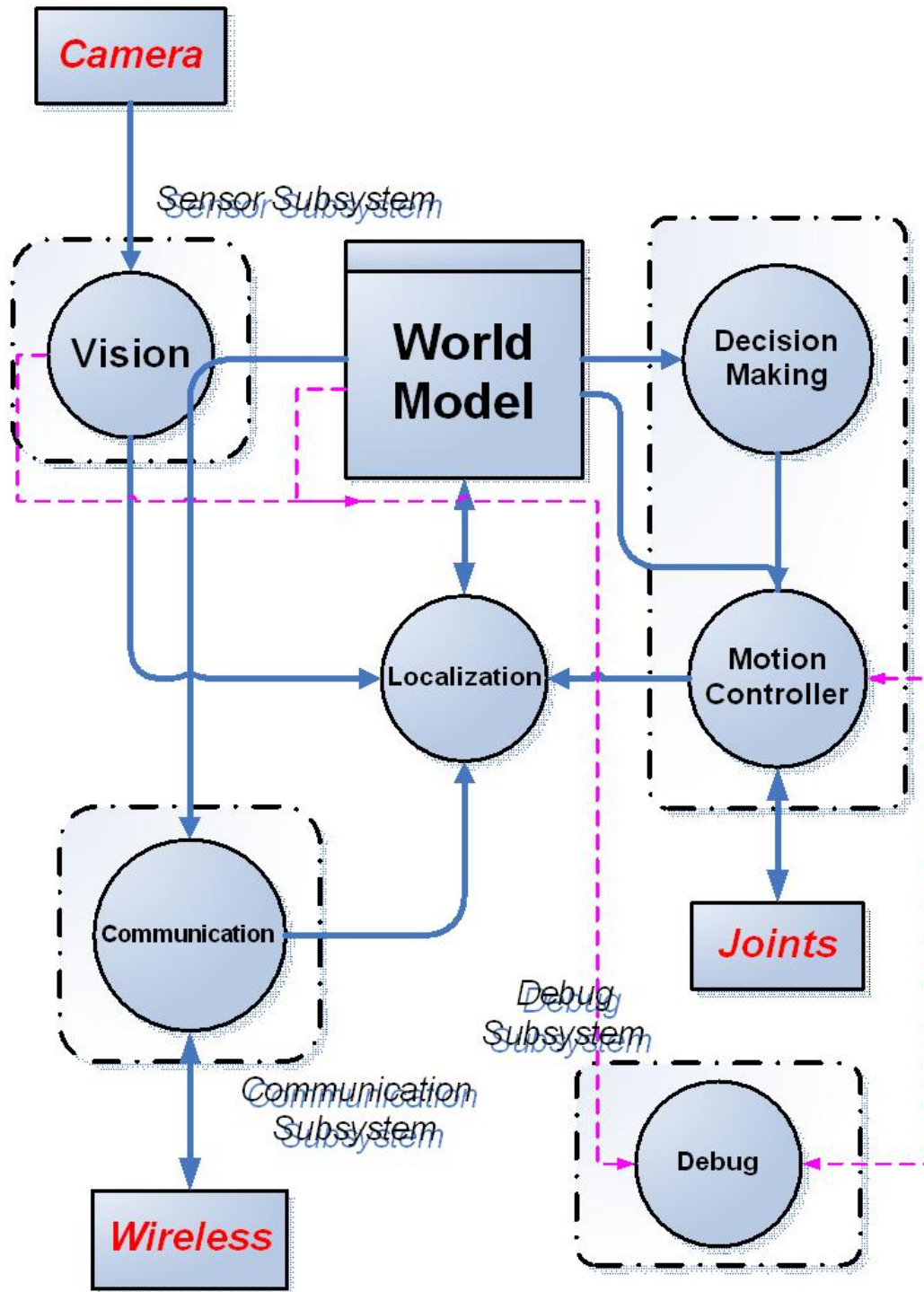


Figure 2-1: *Impossible* World Model Based Architecture (WMBA)

In order to make high level program which means *Decision Making* separate from other part, we use a non Open-R abstract class called *AbstractPlayer*. The definition of *AbstractPlayer* class illustrate in Figure 2-2.

```

class AbstractPlayer {
public:
    AbstractPlayer();
    virtual void decisionMaking() = 0 ;
    virtual void sense(ObjData *input ) = 0 ;

Protected:
    WorldModel *world;

    /*
        Definition of low level motion action
    */
public:
    void setHeadValues();
    void setLegsValues();
}

```

Figure 2-2: Definition of *AbstractPlayer* class

In this class there are two pure virtual functions that should be implemented for a Player. The first one is *sense* which is called every time robot gets some information. These could be vision or sensory data or information came from others via Communications subsystem. Using these data which was coded as *ObjData* robot should update its world model which is a member variable.

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 action such as walk, shoot, look or other action. There are some emergency actions such as standing back to normal position in the case of falling down which is determined automatically so *decisionMaking* is not called in this situations. And also some actions that we called them Blocking Skills will be done completely so *decisionMaking* is not called during these actions (we will discuss about them in more detail in motion section).

3 World Model

In a real world robotics environment such as Aibo 4-legged league, agents have to have interactions with several physical objects, e.g. the orange ball. This interaction is typically implemented as a perception-action loop. Aibo Robots are equipped with sensors that perceive physical characteristics of the environment and they use these percepts to build an internal representation of the environment, i.e. World Model (WM). Once this world model is built, it is possible for agents to exploit in order to accomplish the tasks responsible for producing the required actions to be done by the agent, Aibo robot.

Figure 3-1 illustrate the base hierarchy diagram for objects which are in the 4-legged soccer environment. All objects contain two variables which represents the exact point position of that object. As whole objects are attached to the soccer fields it is enough to consider 2-dimentional points for this purpose. For moving objects the velocity vector of moving object is also needed and for Aibos direction of robot compounding with other variables could determine the position of robot in the soccer field. The robot has bunch of other information for itself which contains inner data and its state. But it has some of this information for its mates (Own Player) which was gained via communication.

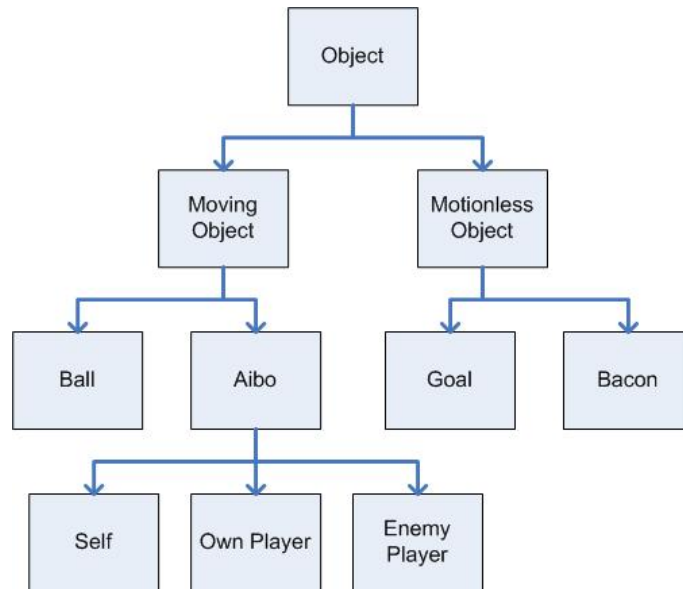


Figure 3-1: Object diagrams of player's world model

Objects could be represented in tow different frame: relative to the robot frame that we called it relative and relative to the reference frame (soccer field frame) that we called it absolute. Although using relative frame and matrix transform from relative frame to reference frame, reference frame could be calculated, we keep the position of each objects in both frame because of uncertainty in the converting matrix

transform (which is equal the position of robot in the soccer field). So in some cases such as ball it is reasonable to use relative frame and in others such as position of other robot gaining from communication using absolute objects seems to be better. Table 1 shows the number of all objects that stored in the world model.

Table 1: table of number of the objects in the world model

Object Type	Relative	Absolute
Ball	1	1
Goal	2	1 (const)
Bacon	2	2 (const)
Self	-	1
Own Player	3	3
Enemy Player	4	4

As a real world environment, Aibo robots receive uncertain information of their surroundings via their sensors, camera and other robots. And also passing time could make the received information uncertain. In order to keep uncertain data, we exploit an extra parameter for each variable of objects which represents the reliability of that variable. Therefore, the above explained world model was designed to support the concept of vagueness. This could be also exploited by our previous fuzzy world model.

4 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. Therefore, since the invention of digital computers, scientific efforts have begun providing visioning capabilities for machines. Many studies have been conducted in this regard aiming at processing raw data from vision sensors (camera) in order to extract useful information.

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. For further information on these robots and their specifications refer to [17].

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 of 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 (green, white and et. al) or to a class allocated to unknown color to specify existing color areas in the image.

Then, existing blobs of each color (connected component of image's point with the same 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.

Figure 4-1 display the architecture of robot's real-time vision system which is based on our research in [11]. Also this system require offline processing regulating some provided algorithms' parameters for various setting prior to the application of software on the robot.

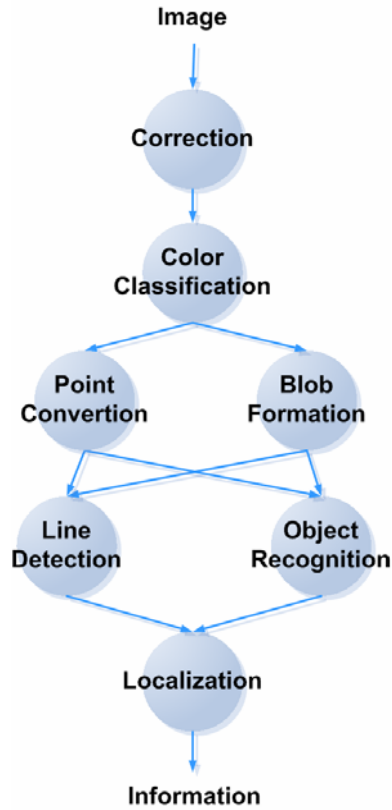


Figure 4-1: architecture of real-time vision system of robot

4.1 Correction of Chromatic Distortion

Aibo robot's camera causes a considerable chromatic distortion in the color of pixels at the corners of images. Figure 4-2(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 Figure 4-2(b) in terms of pixels' distance from image's center (r).

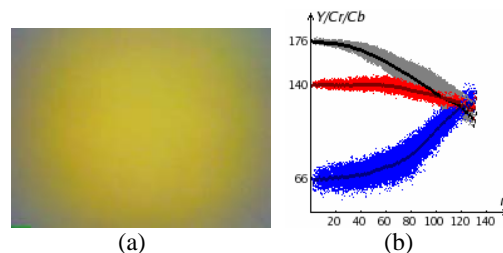


Figure 4-2: (a) Image taken from a uniformly colored yellow page. (b) Values of color components of Figure 2-a's pixels in terms of distance from the center (Y , Cr , and Cb are shown by gray, red, and blue colors respectively).

The thick and fading curves of Figure 4-3(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 Figure 4-3(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 $\gamma \times r^2$ [18]. These curves are shown in Figure 4-3(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 Figure 4-3(b) (the *actual value* of a Cr curve is obtained by averaging the values of $Cr_{observed}$ for pixels up to 10 points far from the center of the corresponding image).

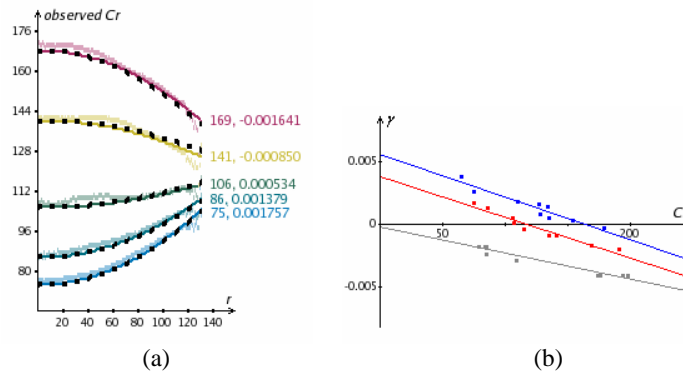


Figure 4-3: (a) Variations of the component Cr in terms of r in a number of uniformly colored pages. (b) γ values in terms of the actual value of Cr .

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 [19]).

Curves approximating $Cr_{observed} - Cr_{actual}$ using the new coefficient (γ' which is in turn obtained by a linear approximation from Cr_{actual}) are displayed in Figure 4-3(a) as thick dotted curves. The maximum error of this two-level approximation in our experiments on uniformly colored images –Figure 4-3(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 4-1 holds for each pixel of an image.

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

$$\Rightarrow Cr_{actual} = \frac{Cr_{observed} - \beta \times r^2}{\alpha \times r^2 + 1} \quad (4-1)$$

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 4-1. Similarly, the actual values of two other components of pixels' colors are obtained and thus the chromatic distortion of the image is corrected. Figure 4-4(a) and (b) depict two sample images captured by Aibo robot's camera and Figure 4-4(c) and (d) depict the result of their correction.

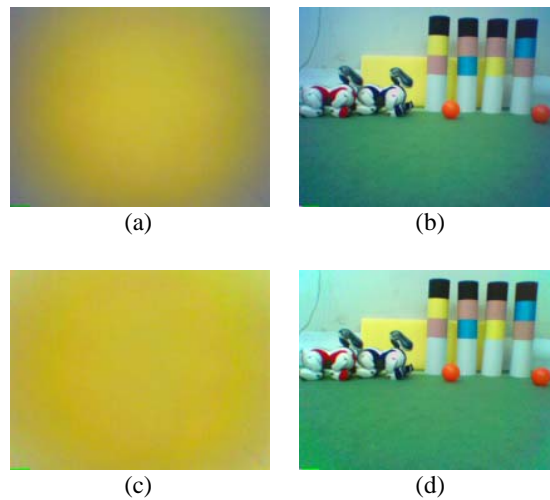


Figure 4-4: Two samples images taken by the robot's camera, before ((a) and (b)) and after ((c) and (d)) correction of chromatic distortion

For speeding up the process of chromatic distortion correction of images in real-time visioning, we use 4 pre-calculated tables (like [18]). The first one is a matrix of size 208×160 (resolution of camera's images), whose cells indicate the distance of pixel (x, y) from the image's center. Three other tables are 131×256 matrices – 131 is the maximum distance of a pixel in the image from its center, and 256 is the number of possible values for a color component – representing the actual values of color components for a pixel with indexes “the observed value of component” and “pixel's distance from the center”. Thus the actual color of each pixel is obtained by four lookups and without conducting time-consuming calculations. Experimental results of the time taken by this correction are presented in Section 4.5.

4.2 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 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 [3]. Moreover, we consider an additional color class named *unknown* for colors similar to more than one of our specified color classes.

Due to the limited memory of Aibo robots, we prepare our color classification table in the size of $128 \times 128 \times 128$, whose cells are representative of cubes of width 2 in the YCrCb color space. 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 color 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 $h: \{c_i | 1 \leq i \leq n\} \rightarrow R$. The value of this function for a (h, s, l) point is calculated using Equation 4-2.

$$h(c_i) = \frac{w_i}{d((h, s, l), (h_i, s_i, l_i))} \quad (4-2)$$

In Equation 4-2, 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 (in the learning phase) color-classified images) using manual tuning tools or intelligent searching algorithms, while the latter is not applicable for our purpose due to the large number of samples and thus the high cost of calculating that difference.

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. Figure 4-5 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.

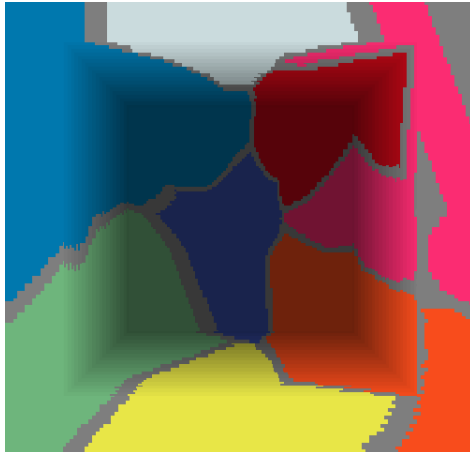


Figure 4-5: Color classes in the three-dimensional space HSL

A sample of color classification (after correction of chromatic distortion) on the image shown in Figure 4-6(a) is depicted in Figure 4-6(b). Gray pixels in the color classified image have unknown color.

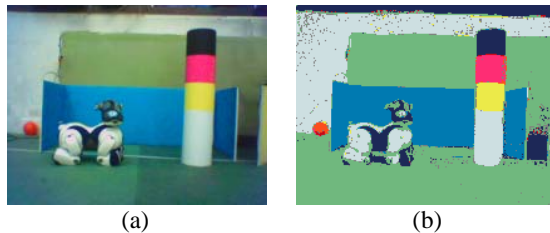


Figure 4-6: (a) Image taken by the robot's camera. (b) The result of color classification of (a)

4.3 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 Figure 4-7(a) for facilitating the calculations ($(0,0)$ is the center of the image). 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 Figure 4-7(a) (the floor is assumed the plane $z = 0$).

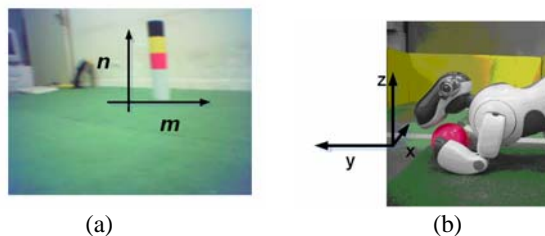


Figure 4-7: coordinates axes in the image and in the actual space

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} : Camera's horizontal angle of view.
- ψ_{ver} : Camera's vertical angle of view.
- $width_{image}$: Image's width.
- $height_{image}$: Image's 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 4-3 holds.

$$x / y = 2m / width_{image} \times \tan(\psi_{hor} / 2) \quad (4-3)$$

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

Therefore, the point is located on the plane represented by Equation 4-4. 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 4-5.

$$height_{image} \times z + 2n \times \tan(\psi_{ver} / 2) \times y = 0 \quad (4-5)$$

The resulted line of crossing these two planes is a locus whose all points are seen at the pixel (m,n) in the image (note that (0,0) is the center of 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 4-6 and 4-7.

$$\theta_x = a \tan(\tan(\psi_{hor} / 2) \times \frac{m}{width_{image}})$$

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

$$x = -\frac{\cos(\theta_x)\sin(\theta_y)\sin(\gamma) + \cos(\theta_y)\sin(\theta_x)\cos(\gamma)}{\cos(\theta_y)\cos(\theta_x)\sin(\gamma) - \cos(\theta_x)\cos(\theta_y)\cos(\gamma)} \times h_c \quad (4-6)$$

$$y = \frac{\cos(\theta_y)\sin(\gamma) \times x + \cos(\theta_y)\cos(\gamma)}{\sin(\theta_y)} \times h_c \quad (4-7)$$

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.

4.4 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.

4.4.1 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. Figure 4-8 shows a color-classified image and its relevant blobs.



Figure 4-8: (a) A color-classified image. (b) Relevant blobs

4.4.2 Specified Objects 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 (the most big and dense, and on the green floor if fully observed) 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 presented in Section 4.3, the ball's actual (x, y) relative to the robot can be calculated using Equations 4-10 and 4-11.

$$x = \frac{-\text{sign}(\gamma) \times (n_c - \frac{m_c}{\tan(\gamma)})}{\sqrt{1 + \frac{1}{\tan(\gamma)^2}}} \times \frac{R_{Ball}}{R} \quad (4-10)$$

$$y = (\frac{33}{2R} \times 50 + 8.14) \times \cos(\beta) \quad (4-11)$$

4.4.3 Recognition of Unspecified 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 4.3 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 Figure 4-9.

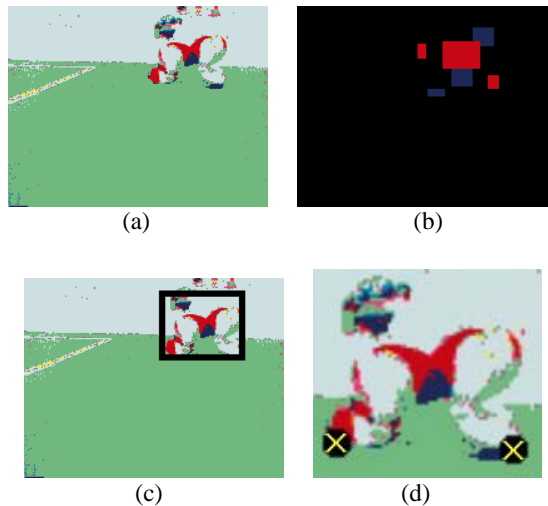


Figure 4-9: Stages of recognition of an unspecified object.

The color-classified image and its relevant blobs are depicted in Figure 4-9(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 Figure 4-9(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 Figure 4-9(d).

4.4.4 Line detection

Field lines are beneficial sources of information for gathering information about robot's location. Regarding the low computation power and limited memory of Aibo robots, we use a fast algorithm for detecting field lines.

We first reduce the color-classified image's size to half of its original size. Each square of side 2 in the original image is viewed as a single pixel whose color is the same as the dominant color of that four-point square (In the case that there was not a dominant color, the representative pixel's color is determined depending on those four pixels' colors. Here, we avoid going into the details). This reduction results in less computations as well as elimination of many noises.

After that, the search for lines begins by scanning 6 columns (at distances of 20 pixels along the reduced image's width) and 5 rows (at distances of 19 pixels along the reduced image's height), so that we look for white pixels in these columns/rows around which green ones exist. Having found such pixel, by a Breadth First Search [20] we traverse white pixels which are surrounded by green pixels (there should exist a green pixel at a bounded distance in six out of eight directions around the white pixel), and their distance to the least squares line of pixels traversed previously is lower than a certain value. By doing this, white pixels within and at the edges of lines are found and green-white edges not belonging to field lines (such as the line of horizon) are ignored, and also the search is stopped by reaching a line break point (i.e. where two half-lines meet). Pixels traversed in one search are not traversed in subsequent searches (begun from other columns/rows).

After the search, if the number of traversed pixels is not less than a certain value, the least squares line of pixels, along with its two endpoints (that are used in detection of two lines intersection type), is recognized as the detected line. Considering a minimum for the number of pixels is for preventing short white lines (such as the edge of a robot's leg) from being confounded with field lines. Figure 4-10 depicts the result of image size reduction, rows and columns to be scanned, and traversal of pixels at lines' edges. Traversed pixels at edges of lines are shown by black points, detected lines by violet lines, and two endpoints and the middle of lines by big brown points.

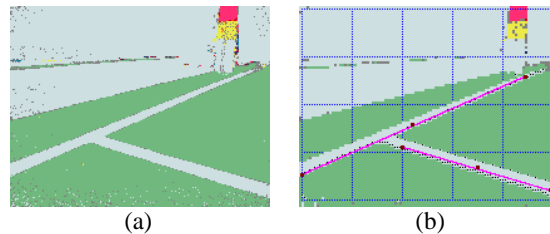


Figure 4-10: (a) The original image. (b) Detecting lines of Figure 10-a after size reduction (this figure is depicted at the doubled size).

After these searches, we obtain a number of lines, where a group of them might belong to just one actual line which is not continuously (in one search) traversed, such as far lines which are thin and are seen interrupted (i.e. discontinuous) in some points because of image size reduction. Therefore, we consider two detected lines with almost same slope and width from the origin a single line. In our experiments in detecting lines as described above, lines within a distance of about 2 meters from the robot were well detected.

Having detected seen lines, we acquire the actual coordinates of their endpoints by the method described in Section 4.3, calculate these lines' equations (from the robot's view), and use them for robot localization. In the case that just one line is seen, the only information obtained is the distance and the angle (i.e. direction) of the robot relative to the line. If the estimation of the robot's location helps us clearly know that which line of the field is seen, that information makes the location estimation more accurate. However, if two lines and their intersection point is seen, given their two endpoints, it is known that their intersection was in the form of T or L (there is no intersection in the form of +) in the field, then considering the point observer of these forms and the current estimation of robot's direction (with an inaccuracy of $\pm 45^\circ$), the accurate location and direction of the robot are found (i.e. the robot is accurately localized).

4.5 Experimental Results

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 2 displays the running time of our visioning stages (accuracies are noted in relevant subsections). According to methods described in Sections 3.1 and 3.2, 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. In the row corresponding to the process of line detection, the running time is calculated for the image depicted in Figure 4-10 which includes two long lines. Objects recognition consists of a few calculations and its running time is negligible in comparison with other visioning algorithms.

Table 2: Running time of different process in Impossible Vision subsystem

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

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.).

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[4]. 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 [6] or sample-based approaches such as Monte-Carlo [5] or Condensation. 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 [7] require the computation of a significant number of samples to support high-performance especially for large areas. However these approaches seem to be good for high-performance or high-memory machines and recently several researches have developed these approaches on different platforms such as Museum Tour-Guide Robots. These kinds of robots have sufficient time to perform their algorithms since there is no necessity for them to interact in real-time situations such as soccer robots.

We use a localization algorithm to localize soccer player robots in the field which is called piecewise Linear Probability Distribution Localization (PLPDL) [10] 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 [8] which localize robot probabilistically. Our approach inherits from Markov localization the ability to localize a robot under global uncertainty. The main idea of this approach is separating probability density functions of random variables which determine robot position. 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 4-legged soccer robots.

5.1 Markov Localization

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 . Initially, $Bel(l)$ reflects the initial state of knowledge: if robot knows its initial position, $Bel(l)$ is centered on the correct position; if the robot does not know its initial position, $Bel(l)$ is uniformly distributed to reflect the global uncertainty. As the robot operates, $Bel(l)$ is incrementally refined.

Markov localization applied two different probabilistic models to update $Bel(l)$, an action model to incorporate movement of the robot into $Bel(l)$ and a perception model to update the belief upon sensory input:

- **Robot motion:** Motions are modeled by conditional probability $P(l|l',a)$, specifying the probability that a measured movement action a , when executed at l' , carries the robot to l . $Bel(l)$ is then updated according to the following general formula, commonly used in Markov chain:

$$Bel(l) \leftarrow \int P(l|l',a)Bel(l')dl' \quad (5-1)$$

The term $P(l|l',a)$ represents the robot's kinematics, whose probabilistic component accounts for errors in odometers.

- **Sensor readings:** Sensor data are integrated with Bays rule. Let s denote a sensor reading and $P(s|l)$ the likelihood of perceiving s given that the robot is at position l , then $Bel(l)$ is update according to the following rule:

$$Bel(l) \leftarrow \alpha P(s|l)Bel(l) \quad (5-2)$$

Here α is used to normalize $Bel(l)$ to insure that $Bel(l)$ integrates to 1.

5.2 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:

$$\begin{aligned} Bel(l_0 = (x_0, y_0, \theta_0)) &= P(x = x_0 \wedge y = y_0 \wedge \theta = \theta_0) \\ &= P(x = x_0)P(y = y_0)P(\theta = \theta_0) \end{aligned} \quad (5-3)$$

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)

$$P(a < x < b) = \int_a^b f_x(x)dx \quad (5-4)$$

Equation 5-4 concludes $f_x(x)$ is non-negative function $\int_{-\infty}^{+\infty} f_x(x)dx = 1$.

Using Equation 5-3 and definition of density function for coordinate random variables we have:

$$\begin{aligned}
& Bel(l_1 = (x_1, y_1, \theta_1) < l < l_2 = (x_2, y_2, \theta_2)) = \\
& P(x_1 < x < x_2 \wedge y_1 < y < y_2 \wedge \theta_1 < \theta < \theta_2) = \\
& P(x_1 < x < x_2)P(y_1 < y < y_2)P(\theta_1 < \theta < \theta_2) = \\
& \int_{x_1}^{x_2} f_x(x)dx \int_{y_1}^{y_2} f_y(y)dy \int_{\theta_1}^{\theta_2} f_\theta(\theta)d\theta
\end{aligned} \tag{5-5}$$

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 Localization [8], 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. $(\Delta x, \Delta y, \Delta \theta)$. Like Markov Localization 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 (It could be better since each non-vision sensor normally supports one parameter of localization). 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:

$$Fx_{New} = Fx_{old} \times (1 - p) + Fx_{Sensor} \times p \tag{5-6}$$

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" [9] 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 standard deviation or something)

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.

5.3 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 Figure 5-1, could be determined by set of following points:

{ (0,0) , (1,.5) , (2,.5) , (3,0) }

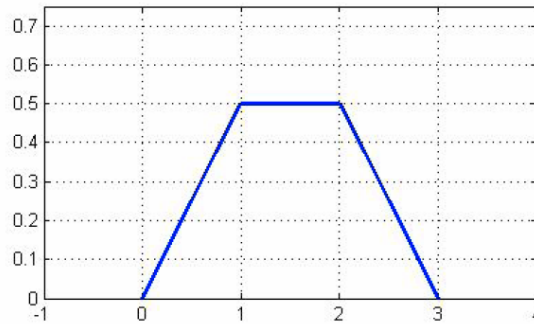


Figure 5-1: A sample piecewise linear probability density function

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 parand-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 Figure 5-2(a) is probability density function of a random variable before movement update. Figure 5-2(b) shows probability

density of movement and Figure 5-2(c) is the final result of movement update. Figure 5-2(d) shows linear approximation of result via PLDL algorithm.

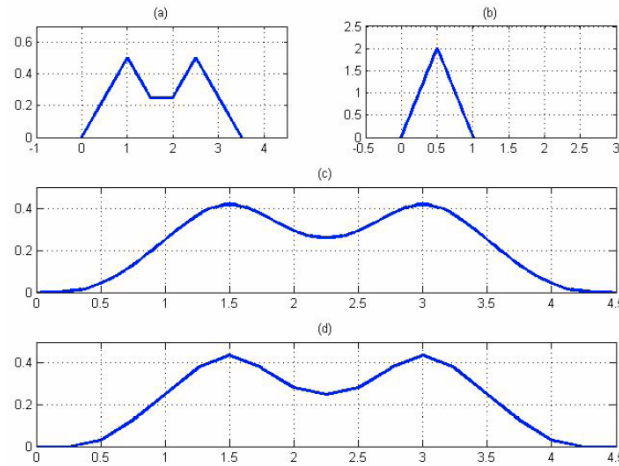


Figure 5-2: An example of Movement Update process.

- 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 Figure 5-3(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. Figure 5-3(c) illustrates result of Sensor Update with $p=0.7$. Figure 5-3(a) is previous density function for a distinct variable and Figure 5-3(b) is sensor data of that distinct variable.

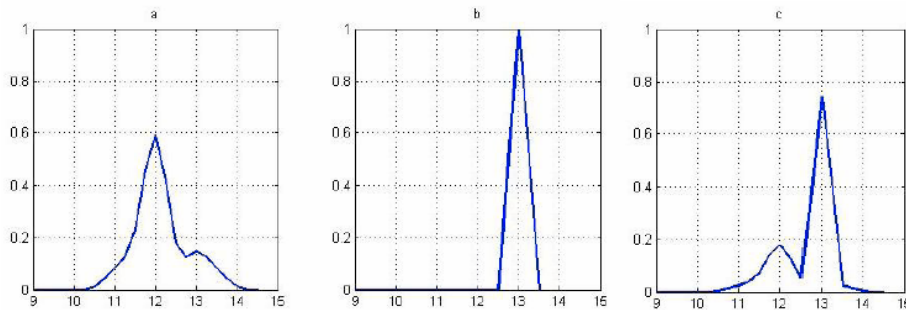


Figure 5-3: Sensor Update example. (a) Previous density function (b) sensor data (c) result by $p=0.7$

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 that express density function so we could control the computational time needed for PLPDL.

PLPDL seems to be great that is compound of Markov Localization and Sensor Resetting Localization with probability theory in the other hand it is compatible with

Probabilistic World Model and can be used real-time. In the next section we are going to discuss about out coming data to our method.

5.4 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. Figure 5-4 demonstrates Self-Localization flowchart using PLPDL method. We explained sensor data which is supplied by vision in Section 4 and movement data will discussed in Section 6. 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.

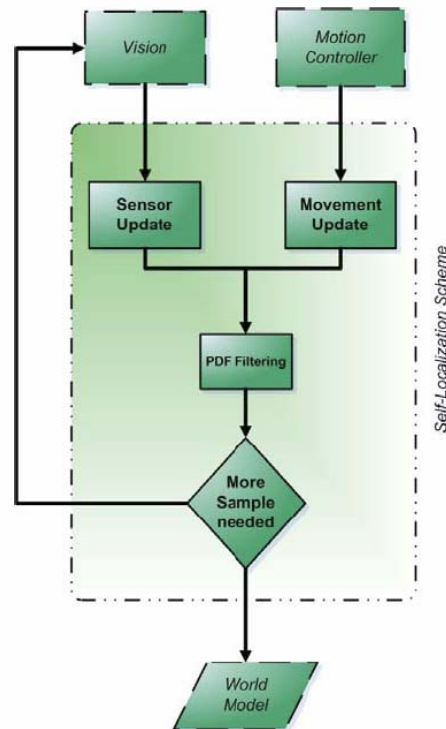


Figure 5-4: Self-Localization Flowchart

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 Motion

We divide usage of motion skills of Aibo robot in two categories: 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 halt the previous action and start performing new commands. Walk and rotate are included in non-blocking skills. On the other hands Blocking Skills are smallest consecutive segment motions which are not needed to be interrupted such as different kind of shoot and blocking ball by goalie.

6.1 Non-Blocking Skills

As mentioned before, skills are categorized into two classes: Blocking skills and non-blockings. 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 halt the previous action and start performing new commands.

Based on above description of a non-blocking skill, all high level commands produced by DM which include an amount of uncertainty are considered so and done in a non-blocking series of actions. For example, preparing the ball for a Chest Shoot is basically a nondeterministic action (i.e. the robot is not sure about the result of ball preparation, since the ball can behave unexpectedly. But Chest Shooting skill is sure about ball's position, thus performs the action undoubtedly in a short amount of time) and therefore this preparation is done in a non-blocking mode.

On the other hand, all non-blocking commands can not have static motional patterns retrieved by a look up table mechanism. All the parameters are dynamic and therefore all calculations are done online. The following sections give brief overview of techniques done in non-blocking modes.

6.1.1 Walk

Walking is really important for robots in a soccer team such that the result of the team is highly dependant on walking styles. Therefore our walking skills have undergone major changes in comparison to last year [#Impossibles 2005]. Better experimental results were experienced by the use of a more technical walk modeling.

We modeled the walk of a robot as movement of Aibo's paws on the perimeter of an ellipse. In this model, movement of the left front leg is synchronized with movement of the right rear leg. This is true for the right front leg and the left rear leg too. This resembles natural behavior of animals in their normal movements.

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
- Semi-minor axis of the ellipse, with the symbolic name of b
- Coordination (x_0, y_0, z_0) of the center of the ellipse. (Coordination system is shown for one of the legs in Figure 6-1)

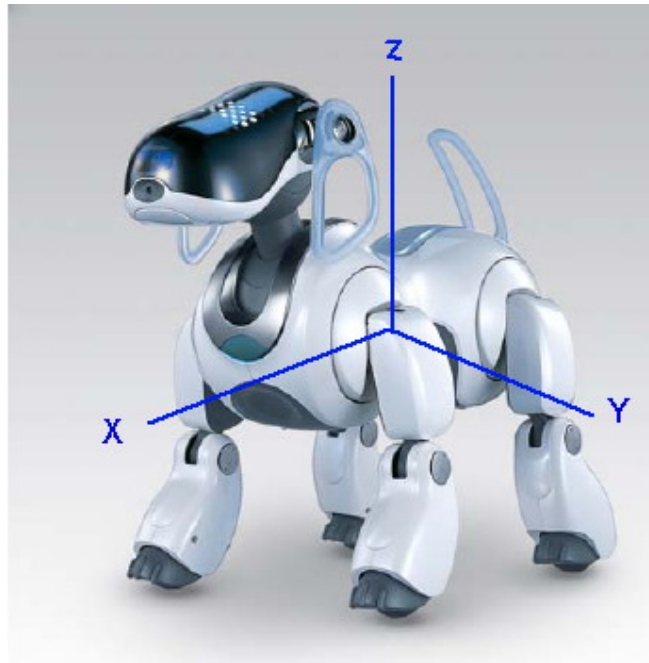


Figure 6-1: Coordination system which is used in walk

Having such parameters leads Walk to form the desired ellipse uniquely for each leg. Methods for setting such parameters are described later in this chapter.

In addition to these parameters for ellipse construction, we need some high level parameters got from the behavior layer. These parameters are somehow the same as those used in our previous motion controller module and are listed below:

- Walking Direction (Θ angle)
- Type of walking (Based on some predefined types of walking)
- Walking speed (if the speed is more than available, robot walks with its maximum available speed)
- If the walking is forward or backward

Types of walking determine how the robot should move to reach to a specific destination, which considers the quality of the walk only (such as Fast Stable Walk, Fastest Available Walk, etc.). This includes the same types as the previous Motion Controller module supported. But the implementations are different from the previous ones.

In order to have high performance walks, we have to use the best values for ellipse's parameters. This is done via the use of Policy Gradient Reinforcement Learning [16]. Thus, we had to convert the continuous space of Walking Directions (Θ) to a discrete set of angles and make the Aibo learn in these conditions. This way parameters a , b and (x_0, y_0, z_0) of the ellipse will be learned.

The process of learning such skills is really time-consuming, since the number of samples the robot should be trained in is dependant on the number of modes the robot can move (backward and forward movement), types of movement and the walking direction Θ . We assumed we always want to reach to the destination with the fastest possible speed based on the quality of the walk, i.e. the type of the walk. If a situation occurs in which the speed must be slower, we simply decrease the power of the walk and assume that the values for ellipse's parameters will not change.

6.1.2 Instantaneous Rotation

Modeling the walk of the robots to an ellipse can be generalized to other non-blocking movements. One of the most important motions of the robot which obviously affects the robot's efficiency is rotation around a rotational axis.

Instantaneous Rotations of the robot can be described simply by the use of ellipse's parameters. In this scenario, an Instantaneous Rotational Center (IRC) is assumed and all other legs should move in especial directions to perform the action of rotation. This idea was based on the rotation of systems with 4 wheels in which the wheel which is located near the center of the rotation usually stops moving and others try to rotate in directions perpendicular to circles around the IRC. By simulating such idea with robot legs, we can gain to a good speed in rotating around a fixed axis.

Rotational speed of the Aibo is increased in such technique. This method is also useful for rotating in other situations such as rotating while holding the ball under the chin. In such conditions, the parameters for ellipse can be also learned. As a result, the model of forward and backward movement has been generalized to rotational motions.

6.1.3 Other special movement

For the defenders, or maybe other teammates, there might be situations in which the robot wants to move across a line perpendicular to its leg's directions. This is the best choice when the opponent should be in Aibo's visiting scope while attacking and the robot can reach a better state in that case.

Such movements can also be modeled using the ellipse's method. We can consider such movements as predefined types of motions and mix them with forward and backward walk.

As a conclusion, generalized modeling of robot's walk helped us so much in designing different walk styles.

6.2 Blocking Skills

Blocking Skills are smallest consecutive segment motions which are not needed to be interrupted. So Decision Making should use them in suitable situation. These actions contain consecutive value of joints which can be utilized from a static look up table that is not change during the game. We use our structure to store these data in files, therefore we have separate file for each blocking skill. These actions contain all type of shoots, block by goalie and stopping the ball.

Figure 6-2 demonstrate one of these actions called block. Figure 6-3 shows the situation that robot is ready to kick the ball and now Decision making could apply one of the shoots.



Figure 6-2: Goalie block the ball.



Figure 6-3: Aibo ready to kick.

These skills are developed using *AIBO Controller* tool which will be described in Section 9 and the result will stored in files for in game usage.

6.3 Kinematics

There will be two kind of problem solving while planning to move from one point or state of the robot to the next: First problem is when we have decided on joint angles and want to compute the position of leg or head relative to the body of robot which is called forward kinematics and the second problem is when we have the position of an effector and want to compute the angle of joints which is called inverse kinematics [15].

The former is needed to compute the exact position of robots part such as position and direction of camera which is critical to calculate Pan, Tilt and Roll of the camera which are used in Vision subsystem. The latter is what we need to determine the value of each joint after considering the trajectory of motion for legs or head.

6.3.1 Forward Kinematics

For each joint we define a three dimensional frame. Each frame is represented using a “4*4” matrix containing the orientation and position of the frame relative to a reference frame. We have defined a point on the robot as reference frame and each frame is computed relative to that frame. This frame's X axis is in direction of the body of the robot and the Y and Z axes are as shown in the following figure.

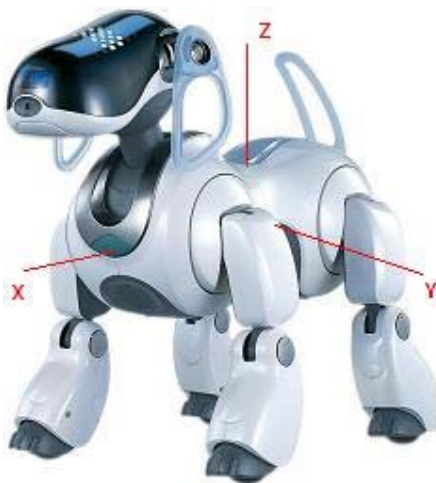


Figure 6-4: Reference Frame of the Aibo Robot

For each leg 3 joint frames exist. Also each joint has an angle. The frame of the shoulder can be computed from the constants defined which present translational matrices which transfer the reference frame to shoulders. We will be able to use D-H method considering Swing and Paw angles to compute the frame for knees. We have defined two polygons one for the front legs and one for the rear legs which are used to achieve the contact point of the leg with ground. A polygon is a sequence of points in a plane.

For head there exist three joints. Computing the position of head relative to the reference frame is too easier. Because it is not in contact with ground and won't be affected as legs do. There will be three transformation matrices from each joint to the other and by multiplying them the coordinate of the head will be achieved.

6.3.2 Inverse Kinematics

For the inverse kinematics we have the positions and want to compute the joint angles of the robot. When we want to break a trajectory to a number of points we have to compute the joint angles for each point to give this sequence of joint angles to the lower level for movement. The problem arises when the leg is on the ground

because of its sophisticated shape. Figure 6-5 illustrates the code of inverse kinematics function of Aibo Leg. The function gets position of the Aibo's toe in three-dimensional relative to reference frame of each leg (which is the junction of leg to body) and as an output, values of three joints of that leg returned.

```

Void inverseKinematicsLegs(double tPoints[3], double legJoints [3]){
    double Qs[3];
    for (int j=0; j<3; j++){
        double x = tPoints[0], y = tPoints[1], z = tPoints[2];
        Qs[2] = abs(rad2deg(acos((pow2(x) + pow2(y) + pow2(z)-
pow2(LEG_LINK1) - pow2(LEGF_LINK2)) / (2 * LEG_LINK1 * LEGF_LINK2)))));
        Qs[1] = rad2deg(asin(y / (LEGF_LINK2 * cos(deg2rad(Qs[2])) +
LEG_LINK1)));
        double a = sin(deg2rad(Qs[2])), b = cos(deg2rad(Qs[2]));
        double A = LEG_LINK1 + LEGF_LINK2 * b;
        Qs[0] = rad2deg(asin((A * x + LEGF_LINK2 * a * z)/(pow2(A) +
pow2(LEGF_LINK2) * pow2(a))));
        legJoints[j] = Qs[j];
        if (j == 2)
            if (0.01<abs(z - front_z_shift))
                legJoints[j] = 122;
    }
}

```

Figure 6-5: Inverse Kinematics function of Aibo Legs

6.4 Head Movement

Head movement usually occurs in three major cases. When we want the robot to trace an object, when the robot should look for an object and when robots want to use its head to control or kick the ball. Decision making layer decides where the robot should look to and just tells the X and Y. in the case of using head for action involved with ball especially in blocking action the head joints controlled such as other joints. But in the case of tracing objects or looking at special position a function called *Look* has been designed in motion module to set robot's joints in order to look at specific place. Decision Making should decide to use *Look* function to get more information or trace an object.

This function gets as its input a point on the ground with its X, Y from a reference point. Our robot is always looking to the ground because everything is on the ground while plying. Also it avoids noises which come when looking to spectators. If the robot can look to the point it sets its three head joints otherwise it rotates its body up to when it can look at that point. It is clear that if we know robots X and Y position relative to the reference point it is easy to compute the angles using inverse kinematics. Figure 6-6 show the mentioned look function which gets position of target points relative to the projection of robot reference frame to the field of game.

```

void Look(double x, double y, double head_joints[3]){
    setHeadFree(false);
    double j2 = (y==0)?0:(rad2deg(atan(x/y)));
    const double front_height = 9.5 + 1.95;
    const double rear_height =
        (LEG_LINK1*cos(deg2rad(getSensorValue(LRLegJ1)/1000000))
+ LEGR_LINK2*cos(deg2rad((getSensorValue(LRLegJ1)+getSensorValue(LRLegJ2))/1000000))
        + (LEG_LINK1*cos(deg2rad(getSensorValue(RRLegJ1)/1000000))
+LEGR_LINK2*cos(deg2rad((getSensorValue(RRLegJ1)+getSensorValue(RRLegJ2))/1000000))
)/20.0;
    const double neck_length = 8.0;
    double alpha = -5, beta, gama = rad2deg(atan((rear_height - front_height)/(13.0)));
    alpha += 5;
    beta = rad2deg(atan((y-neck_length*sin(deg2rad(alpha+gama)))/
(front_height*cos(deg2rad(gama))+neck_length*cos(deg2rad(alpha+gama)))))-
90+(alpha+gama) + 10.75;
    if (beta<-15) beta = -15;
    if (45<beta) beta = 45;
    head_joints[0] = -1 * alpha;
    head_joints[1] = -1 * j2;
    head_joints[2] = beta;
}

```

Figure 6-6: Determining head joint to look at (x,y)

7 Decision Making

Since there is no central processing, decision making is done via distributed manner, each player decides for itself with the information from its sensors, and information from other robots and game status. After deciding, each player informs other robots about the decision so that they can update their status and strategy.

7.1 Architecture

Figure 7-1 illustrates the architecture of DM module in *Impossible* robots.

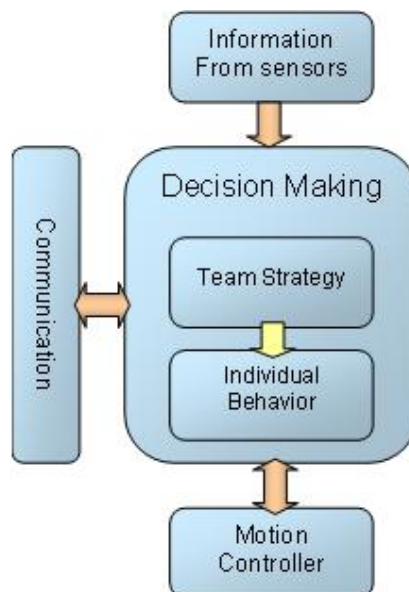


Figure 7-1: architecture of DM module

As shown in the figure, DM gets information from the sensors (including vision), motion controller (information such as amount of movement) and other robots (via communication). After analyzing the data, the first step is to decide the overall team strategy (described later). After processing the overall team strategy, each player role and position and action can be decided. Actions set include shooting the ball, walking to a specified coordinate, finding the ball, grabbing the ball. This information is shared among the other players, for example, when the position of the ball is determined by one of the robots; it informs other players about the position so that they no longer search for the ball.

For example, if result is -2 and time is 5 (with a fixed opponent), overall team strategy would be to attack with a little defense. But when the time is 19 and result is still -2, overall team strategy would be complete attack in order to score.

7.3 Team Strategy Database

Due to CPU usage problem in Aibo robots, Impossible's Aibo robot uses a team strategy database to avoid CPU consumption. Team Strategy Database (TSD) contains pre-calculated and analyzed values which can be used for team strategy. Besides the output values of the fuzzy function described above, TSD also contains proper positions for each player. So when overall team strategy changed, players can find their new positions from TSD quickly without any calculation.

7.4 Individual Behaviors

Individual behaviors are affected by overall team strategy and ball ownership and information from sensors and vision. After analyzing these data, each player decides for itself.

- **Looking**

Each game object such as ball, goals, opponents and landmarks has a parameter called Last Update Time. Whenever Vision Module detects any of these objects, it calls World Model to update this parameter. This parameter can be used to determine the reliability of the data stored in the World Model. Reliability can be calculated using this parameter and current time. A ball which is last update time is 10 seconds ago is no longer reliable. So the ball information should not be used in any calculations or decision makings.

Accordingly, whenever object information is required but the information stored in World Model is not reliable, the player is ordered to refresh the data by trying to look around and finding the object. However there are some factors that can affect this decision such as current position of the player, estimated distance from the object or the ball ownership. For instance, when the player has the ball, turning to find a specific object can lead to losing the ball.

- **Walking**

Walking is costly since it will cause vision not to work correctly and hence Last Update Time of many objects would not be changed. Therefore, walking accuracy is very important. Walking command in *Impossible's* robots needs at least four arguments, degree of the walk, direction of the walk, walking speed and ball ownership status (it is also possible to order the robot to look at a specified point while walking by using two other arguments). Degree of the walk is the amount of rotation while walking. Zero means walk straight and 90 means turn to right while walking. Direction of the walk causes the robot to move to right or left without changing its direction. Speed or power is a number between 0 and 1 which represents the maximum speed that the robot

can run. Ball ownership flag is passed to walk function so that the robot tries to keep the ball under its neck while walking.

Using these parameters walk function orders Motion Module to set the joints and motors by using a function (WalkMaker) which generates trajectory points. WalkMaker uses arguments passed to walk command and generates the path by using inverse kinematics functions.

- **Grabbing the Ball**

Whenever the team loses the ball, the closest player is ordered to grab the ball. Grabbing the ball requires information about the ball. If the information is not reliable the robot first tries to search for the ball (or use information from other agents). After locating the ball no other information is needed. The robot tries to walk to ball whereabouts in order to grab it.

By determining the ball velocity and direction, the robot estimates the ball location and tries to walk to that location instead of following the ball.

- **Shooting**

Shoot command requires one argument which specifies the shoot action. There are few shoot-moves available and each is used in a different position. Shooting straight or shooting to right and left, shooting using neck or using chest are different shoot moves. Decision Module chooses one of this shoots considering robot position, opponent goal position and distance between the player and the goal.

8 Communication

Collective decision making is an essential part of a multi-agent system such as robots. Therefore, the robots have to share information among them. In this section we describe methods for an optimized communication among Aibos. Communication module has not undergone any major changes since last year [3], but it has undergone some minor changes.

8.1 Connection status

During soccer games we noticed that our robots do not act as they were expected to. By a better analysis on modules we found out that connections between 2 or more robots might be lost and therefore there would be no communication among them. Therefore, information in its world model will not be updated and gradually this will result in less knowledge about the environment, so making unsuitable decisions. Since we are using a distributed scenario in Communication module, failures in communicational links might appear critical to our decision making modules. Thus keeping track of connections among robots will help us to act with another behavior during the game.

One of the ideas was to keep track of connections among Aibos with an LED on the face which blinks whenever a message was received. But one LED would not be enough. Since the Aibo might receive messages from two Aibos and the other one might not be alive on the other end of the wireless link. Thus, we determined 3 LEDs on the face to have the status of all connection. While a connection is not dead and there are messages transferred via this link, the corresponding LED will be illuminated and this shows the status of connection. But whenever the Aibo believes that the Aibo on the other end of link is not alive and the connection is lost, the LED will be turned off.

This feature will help us fix connections before a game is started. Even during the game, we can have special techniques to fill the gap of absence of another Aibo. We can incorporate this knowledge into our Aibos behaviors to gain better results.

8.2 Dead Connection Recognition

One of the most simple dead connection recognition techniques is to keep track of the time of last message received from each Aibo. If the difference of current time and those saved times exceeds specific amount, the Aibo can assume the other end of that connection is dead and the connection is lost. Therefore each robot keeps track of how many cycles ago it has heard from the other Aibo on the other end of that specific connection. If this exceeds a certain amount of time, which we call *TIMER_THRESHOLD*, then this Aibo sends an *EMERG_MSG* to the other end. Each

9 Tools

Running compiled code on Aibo itself appears inefficient and time-consuming for debugging purposes. Therefore *Impossibles* spent months on programming software tools that do not run on Aibo platform but helped in development of the soccer software.

A tool for remotely debugging the running code on real robots is presented in Section 9.1. It provides variety of dialogs which allows retrieving robot's information, setting new values for joints, sensors, etc. and testing algorithms separately before using them in the robot.

The Simulator, presented in Section 9.2, makes it possible to simulate up to eight robots. The source code that was developed is compiled and attached to this application. Simulating codes on virtual robots allows testing and debugging algorithms very easily in comparison to running them in a realistic environment with physical robots.

9.1 AIBO Controller

One of the vulnerabilities of *Impossibles* during last year, which slowed down development of codes, was the lack of debugging software. This year, team members spent a lot of time on writing an application which allows debugging the codes, running on real robots, remotely and also gaining access to robots information in the latest frames. It consists of a client program named *AIBO Controller* running on the PC and a *debug* module running on the robot.

All Aibo relevant codes can be traced with this tool. If the code does not contain any vision part, simply we can disable vision debugger and the same thing happens when other parts are not needed. Thus, one of the main advantages of the control tool was the use for challenges in addition to soccer.

9.1.1 DEBUG Module

In addition to all other main modules of soccer software, there would exist a *debug* module and all together will be written to memory stick. This object will run separately and in parallel with other modules in the robot. The task is to gain information from all other modules, joints, camera, sensors, etc. and send them in a predefined format to the client program running on the PC.

9.1.2 Client Application

The *AIBO Controller*, as the client, provides vast of dialogs for retrieving data from the Aibo, as a debugging interface to the robot. Approximately all internal representations of the robot (joint data, images, body sensor data, world states, perceptions) and even internal states of other modules can be visualized and analyzed via the interface.

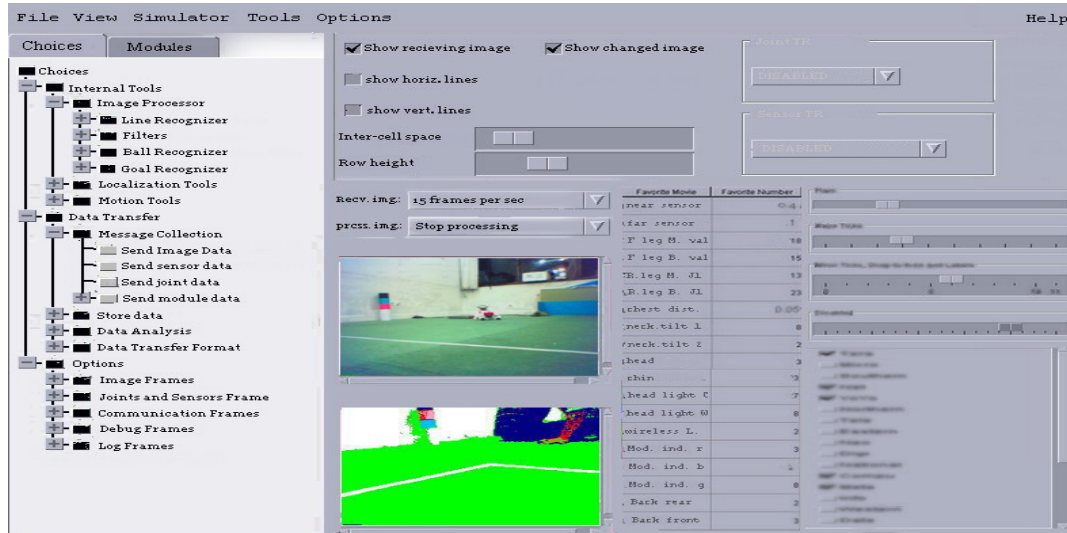


Figure 9-1: AIBO Controller User Interface

Reaching to specific states in a realistic nondeterministic environment is somehow impossible. The need of going back to the state in which a bug occurred is clearly felt. As a result this tool also makes it possible to set the values of intermediate representations of other modules, plus all other joint values. This feature allows changing robot's state to the preferred one and test algorithms in the new condition, so completing debug process.

Although this module is used together with all other running modules on Aibo, it can be also used solely. In this method all other modules are disabled and the only running code is the debug object. So it only collects internal representations of the robot and sends them back to the PC. All algorithms for different modules can be tested separately on the data received. For instance, vision algorithms can be applied to images sent by debug object. In this way the efficiency and correctness of algorithms can be checked without running them on real robots. The same approach is used for motion algorithms by the use of data collected from the joints and setting new data generated after running the algorithms. As a result, algorithms can be tested and the results can be analyzed before merging them together. This approach leads to better code development techniques.

9.1.3 TCP vs. UDP

Statistical analysis of possible links between client and Aibo shows the rate of data loss in different conditions and TCP and UDP throughputs in those scenarios [14]. Since in a wireless connection, data loss occurs in a high rate, choosing an efficient link with an optimized frame size is a must to make data transferred in a real-time process.

Network simulations for a wireless connection led us to an optimized frame size for each link. At last, based on the quality of needed service factors, an acknowledged connectionless wireless link with a non-blocking UDP-based data transfer protocol was the best choice among all others. The rate of data loss is lessened by choosing the optimized packet size and on the other hand the time for transferring data is as short as possible. So transferring data is done in a real time process (i.e. images are viewed by the rate of, nearly, 30 frames per second and joint's, together with sensor values are visualized continuously with nothing to be lost).

9.1.4 Conclusion

Debugging process is done through different dialog boxes as GUI for the front-end part. Each dialog box is connected to corresponding debugging technique on the robot. Thus solving common tasks in debugging robots such as:

- *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 on the collected data on the client side instead of the robot itself.

9.2 AIBO Simulator

Writing the codes to memory sticks after each correction or development is really time-consuming. Even for minor changes, some parts of codes should be rewritten to memory stick and the robot should restart loading modules. So time becomes an important bottleneck in development process. In contrast, simulated environments simplify programmer's job and make it possible to do all steps of running the new code in a short time. Simulation increases the development efficiency dramatically, since we can simulate and debug our codes at any place without the actual robots.

Thus, *Impossibles* decided to develop a simulator which is capable of simulating up to eight robots simultaneously. Code is attached and linked to the simulator first. Then *AIBO Simulator* starts running the modules on virtual robots in the way similar to the real robots.

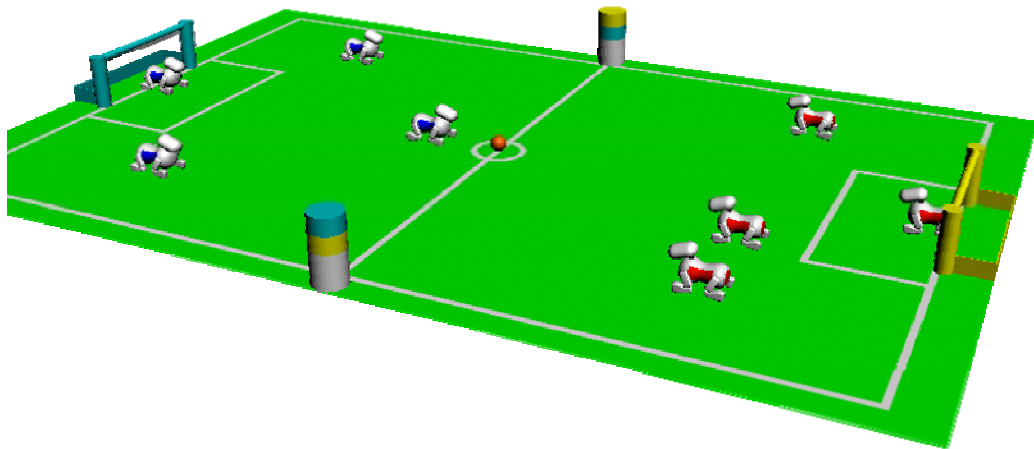


Figure 9-2: AIBO Simulator snapshot

AIBO Simulator consists of three independent modules with well defined interfaces. The *graphical user interface*, the *simulator kernel* with all subsystems to simulate the environment, and, a *controller* part which is provided by user [13].

Figure 9-3 shows the main structure of our simulator. GUI and Controller units possibly have communication with kernel in a bidirectional way, but there would be no need to have communications among these two, separately.

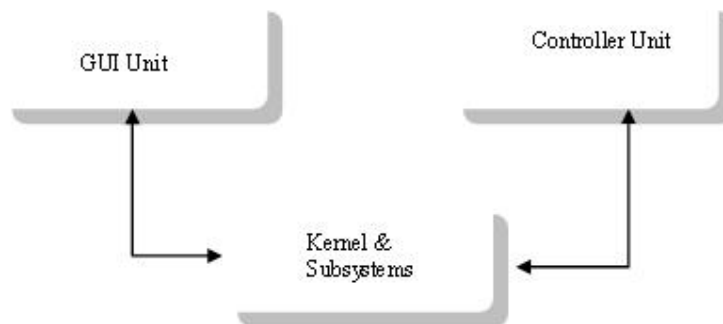


Figure 9-3: Structure of AIBO Simulator

The following sections will give a brief overview of these three modules and discuss how their internal architecture is designed.

9.2.1 Controller Unit

Currently, the code which is aimed to be simulated should be applied some changes. All modules should implement *AbstractAIBOModule* interface in order to have the authority to run inside the simulator, we call these modules *AIBO Module* (AM) objects. So real robots can not participate in games held in simulator and the code developed for robots can not directly be used.

Figure 9-4 shows the structure of Controller Unit based on our architecture in an abstract view.

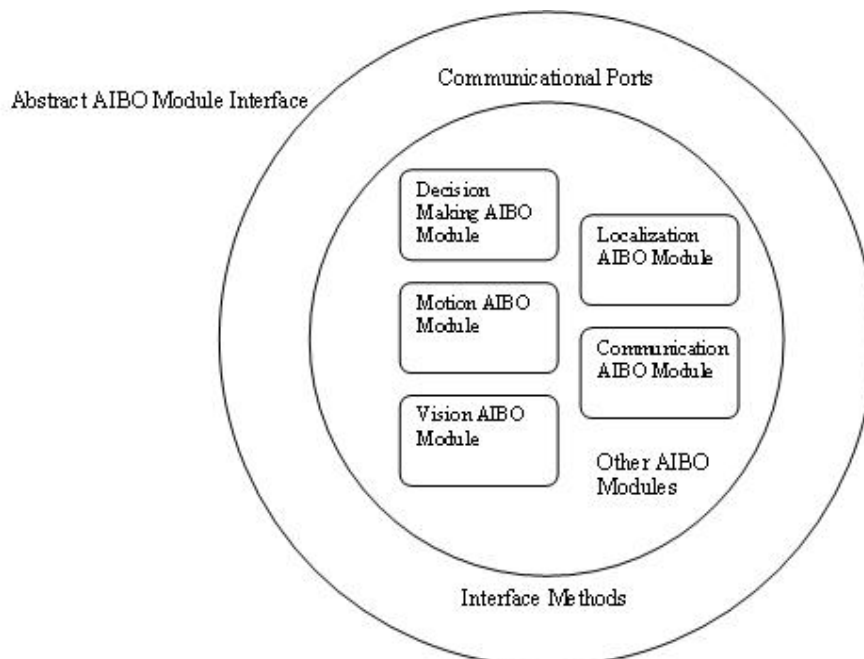


Figure 9-4: Controller Unit structure

Inside the Abstract AIBO Module is the codes developed for simulation purposes. As mentioned earlier, all modules should implement this interface and through which they will connect to simulator's kernel.

Some *ports* are provided for communications. Just like what an Open-R module defines in *stub.cfg*, there will be such configuration file and all these connections will be defined there. Through these ports, data (such as images, joint values, sensor values, low level commands, etc.) will be transferred between kernel and AIBO Modules.

There are also some registered methods based on which simulator runs. Just like an OObject that should have initialization, running and termination methods implemented, our controller codes also should implement some abstract methods.

After providing such methods and ports, the compiled code is linked to the simulator, and then simulation starts.

9.2.2 Kernels and Subsystems

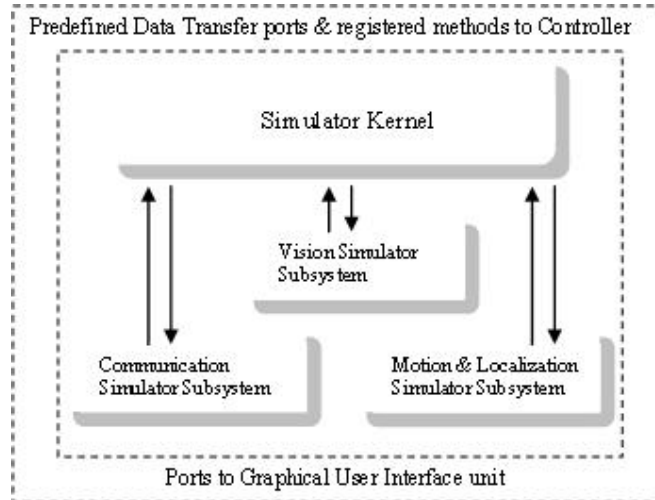


Figure 9-5: Kernel & Subsystems structure

The structure of Simulator's heart is discussed below in details. It consists of three main simulation subsystems and kernel for management purposes. *Vision Simulator Subsystem* prepares virtual images for the simulated robots. Since the simulation is done in a 3D environment, making virtual images is possible and each robot can have its vision module run with these data as inputs. *Communication Simulator Subsystem* is also responsible of communicational links among robots. This subsystem is given all realistic network parameters at first (which contains Bit Error Rate, possibility of data loss and the condition in which the virtual wireless system should work), then it simulates all connections and data transmission among all robots by message passing techniques. There is a *Motion and Localization Simulator Subsystem* (MLSS) which receives data from the kernel and moves the robots to the place where they are expected to be. Based on the fact that even movement of robots might be influenced with errors and this will result in unexpected locations, the MLSS will apply error functions. Therefore, as what happens in a real world, robots might not be exactly in the place they were expected to be. This will make simulation resembles the real world more.

The following sections give a brief overview of Simulator's basic parts and how they are used to simulate a team of robots.

9.2.2.1 Simulator Kernel

Kernel is designed to control other subsystems of simulator, executes commands from Controller and models the environment. The modular design and implementation of subsystems simplifies the task of management algorithms.

Management of subsystems is done by kernel. Each subsystem supports specific simulation module. They all have inputs and outputs, as a collection of robot states. Therefore kernel is responsible to break high level commands of controller into low level commands of all subsystems. Then pass all these parameters to relevant subsystem.

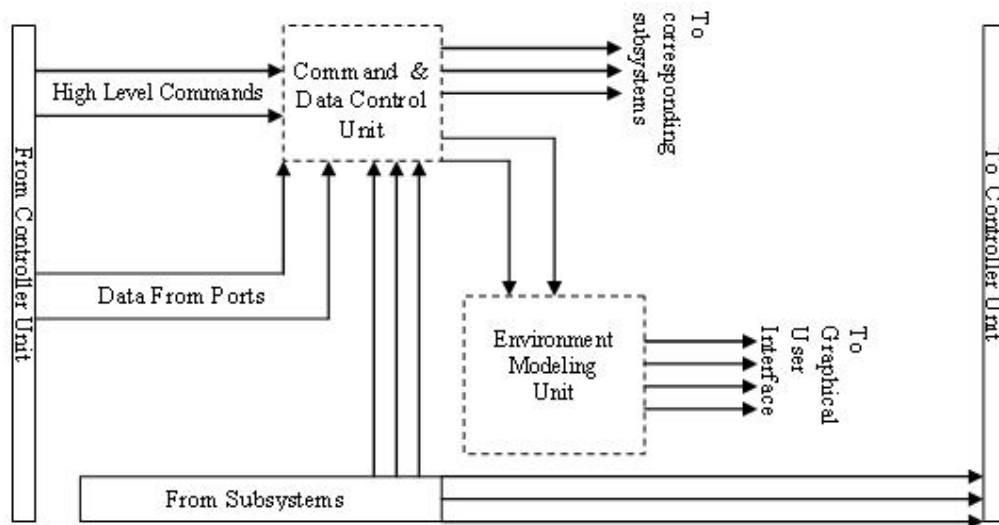


Figure 9-6: Internal view of Kernel

In addition, kernel reports robot's states to the collector through definite ports. It collects data from all subsystems and formats them in a specific frame and then sends them to the relevant ports. For instance, Vision Subsystem sends images to imageObserver port and on the other side, if any listeners are waiting for data from this port; data will be received and used.

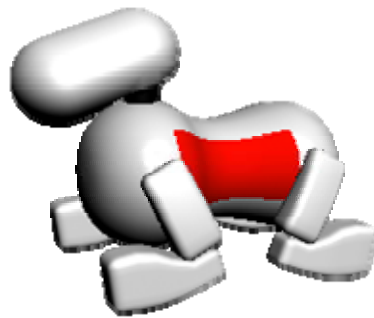


Figure 9-7: Aibo model in our simulator

Modeling the environment is an important task of the kernel. All functions for drawing models are a part of kernel. Through using OpenGL we can have a complete 3D modeling for our environment.

Bodies in our modeling system include some simple cylindrical objects which are connected to each other in a way to form a real robot. Figure 9-7 shows a model of an Aibo robot in our simulator.

9.2.2.2 Vision Subsystem

In a 3D environment, it is possible to create virtual images and use them for image processing purposes. By the knowledge of visiting scope of Aibos and the current state of the camera, in addition to its location in the field, a virtual snapshot can be created and converted to a coloring system. This way, virtual robots can act like real robots and the codes for vision part can be used with no changes.

Since in a real soccer game, robot's vision is highly dependant on lighting condition of the environment and difference in temperature might influence its efficiency, we provided some lighting functions. Light can be adjusted manually at starting point.

9.2.2.3 Communication Subsystem

In a real world, Aibos communicate via a wireless connection. In contrast, in a virtual world, robots should communicate through message passing. Wireless connections and communications should be converted into inter-process communication techniques. The task is done by Communication Subsystem in our simulator.

In order to make it resemble more to real connections, error rate and data collision events would take place in our simulated communications. These parameters are based on statistical analysis of communications in a wireless system.

9.2.2.4 Motion & Localization Subsystem

In a real world, for movement purposes, joints are adjusted with some values and the real robot will move to the approximately desired destination. In comparison to real situations, a virtual system should prepare some tools for the virtual robots to simulate motion. Therefore the MLSS is designed in a way to emulate robot's motion and other movement of legs.

In real displacements of robots, errors might occur. As a result, Aibo would not be exactly located in the expected destination; rather it will be located in a neighborhood of destination. So some noise functions are included to estimate errors in displacements. With these techniques, motion takes place with realistic parameters.

9.2.3 Graphical User Interface

Our user interface includes an editor in which scene files are defined. This increases the flexibility of simulation environment modeling, i.e. we can simulate our robots in a place rather than soccer field.

The user interface provides some functions as the interface to kernel & subsystems. In addition it has some ports to exchange data with the other end of connections. Thus the kernel can do the task of drawing modules by converting them to definite data formats and send them via these communicational ports, then calling relevant functions to complete the task of drawing.

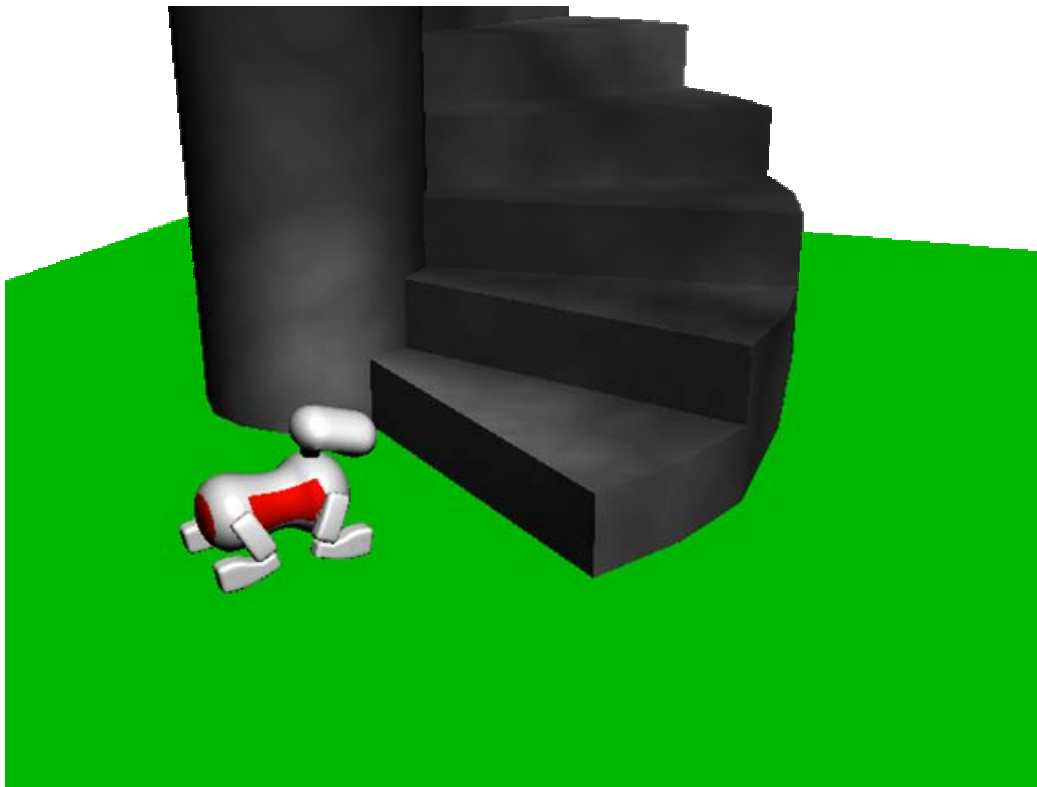


Figure 9-8: Aibo simulation in another environment

9.2.4 Future Works on Simulator

Since in our current simulator we have to change codes in order to be convenient with the kernel, we can not use robot-developed codes directly. Changing codes and rewriting them in a different architecture is really time-consuming. More important is the lack of opportunity to have real robots connected to simulator and playing the role of virtual robots.

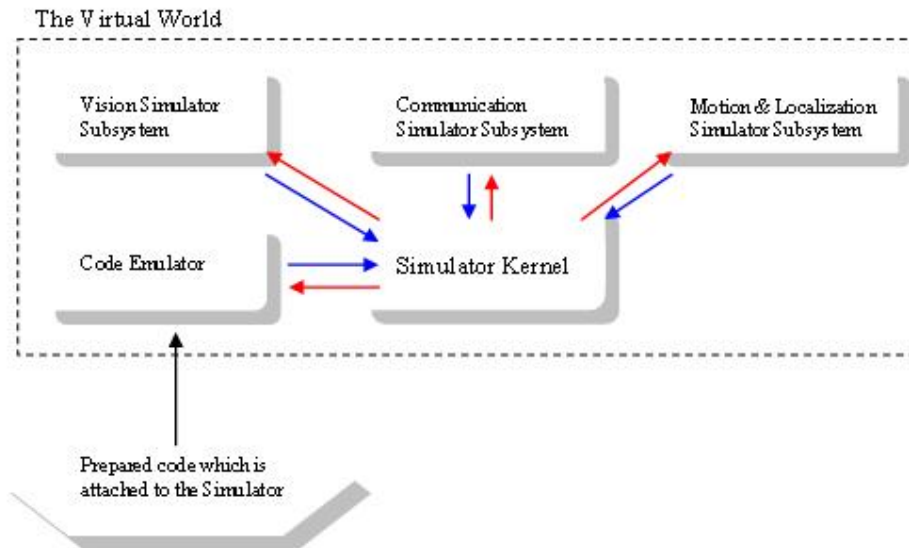


Figure 9-9: Simulator's Structure in the future

Thus, our current project is to extend the structure of simulation system and make it possible to use the original codes developed for Open-R operating system. A solution to above problems is creating a Code Emulator which emulates the code developed for Aibos, with no changes. Then the Code Emulator can communicate with kernel to interpret and send commands, and, receive and deliver data.

The structure of new simulator is shown in Figure 9-9. All subsystems will be almost the same as our current version, but the only part which is changing is the Code Emulator Subsystem, as a layer between the source code and kernel. So removing the Abstract AIBO Module interfaces and makes a direct channel between the codes and kernel.

On one hand, connecting the simulator to the *AIBO Controller* tool also will help us so much in gaining better analytic results. On the other hand, a real robot can also plays the role of a simulated robot in the simulation environment, i.e. we can have both virtual robots and real robots simultaneously playing soccer in a virtual and physical field. The simulator can also be configured to judge a soccer game. All above makes it easier to have contests and practices more and more with no need to have enough ready physical robots.

One of important ambitions of *Impossible's* is to extend simulator's idea for robots other than Aibos. This idea takes a long time to be completed and might need other parts to be added to current structure.

10 References

- [1] <http://openr.aibo.com>
- [2] H. R. Vaezi Joze, S. Aliari Zounoz, S. Rahbar, M. Valipour, A. Fathi, *Impossibles Aibo Four-Legged Team Description Paper RoboCup 2006*, RoboCup 2006, June 2006.
- [3] J. Habibi, S. Aliari Zounoz, H. R. Vaezi Joze, S. Rahbar, M. Valipour, A. Fathi, K. Mokhtarian, A. Kamali, *Impossibles Aibo Four-Legged RoboCup 2006 Report*, RoboCup2006, Bremen, Germany, June 2006.
http://ce.sharif.edu/~impossibles/soccer_2006/impossibles_report_2006.pdf
- [4] D. Kortenkamp, R.P. Bonasso, and R. Murphy, “*AI-based Mobile Robots: Case studies of successful robot systems*”, Cambridge, MA, 1998.MIT Press.
- [5] D. Fox, W. Burgard, F. Dellaert, S. Thrun, *Monte Carlo localization: Efficient position estimation for mobile robots*, In Proceedings of the National Conference on Artificial Intelligence, 1999.
- [6] A. Elfes, *Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation*, PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1989.
- [7] D.B. Rubin, *Using the SIR algorithm to simulate posterior distributions*, In M.H. Bernardo, K.M. an DeGroot, D.V. Lindley, and A.F.M. Smith, editors, Bayesian Statistics 3. Oxford University Press, Oxford, UK, 1988.
- [8] D. Fox, *Markov Localization: A Probabilistic Framework for Mobile Robot Localization and Navigation*, Institute of Computer Science III, University of Bonn, Germany, Doctoral Thesis, December 1998.
- [9] S. Lenser and M. Veloso, *Sensor resetting localization for poorly modelled mobile robots*, In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), San Francisco, CA, 2000.
- [10] H. R. Vaezi Joze, J. Habibi, S. Rahbar, *piecewise linear probability distribution localization: fast and inexpensive mobile robot localization*, (to be submitted)
- [11] K. Mokhtarian, H. R. Vaezi Joze, J. Habibi, *An Inexpensive Approach for Real-Time Vision on Four-legged Footballer Robots*, In Proc. 12th International CSI Computer Conference, Feb 2007.
- [12] H. R. Vaezi Joze, M. Ghodsi, *Design and Implementation of soccer software for Aibo robots*, B.S. thesis, Computer Engineering Department at Sharif University of Technology, 2006.
- [13] T. Rofer et al., “*German Team Technical Report for RoboCup 2005*”, RoboCup, Osaka, Japan, 2005. <http://www.germanteam.org/GT2005.pdf>.

