Impossibles

RoboCup 2006

RoboCup

RoboCup 2006
Bremen · Germany

Rescue · Soccer · Junior

14 - 20 June 2006

Sharif University of Technology

Computer Science Department

Artificial Intelligence & Robotics Laboratory (AIRL)

Supervisor:

Jafar Habibi

Members:

Saman Aliari Zonouz

Hamid Reza Vaezi Joze

Siavash Rahbar

Majid Valipour

Alireza Fathi

Kianoosh Mokhtarian

Ali Kamali

Contents

1 Introduction 9
2 Architecture 11
3 Fuzzy World Model 13
4 Communication 15
4.1 Communication module 15
4.2 Information Level 15
4.2.1 High level commands 16
4.2.2 Low level data 16
4.3 Centralized vs. Distributed Architecture 16
4.4 UDP vs. TCP 18
5 Localization 19
5.1 Self-Localization 19
5.1.1 Probabilistic Distribution Localization (PDL): 19
5.1.2 Triangular Probabilistic Distribution Localization (TPDL): 20
5.2 Object Localization 22
5.2.1 Evidence Theory 22
5.2.1.1 Basic Probability Numbers 23
5.2.1.2 Belief Functions 23
5.2.1.2.1 Bayesian Belief Functions 24
5.2.1.3 Dempster's Rule of Combination 25
5.2.1.3.1 Combining Two Belief Functions 25
5.2.1.3.2 Combining Several Belief Functions from AIBO robots 27
6 Decision Making 29
6.1 Architecture 29
6.2 Team Behavior 30
6.2.1 Determining Factors 30
6.2.1.1 Defense vs. Offense 31
6.2.1.1.1 Caution and Risk 31
6.2.1.1.2 Ball Ownership 36
6.2.1.1.3 Hyperbolic Danger Safety Degree 36
6.2.1.2 Team Fear-Relax Emotional Degree 38
6.2.2 Team Strategy Database 38
6.2.3 Emotional Tactics Selection 41
6.3 Individual Behaviors (Techniques) 42

6.3.1 Predefined Dynamic Assigned Regions 42
6.3.2 Outputs as Decision Making-Motion Engine Interface 42
6.3.2.1 Looking 43
6.3.2.2 Walking & Running 43
6.3.2.3 Ball Grabbing 43
6.3.2.4 Shooting & Passing 43
6.3.2.5 Blocking 44
7 Vision 45
7.1 Architecture 45
7.2 General vs. Specific Vision subsystems 47
7.2.1 General Vision Subsystem 47
7.2.1.1 Color Segmentation 47
7.2.1.2 Blob Formation 52
7.2.1.3 Image Analysis 52
7.2.1.3.1 Object Detection 52
7.2.1.3.2 Line/Corner Detection 52
7.2.2 Specific Vision Subsystem 54
7.2.2.1 Enhancement/Preprocessing 55
7.2.2.1.1 Pixel Transformation 55
7.2.2.1.2 Neighborhood Transformation 56
7.2.2.2 Segmentation 57
7.2.2.2.1 Region Segmentation 57
7.2.2.2.2 Edge Segmentation 59
7.2.2.3 Coding and Feature Extraction 60
7.2.2.3.1 Shape Features 61
7.2.2.3.2 Pattern Matching 61
7.2.2.4 Image Analysis 61
8 Motion 63
8.1 Architecture 63
8.2 Skills 64
8.2.1 Shoot 64
8.2.2 Walk 65
8.2.2.1 Walking Module’s Input 66
8.2.2.2 Walking Module’s Output 66
8.2.2.3 Predefined Walking 66
8.2.2.4 Fast Walking 67
8.2.2.5 Fastest Available Walk 68
8.2.2.6 Fastest Available Stable Walk 68
8.2.3 Looking & Head Movement 68
8.2.4 Grab 68
8.2.5 Block 69
8.3 Kinematics 70
8.3.1 Forward Kinematics 70
8.3.2 Inverse Kinematics 71
9 Acknowledgements 72
10 References 73

Table of Figures

Figure 1: *"Imossibles"* World Model Based Architecture (WMBA)..........................12
Figure 2: Centralized Scenario..17
Figure 3: Distributed Scenario..17
Figure 4: NAM Snapshot of Network Simulator...18
Figure 5: A sample Trapezoidal/Triangular PDF...20
Figure 6: An example of Movement Update process.................................21
Figure 7: An example of Sensor Update..21
Figure 8: Self-Lozalization Flowchart..22
Figure 10: Schema of Combining Two Belief Functions...............................26
Figure 11: Distributed Reasoning with Information Communication..........29
Figure 12: Decision Making Module Architecture..30
Figure 13: Caution-Risk (CR) Fuzzy Controller Scheme..............................31
Figure 14: Time input of the CR Fuzzy Controller..31
Figure 15: Result input of the CR Fuzzy Controller......................................31
Figure 16: Opponent Strength input of the CR Fuzzy Controller.................32
Figure 17: Rule-Base employed for CR Fuzzy Controller.............................33
Figure 18: Caution-Risk output of CR fuzzy Controller................................34
Figure 19: CR(Time, Result) with a fixed Opponent-Strength.....................34
Figure 20: CR(Opponent-Strength, Time) with a fixed Result....................35
Figure 21: CR(Result, Opponent-Strength) in a fixed Time........................35
Figure 22: Hyperbola used to calculate Hyperbolic Factor (HF)..................37
Figure 23: the maximum defensive (MD) team strategy...............................39
Figure 24: General Defensive (GD) team strategy..39
Figure 25: Team strategy 1...40
Figure 26: Team strategy 2...40
Figure 27: Team strategy 3...40
Figure 28: Team strategy 4...40
Figure 29: Midfield Team Strategy 1...41
Figure 30: Midfield Team Strategy 2...41
Figure 31: Offensive Team Strategy 1...41
Figure 32: Offensive Team Strategy 2...41
Figure 33: Intra-Vision Module Architecture...46
Figure 34: Intra-GVS Architecture..47
Figure 35: LAB in luminance of 25%...49

Figure 36: LAB in luminance of 50%......49
Figure 37: LAB of luminance of 75%......49
Figure 38: 1-NN, 2-NN, and 3-NN Samples......51
Figure 39: Smoothing Convolution Mask to remove the holes......51
Figure 40: LFS algorithm......53
Figure 41: Vertical offsets vs. Perpendicular offsets in LFS algorithm......53
Figure 42: Intra-Architecture of SVS......54
Figure 43: Original Image......55
Figure 44: Enhanced Image......55
Figure 45: Subtraction result of Figure 40 from Figure 39......56
Figure 46: Multiplication result of Figure 41 by a Constant......56
Figure 47: Original Image......57
Figure 48: Gray-Scaled Image......57
Figure 49: Histogram Equalized Image......57
Figure 50: After Edge Detection (Sobel)......57
Figure 51: Samples of Segmented Images......58
Figure 52: Color Segmented Image......58
Figure 53: Region Segmented Image......58
Figure 54: Original Image......59
Figure 55: Gray Scaled Image......59
Figure 56: Green Layer of the Image......59
Figure 57: Edge Detected Image......59
Figure 58: Dilated version of the above Image......60
Figure 59: Detected Ball......60
Figure 60: Edge Segmentation point template......60
Figure 62: Chest Shoot......65
Figure 63: "Impossibles" AIBO grabbing the Ball......69
Figure 64: *"Impossibles"* AIBO Blocking the Ball......69
Figure 65: Reference Frame of the AIBO Robot......70
Figure 66: Two snapshots of the AIBO robot while walking......71

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).

Having done the background researches, all of the members decided to exploit their knowledge in a practical and real world environment. Since several teams from Sharif University of Technology had achieved noticeable successes from RoboCup2000 in Melbourne to RoboCup2003 in Padua, RoboCup was selected as the first choice; therefore, we were able to employ their corresponding experiences. Table 1 demonstrates a brief overview of these achievements.

As explained above, RoboCup's interesting features attracted us to begin implementation of our previously designed 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.

Once world championship was achieved, team members made decision on continuing their research objectives through AIBO 4-legged League. AIBO League was preferred over the other RoboCup Leagues because of the following four reasons which are also considered as *"Impossibles"* objectives in AIBO league. AIBO does support the real world challenges, whereas Rescue Simulation does not. Additionally, it is the only physical robot league in which there is no need to get involved into mechanical aspects of the robots' design, so it was the most similar league to the simulation leagues such as Rescue Simulation. Furthermore, AIBO 4-legged league supports most of the research interests of the team members such as machine learning. Lastly, several highly ranked universities (e.g. CMU and Texas-at-Austin) have done research on various branches of AI using AIBO robots; hence, it is thought to be a qualified infrastructure for our team members to do research on. On the other hand, we follow our competitive objective which is to be ranked as one of the first four teams of AIBO league in Bremen, Germany, 2006.

Since Vision and Image Processing were required in order to accomplish the AIBO project, defined in Artificial Intelligence and Robotics Laboratory (AIRL) of Computer Science and Engineering Department at Sharif University of Technology, we came to conclusion to invite some of the members of Vision Group at IPM School of Mathematics Scientific Computing Center.

Table 1: Sharif University of Technology Teams in RoboCup Leagues

Place	Team	League	Rank
RoboCup2000 Melbourne	Sharif CE	Soccer: Middle Size	Third Place
RoboCup2001 Seattle	Arian	Rescue Simulation	Second Place
	Sharif CE	Engineering Challenge	First Place
RoboCup2002 Fukuoka/Busan	Arian	Rescue Simulation	First Place
RoboCup2003 Padua	Arian	Rescue Simulation	First Place
	CEDRA	Rescue Robot	Second Place
RoboCup2005 Osaka	**Impossibles**	Rescue Simulation	First Place
RoboCup2006 Bremen	**Impossibles**	AIBO 4-Legged	???

2 Architecture

Our previous experience in Multi-Agent System (MAS) architecture design in Rescue Simulation Environment leaded us to World Model Based Architecture (WMBA). Having made some subtle modifications in 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

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 and maximum 500Kbps data transmission for wireless communication.

Figure 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.

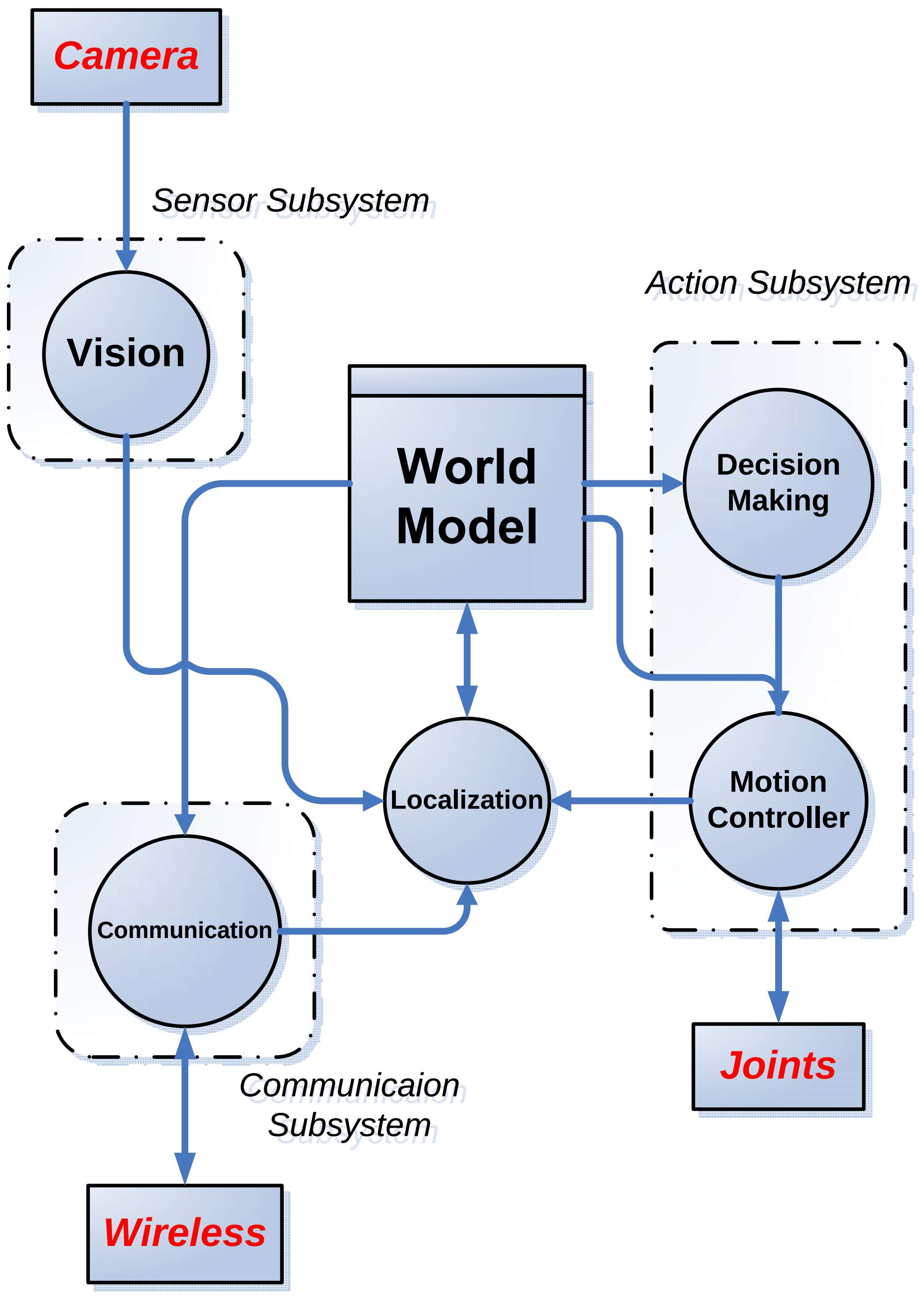


Figure 1: *"Imossibles"* World Model Based Architecture (WMBA)

3 Fuzzy 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.

Generally, the anchoring process consists of the following three steps:

- ***Classification***: each perceived object (i.e. set of features produced by a sensor) is classified according to the predefined features of known objects.

- ***Fusion***: Objects perceived by different sources, such as sensors or camera, that can be associated to the same physical object are merged.

- ***Tracking***: The perceived information via current inputs update the corresponding objects' features in the world model. We assume that smart sensors produce sets of features, where each feature is a triple: $\langle label, v, \rho \rangle$. The label of a feature is its name, v is its numerical value, and ρ is its reliability value, i.e. how the data is assumed to be reliable given the specific sensor and the acquisition situation.

    - If the perceived instances do not match any instance in the world model, a new instance is created with the value of the perceived instance.

    - If an instance in the world model does not match any perceived instance, the reliability values of its attributes are exponentially decreased by a coefficient between zero and one.

    - If a perceived instance matches an instance in the world model, their reliability values are composed by the arithmetic mean.

The classification process matches features provided by the sensors with the predefined properties of objects of interest. The reliability of the concept instance is computed as a ‘ min ’ of the reliability of the features related to substantial properties. The properties have a reliability value equal to the reliability of the corresponding features. The fusion process merges instances, thus obtaining new instances whose reliability is computed using evidence theory (See 5.2.1) and the reliability of the attributes. The perceived instances produced by the fusion module update the values of the instances in the world model.

In this way, the reliability of the instances in the World Model (WM) follows the reliability of the perceived instances, and when an object is no longer perceived, the reliability of the related instance decreases until it is thrown out of the world model. This is needed in dynamical environments, where we cannot expect that data remain the same when we cannot fetch them. The decay is proportional to a constant ‘ γ ’ defined by the user, which can be defined by considering the relative rate of change of the environment with respect to the data acquisition rate.

Although it seems a good idea to collect all information somewhere, it may cause some problems, e.g. mutual exclusion. In order to solve mutual exclusion, two ideas are usually employed. First, World Model is itself considered as an object so it receives updating data and requests to send required data to other parts. Additionally, Data Structure (DS) design is in a way that no more than one part can update the world model; however, there is no limitation on number of parts which are reading data.

As a real world environment, AIBO robots receive uncertain information of their surroundings via their sensors and camera. In order to keep uncertain data, we exploit fuzzy logic; therefore, the above explained fuzzy world model was designed to support the concept of vagueness.

4 Communication

Sony AIBO ERS-7 model have a wireless LAN module (wi-fi certificated) [1] which made if able to communicate which other teammates to share useful information perceived from the environment to improve the quality of each agent's world model eventually resulting in more accurate localization and object detection.

Furthermore, Due to RoboCup 2006 rules, each team has a upper band limit of 500 Kbps for communication among the agents which includes also game manager commands but every team has some special UDP port to broadcast on [2]. So ideally we can count on about 100 Kbps bandwidth for each AIBO robot.

In what follows we will discuss various aspects of communication and will explain the way that our communication module is working.

4.1 Communication module

"Impossibles" AIBO communication module, as an independent module, works in parallel with other modules such as vision, and decision making. It is also in charge of sharing essential data amongst all players. For instance, knowing accurate positions of the players are only possible by having each player report his useful information such as its own position to the others.

The communication module is to be reliable and eventually be aware of the packet loss if any exists. It will repeatedly choose entities from World Model (WM) objects based on their last report time, their reliability measure and also importance of data for other teammates.

4.2 Information Level

There are two general strategies for communication in Multi-Agent Systems (MAS) that a team can employ depending on system's general architecture and also on what kind of data the agents intend to share.

4.2.1 High level commands

This strategy is best applicable in centralized system architecture where a center commands its agents; therefore, in this method, all critical and high level processes and decision makings are made in center. Consequently, only high level commands are sent to agents in order to make them aware of their behavior.

4.2.2 Low level data

In this method communication system is trying to share all raw data that each agent have and every thing could and should be distributed. The latter has several advantages over former in distributed systems which are discussed in “Impossibles” rescue simulation team description paper [3].

4.3 Centralized vs. Distributed Architecture

Generally, we consider the communications amongst players distributed, but due to the large amount of transmitted data and hence time-consuming processes, agents themselves accomplish their own jobs and broadcast the results, i.e. processes data.

If there wasn't any broadcast feature in our access media, having centralized communication may also reduce number of messages which are needed to share all information among agents.

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

- With broadcast message:
  - o Centralized approach – $m = n + 1$
    [n peer to peer message + 1 broadcast]
  - o Distributed approach – $m = n$
    [n broadcast message]

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

As shown in Figure 2, when we are considering our access media properties including its broadcast ability and limited bandwidth and also the fact that defining an agent as center might be unreliable we decide to use distributed communication by broadcasting messages.

The messages contain low level data sensed and acquired by agents from the surroundings such as ball, teammates, and opponent players which are used in localization and updating word model in with each agents self awareness.

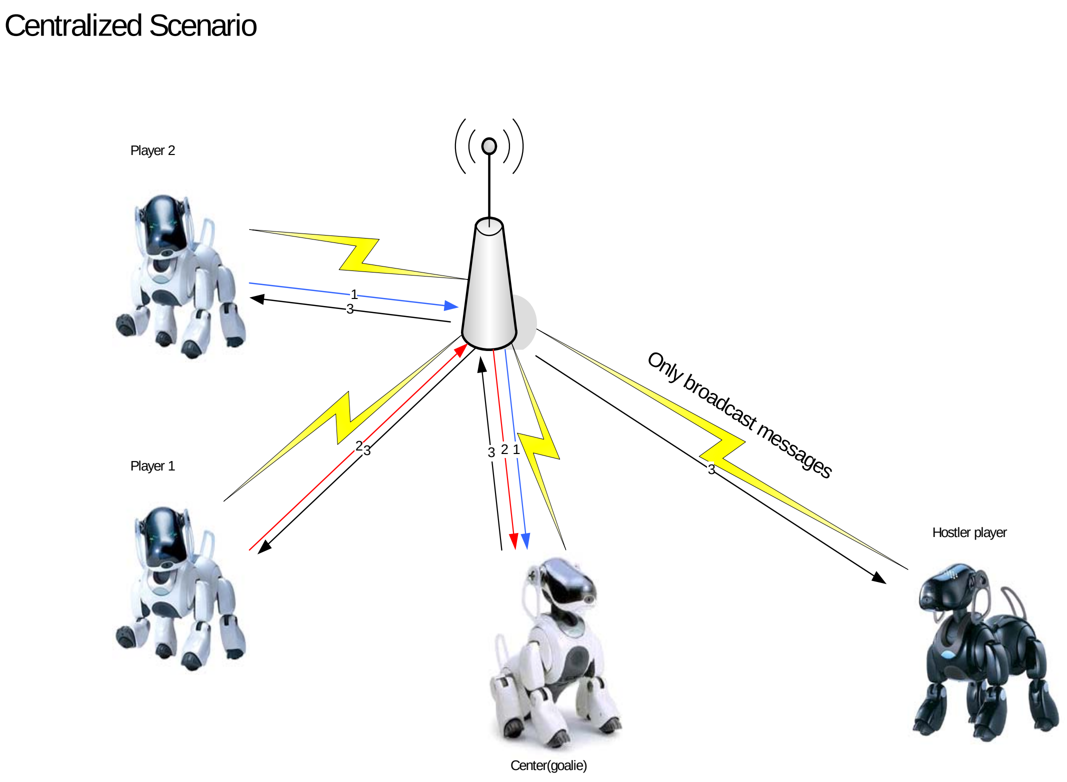


Figure 2: Centralized Scenario

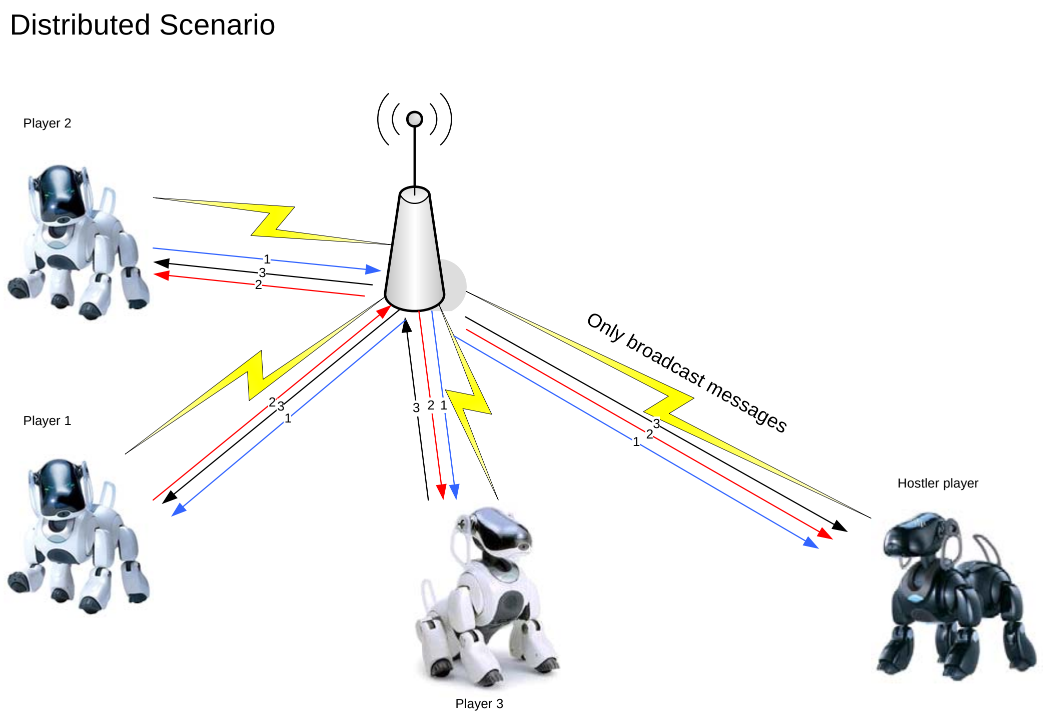
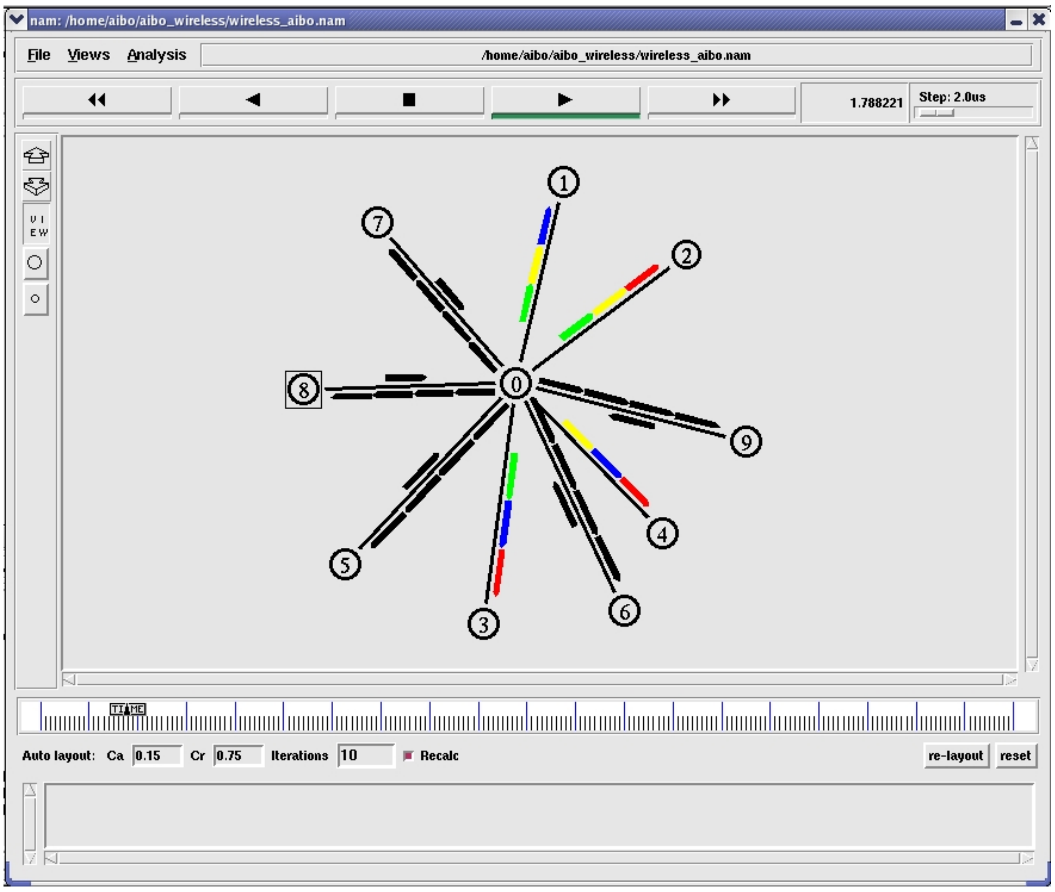


Figure 3: Distributed Scenario

4.4 UDP vs. TCP

Selecting either UDP or TCP is thought to be the primary task, and according to **'NS2'** simulation result for both UDP and TCP scenarios and other teams hints [4, and 5] we decide to use UDP, because of lesser overhead in compare with TCP and ability of broadcasting by UDP which is essential for us to minimize our number of sent messages in our distributed strategy.

NS2 simulator have also been employed to simulate UDP data transfer in wireless mobile networks, in order to select optimized value for our UDP packet size to achieve maximum bandwidth considering possible data collision and opponent team inference. The following figure (Figure 4) demonstrates a snapshot of our simulated situation.


nam: /home/aibo/aibo_wireless/wireless_aibo.nam

/home/aibo/aibo_wireless/wireless_aibo.nam

1.788221 Step: 2.0us

Auto layout: Ca 0.15 Cr 0.75 Iterations 10 Recalc

Figure 4: NAM Snapshot of Network Simulator

In our simulated situation, there is just one access point [node #0], four players [nodes #1 2 3 4], and five other network traffic producers (4 hustler players and one manager). Also the simulated wireless network implements multicast packet switching and 802/11 MAC protocol and random movement for players.

5 Localization

As mentioned before, Localization is in fact the interface for World Model that processes received data and consequently updates the World Model (WM). Unlike other processes running repeatedly without being blocked, Localization is not active until it receives a message from Vision, Motion Controller (MC), or Communication. The most important information is the position of robot itself which should be calculated in a reliable way. In *"Impossibles"* AIBO software, this calculation is done using Self-Localization module, discussed later. Other data such as position of other robots and ball are determined using Object-Localization. Both of these methods are explained in the following sections.

5.1 Self-Localization

"Impossibles" Self-Localization Module (SLM) takes the previous positions and differential locations as its input. Therefore, it receives its inputs from World Model, Vision and Motion Controller (MC) modules. Having processed the inputs, it then updates the World Model. In *"Impossibles"* implemented software positions are stored as (x, y, θ) triples that are 2D position of robot and its direction.

Although, the most popular approach for position estimation of mobile robots is Monte Carlo Localization (MCL) [6] that was widely being used by 4-Legged AIBO soccer teams, we need a method that is compatible with our fuzzy probabilistic world model and also is able to support real time applications. In what follows, we present a new approach that is a probabilistic approach for mobile robot localization.

5.1.1 Probabilistic Distribution Localization (PDL):

It considers a probabilistic distribution function (PDF) for each variable (such as x, y and θ for AIBO). In Monte-Carlo Localization (MCL), samples are stored by (x, y, θ) triples and a weight factor (p>0). In contrast, in Probabilistic Distribution Localization (PDL), we have three PDF for each sample (one PDF for each of x, y and θ). Also each differential motion, i.e. (Δx, Δy, Δθ), contains three corresponding PDFs. So we need to update the PDFs after movement update (from Motion Controller) and sensor update (from Vision).

- ***Movement Update:*** We consider 'X' a random variable for probabilistic distribution of 'x' position and 'ΔX' as a random variable for probabilistic distribution of movement of 'x' so the new value for 'X' will be 'X+ΔX'. In this way the corresponding PDF for x is obtained.
- ***Sensor Update:*** As mentioned above, each perceived data by vision module in *"Impossibles"* software contains a PDF for each variable for example 'x' and 'p' that is the probability that this sample is correct. Now we create a new PDF for 'x' by the following formula:

$$f_{Xnew}(x) = p \times f_{Xvision}(x) + (1-p) \times f_{Xold}(x) \qquad (1)$$

Our PDF may become worthless after too many movements or sensor updates with small ‘p’. So we use the idea of "Sensor Resetting Localization" [7] that considers a threshold for average of ‘p’. Some new samples must replace when it becomes lesser than the assigned threshold. Similarly in PDL in such cases, more samples are fed by vision module.

As explained before, Self-Localization’s main output is a probability distribution function (PDF) and not a crisp value, but PDL is required to provide more suitable results for other modules. So a kind of clustering algorithm can be employed.

5.1.2 Triangular Probabilistic Distribution Localization (TPDL):

In this section we are going to change PDL approach in such a way that it becomes simple and suitable for real-time applications, e.g. mobile robot such as AIBO. In order to simplify the PDL process, we employ triangular PDFs in order to make our calculation and storing much simpler. For storing PDF it is enough to store some point. For instance, for function shown in Figure 5, it is enough to store following points: ((0,0) , (1,.5) , (2,.5) , (3,0)).

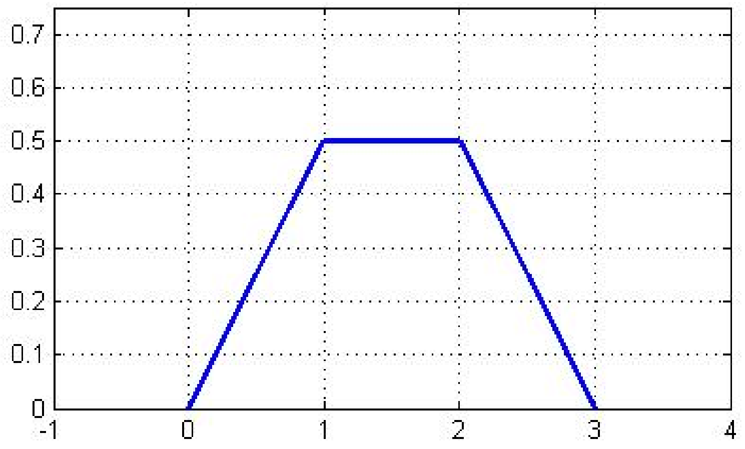


Figure 5: A sample Trapezoidal/Triangular PDF

- ***Movement Update:*** With us considering both of random variables X and Y triangular PDF, sum of them is paranoid-segment PDF. To simplify the process, we approximate such functions to be triangular function. It could be done using convolution of these PDFs as a vector of possibility for each PDF. As an example Figure 6(a) is PDF of a variable before movement update. Figure 6(b) shows PDF of movement and Figure 6(c) is the final result of movement update. Figure 6(d) shows linear approximation of result via TPDL algorithm.

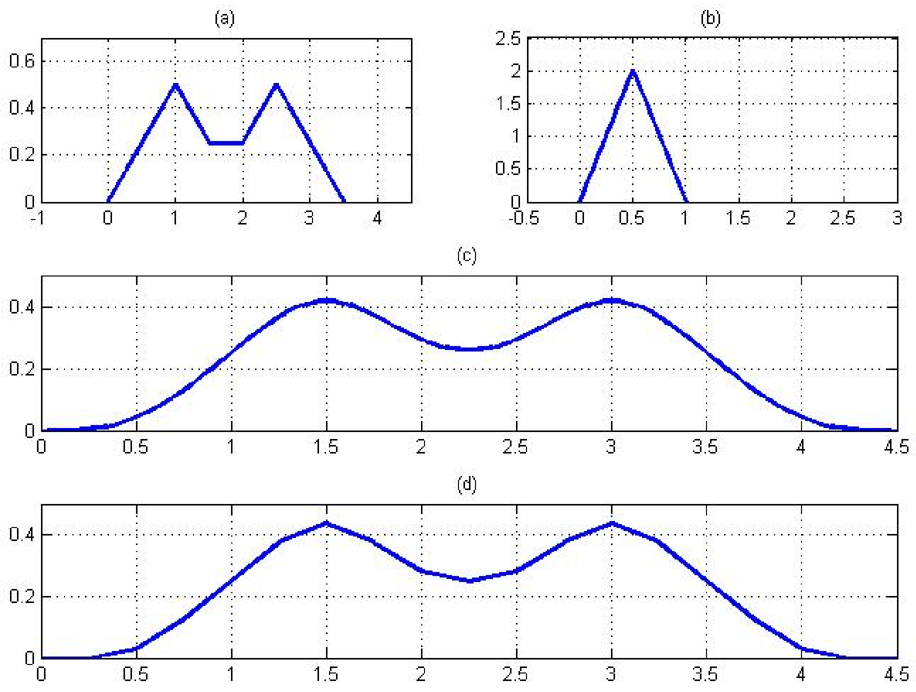


Figure 6: An example of Movement Update process.

- ***Sensor Update:*** This step is straightforward in triangular PDF. A set of 'x' points, stored, are union of this 'x' of both PDF and the corresponding 'y' could be calculated by sum of corresponding 'y'. Figure 7 illustrates a process of Sensor Update with p=0.7 and Figure 7 (c) is the final result.

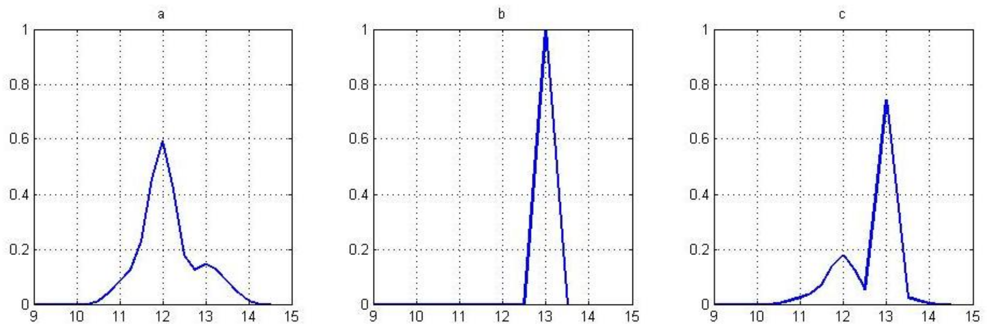


Figure 7: An example of Sensor Update. (a) old PDF (b)sensor PDF (c) result by p=.7

Also to decrease required memory for storing triangular PDF we omit points that have small magnitudes. Additionally, the function is scaled in a way that integral of the function becomes one.

TPDL seems to be great that is compound of Monte-Carlo Localization and Sensor Resetting Localization with probability theory in the other hand it is compatible with our Probabilistic World Model and can be used real-time.

Figure 8 demonstrates our Self-Localization flowchart using TPDL method. We explained sensor and movement updating and also storing triangular PDFs as a set of points. The next submodule is "PDF Filtering" which filters PDF in order to omit small values and also non-reliable ones. Then, it is determined if some extra samples are required from vision. This decision is made using a threshold on PDF variance after clustering it.

When it needs more samples it sends a signal to vision module to provide localization module with some extra samples. This signal also contains information about the accuracy of such a sample that may cause executing time-consuming, i.e. Specific Vision Subsystem (SVS) which is computationally more expensive than GVS (See General vs. Specific Vision subsystems).

Although we do not use information about positions of AIBO robots from its teammates, it can be employed if vision part is optimized so have enough time for more calculations; however it seems impossible with current CPUs of AIBO robots. Therefore, in TPDL we consider this information to be samples that of course are not as precise as robot vision. In this case, self-localization method may be fed from Communication subsystem too.

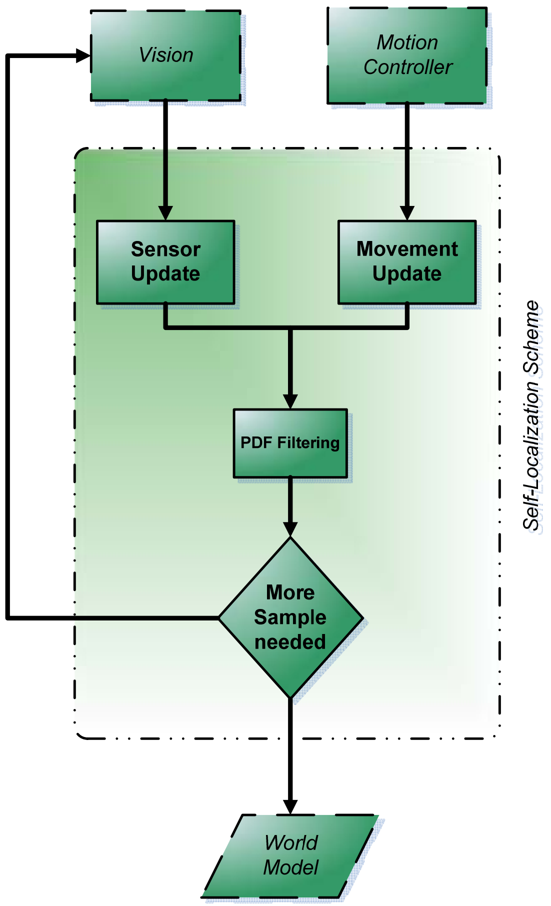


Figure 8: Self-Lozalization Flowchart

5.2 *Object Localization*

Vision module supports data for possibly position of that robot that is needed by self-localization and also possible position of other objects such as ball and hostler AIBO, because the teammates' locations are announced by wireless communication. Object localization is responsible for collecting data about object position from his vision submodule and communication from other teammates in order to estimate these positions.

5.2.1 Evidence Theory

We employed evidence theory [8] in order to estimate the locations of objects of interest on the field. Evidence theory begins with the familiar idea of using a number between zero and one to indicate the degree of support a body of evidence provides for a proposition, i.e. the degree of belief one should accord the proposition on the basis of the evidence. Evidence theory focuses on the combination of degrees of belief or support based on one body of evidence with those based on an entirely distinct body of evidence. The heart of the theory is Dempster's rule for effecting this combination. What follows is a brief explanation of evidence theory. The mathematical proofs of theorems are not given in details.

5.2.1.1 Basic Probability Numbers

We want to partition all of one's belief among the different subsets of Θ, assigning to each subset 'A' that portion that is committed to 'A' and to nothing smaller. This suggests the following definition:

- Definition: if Θ is a frame of discernment, then a function $m: 2^{\Theta} \rightarrow [0,1]$ is called a basic probability assignment whenever (1) $m(\varnothing)=1$, and (2) $\sum_{A \subset \Theta} m(A)=1$.

The quantity $m(A)$ is called A's basic probability number, and it is understood to be the measure of the belief that is committed exactly to A. Condition (1) reflects the fact that no belief ought to be committed to Θ, while (2) reflects the convention that one's total belief has measure one.

To reiterate, the quantity $m(A)$ measures the belief that one commits exactly to A, not the total belief that one commits to A. To obtain the measure of the total belief committed to A, one must add to $m(A)$ the quantities $m(B)$ for all proper subsets B of A:

$$Bel(A) \quad = \quad \sum_{B \subset A} m(A) \tag{2}$$

A function $Bel: 2^{\Theta} \rightarrow [0,1]$ is called a belief function over Θ if it is given by above equation for some basic probability assignment $m: 2^{\Theta} \rightarrow [0,1]$.

The belief function with the simplest structure is surely the one obtained by setting $m(\Theta)=1$ and $m(A)=0$ for all $A \neq \Theta$. Since this belief function seems appropriate when one has no evidence, it is called the vacuous belief function.

5.2.1.2 Belief Functions

According to *"Impossibles"* AIBO architecture, the concept of belief functions is employed. As discussed earlier, the vision module of each agent (i.e. AIBO robot) provides a set of tuples in each of which a fuzzy reliability/belief degree is present. Each object on the field is represented by tuple.

- Theorem 5.2.1.2.1: If Θ is a frame of discernment, then a function $Bel: 2^{\Theta} \to [0,1]$ is a belief function if and only if it satisfies the following conditions:

  (1) $Bel(\varnothing) = 0$
  (2) $Bel(\Theta) = 1$
  (3) For every positive integer n and every collection $A_1, \cdots, A_n$ of subsets of Θ,

$$Bel(A_1 \cup \cdots \cup A_n) \quad \geq \sum_{\substack{I \subset |1, \cdots, n| \\ I \neq \varnothing}} (-1)^{|I|+1} Bel\left(\bigcap_{i \in I} A_i\right).$$

Furthermore, the basic probability assignment, which produces a given belief function, is unique and can be recovered from the belief function:

- Theorem 5.2.1.2.2: Suppose $Bel: 2^{\Theta} \to [0,1]$ is the belief function given by the basic probability assignment $m: 2^{\Theta} \to [0,1]$. Then

$$m(A) \quad = \quad \sum_{B \subset A} (-1)^{|A-B|} Bel(B) \qquad (3)$$

  For all $A \subset \Theta$.

A subset 'A' of a frame Θ is called a *focal element* of a belief function Bel over Θ if $m(A) > 0$. The union of all the focal elements of a belief function is called its core.

5.2.1.2.1 Bayesian Belief Functions

In order to simplify the procedure, we decided to employ Bayesian belief functions. We are going to discuss Bayesian belief functions here. The first three of Bayes' rules can also be expressed in terms of a frame of discernment:

- Definition: if Θ is a frame of discernment, then a function $Bel: 2^{\Theta} \to [0,1]$ is called a Bayesian belief function if

(1) $Bel(\varnothing) = 0$,
(2) $Bel(\Theta) = 1$,
(3) $Bel(A \cup B) = Bel(A) + Bel(B)$ whenever $A, B \subset \Theta$ and $A \cap B = \varnothing$.

- Theorem 5.2.1.2.1.1: A Bayesian belief function is a belief function.

- Theorem 5.2.1.2.1.2: Suppose $Bel: 2^{\Theta} \to [0,1]$ is a belief function. Then the following assertions are equivalent:

  (1) Bel is Bayesian.
  (2) All of Bel's focal elements are singletons.

- Theorem 5.2.1.2.1.3: A function $Bel : 2^{\Theta} \to [0,1]$ is a Bayesian belief function if and only if there exists a function $p : \Theta \to [0,1]$ such that

$$\sum_{\theta \in \Theta} p(\theta) = 1, \qquad \text{and} \qquad Bel(A) = \sum_{\theta \in A} p(\theta)$$

  for all $A \subset \Theta$.

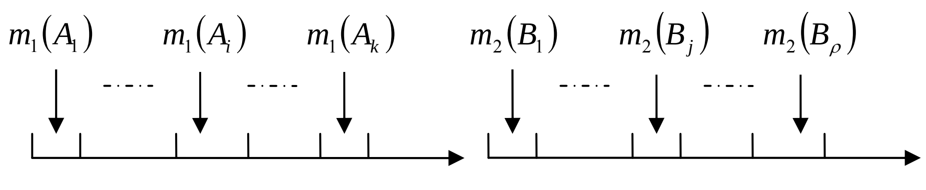
5.2.1.3 Dempster's Rule of Combination

The concept of data fusion was explained in world model chapter. Having a set of belief functions by AIBO robots, we are to combine these belief functions in order to be able to exploit them. Belief functions are adapted to the representation of evidence because they admit a genuine rule of combination. Given several belief functions over the same frame of discernment but based on distinct bodies of evidence, Dempster's rule of combination enables us to compute their orthogonal sum, a new belief function based on the combined evidence. In the special case of a frame of discernment containing only two elements, Depster's rule of combination was accurately stated and used by J. H. Lambert [9].

5.2.1.3.1 Combining Two Belief Functions

Dempster's rule is most accessible to the intuition when it is expressed in terms of the basic probability numbers, and especially when these basic probability numbers are depicted geometrically.

Suppose m_1 is the basic probability assignment for a belief function Bel_1 over a frame Θ, and denote Bel_1's focal elements by $A_1, \cdots, A_k$. Then the probability masses measured by the basic probability numbers $m_1(A_1), \cdots, m_1(A_k)$ can be depicted as segments of a line segment of length as segments of a line segment of length one, as in Figure 9.



In order to carry out the combination of Bel_1 and Bel_2, we now think of the square as representing our total probability mass and suppose that Bel_1 commits vertical strips to its focal elements, while Bel_2 commits horizontal strips to its focal

Figure 9: Probability Measures

elements. Figure 10 singles out, for example, a vertical strip of measure $m_1(A_i)$ that is exactly committed to A_i by m_1 and a horizontal strip of measure $m_2(B_j)$ that is exactly committed to B_j by m_2. The intersection of these two strips has measure $m_1(A_i)\times m_2(B_j)$, and since it is committed both to A_i and to B_j, we may say that the joint effect of Bel_1 and Bel_2 is to commit it exactly to $A_i \cap B_j$.

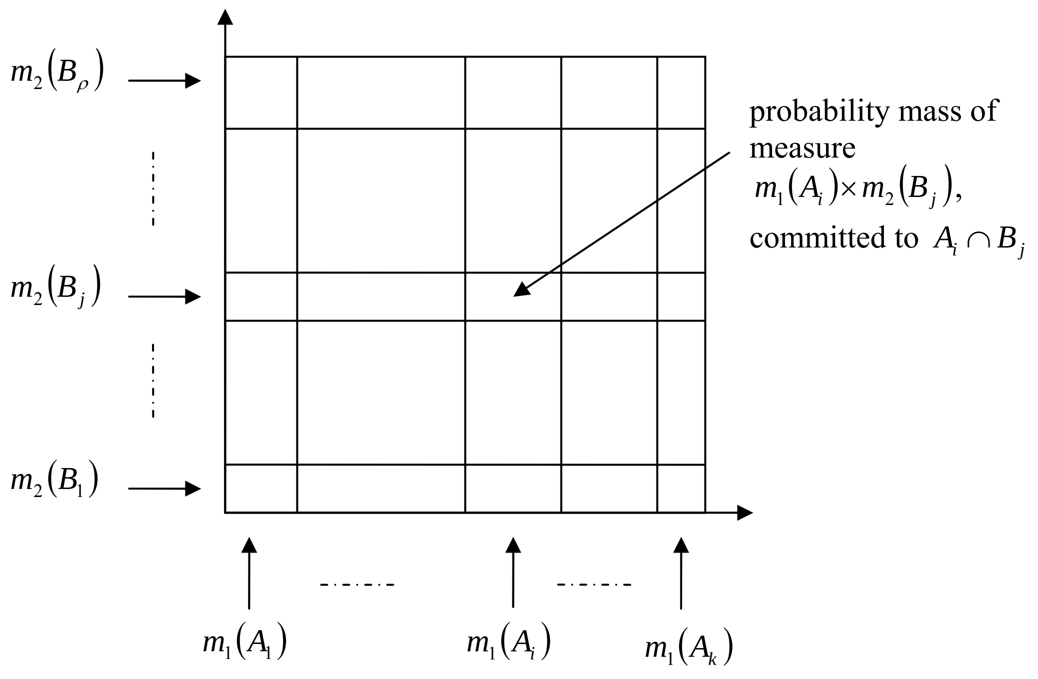

$m_2(B_\rho)$
$m_2(B_j)$
$m_2(B_1)$
probability mass of measure $m_1(A_i)\times m_2(B_j)$, committed to $A_i \cap B_j$
$m_1(A_1)$
$m_1(A_i)$
$m_1(A_k)$

Figure 10: Schema of Combining Two Belief Functions

Similarly, we can specify the exact commitment of every rectangle in Figure 10. A given subset A of Θ may have more than one of these rectangles exactly committed to it, of course; the total probability mass exactly committed to A will have measure

$$\sum_{\substack{i,j \\ A_i \cap B_j = A}} m_1(A_i)m_2(B_j).$$

The only difficulty with this schema is that it may commit some of the square to the empty set $\varnothing$. For there may well be a focal element A_i of Bel_1 and a focal element B_j of Bel_2 such that $A_i \cap B_j = \varnothing$, in which case

$$\sum_{\substack{i,j \\ A_i \cap B_j = \varnothing}} m_1(A_i)m_2(B_j) > 0.$$

The only remedy is to "discard" all the rectangles thus committed to $\varnothing$. If not all the rectangles are thus discarded, the measures of the remaining rectangles can then be inflated by multiplying them by the following factor.

$$\left(1-\sum_{\substack{i,j\\A_i\cap B_j=\varnothing}} m_1(A_i)m_2(B_j)\right)^{-1} \tag{4}$$

So that the total probability mass will again have measure one. As the next two theorems show, this construction does indeed lead to a new basic probability assignment, provided only that Bel_1 and Bel_2 do not flatly contradicted each other.

- Theorem 5.2.1.3.1.1: Suppose Bel_1 and Bel_2 are belief functions over the same frame Θ, with basic probability assignment m_1, m_2, and focal elements $A_1,\cdots,A_k$ and $B_1,\cdots,B_\rho$ respectively. Suppose

$$\sum_{\substack{i,j\\A_i\cap B_j=\varnothing}} m_1(A_i)m_2(B_j)<1$$

Then the function $m:2^\Theta\to[0,1]$ defined by $m(\varnothing)=0$ and

$$m(A) \quad = \quad \frac{\displaystyle\sum_{\substack{i,j\\A_i\cap B_j=A}} m_1(A_i)m_2(B_j)}{1-\displaystyle\sum_{\substack{i,j\\A_i\cap B_j=\varnothing}} m_1(A_i)m_2(B_j)} \tag{5}$$

for all non-empty $A\subset\Theta$ is a basic probability assignment. The core of the belief function given by m is equal to the intersection of the cores of Bel_1 and Bel_2.

The belief function given by m is called the orthogonal sum of Bel_1 and Bel_2 and is denoted $Bel_1\oplus Bel_2$. If Equation.4 does not hold, then we say that the orthogonal sum $Bel_1\oplus Bel_2$ does not exist.

- Theorem 5.2.1.3.1.2: Suppose Bel_1 and Bel_2 are belief functions over the same frame Θ, then the following conditions are all equivalent:

(1) $Bel_1\oplus Bel_2$ does not exist.
(2) The cores of Bel_1 and Bel_2 are disjoint.

5.2.1.3.2 Combining Several Belief Functions from AIBO robots

The rule of combination described in the preceding sections is a rule for combining a pair of belief functions, but by repeatedly applying it one can obviously combine any number of belief functions. Indeed, in order to combine a collection $Bel_1,\cdots,Bel_n$ of belief functions, one can form the pairwise orthogonal sums:

$$Bel_1 \oplus Bel_2,$$

$$(Bel_1 \oplus Bel_2) \oplus Bel_3,$$

$$((Bel_1 \oplus Bel_2) \oplus Bel_3) \oplus Bel_4,$$

etc. containing until all the Bel_i are included. If Dempster's rule meets its purpose, then each stage of this process should correspond to the addition of the evidence underlying the Bel_i entered at that stage, and the belief function Bel finally issuing from the process should represent the pooled evidence from all the Bel_i.

6 Decision Making

As explained in our architecture (chapter.2), Decision Making (DM) module plays the key role in logical decisions made by agents in a multi-agent environment such as AIBO soccer. DM is done in a completely distributed manner in "*Impossibles*" AIBO robots; however, communication is employed in order to propagate the information obtained from vision, sensors, and communication modules. Consequently, information is propagated by communication and decisions are made by agents themselves (Demonstrated in Figure 11). This section is organized as follows: The intra-DM architecture is explained in details in subsection 1. Team behavior is given in subsection 2. Subsection 3 is devoted to individual behaviors. Finally, goalie will be discussed independently in subsection 4.

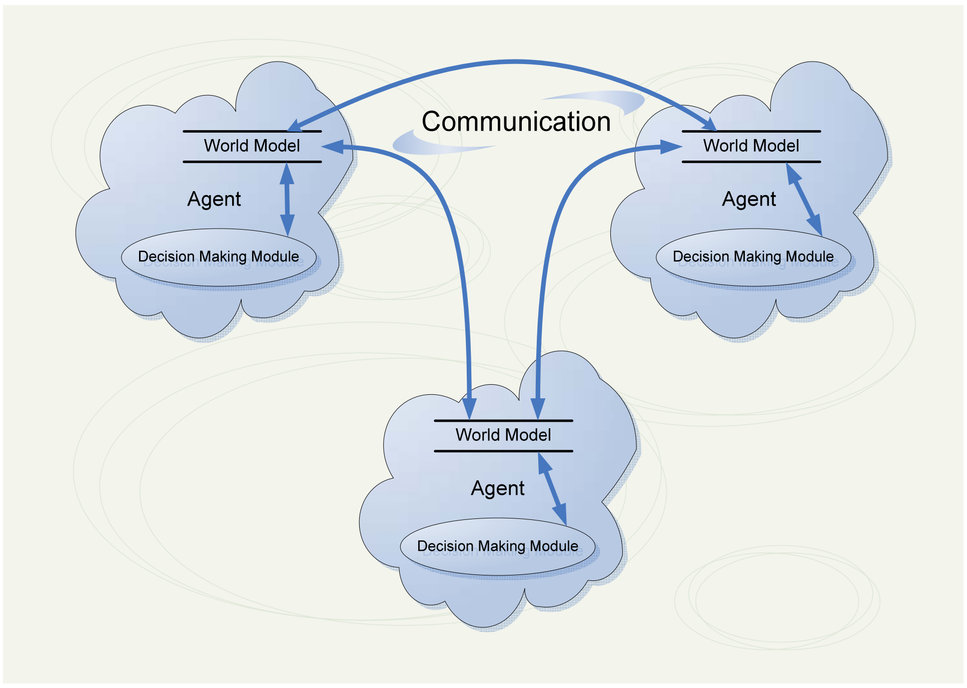


Figure 11: Distributed Reasoning with Information Communication

6.1 Architecture

Intra-DM module in *"Impossibles"* AIBO robots have a hierarchical layered architecture (shown in Figure 12). In fact, DM module consists of two major layers. Team Behavior (TB), i.e. tactics, layer is the highest one which determines the tactics of the soccer team. On the other words, team behavior layer plays a role similar to coach in real world soccer. Secondly, Individual Behavior (IB) layer is the techniques employed by individual players.

As demonstrated in Figure 12, Decision Making (DM) module gets its input from the system's world model including opponent players', teammates', and ball's locations accompanying with some degree of belief which is due to existing uncertainty in real system environment such as AIBO soccer.

Having gotten the inputs from its world model, the AIBO robot will analyze the input in a two-step procedure. Team behavior (TB) sub-module gets DM inputs from world-model and then resolves the whole team behavior, e.g. tactics stored in a database (Section 6.2.3). Finally, TB passes the team behavior and world model information to the lower layer that is Individual Behavior (IB).

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

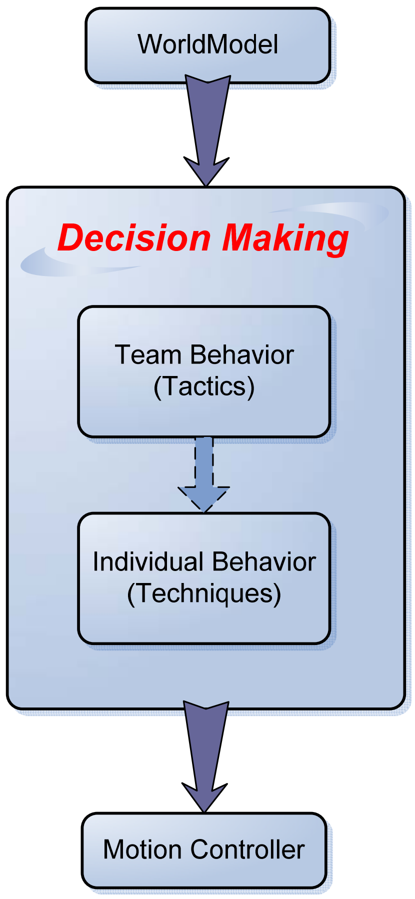


Figure 12: Decision Making Module Architecture

6.2 *Team Behavior*

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

6.2.1 Determining Factors

Tactics selection step needs two parameters. First, fuzzy membership degree in offense set is to be determined, i.e. Defense-Offense (DO). It is given in subsection 6.2.1.1. Teammates' and Players' sites is defined to be the second parameter (see subsection 6.2.1.2).

6.2.1.1 Defense vs. Offense

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

6.2.1.1.1 Caution and Risk

The first parameter which contributes to obtain DO is Caution-Risk (CR) fuzzy membership degree. We have employed a fuzzy logic controller (see Figure 13). In fact, the CR degree (demonstrated in Figure 18) is the output of a fuzzy controller which gets three inputs (shown in Figure 14 through Figure 16). The result of the game, time, and opponent's strength are the mentioned inputs. Fuzzification, inputs, rule base, outputs, and defuzzification of the fuzzy controller are shown in Figure 13 through Figure 17.

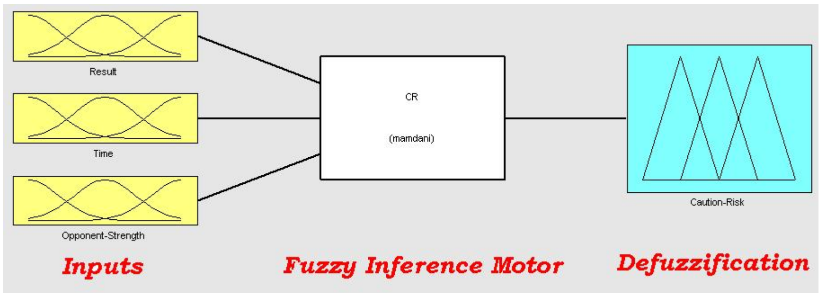


Figure 13: Caution-Risk (CR) Fuzzy Controller Scheme

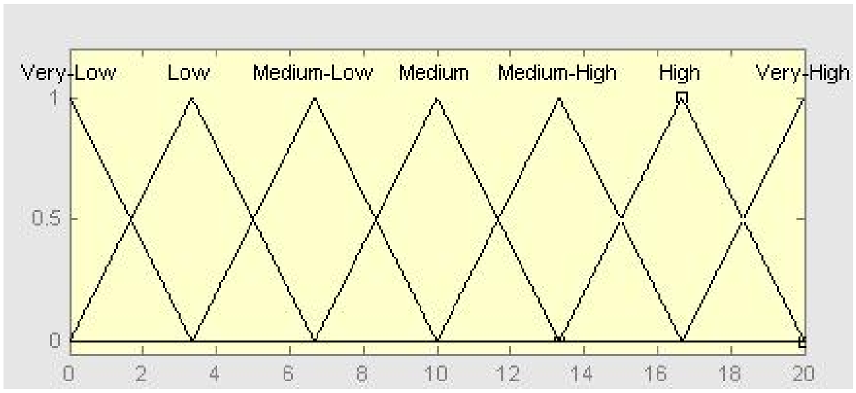


Figure 14: Time input of the CR Fuzzy Controller

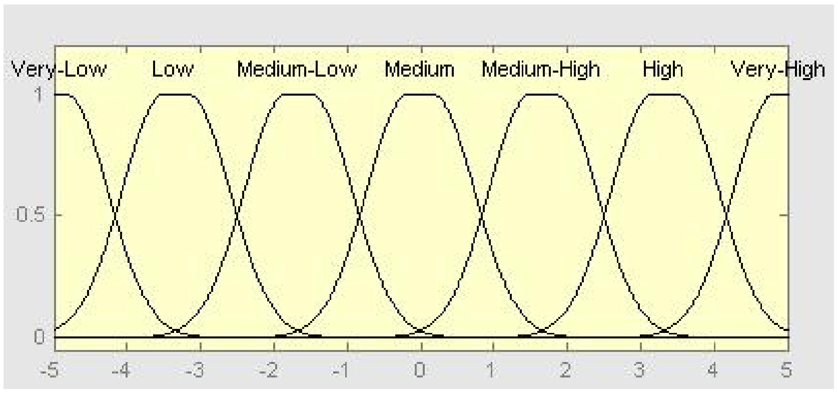


Figure 15: Result input of the CR Fuzzy Controller

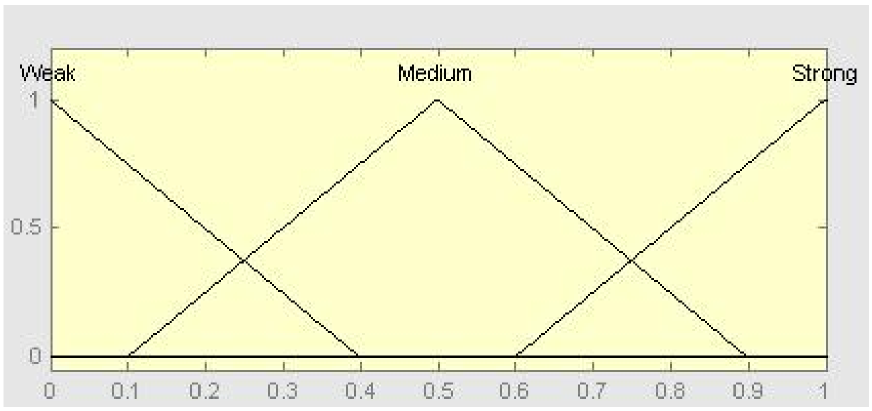
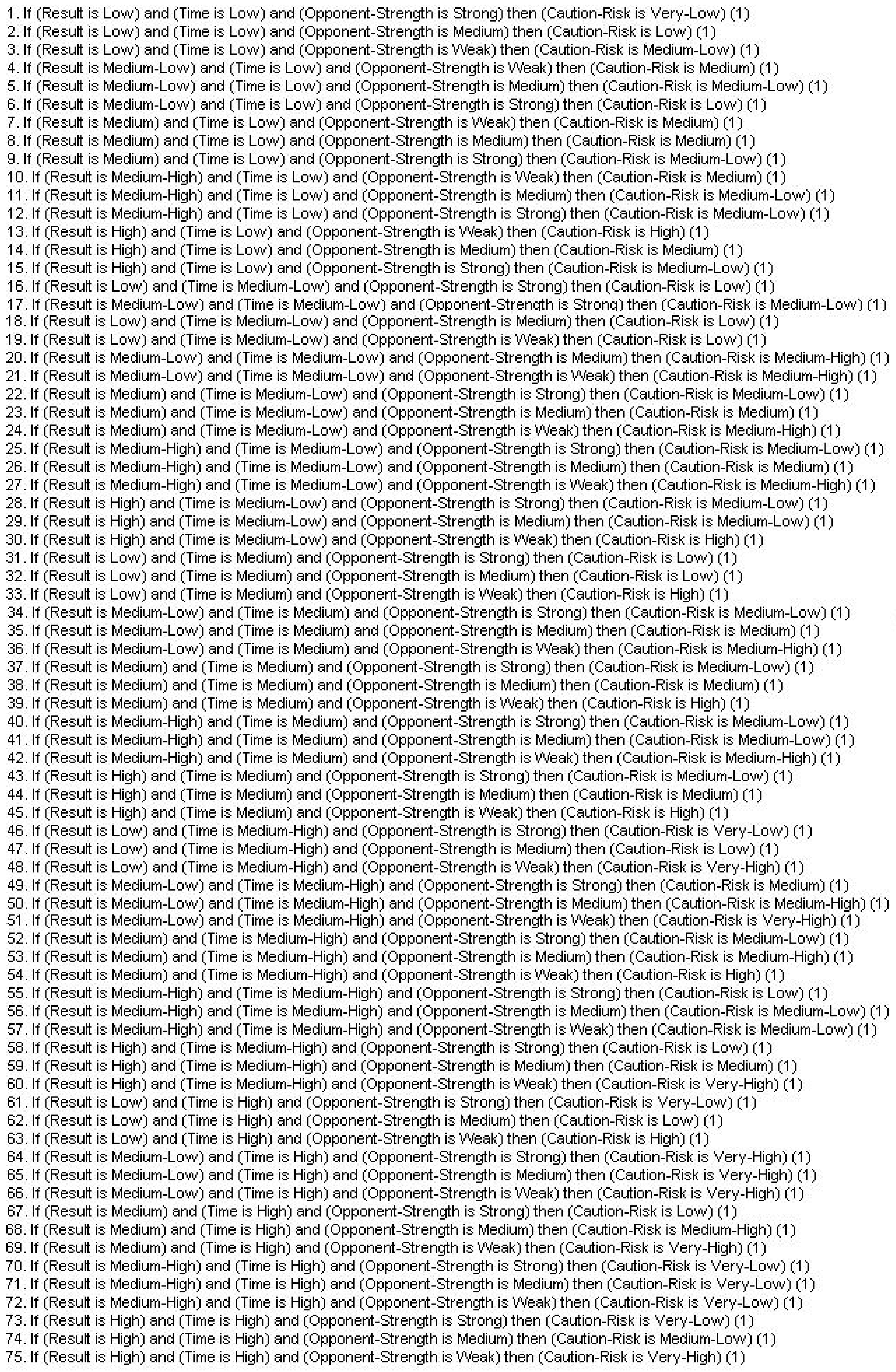


Figure 16: Opponent Strength input of the CR Fuzzy Controller

Passed time (Figure 14) in a game has been represented as seven fuzzy membership functions covering the whole range between 0 and 20. Figure 5 shows the result as an important factor determining the caution-risk (CR) fuzzy membership degree of final strategy. Seven Possibility Distribution Functions (PDFs) are employed to symbolize the result; however, the result input in fact is integer number which represents the difference between goals scored by two teams. For instance, if the result input is '-3', so our AIBO robots have scored three goals less than the opponent team.



1. If (Result is Low) and (Time is Low) and (Opponent-Strength is Strong) then (Caution-Risk is Very-Low) (1)
2. If (Result is Low) and (Time is Low) and (Opponent-Strength is Medium) then (Caution-Risk is Low) (1)
3. If (Result is Low) and (Time is Low) and (Opponent-Strength is Weak) then (Caution-Risk is Medium-Low) (1)
4. If (Result is Medium-Low) and (Time is Low) and (Opponent-Strength is Weak) then (Caution-Risk is Medium) (1)
5. If (Result is Medium-Low) and (Time is Low) and (Opponent-Strength is Medium) then (Caution-Risk is Medium-Low) (1)
6. If (Result is Medium-Low) and (Time is Low) and (Opponent-Strength is Strong) then (Caution-Risk is Low) (1)
7. If (Result is Medium) and (Time is Low) and (Opponent-Strength is Weak) then (Caution-Risk is Medium) (1)
8. If (Result is Medium) and (Time is Low) and (Opponent-Strength is Medium) then (Caution-Risk is Medium) (1)
9. If (Result is Medium) and (Time is Low) and (Opponent-Strength is Strong) then (Caution-Risk is Medium-Low) (1)
10. If (Result is Medium-High) and (Time is Low) and (Opponent-Strength is Weak) then (Caution-Risk is Medium) (1)
11. If (Result is Medium-High) and (Time is Low) and (Opponent-Strength is Medium) then (Caution-Risk is Medium-Low) (1)
12. If (Result is Medium-High) and (Time is Low) and (Opponent-Strength is Strong) then (Caution-Risk is Medium-Low) (1)
13. If (Result is High) and (Time is Low) and (Opponent-Strength is Weak) then (Caution-Risk is High) (1)
14. If (Result is High) and (Time is Low) and (Opponent-Strength is Medium) then (Caution-Risk is Medium) (1)
15. If (Result is High) and (Time is Low) and (Opponent-Strength is Strong) then (Caution-Risk is Medium-Low) (1)
16. If (Result is Low) and (Time is Medium-Low) and (Opponent-Strength is Strong) then (Caution-Risk is Low) (1)
17. If (Result is Medium-Low) and (Time is Medium-Low) and (Opponent-Strength is Strong) then (Caution-Risk is Medium-Low) (1)
18. If (Result is Low) and (Time is Medium-Low) and (Opponent-Strength is Medium) then (Caution-Risk is Low) (1)
19. If (Result is Low) and (Time is Medium-Low) and (Opponent-Strength is Weak) then (Caution-Risk is Low) (1)
20. If (Result is Medium-Low) and (Time is Medium-Low) and (Opponent-Strength is Medium) then (Caution-Risk is Medium-High) (1)
21. If (Result is Medium-Low) and (Time is Medium-Low) and (Opponent-Strength is Weak) then (Caution-Risk is Medium-High) (1)
22. If (Result is Medium) and (Time is Medium-Low) and (Opponent-Strength is Strong) then (Caution-Risk is Medium-Low) (1)
23. If (Result is Medium) and (Time is Medium-Low) and (Opponent-Strength is Medium) then (Caution-Risk is Medium) (1)
24. If (Result is Medium) and (Time is Medium-Low) and (Opponent-Strength is Weak) then (Caution-Risk is Medium-High) (1)
25. If (Result is Medium-High) and (Time is Medium-Low) and (Opponent-Strength is Strong) then (Caution-Risk is Medium-Low) (1)
26. If (Result is Medium-High) and (Time is Medium-Low) and (Opponent-Strength is Medium) then (Caution-Risk is Medium) (1)
27. If (Result is Medium-High) and (Time is Medium-Low) and (Opponent-Strength is Weak) then (Caution-Risk is Medium-High) (1)
28. If (Result is High) and (Time is Medium-Low) and (Opponent-Strength is Strong) then (Caution-Risk is Medium-Low) (1)
29. If (Result is High) and (Time is Medium-Low) and (Opponent-Strength is Medium) then (Caution-Risk is Medium-Low) (1)
30. If (Result is High) and (Time is Medium-Low) and (Opponent-Strength is Weak) then (Caution-Risk is High) (1)
31. If (Result is Low) and (Time is Medium) and (Opponent-Strength is Strong) then (Caution-Risk is Low) (1)
32. If (Result is Low) and (Time is Medium) and (Opponent-Strength is Medium) then (Caution-Risk is Low) (1)
33. If (Result is Low) and (Time is Medium) and (Opponent-Strength is Weak) then (Caution-Risk is High) (1)
34. If (Result is Medium-Low) and (Time is Medium) and (Opponent-Strength is Strong) then (Caution-Risk is Medium-Low) (1)
35. If (Result is Medium-Low) and (Time is Medium) and (Opponent-Strength is Medium) then (Caution-Risk is Medium) (1)
36. If (Result is Medium-Low) and (Time is Medium) and (Opponent-Strength is Weak) then (Caution-Risk is Medium-High) (1)
37. If (Result is Medium) and (Time is Medium) and (Opponent-Strength is Strong) then (Caution-Risk is Medium-Low) (1)
38. If (Result is Medium) and (Time is Medium) and (Opponent-Strength is Medium) then (Caution-Risk is Medium) (1)
39. If (Result is Medium) and (Time is Medium) and (Opponent-Strength is Weak) then (Caution-Risk is High) (1)
40. If (Result is Medium-High) and (Time is Medium) and (Opponent-Strength is Strong) then (Caution-Risk is Medium-Low) (1)
41. If (Result is Medium-High) and (Time is Medium) and (Opponent-Strength is Medium) then (Caution-Risk is Medium-Low) (1)
42. If (Result is Medium-High) and (Time is Medium) and (Opponent-Strength is Weak) then (Caution-Risk is Medium-High) (1)
43. If (Result is High) and (Time is Medium) and (Opponent-Strength is Strong) then (Caution-Risk is Medium-Low) (1)
44. If (Result is High) and (Time is Medium) and (Opponent-Strength is Medium) then (Caution-Risk is Medium) (1)
45. If (Result is High) and (Time is Medium) and (Opponent-Strength is Weak) then (Caution-Risk is High) (1)
46. If (Result is Low) and (Time is Medium-High) and (Opponent-Strength is Strong) then (Caution-Risk is Very-Low) (1)
47. If (Result is Low) and (Time is Medium-High) and (Opponent-Strength is Medium) then (Caution-Risk is Low) (1)
48. If (Result is Low) and (Time is Medium-High) and (Opponent-Strength is Weak) then (Caution-Risk is Very-High) (1)
49. If (Result is Medium-Low) and (Time is Medium-High) and (Opponent-Strength is Strong) then (Caution-Risk is Medium) (1)
50. If (Result is Medium-Low) and (Time is Medium-High) and (Opponent-Strength is Medium) then (Caution-Risk is Medium-High) (1)
51. If (Result is Medium-Low) and (Time is Medium-High) and (Opponent-Strength is Weak) then (Caution-Risk is Very-High) (1)
52. If (Result is Medium) and (Time is Medium-High) and (Opponent-Strength is Strong) then (Caution-Risk is Medium-Low) (1)
53. If (Result is Medium) and (Time is Medium-High) and (Opponent-Strength is Medium) then (Caution-Risk is Medium-High) (1)
54. If (Result is Medium) and (Time is Medium-High) and (Opponent-Strength is Weak) then (Caution-Risk is High) (1)
55. If (Result is Medium-High) and (Time is Medium-High) and (Opponent-Strength is Strong) then (Caution-Risk is Low) (1)
56. If (Result is Medium-High) and (Time is Medium-High) and (Opponent-Strength is Medium) then (Caution-Risk is Medium-Low) (1)
57. If (Result is Medium-High) and (Time is Medium-High) and (Opponent-Strength is Weak) then (Caution-Risk is Medium-Low) (1)
58. If (Result is High) and (Time is Medium-High) and (Opponent-Strength is Strong) then (Caution-Risk is Low) (1)
59. If (Result is High) and (Time is Medium-High) and (Opponent-Strength is Medium) then (Caution-Risk is Medium) (1)
60. If (Result is High) and (Time is Medium-High) and (Opponent-Strength is Weak) then (Caution-Risk is Very-High) (1)
61. If (Result is Low) and (Time is High) and (Opponent-Strength is Strong) then (Caution-Risk is Very-Low) (1)
62. If (Result is Low) and (Time is High) and (Opponent-Strength is Medium) then (Caution-Risk is Low) (1)
63. If (Result is Low) and (Time is High) and (Opponent-Strength is Weak) then (Caution-Risk is High) (1)
64. If (Result is Medium-Low) and (Time is High) and (Opponent-Strength is Strong) then (Caution-Risk is Very-High) (1)
65. If (Result is Medium-Low) and (Time is High) and (Opponent-Strength is Medium) then (Caution-Risk is Very-High) (1)
66. If (Result is Medium-Low) and (Time is High) and (Opponent-Strength is Weak) then (Caution-Risk is Very-High) (1)
67. If (Result is Medium) and (Time is High) and (Opponent-Strength is Strong) then (Caution-Risk is Low) (1)
68. If (Result is Medium) and (Time is High) and (Opponent-Strength is Medium) then (Caution-Risk is Medium-High) (1)
69. If (Result is Medium) and (Time is High) and (Opponent-Strength is Weak) then (Caution-Risk is Very-High) (1)
70. If (Result is Medium-High) and (Time is High) and (Opponent-Strength is Strong) then (Caution-Risk is Very-Low) (1)
71. If (Result is Medium-High) and (Time is High) and (Opponent-Strength is Medium) then (Caution-Risk is Very-Low) (1)
72. If (Result is Medium-High) and (Time is High) and (Opponent-Strength is Weak) then (Caution-Risk is Very-Low) (1)
73. If (Result is High) and (Time is High) and (Opponent-Strength is Strong) then (Caution-Risk is Very-Low) (1)
74. If (Result is High) and (Time is High) and (Opponent-Strength is Medium) then (Caution-Risk is Medium-Low) (1)
75. If (Result is High) and (Time is High) and (Opponent-Strength is Weak) then (Caution-Risk is Very-High) (1)

Figure 17: Rule-Base employed for CR Fuzzy Controller

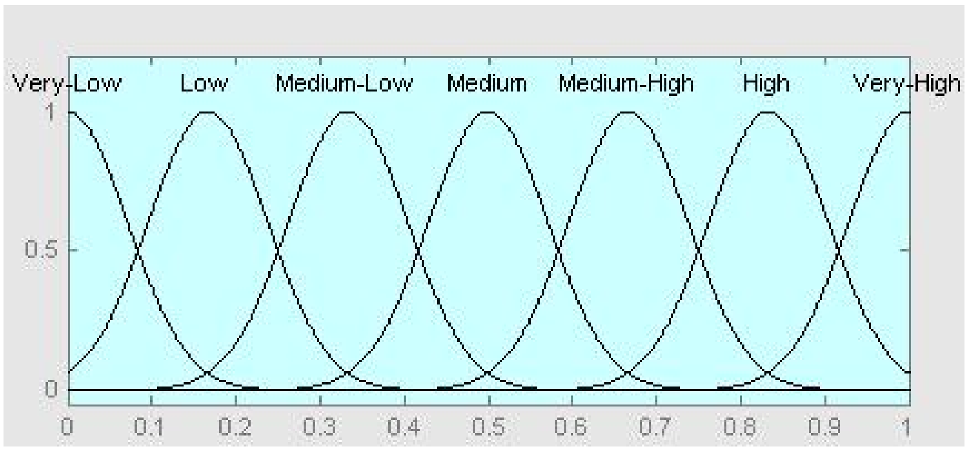


Figure 18: Caution-Risk output of CR fuzzy Controller

To prevent repeating the above explanations, Figure 18 is not given in details. Lastly, Caution-Risk (CR) fuzzy logic controller produces the following surfaces (from Figure 19 to Figure 21) using its Rule-Base (Figure 17). Figure 19 illustrates caution-risk membership degree of variable time and result against a fixed opponent. The most interesting part of the Figure 19 is moving along the line 'Result = -2'. As demonstrated below, our AIBO robots will attack in the middle first half time of the game; however, they will choose a more defensive strategy around minute 10 in order to let opponent team players to distribute all over the field to make it possible to find more open spaces in the opponent's defensive third. The team will switch to the full-attack strategy if it seems impossible for the robots to score. As shown in Figure 19 robots with 'Result = -2 & time = 19' will attack completely to score, because the team hope to get the medium result.

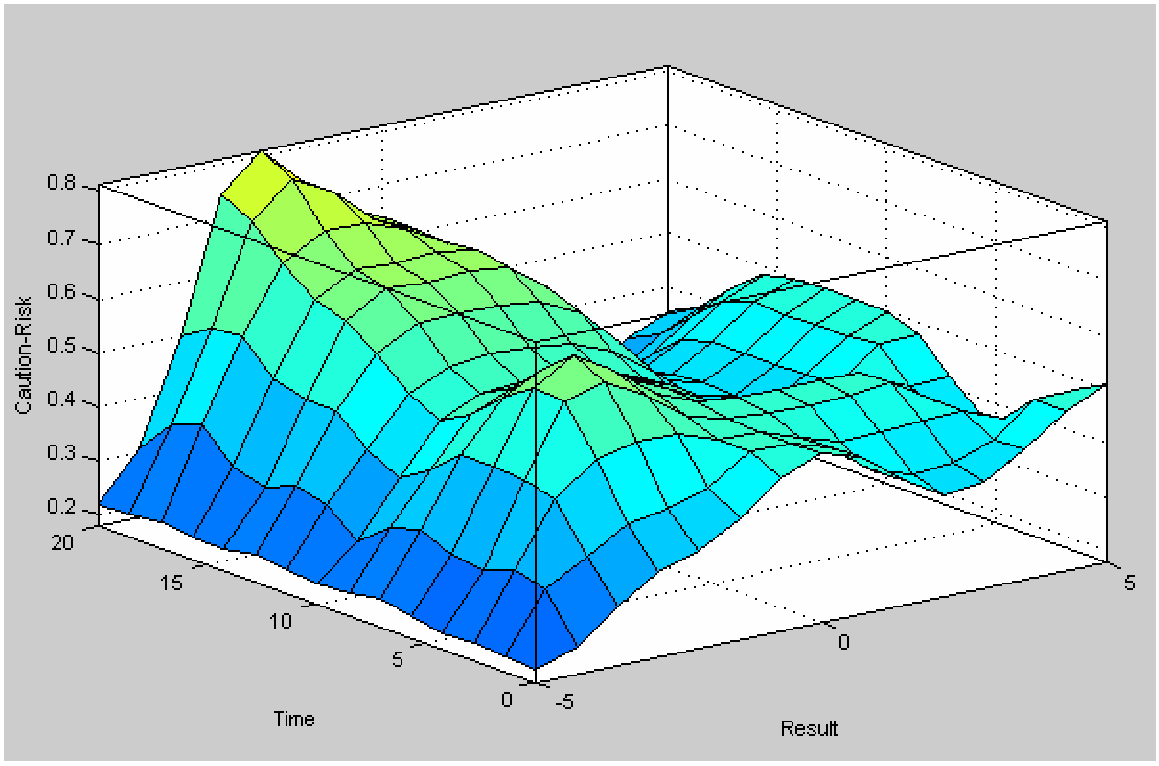


Figure 19: CR(Time, Result) with a fixed Opponent-Strength

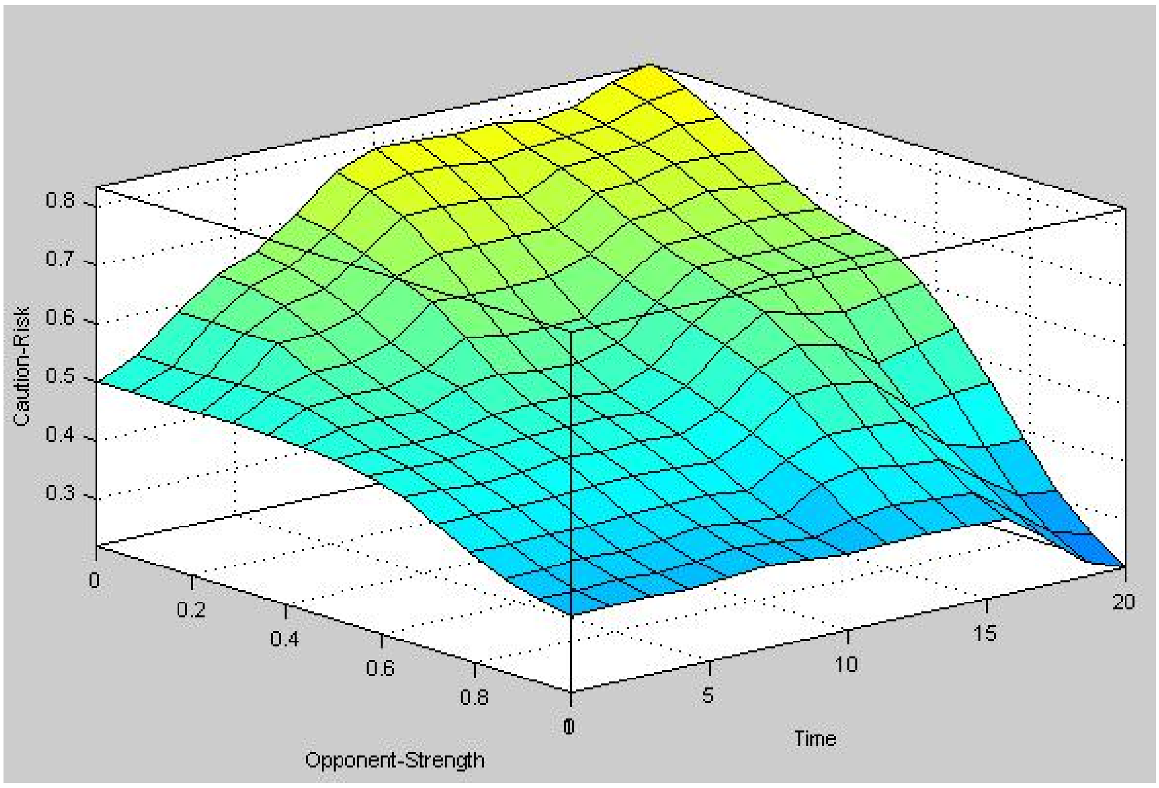


Figure 20: CR(Opponent-Strength, Time) with a fixed Result

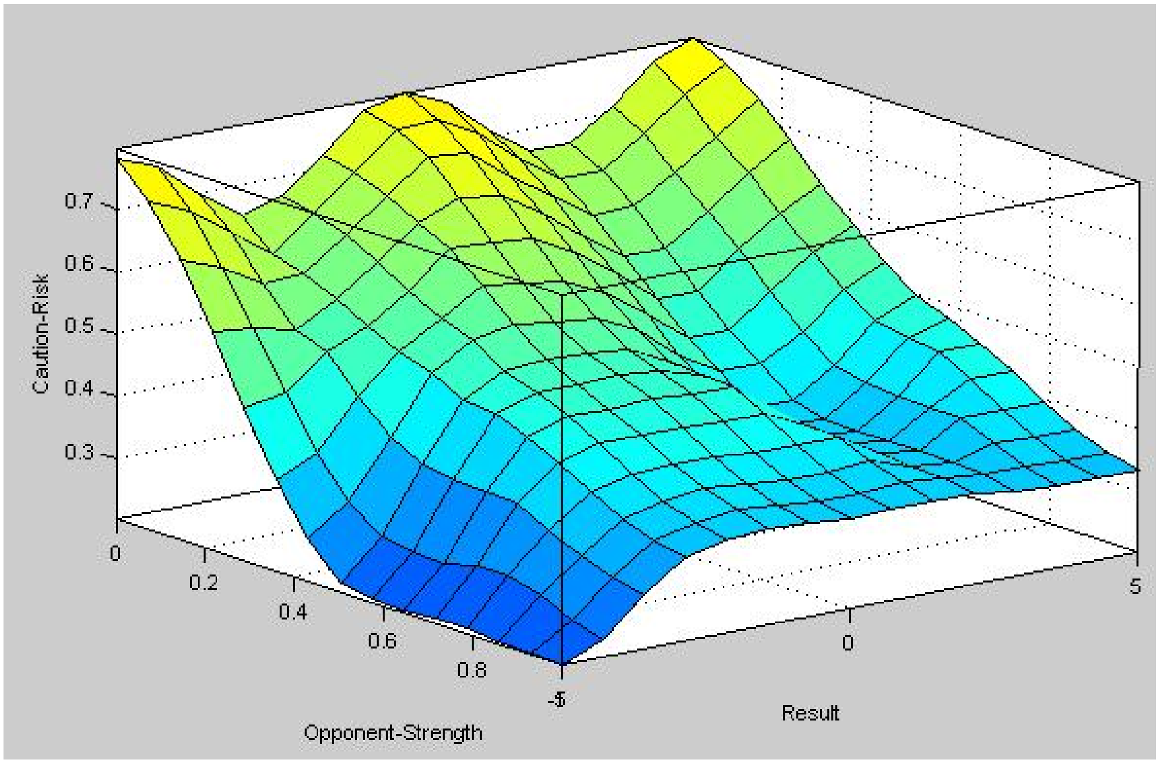


Figure 21: CR(Result, Opponent-Strength) in a fixed Time

6.2.1.1.2 Ball Ownership

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

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

$$BO(Team_i) = \sum_{m_j \in Team_i} \frac{1}{d_j^{\alpha}} \qquad (6)$$

The final Ball Ownership (BO) factor is obtained by means of the following equation as it follows:

$$BO = \frac{BO\,(\text{Our Team})}{BO\,(\text{Opponent Team})} \qquad (7)$$

6.2.1.1.3 Hyperbolic Danger Safety Degree

As explained above, Ball Ownership (BO) is a factor is due to players' rational locations to the ball; however, players' absolute locations are also important. In order to accomplish the job a Hyperbolic Factor (HF) is defined.

A hyperbola [10] is a conic section defined as the locus of all points P in the plane the difference of whose distances $r_1 = F_1P$ and $r_2 = F_2P$ from two fixed points (the foci F_1 and F_2) separated by a distance $2c$ is a given positive constant k [11],

$$r_2 - r_1 = k\,.$$

Letting P fall on the left x-intercept requires that

$$k = (c + a) - (c - a) = 2a\,.$$

So the constant is given by $k = 2a$, i.e., twice the distance between the x-intercepts (left figure below).

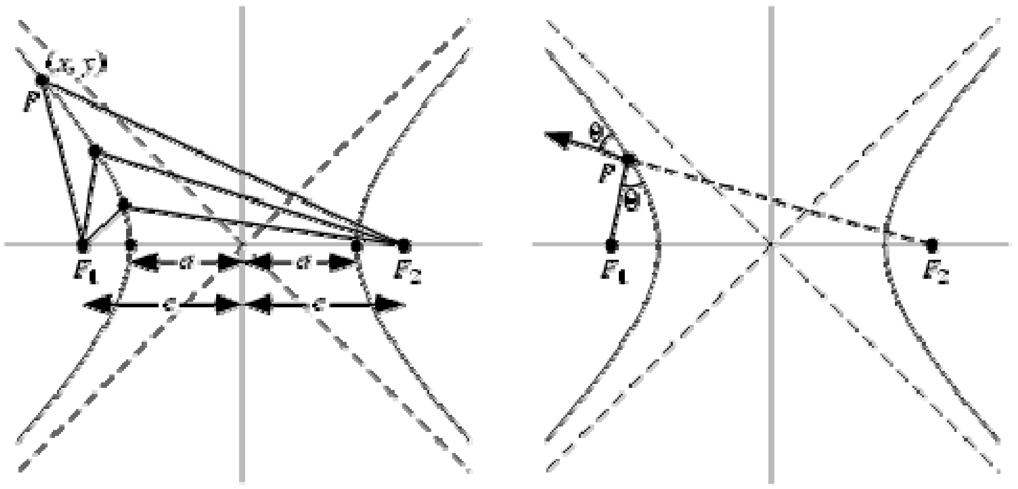


Figure 22: Hyperbola used to calculate Hyperbolic Factor (HF)

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

$$DD(P_i) = \frac{\text{distance}(P_i, \text{opponent Goal}) - \text{distance}(P_i, \text{our Goal})}{Field\_Length} \tag{8}$$

$$SD(P_i) = \frac{\text{distance}(P_i, \text{our Goal}) - \text{distance}(P_i, \text{opponent Goal})}{Field\_Length} \tag{9}$$

According to above equations, points on a hyperbolic locus with a constant 'a' will have equal Danger Degrees (DD) in the case of having an opponent player in the point. Similarly, the points have equal Safety Degree (SD) if one of our team members is situated in one of those points.

The final Hyperbolic Danger-Safety Degree (HDSD) factor is calculated employing players individual Danger Degree (DD), or Safety Degree (SD).

$$HDSD = \frac{f_1(SD(\text{our team members}))}{f_2(DD(\text{opponent team players}))} \tag{10}$$

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

$$HDSD = \frac{\sum_{m_i \in \text{our team}} SD(m_i)}{\sum_{m_i \in \text{opponent team}} DD(m_i)} \tag{11}$$

6.2.1.2 Team Fear-Relax Emotional Degree

As explained in 1.2.1.1, team behavior is determined according to Defense-Offense Degree (DOD) which lies in the range of 0 to 1. In above subsections, three determining factors were defined: Caution-Risk (CR), Ball Ownership (BO), and Hyperbolic Safety Danger Degree (HDSD). Now these three parameters are to be combined to represent the final Team Behavior Defense Offense Degree (TBDOD).

6.2.2 Team Strategy Database

In AIBO robots, we usually face CPU over usage problems; in contrast, memory over usage does not seem to be a critical problem. In order to avoid having CPU over usage, we save predefined team strategies in a Team Strategy Database (TSD). In each moment of the game, a linear combination of the proper strategies is computed based on TBDOD and players’ locations. Selecting from TSD offers two priorities over computing dynamic team strategies. First it supports to have a more flexible team behavior, because further team strategies can be added to TSD later. Secondly, this approach helps to decreases the CPU usage.

Generally, team strategies are categorized into three groups. Defensive strategies, midfield strategies, and offensive strategies are the mentioned groups. Two independent defensive team strategies (DTS) of *“Impossibles”* AIBO robots are presented. Figure 13 demonstrates the first DTS in which our players try to defend opponent’s forward players reaching the goal along a line from the center of the field to our goal. It is the most defensive strategy of the team employed in critical circumstances.

Note that the following surfaces represent the values of points on the soccer field according to the team strategy. For instance, the dark red points in the diagrams indicate the most important regions of the field. On the other hand, the blues ones denote the regions which are not considered as significant regions. To clarify the problem it may be useful to declare that (0, 2.7) is the center of our goal.

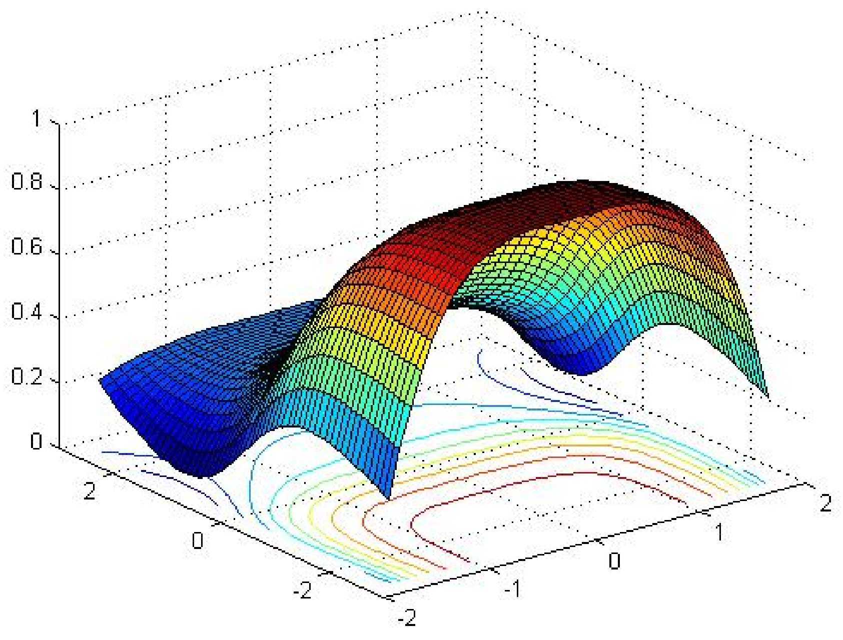


Figure 23: the maximum defensive (MD) team strategy

Figure 24 demonstrates the general defensive (GD) strategy of the *"Impossibles"* AIBO robots employed to some objectives such as preventing the game result being changed.

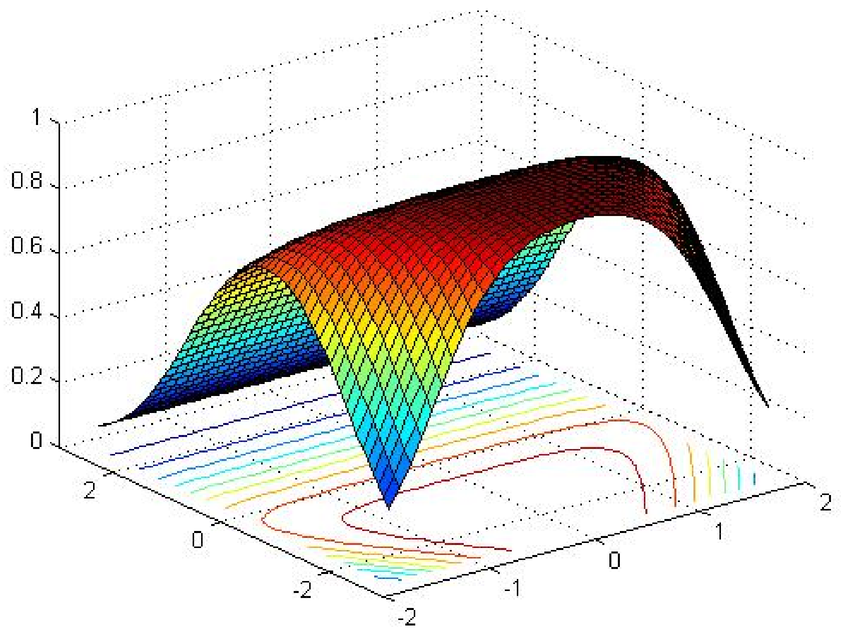


Figure 24: General Defensive (GD) team strategy

Midfield and offensive team strategies of *"Impossibles"* are in fact generated easily as a combination of the following basic strategies (from Figure 25 to Figure 28).

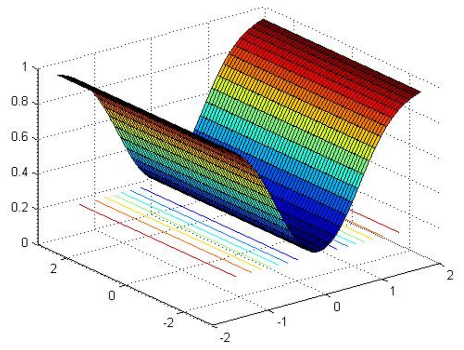


Figure 25: Team strategy 1

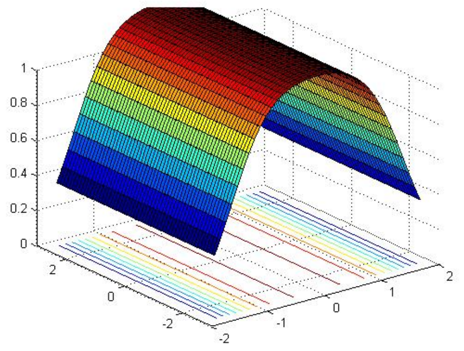


Figure 26: Team strategy 2

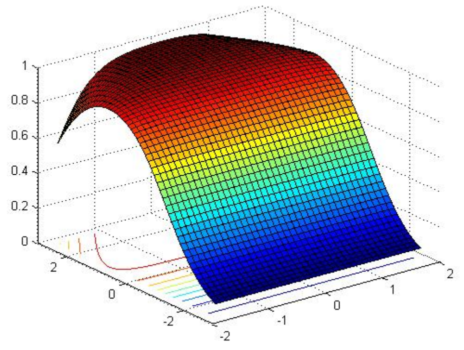


Figure 27: Team strategy 3

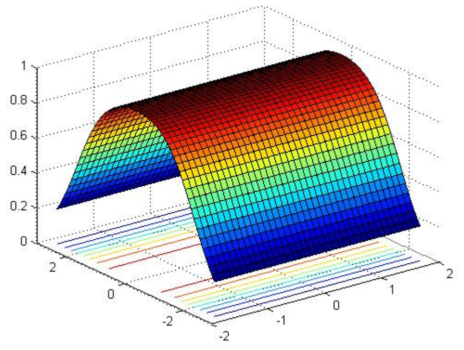


Figure 28: Team strategy 4

The following equations are employed to generate the above shown team strategies.

$$MD = \left(0.4 + 0.03 \times (Y-3)^{0.8} \times X^{4} + \left(\frac{1}{1+e^{2\times Y}}\right)\right) \times 0.7$$

$$GD = \left(1 - \left(\frac{1}{1+e^{-1.5\times Y+1.5}}\right)\right) + \left(\frac{1}{1+e^{-1.5(2\times X+Y+5.5)}}\right) + \left(\frac{1}{1+e^{-1.5\times(-2\times X+Y+5.5)}}\right) - 2$$

$$W = \left(\frac{1}{1+e^{-3\times(X-0.7)}}\right) + \left(\frac{1}{1+e^{3\times(X+0.7)}}\right)$$

$$NW = \left(\left(\frac{1}{1+e^{3\times(X-1.5)}}\right) + \left(\frac{1}{1+e^{-3\times(X+1.5)}}\right) - 1\right) \times 1.25$$

$$C = \left(\frac{1}{1+e^{2\times(Y-2)}}\right) + \left(\frac{1}{1+e^{-2\times(Y+2)}}\right) - 1$$

$$A = \left(1-\left(\frac{1}{1+e^{1.5\times Y+1.5}}\right)\right)+\left(\frac{1}{1+e^{-1.5\times(2\times X-Y+6.5)}}\right)+\left(\frac{1}{1+e^{-1.5\times(-2\times X-Y+6.5)}}\right)-2$$

In Figure 29 to Figure 32 denote midfield and offensive strategies of the team. These strategies are generated using the above basic strategies. Here we have employed the simplest operation, i.e. multiplication, to produce midfield and offensive team behaviors.

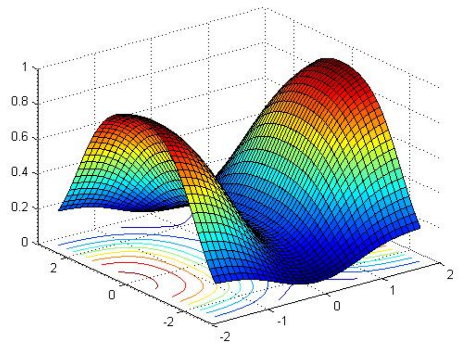


Figure 29: Midfield Team Strategy 1

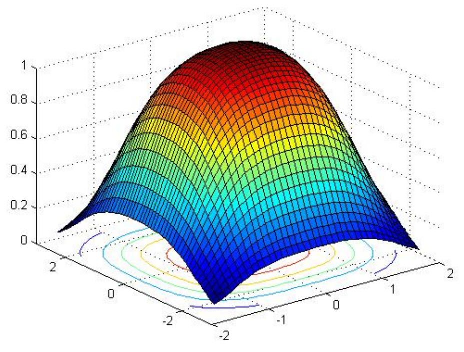


Figure 30: Midfield Team Strategy 2

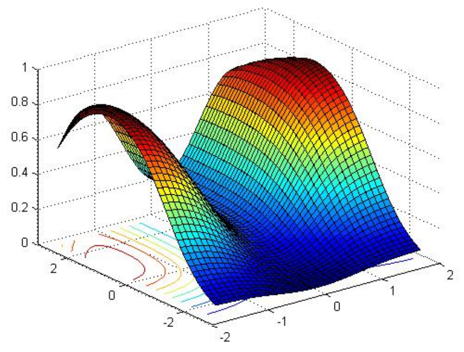


Figure 31: Offensive Team Strategy 1

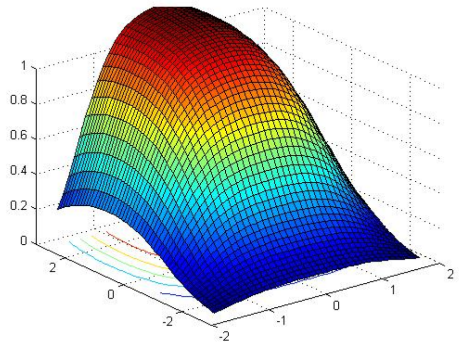


Figure 32: Offensive Team Strategy 2

6.2.3 Emotional Tactics Selection

One of the most important aspects of human decision making is the role of emotions in its behavior and reactions [12]. Humans choose their actions and make their decisions due to several internal variables called emotions. Emotional decision making is employed by humankind to avoid time-consuming and computationally-expensive approaches, e.g. using mathematical equations in decision making, to optimize the final result in critical circumstances such as danger [13].

In AIBO soccer environment, CPU usage is a critical criterion; while memory usage is not. So in order to reduce the computation burden of the team behavior generation, we exploit Emotional Tactics Selection (ETS) approach. As presented in [12], agents' Emotional Decision Making (EDM) is based on their different states, called emotions. As explained in 6.2.1.1.2, up to now the robots have calculated the fuzzy emotional Fear-Relax (FR) degree of the whole team. Given this degree, robots are to compute the team strategy, i.e. a linear combination of Team Strategy Database (TSD) elements. Fundamentally, in [12], transitions from one emotional state to the other are thought of being a gradual change, i.e. not suddenly, as it is done in real world animals and human. In other words, emotions are believed to have inertia while changing. Therefore, we avoid quantization of the given FR degree. In contrast, based on the given FR degree, a linear combination of the strategies in TSD is computed as the whole team behavior, i.e. team strategy.

6.3 Individual Behaviors (Techniques)

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

6.3.1 Predefined Dynamic Assigned Regions

Generally, in multi-agent systems, decision making can be accomplished using one of these four solutions [14]: (1) No Sharing Decision Making, (2) Information Sharing Decision Making, (3) Centralized Decision Making, (4) Fully Centralized Decision making. *"Impossibles"* AIBO robots use the second above approach. In this method, communication is employed just for transporting the information; therefore, neither commands nor decisions made by center are transmitted.

With us using Information Sharing decision making solution, the most critical problem was similar behaviors of the robots in the same situations. In other words, cooperation of the robots was not supported. So robots are assigned predefined roles as in real world soccer.

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

$$\text{Tendency}(P_i) \;=\; K \times \text{distance}(\text{Location}(P_i),\ \text{Assigned\_Region}(P_i))^{\alpha} \qquad (12)$$

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

6.3.2 Outputs as Decision Making-Motion Engine Interface

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

MC modules. In this section the employed Individual Behaviors are explained logically.

6.3.2.1 Looking

The objects stored in World Model (WM) own a saved parameter called Update Time (UT). It denotes the last time when a particular object has been seen by an agent. So it may be necessary for agents to refresh their knowledge about their surroundings limited to their vision capability, i.e. approximately 1.5 meters. 'LOOK' is called by Decision Making (DM) module with an argument denoting the coordination or direction of the region to be looked. When calculating this direction, three major parameters are considered. First of all, possibility degree to which a particular region is seeable. Possibility degree itself is calculated using the agent's distance from the region to be looked and other players acting as obstacles. Moreover, the turning cost which the agent pay in order to look at the region. Additionally, importance which the region has is to be mulled over. For instance, looking at the ball is usually thought to a significant look.

6.3.2.2 Walking & Running

Walking and Running as two basic categorizations of motion should be implemented efficiently because of their importance. A lot of learning algorithms on AIBOs have been presented during the past few years [15, and 16].

In *"Impossibles"* AIBOs, Walking and Running are called via just one command, i.e. 'WALK'. 'WALK' takes three arguments. First, a trajectory is passed through which robot is to walk to the lower layer module, i.e. Motion Controller (MC). Trajectory is represented by an array of points in the field. Moreover, a flag is set to either 'true' or 'false' as the second argument. In fact, this flag shows if the player owns the ball while moving or not. Finally, a power degree is passed as the third argument. Power takes fuzzy values between 0 and 1, where 1 represents the highest speed with which an agent can run.

6.3.2.3 Ball Grabbing

In order to get the ball ownership, Decision Making (DM) module of a player passes 'GRAB' to the lower level module, i.e. Motion Controller (MC). In fact, 'GRAB' takes no argument as input, because the motion controller gets one of its inputs from the World Model (WM) of the agent; therefore, no extra information is required to grab the moving/stationary ball on the field.

6.3.2.4 Shooting & Passing

Like 'WALK', Shooting and Passing are both called through 'SHOOT' which takes two arguments. Direction and the power degree are passed as the arguments of 'SHOOT'. According to the above explained details, no further details are needed here to clarify.

6.3.2.5 Blocking

As a defensive behavior, 'BLOCKING' has been implemented. Its only argument is the direction. 'BLOCKING' is explained in details in Motion Controller (MC) module chapter of this article.

7 Vision

According to *“Impossibles”* AIBO architecture (Chapter 2), each robot updates its world model using three inputs from its surroundings. Sensors, wireless communication, and vision are responsible for making these inputs ready to exploit; however, sound is also addable to the architecture which has not been implemented yet. In fact, the sensors and vision build the main inputs of the robot, because communication is only employed to propagate the information which is gathered by vision and sensors. Therefore, vision, as one the input modules of the architecture, plays a chief role in gathering information from the surroundings of the robot.

Vision in AIBO robots is in charge of receiving two inputs and producing a set of two outputs [17]. These inputs and outputs are as follows:

1. *Inputs:*
   - A stream of images taken by robot’s camera. Surely, these images contain a large amount of noise which has been caused by some issues such as robot’s motion or distance of the objects in image from the robot’s location in the field.
   - AIBO robots’ sensors provide us with a set of joints’ angles over time. So, direction of the camera and current condition of the robot is identified using this type of input.

2. *Outputs:*
   - Distances and angles to a fixed set of color-coded objects with known locations, which can be used to *localize* the robot on the field.
   - Distances and angles for a varying set of mobile objects.

Vision module of the *“Impossibles”* AIBO architecture (Chapter 2) will be clarified in this chapter. The following subsections present different approaches implemented in our vision subsystem. These approaches are categorized into to groups: General Vision Subsystem (GVS), and Specific Vision Subsystem (SVS) which are going to be explained later in this chapter. The GVS algorithms are mainly based on the UT Austin Villa vision system [18, and 17]. We have implemented SVS approaches which cannot be employed generally by robots, because of their overtime-consumption. Hence, SVS approaches are used in special cases which will be given in the following subsections.

7.1 Architecture

Vision Module Architecture (VMA) will be briefly explained here to make the later subsections easy to understand. Generally, VMA consists of three major subsystems. Case Detection (CD) submodule, General Vision Subsystem (GVS), and

Specific Vision Subsystem (SVS) are the mentioned subsystems of VMA. First of all, the state of the AIBO robot is to be determined. State can be assigned one of the following values: Free Playing (FP), Blocked, and post-Kidnapped.

AIBO robots are usually in FP state. In other words, robots are playing freely most of the time without other players interfering. GVS approaches are used in such situations in which robots have freedom of action, i.e. they can move towards every direction. As a matter of fact, GVS approaches are employed in such situations because they are not considered computationally expensive; therefore, as a limited resource, CPU can be used by other processes such as Decision Making (DM) module.

First, Blocked state is encountered in situation that the AIBO has failed to move after trying for some time. SVS approaches are run in these cases to realize the reason of being blocked. SVS is exploited, because GVS has failed to detect objects exactly in order to let DM module decide what to do properly. Also post-Kidnapped state is happened in few moments. As a case in point, having booked, the robot is placed out of play for thirty seconds. In this status, the robot state is thought to be kidnapped. After repositioning on the field, the robot will make use of SVS approaches in the first moments to let the localization module self-localize exactly. Exact self-localization in the first few seconds of being repositioned on the field is an important factor. If the first self-localization is not done properly, the fault can be propagated until being in the situation that a land mark is recognizable by GVS.

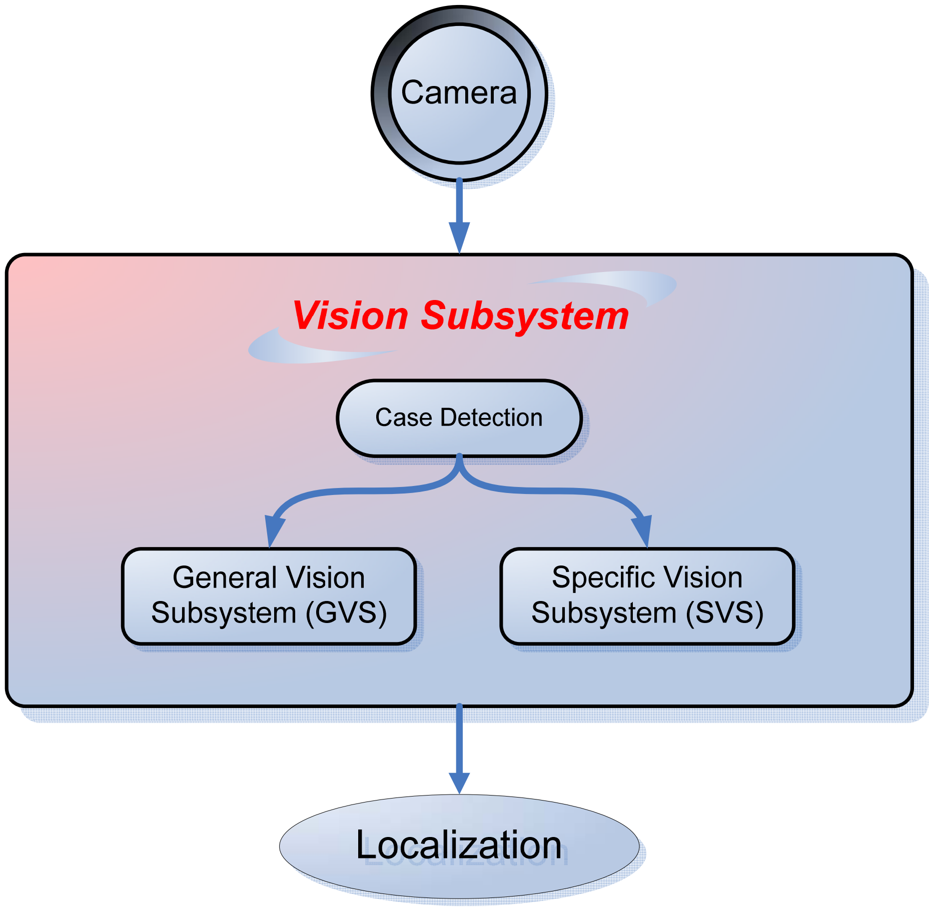

Camera
Vision Subsystem
Case Detection
General Vision Subsystem (GVS)
Specific Vision Subsystem (SVS)
Localization

Figure 33: Intra-Vision Module Architecture

7.2 General vs. Specific Vision subsystems

Up to now, we have provided a brief explanation of situations in which each of General Vision Subsystem (GVS) and Specific Vision Subsystem (SVS) are employed. In this subsection these sub-modules are given in detail.

7.2.1 General Vision Subsystem

The fundamental idea of our General Vision Subsystem (GVS) is based on UT Austin Villa Vision system. Figure 34 demonstrates the architecture of GVS. In fact, GVS consists of three major parts. Color segmentation, as the first step, will be discussed first. Second subsection is devoted to blob formation. Finally, object detection is given in the last part which itself is categorized into two subjects: marker detection and line detection.

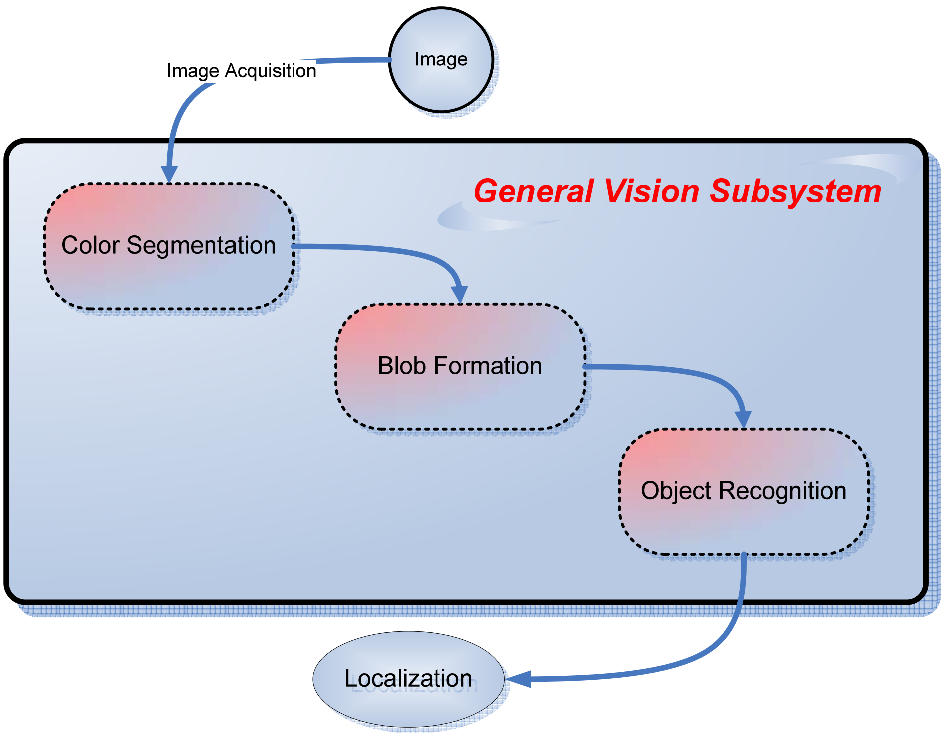


Figure 34: Intra-GVS Architecture

7.2.1.1 Color Segmentation

Image segmentation methods can be categorized as follows:

- Histogram thresholding: assumes that images are composed of regions with different gray (or color) ranges, and separates it into a number of peaks, each corresponding to one region.

- Edge-based approaches: use edge detection operators such as Sobel, Laplacian for example. Resulting regions may not be connected; hence edges need to be joined.
- Region-based approaches: based on similarity of regional image data. Some of the more widely-used approaches in this category are: Thresholding, Clustering, Region growing, Splitting and merging.
- Hybrid: consider both edges and regions.

Vision systems employing region segmentation by color are crucial in real-time interactive mobile robot applications such as object tracking (e.g. the ball in RoboCup soccer) and various forms of robot/human interaction. An important first step in many color vision tasks is to classify each pixel in an image into one of a discrete number of color classes. The leading approaches to accomplishing this task include linear color thresholding, nearest neighbor (NNr) classification, color space thresholding and probabilistic methods.

Linear color thresholding works by partitioning the color space with linear boundaries (e.g. planes in 3dimensional spaces). A particular pixel is then classified according to which partition it lies in. This method is convenient for learning systems such as neural networks (NNs), or multivariate decision trees (MDTs) [19].

A second approach is to use nearest neighbor (NNr) classification. Typically several hundred pre-classified exemplars are employed, each having a unique location in the color space and an associated classification. To classify a new pixel, a list of the K nearest exemplars is found, and then the pixel is classified according to the largest proportion of classifications of the neighbors [20]. Both linear thresholding and nearest neighbor classification provide good results in terms of classification accuracy, but do not provide real-time performance using off-the-shelf hardware.

Another approach is to use a set of constant thresholds defining a color class as a rectangular block in the color space [21]. This approach offers good performance, but is unable to take advantage of potential dependencies between the color space dimensions. A variant of the constant thresholding has been implemented in hardware by Newton Laboratories [22]. Their product provides color tracking data at real-time rates, but is potentially more expensive than software only approaches on general purpose hardware.

A final related approach is to store a discrete version of the entire joint probability distribution [23]. So, for example, to check whether a particular pixel is a member of the color class, its individual color components are used as indices to a multidimensional array. When the location is looked up in the array the returned value indicates probability of membership. This technique enables a modeling of arbitrary distribution volumes and membership can be checked with reasonable efficiency. The approach also enables the user to represent unusual membership volumes (e.g. cones or ellipsoids) and thus capture dependencies between the dimensions of the color space. The primary drawback to this approach is high memory cost—for speed the entire probability matrix must be present in RAM.

Color segmentation is the first step of our vision system; however, to reduce the computation burden of the algorithms we have employed mapping technique. As a brief explanation of mapping, the color of each pixel, which is in YCbCr, is mapped into a color class label of a set of limited number of classes, e.g. a 10-element set of colors defined in a LAB color space [24] explained below.

CIE L*a*b* (CIELAB) is the most complete color model used conventionally to describe all the colors visible to the human eye. The three parameters in the model represent the luminance of the color (L, L=0 yields black and L=100 indicates white), its position between red and green (a, negative values indicate green while positive values indicate red) and its position between yellow and blue (b, negative values indicate blue and positive values indicate yellow).

The Lab color model has been created to serve as a device independent, absolute model to be used as a reference. Therefore it is crucial to realize that the visual representations of the full gamut of colors in this model are never accurate. They are there just to help in understanding the concept, but they are inherently inaccurate. So we employed Lab color model to recognize the objects in the image.

Since the Lab model is a three dimensional model, it can only be represented properly in a three dimensional space. A useful feature of the model however is that the first parameter is extremely intuitive: changing its value is like changing the brightness setting in a TV set. Therefore only a few representations of some horizontal "slices" in the model are enough to conceptually visualize the whole gamut, assuming that the luminance would be represented on the vertical axis.

CIE 1976 L*a*b* is based directly on the CIE 1931 XYZ color space [24]; however, we are not going to explain the details about XYZ color space model here.

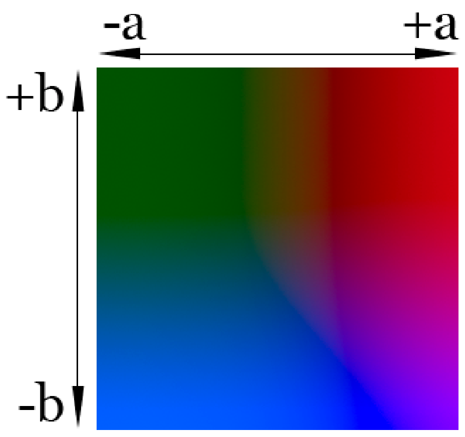


Figure 35: LAB in luminance of 25%

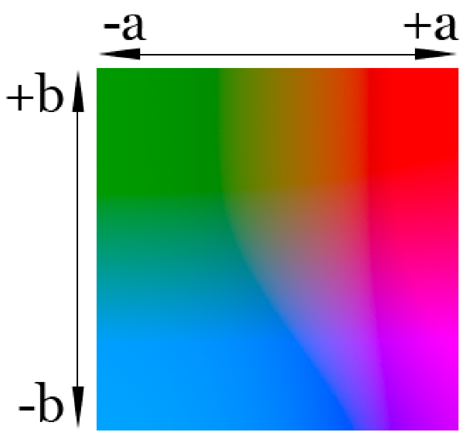


Figure 36: LAB in luminance of 50%

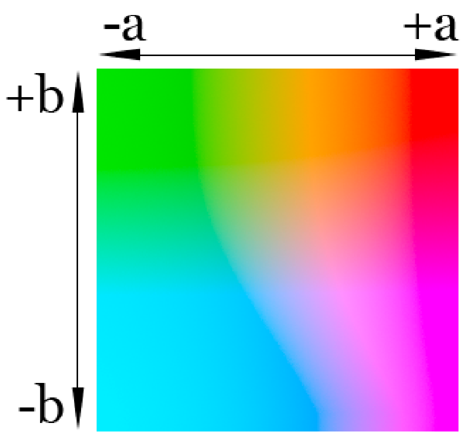


Figure 37: LAB of luminance of 75%

Given the RGB magnitudes of a pixel, to obtain the Lab values, one has to pass a two-step procedure. First, XYZ scale should be calculated using the given RGB values (See Equation.1). Finally, using the achieved XYZ values, Lab magnitudes are computed (Equation.2-4).

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \frac{1}{b_{21}} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \frac{1}{0.17697} \begin{bmatrix} 0.49 & 0.31 & 0.20 \\ 0.17697 & 0.81240 & 0.01063 \\ 0.00 & 0.01 & 0.99 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{13}$$

$$L = 116 \times f\left(\frac{Y}{Y_n}\right) - 16 \tag{14}$$

$$a = 500 \times \left[f\left(\frac{X}{X_n}\right) - f\left(\frac{Y}{Y_n}\right) \right] \tag{15}$$

$$b = 200 \times \left[f\left(\frac{Y}{Y_n}\right) - f\left(\frac{Z}{Z_n}\right) \right] \tag{16}$$

Where,

$$f(t) = \begin{cases} t^{\frac{1}{3}} & \text{t} > 0.008856 \\ 7.787 \times t + \frac{16}{116} & \text{otherwise} \end{cases} \tag{17}$$

Mapping is done using 1-Nearest Neighbor (1-NNr) [25] algorithm with threshold in Lab color space model. Lab color space is used, because it is known as the most robust color space model [26, and 27]. For instance, ball color is usually realized to be red by the robot vision system in the case of YCbCr color space model. Nearest Neighbor (NNr) will be explained here briefly.

To demonstrate a *k*-nearest neighbor (K-NNr) analysis, let's consider the task of classifying a new object (query point) among a number of known examples. This is shown in the figure below (Figure 38), which depicts the examples (instances) with the plus and minus signs and the query point with a red circle. Our task is to estimate (classify) the outcome of the query point based on a selected number of its nearest neighbors. In other words, we want to know whether the query point can be classified as a plus or a minus sign.

To proceed, let's consider the outcome of KNN based on 1-nearest neighbor (1-NNr). It is clear that in this case KNN will predict the outcome of the query point with a plus (since the closest point carries a plus sign). Now let's increase the number of nearest neighbors to 2, i.e., 2-nearest neighbors (2-NNr). This time KNN will not be able to classify the outcome of the query point since the second closest point is a minus, and so both the plus and the minus signs achieve the same score (i.e., win the same number of votes).

For the next step, let's increase the number of nearest neighbors to 5, i.e. 5-nearest neighbors (5-NNr). This will define a nearest neighbor region, which is indicated by the circle shown in the figure above. Since there are 2 and 3 plus and minus signs, respectively, in this circle KNN will assign a minus sign to the outcome of the query point.

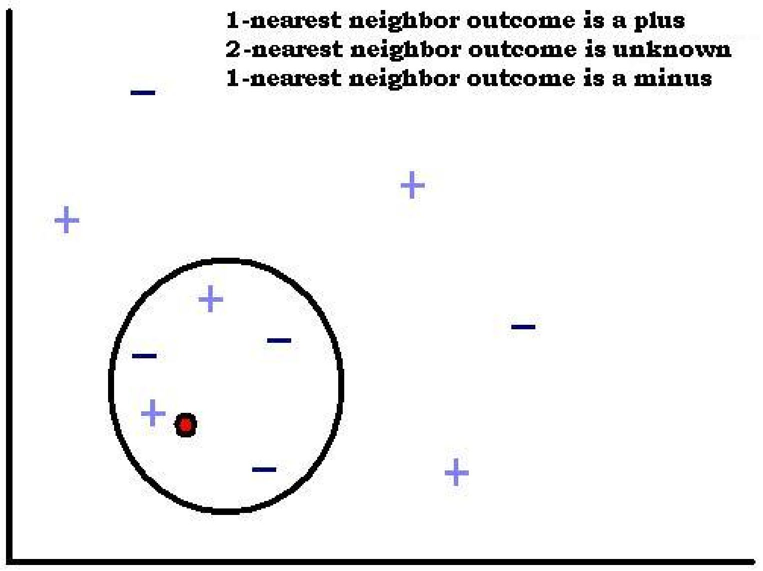

1-nearest neighbor outcome is a plus
2-nearest neighbor outcome is unknown
1-nearest neighbor outcome is a minus
–
+
+
+
–
–
–
+
–
+

Figure 38: 1-NN, 2-NN, and 3-NN Samples

In order to remove the holes, the following convolution mask is employed (Figure 39).

1	**3**	**4**	**3**	**1**
3	**6**	**7**	**6**	**3**
4	**7**	**10**	**7**	**4**
3	**6**	**7**	**6**	**3**
1	**3**	**4**	**3**	**1**

Figure 39: Smoothing Convolution Mask to remove the holes

A sample process of color segmentation is shown in Figure 51 and Figure 52. In order to improve the efficiency of the algorithm, the following three-step color cube generation approach is employed [17]:

- The off-board training phase in Lab color space results in an initial painting.
- Nearest Neighbor classification is performed in the Lab color space.
- Each cell in the output YCbCr color cube is labeled based on the value in the corresponding cell in the Lab color cube.

7.2.1.2 Blob Formation

As the second phase of our vision system, blob formation is essential for object recognition; therefore, an efficient clustering algorithm is required such as K-Means clustering algorithm [28] to cluster pixels of the same color into meaningful groups, but it is known as a computationally-expensive technique to be implemented in AIBO robots; therefore, it cannot be used. Our heuristic blob formation algorithm is carried out on the output of the previous step, i.e. color segmentation; hence, only regions thought to be objects, in which robots are interested, are clustered in separated blobs. Additionally, we intend to employ run-length encoding to compress data in order to have an efficient memory usage. It has not been implemented yet. The following figure shows the outcomes of the blob formation sub-module. As shown in this figure, the pixels have been clustered to identify the ball on the field.

7.2.1.3 Image Analysis

7.2.1.3.1 Object Detection

Once blobs are obtained, the objects of interest are to be recognized. The current object recognition techniques such as [29] cannot be used here, because they are time-consuming and computationally-expensive algorithms which prevent to be exploited in real-time environments such as AIBO soccer robots.

In addition to costs of the current algorithms, since all the objects in AIBO robots' environment are color coded, these algorithms may not be needed. Therefore, we employ a heuristic approach in order to decrease the processing cost. This approach gets the blobs, produced in the previous phase, i.e. blob formation. First, some of the blobs, which seem not be the objects of interest (e.g. the ball), are removed from the image. Some parameters such as size and position in the image are considered while removing these blobs. Then, the rest of the blobs are mapped into objects of interest on the field. Furthermore, we exploit their features as the inputs of our localization module in "Impossibles" AIBO architecture (chapter.2). For instance, looking at a marker, the AIBO robot can estimate its distance from the object, i.e. the marker.

7.2.1.3.2 Line/Corner Detection

Lines and corners of the field can be considered a type of information source which is passed into our localization module. Consequently, if they are detected precisely, an accurate localization module is supported.

Lines in AIBO environment is considered to be two parallel edges. These edges consist of green-white and white-to-green transitions. However, these two parallel edges change into one edge when the robot is far from the line.

First of all, we employ the vertical scanning algorithm to detect the lines. During each scan, Green-to-white and white-to-green edges are searched for along the vertical scanning line. Since closer lines give more reliable information than the farther lines, an algorithm is used to label the lines based on their distance from the

AIBO robot. According to this idea, vertical scanning lines are processed from bottom of the image to its top.

Then, having detected some white points along vertical scanning lines, we eliminate some these detected points which are not thought to be a point on a line of the field. Most of these employed algorithms are heuristic.

After omitting the useless detected points, the main sub-procedure starts to function. Since the images are usually taken in an ambiguous environment, i.e. AIBO robots, the above explained algorithm outputs some inexact information about the points. Hence, we employed Least Square Fitting (LSF) [30] technique to find the best fitting line which passes through detected points. In the following subsection, LSF algorithm is explained briefly.

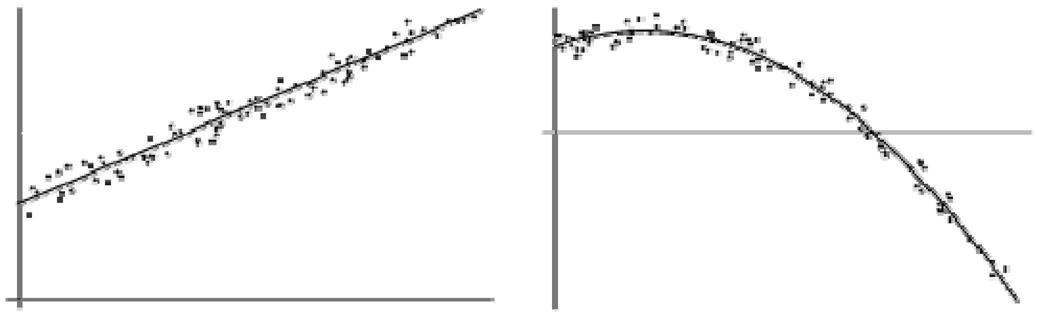


Figure 40: LFS algorithm to find the best fitting line passing through a set of points

As mentioned above, Least Fitting Square (LFS) is a mathematical procedure for finding the best-fitting curve to a given set of points by minimizing the sum of the squares of the offsets ("the residuals") of the points from the curve. The sum of the squares of the offsets is used instead of the offset absolute values because this allows the residuals to be treated as a continuous differentiable quantity. However, because squares of the offsets are used, outlying points can have a disproportionate effect on the fit, a property which may or may not be desirable depending on the problem at hand.

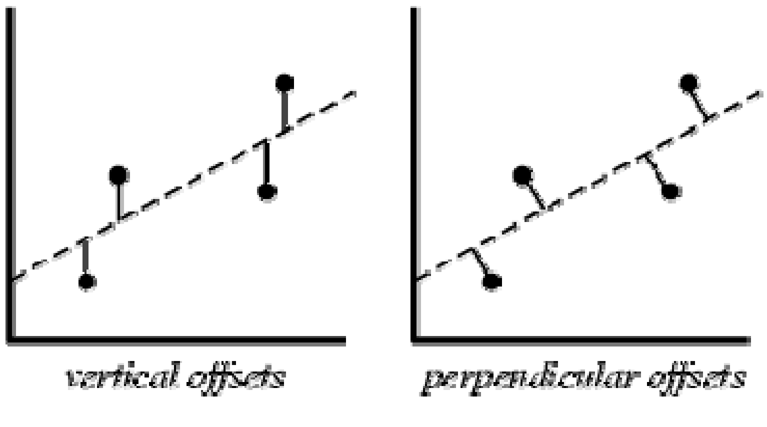


Figure 41: Vertical offsets vs. Perpendicular offsets in LFS algorithm

In practice, the vertical offsets from a line (polynomial, surface, hyper plane, etc.) are almost always minimized instead of the perpendicular offsets. This provides a fitting function for the independent variable 'X' that estimates 'y' for a given 'x' (most often what an experimenter wants), allows uncertainties of the data points along the 'x' and 'y' axes to be incorporated simply, and also provides a much simpler analytic form for the fitting parameters than would be obtained using a fit based on perpendicular offsets. In addition, the fitting technique can be easily generalized from a best-fit line to a best-fit polynomial when sums of vertical distances are used. In any case, for a reasonable number of noisy data points, the difference between vertical and perpendicular fits is quite small.

The linear least squares fitting technique is the simplest and most commonly applied form of linear regression and provides a solution to the problem of finding the best fitting straight line through a set of points. In fact, if the functional relationship between the two quantities being graphed is known to within additive or multiplicative constants, it is common practice to transform the data in such a way that the resulting line is a straight line.

7.2.2 Specific Vision Subsystem

SVS is *"Impossibles"* AIBO robots Vision's second submodule which is under implementation and has not been completed yet. As a matter of fact, SVS performs object detection much more accurately than General Vision Subsystem (GVS); however, SVS consists of a sequence of computationally-expensive and time-consuming techniques; therefore, it seems impossible to generally employ SVS in games. Hence, as explained in vision architecture subsection, we only exploit SVS after specific situations such as positioned after being kidnapped. Also SVS is used when localization module sends a signal that it is unable to self-localize the AIBO robot; therefore, it is reasonable to employ SVS to detect objects more accurately; however, it causes to vision subsystem have more CPU usage than GVS submodule.

As briefly discussed above and shown in Figure.10, Specific Vision Subsystem (SVS) consists of three major steps in order to accomplish the assigned task, i.e. proper object recognition.

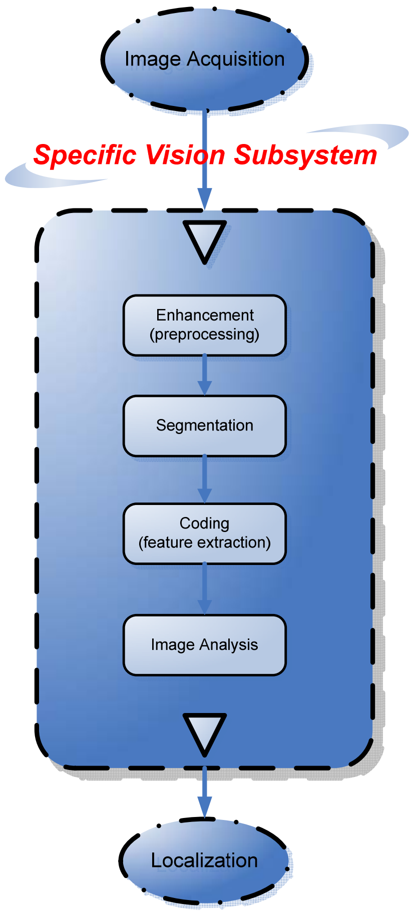


Image Acquisition
Specific Vision Subsystem
Enhancement (preprocessing)
Segmentation
Coding (feature extraction)
Image Analysis
Localization

Figure 42: Intra-Architecture of SVS

First of all, as a preprocessing step, image enhancement algorithms are employed to result an enhanced image in which it is much easier to detect objects. This step has been implemented completely. Second, segmentation phase is carried out. Now, we are discussing the algorithm to be implemented.

Coding, i.e. a type of image feature extraction and image analysis are the third and fourth steps of the SVS. These phases will be explained completely in the following subsections. The final results of SVS is passed through Localization module which is responsible for updating world model of *"Impossibles"* AIBO architecture (chapter.2).

7.2.2.1 Enhancement/Preprocessing

As in most of image processing procedures, enhancement/preprocessing are considered as the first phase in *"Impossibles"* SVS subsystem. In fact, enhancement/preprocessing are the whole operations employed to produce other images from the original ones. These produced images contain only necessary information to be analyzed later in the following steps of SVS.

As mentioned above, enhancement techniques transform an image into a "better" image, or one more suitable for subsequent processing to assure repeatable and reliable decisions. As a case in point, Figure 43 demonstrates a sample image and its corresponding enhanced version (Figure 44). There are three fundamental enhancement procedures: pixel or point transformations, image or global transformations, neighborhood transformations. *"Impossibles"* employ pixel and neighborhood transformations which are explained here.

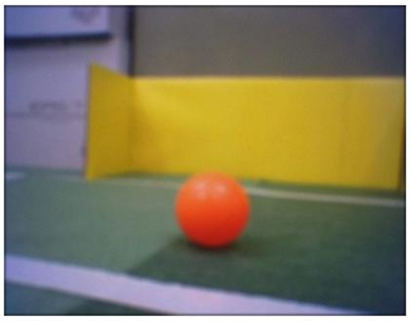


Figure 43: Original Image

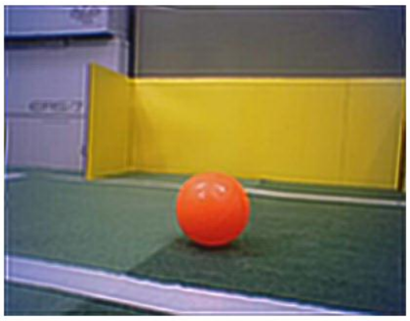


Figure 44: Enhanced Image

7.2.2.1.1 Pixel Transformation

There are a number of image enhancement routines that can be applied to improve the content of the image data before coding and analysis. Contrast and brightness enhancements alter an image's gray scale. Single pixel operators transform an image, pixel by pixel, based on one-to-one transformations of each pixel's gray level value. Some such operations include: Scaling, Addition or Subtraction of a constant to each pixel and inverting. Figure 45 denotes the result of subtraction of Figure 44 from Figure 43. Moreover, a sample multiplication result is shown in Figure 46.

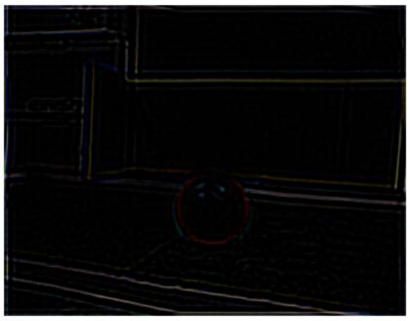


Figure 45: Subtraction result of Figure 44 from Figure 43

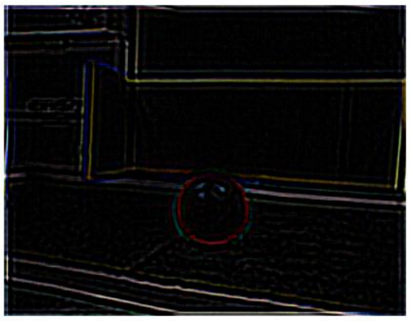


Figure 46: Multiplication result of Figure 45 by a Constant

7.2.2.1.2 Neighborhood Transformation

These take two forms, binary and gray scale processing. In both cases, operators transform an image by replacing each pixel with a value generated by looking at pixels in that pixel's neighborhood.

Gray Scale Neighborhood Processing is the first neighborhood transformation explained here. A neighborhood is passed over the input image, except that the image is gray scale, the coefficients assigned to the neighborhood are real numbers, and the function is generally arithmetic. As the most important Neighborhood Processing techniques, Sobel Spatial Filtering, as an edge detector, will be employed. As an example, edge detection process is shown in the following figures.

Moreover, binary neighborhood processing is mainly employed in *"Impossibles"* Specific Vision Subsystem (SVS). A binary image is transformed, pixel by pixel, into another binary image. At each pixel, the new value is generated by the old value of each pixel in the neighborhood. This can be thought of as a square matrix passing over the old image. At each pixel, all values inside the matrix are combined, giving one new value. The matrix moves to the next pixel, and the process repeats until the new image is generated. The following techniques are exploited in SVS subsystem: Dilation (Growing), Erosion (Shrinking), and Skeletonization.

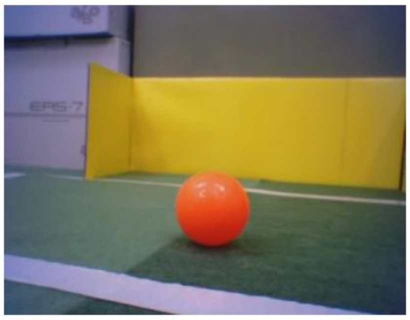


Figure 47: Original Image

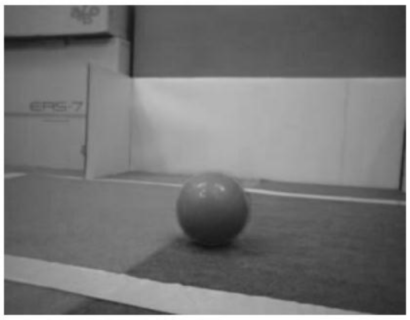


Figure 48: Gray-Scaled Image

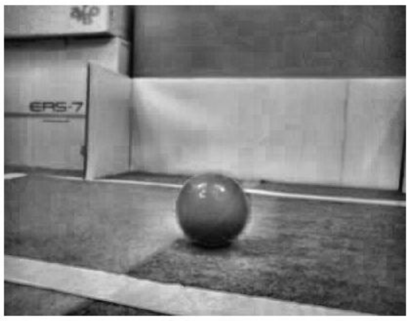


Figure 49: Histogram Equalized Image

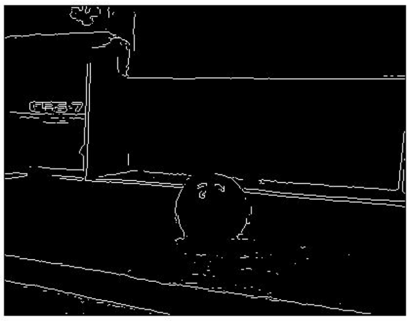


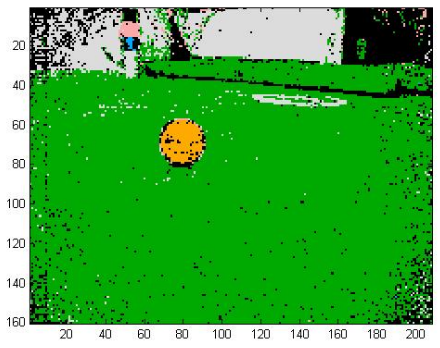
Figure 50: After Edge Detection (Sobel)

7.2.2.2 Segmentation

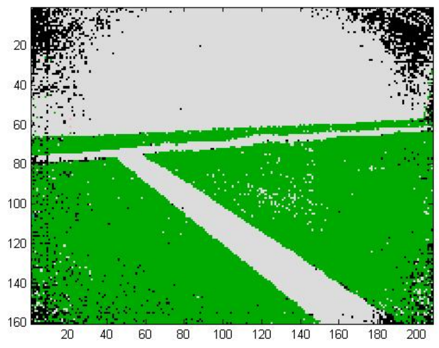
A scene can be segmented by color, windows, regions, or boundaries. Boundary, color, and region segmentation are utilized in SVS submodule. Color Segmentation was discusses in GVS subsection. Region and boundary segmentation are briefly explained here.

7.2.2.2.1 Region Segmentation

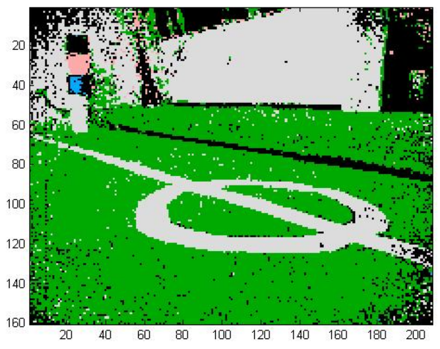
This is the process of partitioning an image into elementary regions (adjacent pixels) with a common property (such as specific gray level or color range), and then successively merging adjacent regions having sufficiently small differences in the selected property until only regions with large differences between them remain. A popular execution of this segmentation is based on using thresholding techniques to establish a binary image. Figure 51, Figure 52, and Figure 53 demonstrate some samples of color segmented images.



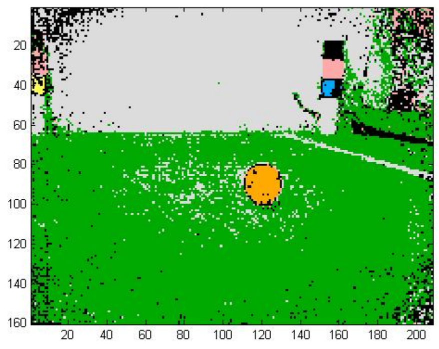
(a)



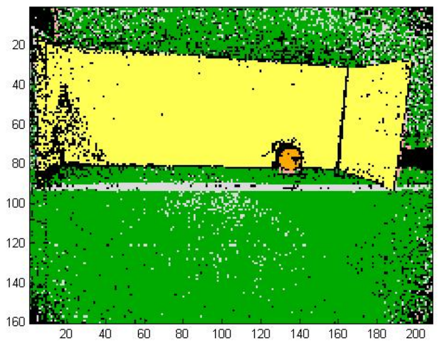
(b)



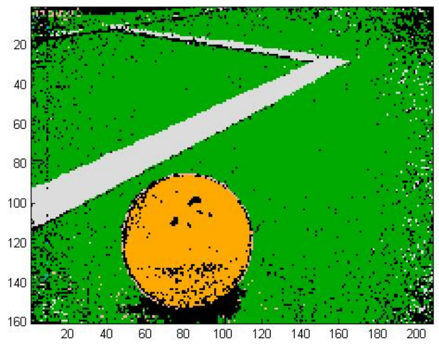
(c)



(d)



(e)



(f)

Figure 51: Samples of Segmented Images

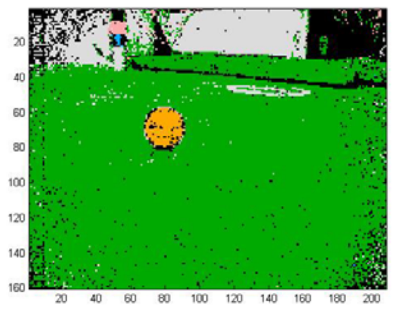


Figure 52: Color Segmented Image

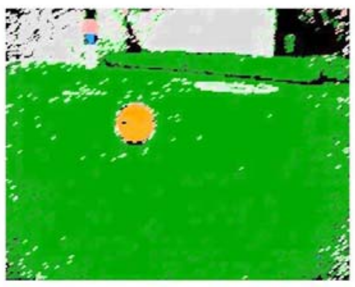


Figure 53: Region Segmented Image

Thresholding is the process of assigning a predefined value to each pixel in the image with value above a particular value. That particular value is the threshold. For instance, Areas that are lighter than the threshold become white; areas darker than the threshold become black. The resulting image, consisting of only black and white, is called a binary image. Thresholding was the first segmentation technique used, and almost all systems use it to some extent. It has a simplifying effect on the image. The number of pixels in the image does not change, but each pixel can now have one of only two values, usually written one or zero.

7.2.2.2.2 Edge Segmentation

Features can also be extracted based on edges. Again edges can be obtained from a binary image based on transition locations in a gray scale image. In the case of the latter, points of rapid change characterize an edge in gray level intensity. While sensitive to changes in pixel intensity of a single pixel, edge detection is not related to the individual intensities within patterns. Analysis of the edge intensity within a single pixel results in sub-pixel calculations of the location of an edge. The process is shown from Figure 54 to Figure 59.

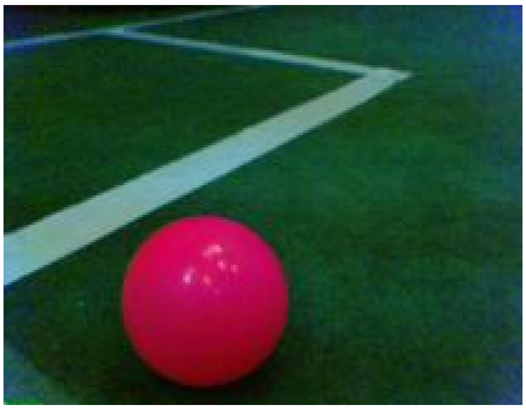


Figure 54: Original Image

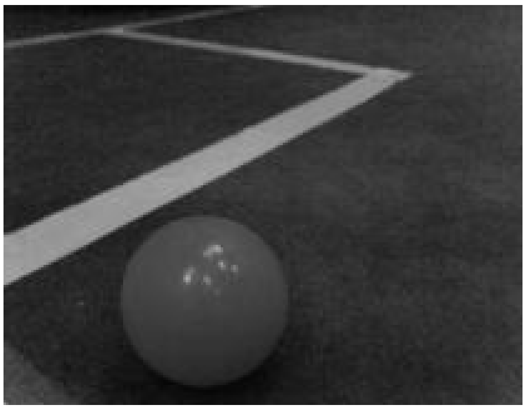


Figure 55: Gray Scaled Image

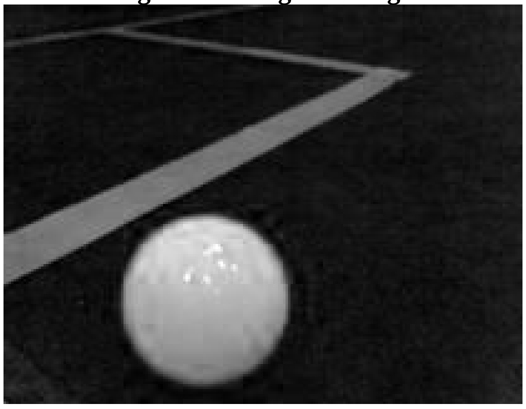


Figure 56: Green Layer of the Image

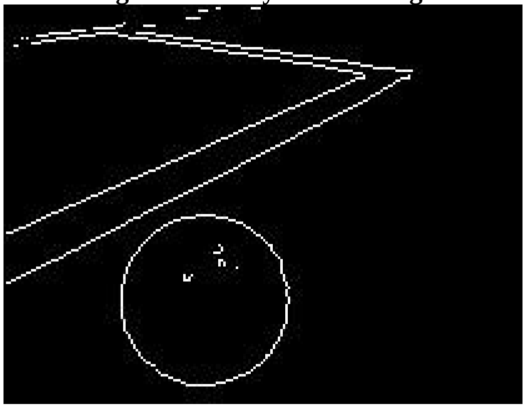


Figure 57: Edge Detected Image

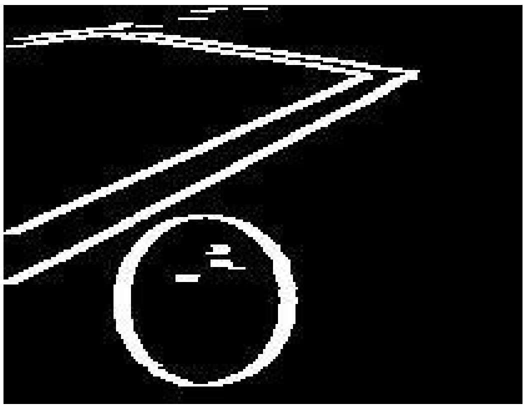


Figure 58: Dilated version of the above Image

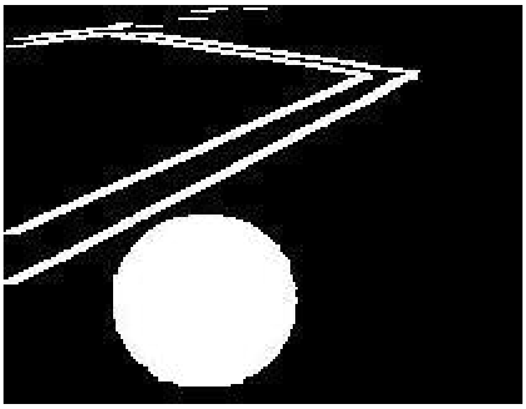


Figure 59: Detected Ball

Many edge-segmenting systems are based on detecting patterns of increasing and decreasing intensities or gradients generally found at the edges of objects. Since they are based on gradients, they are less sensitive to illumination variations and can handle lower contrast scenes.

Neighborhood processing techniques have evolved which are generally employed in conjunction with edge segmentation systems. These techniques involve evaluating each individual pixel according to its relation with its nearest neighbor pixels using a template or an array designed to detect some invariant regional property to perform the convolution. Figure 60 depicts a point template. The idea is to get a template response at every pixel location by centering the mask over each pixel in the image.

Point template

-1	**-1**	**-1**
-1	**8**	**-1**
-1	**-1**	**-1**

Coefficients may be arranged in vector form

$$W = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_9 \end{bmatrix}$$

$$w_i = \begin{cases} 8 & \text{i} = 5 \\ -1 & \text{otherwise} \end{cases}$$

Figure 60: Edge Segmentation point template

7.2.2.3 Coding and Feature Extraction

The next step in *"Impossibles"* SVS subsystem is feature extraction which is the process of deriving some values from the enhanced and/or segmented image. Some feature extraction methods require a binary image, while others operate on gray scale intensity or gray scale edge-enhanced images. As explained above, we have employed edge-enhanced images. The coding methods include: miscellaneous scalar features; shape features; and pattern matching extraction. Of the mentioned features, shape features and pattern matching are explained below.

7.2.2.3.1 Shape Features

Some computationally more intensive image analysis approaches are employed in SVS subsystem. They are based on extracting geometric features. We will employ the most popular approach (developed at Stanford Research Institute International) which involves performing global feature analysis (GFA) on a binary picture. In this case, the features are geometric: centroid, area, radius, perimeter, and so on. As a case in point, we exploit detected ball's radius to find out the location of the ball. In GFA, no inferences are made about the spatial relationships between features, and generally the parts are isolated.

7.2.2.3.2 Pattern Matching

As our last try to identify the template objects on the field such as ball, goal and opponent players we employ pattern matching. Pattern matching, also called correlation, pattern recognition, or template matching is a mathematical process for identifying the region in an image that looks most like a given reference subimage, e.g. ball. The reference subimage, or template, is overlaid on the image at many different locations. At each, goodness of match is evaluated. The location with the best is recorded, and the process is complete. Notice that process is inherently robust since it uses all the information in the image. Note, however, that a "match" operation, involving the template and part of the subimage, must be performed for each location in the image; this is a very time-consuming task.

7.2.2.4 Image Analysis

For some applications, the features, as extracted from the image, are all that is required. Most of the time, however, one more step must be taken; classified interpretation.

The most important interpretation method is conversion of units. Rarely will dimensions in "pixels" or "gray levels" be appropriate for an industrial application. As part of the *"Impossibles"* AIBO software, a calibration procedure will define the conversion factors between vision system units and real world units. Most of the time, conversion simply requires scaling by these factors. Occasionally, for high accuracy systems, different parts of the image may have slightly different calibrations (the parts may be at an angle, etc.). In any case, the system should have separate calibration factors in X and Y.

Reference points and other important quantities are occasionally not visible on the part, but must be derived from measurable features. For instance, a reference point may be defined by the intersection of two lines. To derive the location of this point, enough points must be measured to define the two lines and find their intersection by geometry.

Another common indirect measurement is to locate the center of a circle (e.g. Ball on the field) by finding points on its perimeter. Most systems have fast methods for doing this. In fact, indirect measurement calculations should present no problem to an experienced applications engineer.

8 Motion

"Impossibles" AIBO robots employ a Layered Motion Controlling approach (LMC). This section discusses the architecture and abilities of this system.

8.1 Architecture

The architecture of Motion Controller system is shown in Figure 61. Motion Controller (MC) system consists of two submodules: Skills and Inverse Kinematics [36] module. There are five skills in Skills part: Shooting, Walking, Looking, Blocking and Grabbing. Each of the skills could be used by Decision Making system to move the robot.

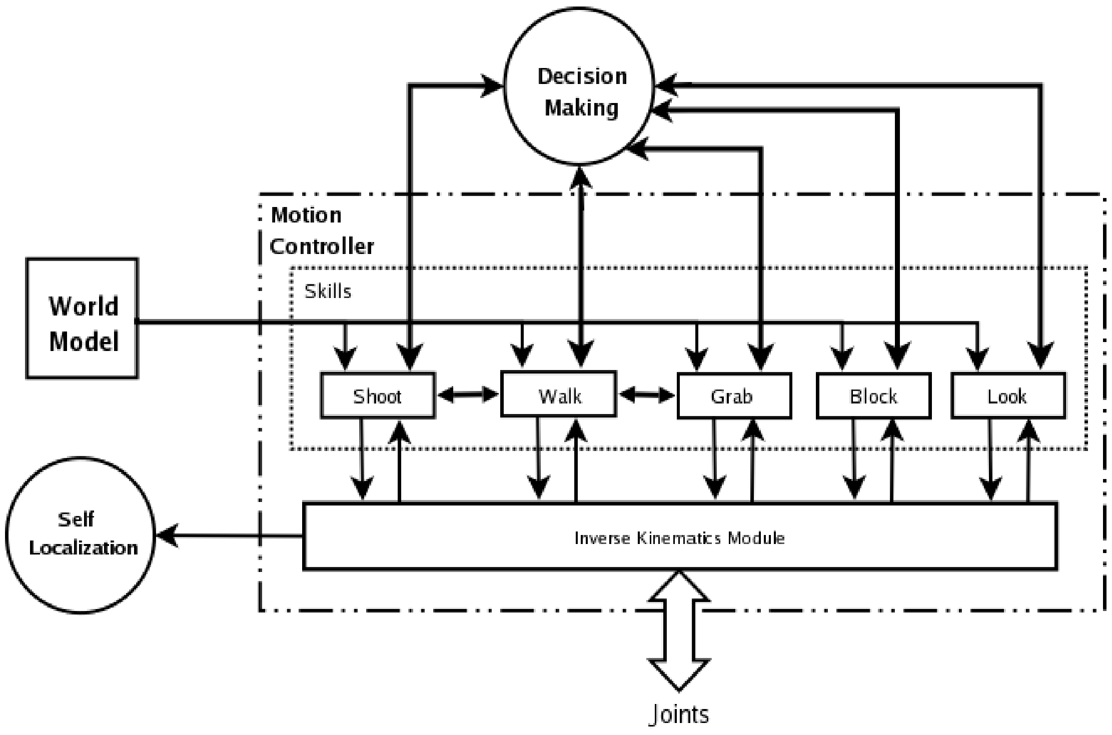


Figure 61: Architecture of Motion Controller

Skills are OPEN-R [37] objects and are run concurrently. They give the necessary information about the actions from Decision Making (DM) submodule and produce trajectories for joints.

Using concurrent Skills as separate modules makes development easier but has some disadvantages. By using this architecture, adding a new skill requires some changes in World Model and Inverse Kinematics module.

Like Skills, Inverse Kinematics module is also an OPEN-R object. It sends commands to the joints and receives joint values. As discussed earlier in this report, Self-Localization system uses joint values; therefore, having received the joints' values, the Inverse Kinematics module sends the values of the joints to the Self-Localization system. Skills Conflict Prevention (SCP) is also done by Inverse Kinematics submodule. As a matter of fact, it guaranties not to have conflicts amongst skills. When two skills are trying to simultaneously use a joint, Inverse Kinematics submodule selects the more important skill and reports failure to the skill with the lower priority. This module will be discussed later.

8.2 Skills

Each skill has its own input parameters and uses specific information from World Model (WM). Skills are responsible for executing the received commands from Decision Making (DM) subsystem and reporting the state of the robot's joints when executing the command and notifying the Decision Making (DM) module when execution is finished. In what follows, skills are given in words.

8.2.1 Shoot

Shooting skill is considered as one of the most important skills in the AIBO soccer environment. *"Impossibles"* AIBO robots have about fifteen methods for shooting the ball. They differ in delay, speed, stability of the robot and accuracy. We classify our shooting methods into two groups: Controlled shoots and Non-controlled shoots.

In Controlled shoots robot should own the ball before shooting. In this type, motions of robot after grabbing ball are predefined and joint trajectories could be looked up from a Look Up Table (LUT). Although these shoots have more delay, they are accurate. The famous example of this type is the Chest shoot witch is widely used in competitions. Another example is UMIGAME [38] that is a backward shooting method.

Non-controlled shoots are faster but they are inaccurate and need good prediction of ball movement. They could be used in cases that the robot is far from ball and can't reach ball (maybe because of obstacles or speed). One of the best examples for this type is German Team [39]'s One-hand shooting method.

Shooting Skills receives two parameters from Decision Making (DM) subsystem; Direction and Power. Currently the power parameter is ignored and robots always use maximum power for shooting. Steps of chest shoot are shown in Figure 62. Steps for Chest Shoot:

1. **Holding the ball with chin:** This is the start state of shooting. It returns failure signal if the ball was not grabbed before.
2. **Turning to direction:** Then the robot should turn to the desired direction without releasing the ball. This is just like normal

turning. The most important difference is that robot should hold the ball with its chin.

3. **Kick the ball:** The last part is the final shooting step. Joint trajectories for this part is predefined and are tuned for maximum speed of the ball after pushing.

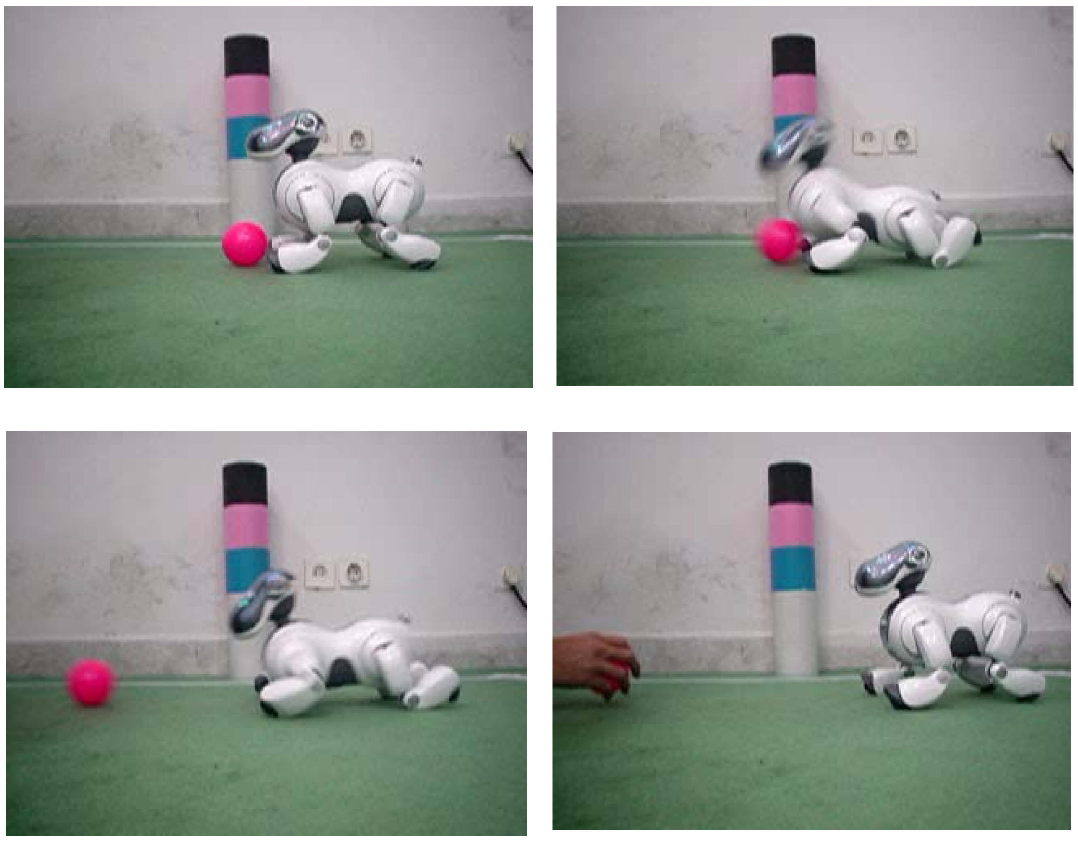


Figure 62: Chest Shoot

This type of kick is accurate and straightforward but the robot should completely grab the ball before the shoot. Sometimes this limitation makes the performance slow and even sometimes the failure. So it seems that another shoot method is necessary for such cases.

Currently we are working on a faster and more stable shoot method that uses just one leg; however, this shoot is less accurate.

8.2.2 Walk

Moving for legged robot is too different from wheeled robots. Legged robots have capabilities that wheeled robots don't like changing their height but on the other hand creation and optimization of walking for legged robots is more challenging. This is because legged robots have many degrees of freedom. Number of parameters that affect the style of walking, its speed and stability is too big that tuning the walking manually is too hard and time consuming. We are using learning techniques that we have described in more detail below.

Walking is too important for robots of a soccer team and strongly affects the result of the team. Not only it is too important to walk fast when racing with rival

dogs to reach to ball but also it is important to have a smooth walking which not only doesn't make problems for vision but also makes the robot stable. We think it is important to be stable while walking because many times the dog will be in a situation that other dogs hit it. As mentioned above our walking gets its parameters from the behavior layer.
These parameters are as follows:

- Walking Direction (θ angle)
- Type of walking (We have some handmade walking styles and also we have reached to a walking style using parameter learning)
- Height of rear and front hips
- Walking Speed(If the speed is more than available robot walks with its maximum available speed)
- If the walking is backward or forward

The walking engine decides the walking style considering these parameters. We have taken two sub-modules for walking. The first module of walking is Predefined Walk which gets all needed parameters from the upper layer. The second type is Fast Walk. Also we have two types of fastest walk, the Fastest Available Walk and the Fastest Stable Walk. The former is used when we want the robot to reach a far point in minimum time and the latter is responsible for the fastest walk which is resistant against falling when the robot is pushed by others also is smooth to avoid problems for vision.

8.2.2.1 Walking Module's Input

The input variables for this module are *power* and an *array of points* from the Decision Making layer. The power is a double variable between 0 and 1 and defines how fast the robot should move. Considering the current position and direction of the robot and the next position, θ is computed. The walking module uses data from world model to decide the type of walking, height of rear and front hips and also if the walking is backward or forward.

8.2.2.2 Walking Module's Output

Output of the walking module is an array of joint angles which is given to the lower level. These joint angles are computed using inverse kinematics. The distance and number of points are dependent to the trajectory and speed required. Joint angles are attended to be in the available range considering the degree of freedom for each joint.

8.2.2.3 Predefined Walking

As described above, all parameters are passed to this walk engine. We are working on Policy Gradient Reinforcement Learning [42] to define the trajectory of transition from the states of walking to one another for the fastest stable walk.

8.2.2.4 Fast Walking

The angle of swing-joints in rear legs affects the height of front and rear hips of robot. Height of the rear and front hips are effective on the distribution of the robot’s weight on each leg. Of course the posture of legs has a direct relation with the height of hips but we can’t look at them as a unique issue. This is because there isn’t a one-to-one mapping between the height of a leg and its joint angles. The best style of standing should be computed or defined using learning algorithms. The style of standing is not very related to the friction or type of the surface. A larger distribution of weight on a leg causes bigger friction force. Rear legs can move with a less slip when they have a larger friction with the land surface.

However; when the weight distribution on a leg is big, lifting that leg may cause the AIBO to fall also the robot will be less stable than when legs which cause motion suffer less pressure from the body and are farther from the center of gravity. Less pressure in front legs benefit to less friction and they will slip easier on the ground. So it is a tradeoff!

If the behavior level defines the height of front and rear hips there is no need for making decision on how to distribute the weight. If not it needs to be decided. There are some parameters that need to be defined. These parameters are listed below:

- **Posture of leg while standing**: This parameter is very important. Because it affects many other parameters. It defines the back and front hip height. This is important as mentioned above because it affects the friction of legs with the ground while lifting the ground for rear legs and while slipping on the ground for front legs. Also height of the robot is one of the three parameters that affect the stability of the robot.
- **Cycle of a walk**: This parameter is important because it affects the stability of the robot. When the number of frames a leg is not on the ground is big it makes the AIBO unstable and will fall easier by a hit. Also if cycles a leg is not on the ground are large, robot's speed will be higher when using a good trajectory.
- **Trajectory of moving of the rear legs**: This parameter is very important and we are using temporal learning to choose the best trajectory. The trajectory is important from two points of views. The former is when lifting the ground and is very important because it may cause the leg to slip on the ground. The latter is the length of the path and trajectory of moving between the Cartesian points defined for each frame. Not only we need a short path but also we need to use the motors efficient. It means ideally we have to use each motors maximum potential in each frame.
- **Friction of surface**: Some technical reports have noted that when a leg wants to start a step it is better to lift it vertical from the ground to avoid its slipping. But it is clear it can be inclined to a degree considering the friction of ground. By making the lifting oblique the speed of moving forward will increase. But this angle should be identified using learning on each surface. Also when one leg is moving on the space the other leg(s) are support and their friction will stop the

robot from slipping. So it is important to calculate how the style of legs should be to avoid slipping.

Considering the input variables for Fast Walking we reached two styles of fast walking which are described below:

8.2.2.5 Fastest Available Walk

This style of walking is used when there is a long distance to the goal (e.g. ball). Also this walking is not smooth and do not consider if the head will have shaking. We are using learning for defining the best trajectory and cycle time. As mentioned above firstly by learning we define the “alpha” angle for the surface. Then the trajectory has to be tuned using learning to reach the best speed.

8.2.2.6 Fastest Available Stable Walk

This walking style is very different from fastest available walk. Comparing to fastest available walk it is very smooth and stable. Steps are too much smaller and hips are closer to the ground. We have two goals in this kind of fast walking learning. Not only we want to have the fastest available walk but also we want the smoothest walk. We used some calculations in our vision module to define how smooth is the walking. For our learning the robot starts from its penalty region and walks to the other goal. We defined that the upper line of the goal must be in the middle of the camera picture. The diversity of the line from the center of the picture defines how much the walking is smooth. Also we took an upper limit for the step length of a foot to get to a stable walking. We choose the fastest walking which its smoothness is not less than a threshold.

8.2.3 Looking & Head Movement

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.

Head movement usually occurs in two major cases. When we want the robot to trace an object and when the robot should look for an object. Decision making layer decides where the robot should look to and just tells the X and Y to this layer as described so this layer just decides the style of movement.

8.2.4 Grab

To grab the ball first the robot should chase the ball and then catch it. First, the robot should determine the ball’s position to call a proper walk skill command to reach the ball. Doing the second part is not straightforward when the ball is moving fast. In such cases we should use Block skill first. But when it is fixed the robot catches it with its hands and chin.

In AIBO 4-legged soccer matches the ball is light and doesn't move straight. So using the way explained above will fail most of the times. So currently our robots just try to get close to the ball. And catch the ball when it is in a grabable range. Figure 63 shows an *"Impossibles"* AIBO robot grabbing the ball.

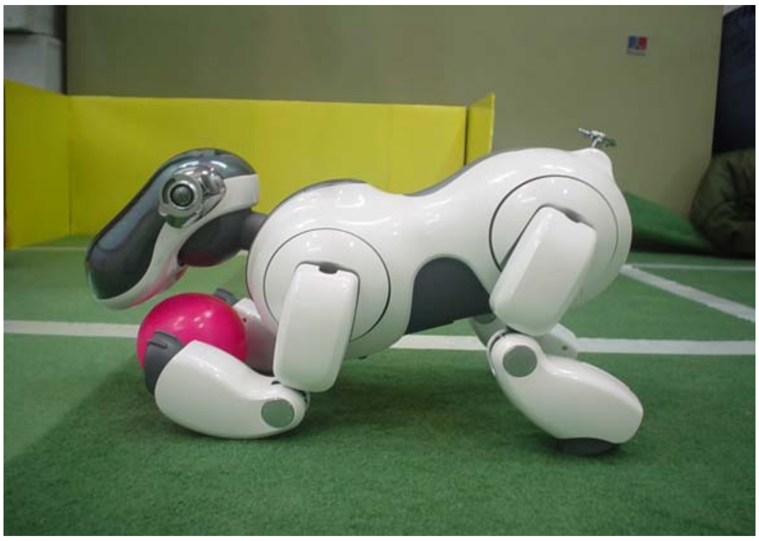


Figure 63: "Impossibles" AIBO grabbing the Ball

8.2.5 Block

Due to uncertainty of ball velocity and position in AIBO robots' real world environment there are some difficulties with grabbing the ball when it is moving with high speed. For example, catching the opponent's shoot for the goalie is hard. So we use another action in such cases. Block action tries to avoid fast moving balls. We use an estimate of ball position and velocity for blocking. Better estimate results in more chance of success.

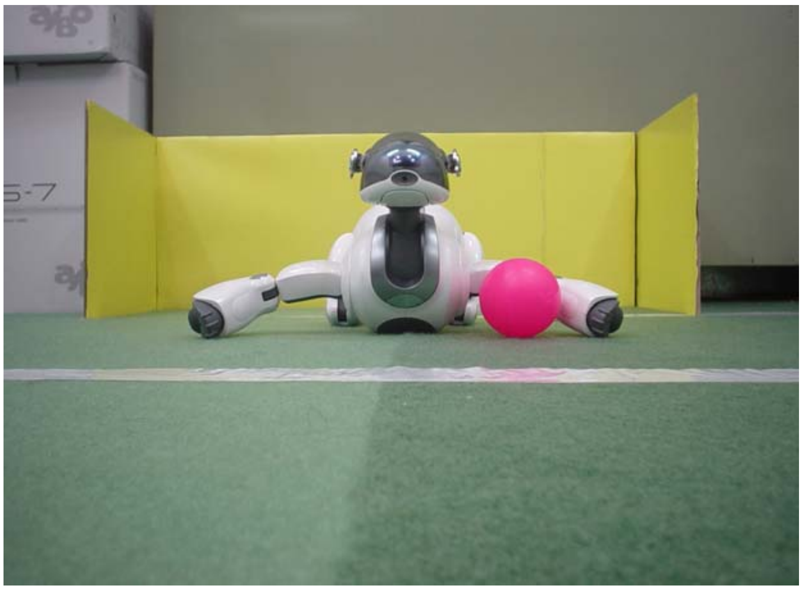


Figure 64: *"Impossibles"* AIBO Blocking the Ball

Blocking is useful but to do this the robot should fall and stand up again; therefore, it is a time-consuming skill. So it should be used when it is required. As discussed earlier, Decision Making (DM) module is responsible for this part.

8.3 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.

8.3.1 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.

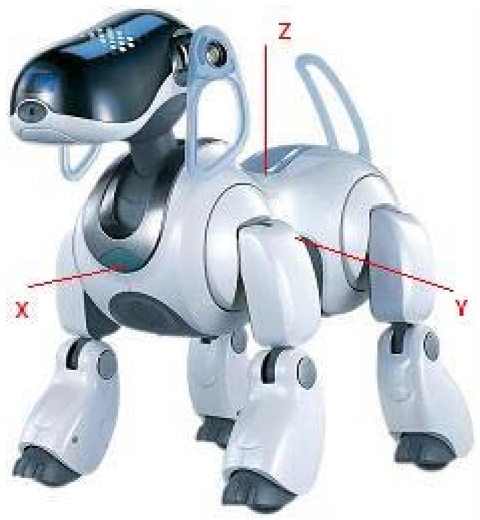


Figure 65: Reference Frame of the AIBO Robot

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.

$$T_{frame} = A_1 \times A_2 \times A_3 \times T_{reference} \tag{18}$$

$$A_i = Rot_{\theta i} \times Trans_a \times Trans_d \times Rot_\alpha \tag{19}$$

8.3.2 Inverse Kinematics

For the inverse kinematics we have positions and want to compute the joint angles of the robot. Computation of inverse kinematics for legs is too challenging and hard. 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. To solve this by inverse kinematics we have created a lookup table which robot sets its joints to the most similar condition in the lookup table.

Computation of the joint angles for head is too much easier. It is available to compute it using the inverse matrices in the equations above;

$$T_{frame} = A_1^{-1} \times A_2^{-1} \times A_3^{-1} \times T_{reference} \tag{20}$$

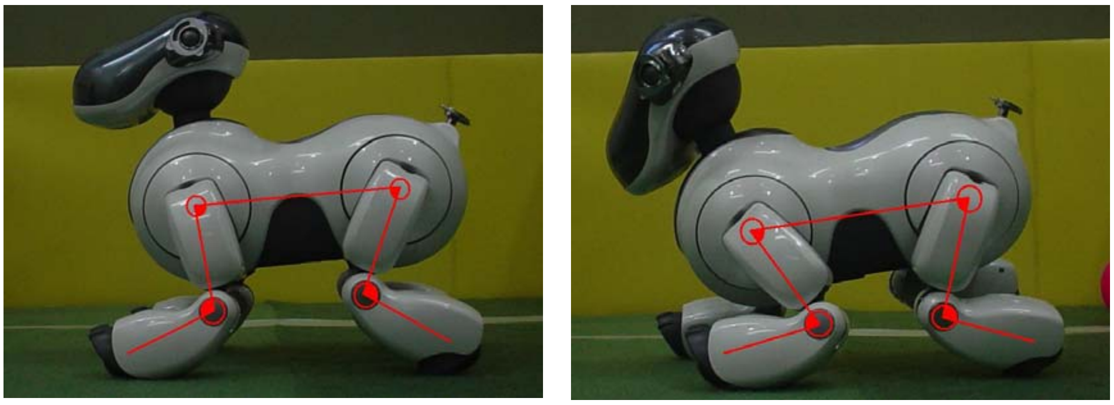


Figure 66: Two snapshots of the AIBO robot while walking

9 Acknowledgements

We would like to thank the following people for their valuable comments on specified fields:

1. Peter Stone	Reinforcement Learning	pstone@cs.utexas.edu
2. Caro Lucas	Emotional Control	lucas@ipm.ir
3. Saeed Bagheri	Fuzzy Logic Control	bagheri-s@sharif.edu
4. Mansour Jamzad	Machine Vision	jamzad@sharif.edu
5. Ramin Halavati	Machine Learning	halavati@ce.sharif.edu
6. Saman Haratizadeh	Emotional Control	harati@ce.sharif.edu
7. Parisa Haghany	Evidence Theory	haghany@ce.sharif.edu
8. Mohsen Amiri	Machine Vision	mo_amiri@ce.sharif.edu
9. Reza Hesamifard	Fuzzy FSM	hesami@ce.sharif.edu
10. Elyas Khameneh	Machine Vision	khameneh_e@mehr.sharif.edu

Additionally, “*Impossibles*” appreciate H. Moradi because of his assistance in robots’ on time delivery.

10 References

[1] http://openr.aibo.com

[2] http://www.tzi.de/4legged/pub/Website/Downloads/Rules2006.pdf.

[3] Jafar Habibi, Saman Aliari Zonouz, Hamid Reza Vaezi Joze, Majid Valipour, Moahammad Reza Ghodsi, Alireza Fathi, "*Impossibles Team Description*", RoboCup, Osaka, Japan, 2005.

[4] Thomas Rofer et al., "German Team Technical Report for RoboCup 2005", RoboCup, Osaka, Japan, 2005.

[5] Peter Stone, Peter Stone, Kurt Dresner, Peggy Fidelman, Nate Kohl, Gregory Kuhlmann, Mohan Sridharan, and Daniel Stronger, "The UT Austin Villa 2005 RoboCup Four-Legged Team", RoboCup, Osaka, Japan, 2005.

[6] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte Carlo localization: Efficient position estimation for mobile robots", In Proceedings of the National Conference on Artificial Intelligence, 1999.

[7] S. Lenser, J. Bruce, and M. Veloso, "CMPack: A complete software system for autonomous legged soccer robots", In Autonomous Agents, 2001.

[8] Glann Shafer, A mathematical theory of evidence, Princeton University Press, 1976.

[9] Johann Heinrich Lambert, "Neues Organon", 1764.

[10] Gray, A., "Modern Differential Geometry of Curves and Surfaces with Mathematica", Boca Raton, CRC Press, 1997.

[11] Hilbert, D. and Cohn-Vossen, S., "Geometry and the Imagination", New York: Chelsea, pp. 3-4, 1999.

[12] Saman Harati Zadeh, Ramin Halavati, Saeed Bagheri Shouraki, Caro Locus, "Emerging Simple Emotional States in Zamin Artificial World",

[13] Caro Locus, Danial Shahmirzadi, and Nima Sheikholeslami, Introducing a Controller Based on Brain Emotional Learning Algorithm: BELBIC (Brain Emotional Learning Based Intelligent Controller), International Journal of Intelligent Automation and Soft Computing (AutoSoft), August 2002, USA.

[14] J. Habibi, and P. Nayeri, "Centralized vs. Non-Centralized Decision-Making in Multi-Agent Environments", 11th CSI computer Conference, January, 2006.

[15] Peggy Fidelman and Peter Stone, "Layered Learning on a Physical Robot", February 2005.

[16] Nate Kohl and Peter Stone, “Machine Learning for Fast Quadrupedal Locomotion” In Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI), San Jose, CA, July 2004

[17] Mohan Sridharan, Peter Stone, “Real-Time Vision on a Mobile Robot Platform”, IROS, 2005.

[18] Peter Stone, Kurt Dresner, Peggy Fidelman, Nate Kohl, Gregory Kuhlman, Mohan Sridharan, Daniel Stronger, “UT Austin Villa 2005 RoboCup Four-Legged Team Report”, 4-Legged Soccer League RoboCup, Osaka, 2005.

[19] Brodley, C., and Utgoff P., ”Multivariate decision trees. Machine Learning”, 1995.

[20] Brown, T., and Koplowitz, J. 1979. The weighted nearest neighbor rule for class dependent sample sizes. IEEE Transactions on Information Theory 617–619.

[21] Jain, R., Kasturi, R., and Schunck, B., “Machine Vision”. McGrawHill, 1995.

[22] Laboratories, N. 1999. Cognachrome image capture device. http://www.newtonlabs.com.

[23] Silk, E. 1999. Human detection and recognition for an hors d’oeuvres serving robot. http://www.cs.swarthmore.edu/ silk/robot/.

[24] FORSYTH D. A., PONCE J.: *Computer Vision: A Modern Approach*. Prentice Hall, 2002.

[25] Nearest Neighbor Pattern Classification, T. M. Cover, and P. E. Hart.

[26] Jeff Hyams, Mark W. Powell, and Robin R. Murphy. Cooperative navigation of micro-rovers using color segmentation. *In Journal of Autonomous Robots*, 9(1):7–16, 2000.

[27] B. W. Minten, R. R. Murphy, J. Hyams, and M. Micire. Low-ordercomplexity vision-based docking. *IEEE Transactions on Robotics and Automation*, 17(6):922–930, 2001.

[28] Ana Fred, and Anil K. Jain, “Evidence Accumulation Clustering based on the K-Means Algorithm”.

[29] R. Bergevin and M. D. Levine. Generic object recognition: Building and matching coarse 3d descriptions from line drawings. IEEE Transactions on Pattern Analysis and Machine Intelligence, 15:19--36, January 1993.

[30] York, D. "Least-Square Fitting of a Straight Line." Canad. J. Phys. 44, 1079-1086, 1966.

[31] Nello Zuech, “Understanding and Applying Machine Vision”, Second Edition, Marcel Dekker, Inc.

[32] Becher, W. D., "Cytocomputer, A General Purpose Image Processor," ASEE 1982, North Central Section Conference, April 1982.

[33] Lougheed, R. M. and McCubbrey, D. L., "The Cytocomputer: A Practical Pipelined Image Processor," Proceedings of the 7th Annual International Symposium on Computer Architecture, 1980.

[34] Sternberg, S. R., "Language and Architecture for Parallel Image Processing," Proceedings of the Conference on Pattern Recognition in Practice, Amsterdam, The Netherlands, May 21–30, North-Holland Publishing Company, 1980.

[35] Sternberg, S. R., "Architectures for Neighborhood Processing," IEEE Pattern Recognition and Image Processing Conference, August 3–5, 1981.

[36] S. B. Niku, Introduction to Robotics; Systems and Applications, Jul 2001, Prentice Hall.

[37] OPEN-R SDK. http://www.openr.org.

[38] H. Fukumoto et al., Baby Tigers Dash: Osaka 4-Legged Robot Team, RoboCup, Osaka Japan, 2005.

[39] http://www.germanteam.org/tiki-index.php.

[40] M. S. Kim, and William Uther, "Automatic Gait Optimisation for Quadruped Robots", Proceedings of the 2003 Australian Conference on Robotics & Automation.

[41] J. Bruce, S. Lenser, and M. Veloso, "Fast Parametric Transitions for Smooth Quadrupedal Motion", IEEE Transactions on Robotics, April 2005.

[42] Nate Kohl and Peter Stone, "Policy Gradient Reinforcement Learning for Fast Quadrapedal Locomotion", In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), New Orleans, LA, May 2004.

[43] S. Chernova, and M. Veloso, "An Evolutionary Approach to Gait Learning for Four-Legged Robots", In Proceedings of IROS'04, September 2004.