# CERBERUS 2001 TEAM REPORT


H. Levent Akın[1]*,
Ersin Başaran[1]
Tamer Demir[1]
Onur Dikmen[1]
Çiğdem Gündüz[1]
Okyay Kaynak[2]
Hatice Köse[1]
Albert Ali Salah[1]
Önder Tombuş[3]
Olcay Taner Yıldız[1]

Ivan Kavalchev
Petya Emilova Pavlova
Christo Iliev Radev
Anastasia Christova Radeva
Nikola Georgiev Shakev
Jordan Kirilov Tombakov
Andon Venelinov Topalov

[1] Department of Computer Engineering, Boğaziçi University, 80815, Bebek, Istanbul, Turkey

[2] Department of Electrical and Electronics Engineering, Boğaziçi University, 80815, Bebek, Istanbul, Turkey

[3] Department of Industrial Engineering, Boğaziçi University, 8*815, Bebek, Istanbul, Turkey

Department of Control Systems, Technical University Sofia, Plovdiv branch, 61, St. Petersburg Blvd, Plovdiv 4000, Bulgaria

**September 30, 2001**

---

* Corresponding Author

**ACKNOWLEDGEMENTS**

The members of the Boğaziçi University team would like to thank Prof. Sabih Tansal, Rector of Boğaziçi University and Prof. Z. İlsen Önsan Vice Rector and Chairperson of the Executive Committee of the Boğaziçi University Research Fund and Candan Fetvaci, Coordinator of the Boğaziçi University Foundation for their kind support throughout all the phases of the project . We would also like to thank Profs. Ethem Alpaydın, Fikret Gürgen and Lale Akarun for the valuable discussions.

We gratefully acknowledge the support of our work by the Boğaziçi University Research Fund through project 01A101, Boğaziçi University Foundation, and Boğaziçi University Student Fund.

## ABSTRACT

Cerberus is one of the four new teams in the RoboCup Tournament, Legged Robot Category. The team is a joint effort of Boğaziçi University (Turkey) and Technical University of Sofia, Branch Plovdiv (Bulgaria). We have divided the robot software design into six major tasks, and worked on those tasks in parallel. With limited time in our hands, our aims were a clear object oriented design, readable behavior architecture.

## TABLE OF CONTENTS

# I. INTRODUCTION

Robotic soccer as first proposed by (Mackworth, 1993), is a challenging research domain that is particularly appropriate for studying a large spectrum of issues of relevance to the development of complete autonomous agents (Asada *et al*, 1998). The Sony Legged Robot League is an international robot competition that has been launched within the RoboCup initiative (http://www.robocup.org). Sony's quadruped robot AIBO has been adopted as a hardware platform, so the competition in this league is on a software level.

The "Cerberus[1] 2001" team made its debut in RoboCup 2001 competition. This was the first international team participating in the league as result of the joint research effort of a group of students and their professors from Bogazici Univerity (BU), Istanbul, Turkey, and Technical University Sofia, Plovdiv branch (TUSP), Plovdiv, Bulgaria.

The challenge of designing a robotic soccer team is best met with a lot of hands-on experience. The noisy real-time environment, the dynamic nature of the opponents, the different lighting conditions all necessitate robust algorithms for the robot and require arduous fine-tuning. The initial designs are usually based on theoretical postulates and assumptions. These prove to be ineffective, as noise prevails. The performance can only be improved by patches to handle special cases, and feedback from observing the system in action.

Unfortunately, the ideal design-observe-correct cycle is not applicable if there is only a limited time available. The scheme has to be replaced by a trial and error approach, as we did. Between the possible two approaches of taking some other team's code and improving upon that and developing a completely new one based on our previous experience we mainly chose to develop new algorithms wherever possible and use old code when absolutely necessary. Our general strategy was to divide the robot software into six major tasks, and assign one team for each task. Each team started out by developing some working code for the task, and by integrating it to produce a working system to begin with. Then, each team gradually improved the modules.

---

[1] In Greek mythology, Cerberus was the most dangerous labor of Hercules. It was a vicious beast that guarded the entrance to Hades and kept the living from entering the world of the dead. It had three heads of wild dogs, a dragon or serpent for a tail, and heads of snakes all over his back.

The groups were locomotion, localization, vision, behavior control, learning and communication. Locomotion and vision were low-level tasks that involved interfacing the sensors and actuators of the AIBO robot via middleware. Localization relies on the information from vision and locomotion to determine robot's position. Learning was used whenever necessary. The majority of the Turkish team has machine learning as their primary research area, whereas the Bulgarian team mostly specializes on control. The behavior control involves the design of a conceptual finite state machine to guide the behavior of the robot. Cooperation and communication can be seen as integrated parts of the behavior.

# II. THE DEVELOPMENT TEAM

The Cerberus team is a joint effort of Boğaziçi University (Turkey) and Technical University of Sofia, Branch Plovdiv (Bulgaria).The head of the Cerberus project is Prof. Okyay Kaynak, faculty member at the Electrical and Electronics Department of Boğaziçi University and is the UNESCO chair on Mechatronics.

## II.1 Turkish Team

Assoc. Prof. H. Levent Akın, the head of the Artificial Intelligence Laboratory at the Department of Computer Engineering, leads the Turkish team. The Turkish team has six PhD students. Hatice Köse works on mobile robot control and obstacle avoidance. Çiğdem Gündüz has an M.Sc. on value of information in classifiers. Olcay Taner Yıldız works on graphical models and decision trees. Ersin Başaran applies machine learning techniques to chromosome classification and sequence alignment. Albert Ali Salah works on selective attention and unsupervised learning. Önder Tombuş conducts research on multi-agent systems in the Industrial Engineering Department.

The other members are MS students. Onur Dikmen's research involves rule-based reasoning. Tamer Demir specializes in networking, e.g. performance issues in ATM networks.

## II.2 Bulgarian Team

The Bulgarian team is lead by Assoc. Prof. Andon Venelinov Topalov from the Department of Control Systems at the Technical University of Sofia Plovdiv Branch. The other members of the team are Christo Iliev Radev – computer sciences consultant, Petya Emilova Pavlova – lecturer in image processing, Nikola Georgiev Shakev – Ph. D. student from the Department of Control Systems and three senior students from the Department of Computer Engineering - Ivan Kavalchev, Jordan Kirilov Tombakov and Anastasia Christova Radeva. The Bulgarian team specializes in robot motion control (A. Topalov, Ch. Radev, J. Tombakov and N. Shakev), communications between robots (Ch. Radev and A. Radeva), image preprocessing algorithms (P. Pavlova and I. Kalvachev) and low level behavior control (A. Topalov, N. Shakev and I. Kalvachev).

# III. THE ALGORITHMS

Our main approach was to come up with new ideas that have not been examined previous. In the development of the Cerberus team entry we tried to follow as closely as possible object oriented software analysis and design techniques. But since one team consisted of mainly control engineers we had difficulties in strictly implementing the chosen methodology. Nevertheless, we first made a requirements analysis, then designed the architecture which were followed by the implementation and testing with continuous feedback from each and every phase to the previous ones.

## III.1 Requirements Analysis

Since this was our first time we initially watched the videos of the 2000 games and examined the reports and codes of the competing teams to determine the requirements and implement the system accordingly. By observing the games, we deduced some possible rules to be used as guidelines in behavior design. Not all of them were implemented due to lack of time. Below a partial list of these are given:

- Backwards movement: The CM-United team did not implement a backwards movement. However, it can be useful, especially if the robot sees a teammate, but not the ball. It has the immediate benefit of expanding the visual field, and it's also useful when a robot blocks the ball from the other.
- We need to determine a handful of initial robot configurations. The first one is the case where our team has the center kick. This is an offensive position, and a locker-room agreement should be made for the rapid progress towards the opposing goal. A second initial position is the defensive configuration. Blocking the ball or the opposing team members, a good positioning of the goalie are useful. We may consider to position one of our players towards our own goal for fast defensive positioning, or a preset backwards-movement may be used.
- The best goal shooting path seems to be the corner on which the goalie is positioned. The goalie can be pushed inside the goal area with the ball, or the ball is shot towards the goalie, and scoring is made possible because the goalie is not very accurate in its

movements.

- A fast ball-reversal algorithm is useful, in case our robot catches a ball while facing its own goal. Somehow jumping over the ball, falling towards the ball so that it changes direction or a special kick can be considered.
- A robust and strong kicking algorithm should be implemented. We consider pressing on the ball to move it forward with great speed. The momentum of the robot can be employed in this case.
- Immobilizing the head to sample visual information is a waste of time during the game. Successful teams sample visual scenery on the run. This means that the images are really erratic, there is vibration in three dimensions. We can use a voting algorithm which looks at a window of successive images, detect blobs and votes to indicate a granular direction. If a more accurate prediction is necessary for the kicking algorithm, we can increase the processing load, and sample more images, and also throw in corrective terms from the leg and head joint positions. An ideal case behavior would be not to synchronize four legs, but four legs plus the head, so as to minimize the vertical and horizontal disturbance on the sampled images.
- A sideways hit with the head is a most effective strategy if the robot is near the goal area. We should identify "comfortable goal positions". These include cases where our robot faces the goal while controlling the ball, and the goalie is disabled. Some robust and fast strategy must be used to score such secure goals. The robot should not waste time to search for other robots or beacons.
- Constantly checking for the white defense line may be inefficient, because the head position must be towards the ground, which makes the robot lose sight of the ball (normally, we can strive to keep the ball in the center of the visual field by feedback from vision and use the head position parameters to orient the robot towards the ball rapidly. This scheme will fail if the robot constantly searches for other features on the scene). Some teams completely ignore the white line, and referees re-position the robots that cross the line. Maybe we can use this feature to rapidly come to the half-line.
- If the robot in not able to see the ball, it should turn its body, and not just the head. Always consider cases where the ball is just beneath the robot or very close to it. Moving the body constantly helps.
- The goalie should identify some specific positions, and act accordingly. It should position itself correctly to defend against shots from afar, and switch to another strategy is if the ball and attackers are close to it. Consider the case where two attackers and the

goalie are within the defense line. Ideally, the goalie should be able to block the goal and remove the ball from the defense zone. The movements should be fast, but very robust in order to prevent a self-goal. A problem occurs if the goalie loses sight of the ball, and tries to search it by turning around. These usually lead to easy goals for the opposing team. There must be "return to the goal area" mode for the aggressive goalie, which must implement obstacle avoidance.

- The straight and fast movement of any robot usually indicates the position of the ball. If the ball is not visible, the robot may check the direction of movement of the other objects (possibly robots), and turn towards that direction.
- We have observed that a static scan mode for the goalie, where the goalie just stays immobile and scans the visual field in six steps (3 low, 3 high), is not very efficient. Especially if the opposing team has advanced sufficiently, this immobility of the goalie is a great disadvantage.
- A gait which is close to the ground allows a better visual recognition, since the variations on the vertical axis are not as great. It also allows a fast way of hitting the ball with the head. One advantage of "crawling" is that the attacker can dribble the ball into the goal. Since the ball cannot enter the reduced space between the belly of the robot and the ground, this procedure does not result in a foul.
- Before the start of the game, robots are driven through an initialization process. They take in the visual field, and possibly initialize the parameters for the visual system and localization. The initial positions should be recognized with ease.
- If an attacker approaches, the goalie should enter the goal area, but not too much. It should defend the front line of the goal area.
- The ball should rarely be approached directly. It is faster to calculate an intention first, and move accordingly. Splines may be used to move towards the ball in an arc.
- We should allow ample time for code optimization. Chunks of code may be written in C++, different routines and data structures can be tested for efficiency. It is important that we observe performance bottlenecks during all our real-world tests, and identify parts of the code that need optimization. Note that this can only be achieved if we look for those bottlenecks constantly, and always bear in mind that there is almost always room for optimization in the code.
- We will implement objects that read sensor values and derive "virtual sensor" values from those. That means some high-level variables will be derived from selectively scanning and combining the sensor data. These variables can be stored with appropriate

time-stamps. Then, if the virtual sensor value is needed, the time-stamp can be used to check whether the value is still valid or not. The position of the goal, for example, can be such a variable, and will not change easily. Therefore a high timeout can be associated with it, and the robot will not re-calculate the position repeatedly.

### III.2 The Architecture

After the requirements analysis phase the agent architecture to be used was determined. The Cerberus team has developed a behavior-based architecture (Arkin 1998) for the robots. The main objective of our team was to build a research platform, which allows robust and efficient carriage of robots playing football. A modular architecture has been adopted and implemented. The following are the main system components: Vision, Localization, Communication, Planner, Behaviors, and Locomotion. The relationship between them can be seen in Figure 1.

The perception level of the control system consists of the Vision and Localization modules. The Vision module accepts images from the camera and identifies objects on the field that are visible. It delivers to the Planner module and Localization module the identified objects as well as the distance to the object and its relative direction. The Localization module receives from the Vision module the relative position of the landmarks around the play field. This information is used to approximate the position and heading of the robot on the field.

The Planner and Communication module form the logical level. They incorporate the memory of the robot, decision-making and inter-robot communication. The Planner module collects the information about the visible objects from the Vision module and stores it in the local map. The local map presents the objects around the robot in polar coordinates. Also the Planner receives from Localization the position and heading of the robot. Using this information it decides what have to be done and invokes the appropriate behavior. Communication module enables an acoustic communication between teammate robots. Exchanging of information can be used to fill the local map or to facilitate the cooperative operations.

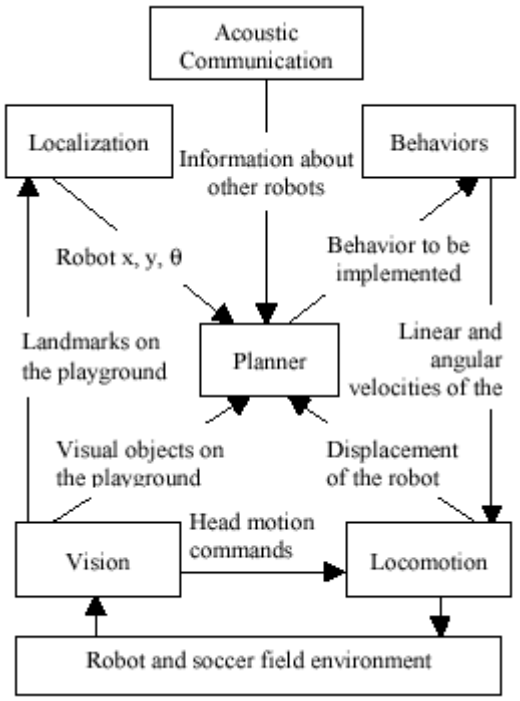



Figure 1. The architecture of the control system

The implementation level consists of the Behavior module and Locomotion module. The Behavior module supports a set of reactive behaviors that can be implemented. It accepts abstract commands like “Dribble with the ball” and transforms them in values of angular and linear velocities of the robot. The Locomotion module receives the values of the linear and angular velocities that have to be implemented and makes the appropriate movement. It also delivers to the Planner module an information about the displacement of the robot in order to be actualized the information in the local map.

### III.3 Vision

We rely on robust vision algorithms to drive the robot behavior. The Turkish team worked on the vision system. Decision trees and Multi Layer Perceptrons were compared for object recognition. Decision trees were found to be faster and more robust for the successful classification of colors. After classification, we implemented the region finding codes. We find regions with areas that exceed some threshold.

#### III.3.1 Color Recognition System

We did not use the built-in color detection table and developed a new color recognition system. For color classification, we tested mainly decision trees and artificial neural networks (linear and multilayer perceptrons). In both cases YUV values received by the camera are the inputs and color label is the output.

For both cases, it was necessary to train our systems for adjusting the parameters of the classification. We took data from the camera images and used them for training. For this purpose special programs were developed C and Delphi environments. These programs generated the decision tree and neural network testing codes to be used in OPENR environment.

After these codes were embedded in OPENR environment, we could classify each pixel of images by using the parameters learned. YUV values were the inputs and color label is the output of these functions.

We also compute the confidence values for classification of each pixel. The probability values for neural networks and the class percentages at each decision leaf taken as probability values for decision tree are used as confidence values.

#### III.3.1.1 Classification using Neural Networks and Decision Trees

Data that are taken from different images were separated into two dataset, namely the training and the test datasets. They were trained by using linear perceptron and the multilayer perceptrons with different number of hidden units. The online learning method was used for training the perceptrons, we also used momentum and adaptive learning factor in training. Several runs were done with different constants of learning and momentum factors. Some of the results taken from these learners with their constants are reported in the following table:

Table 1. Results for different neural classifiers.

| | **learning factor** | **momentum factor** | **Training accuracy** | **Test accuracy** | **Cost** |
|---|---|---|---|---|---|
| LP | 0.10 | 0.7 | 95.21 | 93.89 | 30 |
| MLP1 | 0.08 | 0.5 | 77.12 | 72.71 | 13 |
| MLP2 | 0.04 | 0.5 | 94.06 | 95.02 | 26 |
| MLP3 | 0.20 | 0.5 | 95.93 | 95.83 | 39 |
| MLP4 | 0.30 | 0.7 | 95.37 | 95.46 | 52 |
| MLP5 | 0.30 | 0.5 | 96.84 | 96.56 | 65 |
| MLP6 | 0.30 | 0.7 | 96.43 | 96.54 | 78 |
| MLP7 | 0.20 | 0.7 | 97.36 | 95.39 | 91 |
| MLP8 | 0.40 | 0.7 | 95.82 | 96.22 | 104 |
| MLP9 | 0.40 | 0.7 | 96.45 | 96.37 | 117 |
| MLP10 | 0.20 | 0.5 | 97.45 | 95.33 | 130 |

The costs are computed in big-*O* notation for linear and multilayer perceptrons as follows:

**LP classification**: $c * (d + 1) + c + c*1$
: $c * (d + 3)$
: $O(c * d)$

**MLP classification**: $s * (d + 1) + s * 1 + c * (s + 1) + c + c * 1$
: $c * 3 + s * (d + c + 2)$

*: O(s * d + s * c)*

where *d, c* and *s* are the numbers of the input, output and hidden units respectively.

For the linear perceptron and the multilayer perceptrons with 3 and 5 hidden units, there are 10 independent runs. The average of the accuracies are given in the following table. The values in parenthesis are the standard deviations for these runs.

Table 2. Means and standard deviations for some of the runs.

| | learning factor | momentum factor | Training accuracy | Test accuracy | Cost |
|---|---|---|---|---|---|
| LP | 0.10 | 0.7 | 95.17(0.34) | 92.94(0.37) | 30 |
| MLP3 | 0.20 | 0.5 | 95.56(0.76) | 90.09(2.66) | 39 |
| MLP5 | 0.30 | 0.5 | 96.57(0.26) | 93.71(1.04) | 65 |

From these experiments, we concluded the following:

1. Green for the beacon and the field is almost same, so we can use only one green class for both of them.
2. Gray for robots are frequently mixed with different colors because of the lighting conditions.
3. The lines of the field can not be distinguished most of the time. White for lines can be confused with field color, pink or light blue. Also the edge detection will be too costly.
4. Dark blue uniform is sometimes confused with gray. So we can ignore gray to solve this problem.
5. Orange is confused with other colors, so we can use another classifier that labels the colors as orange and not orange.

In the following table, the average misclassifications rates are given for each color with the misclassified colors and their percentages in the whole incorrect labels. The values in parenthesis indicate the standard deviations. We use multilayer perceptron with 5 hidden units for training our system.

Then we decide not to use white, gray and beacon gray in our classification. The results which we have seven colors are given in the following table.

Table 3. Average misclassification rates for different colors.

| Correct Color | Misclassification Percentages | Incorrect Color Assigned For Misclassification | Percentage Of Selecting The Incorrect Color |
|---|---|---|---|
| white | 11.12 (6.61) | gray | 83.68 |
| | | orange | 13.81 |
| | | other | 2.51 |
| field green | 2.08 (1.09) | white | 40.51 |
| | | yellow | 58.75 |
| | | other | 0.74 |
| Gray | 10.14 (2.56) | white | 59.44 |
| | | dark blue | 11.81 |
| | | blue | 17.37 |
| | | other | 11.38 |
| Pink | 3.52 (0.95) | white | 32.03 |
| | | gray | 19.61 |
| | | orange | 47.71 |
| | | other | 0.65 |
| Red | 5.80 (1.63) | pink | 89.43 |
| | | other | 10.57 |
| dark blue | 13.23 (3.27) | gray | 91.42 |
| | | other | 8.58 |
| Blue | 0.29 (0.17) | gray | 84.78 |
| | | pink | 15.22 |
| Orange | 19.34 (12.77) | pink | 40.10 |
| | | yellow | 30.99 |
| | | red | 23.53 |
| | | other | 5.38 |
| beacon green | 12.17 (2.04) | field green | 77.01 |
| | | gray | 22.99 |
| Yellow | 1.46 (0.71) | orange | 85.57 |
| | | other | 14.43 |

After testing decision trees, linear perceptrons and multilayer perceptrons which have different number of hidden units, we observed that the accuracy of all algorithms were very close to each other, and the space complexities were negligible. Although the time complexities were of the same order in the real time implementation there were significant differences in performance especially the exponential function that should be used for neural network is too costly. At the end, we decided to use decision trees in color classification, which is based on comparison operations.

Table 4. Results after removing some of the colors.

| | learning factor | momentum factor | Training accuracy | Test accuracy | Cost |
|---|---|---|---|---|---|
| LP | 0.0.2 | 0.5 | 96.45 | 97.77 | 21 |
| MLP1 | 0.01 | 0.7 | 83.17 | 80.31 | 10 |
| MLP2 | 0.10 | 0.7 | 96.24 | 98.07 | 20 |
| MLP3 | 0.40 | 0.5 | 97.79 | 98.67 | 30 |
| MLP4 | 0.40 | 0.5 | 96.78 | 98.79 | 40 |
| MLP5 | 0.08 | 0.7 | 98.21 | 99.03 | 50 |
| MLP6 | 0.30 | 0.7 | 98.46 | 99.26 | 60 |
| MLP7 | 0.40 | 0.7 | 98.70 | 98.97 | 70 |
| MLP8 | 0.30 | 0.5 | 98.83 | 99.03 | 80 |
| MLP9 | 0.40 | 0.7 | 96.78 | 99.33 | 90 |
| MLP10 | 0.40 | 0.7 | 98.85 | 99.15 | 100 |

Table 5. Classification results for decision trees

| | Mean | Standard Deviation |
|---|---|---|
| Testing Accuracy | 96.65 | 0.26 |
| Number of nodes | 78.60 | 18.49 |
| Training time (sec) | 216.73 | 32.70 |

#### III.3.1.2 Vision Modes

The next step was to implement some modes for vision. Since we did not use the built-in color recognition system, we could not use the track or search modes that were available. So we had to implement the normal, search and track modes from scratch. In the normal mode, we classify the pixels by using the decision tree that is trained according to all colors available. But for the search and track modes, we train our decision tree such that it discriminates the colors as orange and others. So it was possible to decrease significantly the time that is needed for color classification.

### III.3.2 Object Recognition System

After color classification, we find regions in an image. We need to have two passes for region finding. In the first pass, pixels in the top row and the left column are labeled according to their classes. If a pixel has the same color with the one on its left or above, it is given the same label as them. Then all pixels are processed row by row. After this process, each region has assigned to it region label, region color, area (number of the pixels in it), confidence (∑pixel confidence in this region / area), bounding box (it has X and Y

coordinates of the top-left and the right-bottom pixels) and the centroid (X and Y coordinates of centroid).

The second pass is to merge the regions with the same color but with different region labels. If two regions are to be merged, new area, confidence, bounding box and centroid are calculated.

These regions are sorted in descending order according to their area. The regions smaller than a predefined area and confidence thresholds are discarded. The long and thin areas are also discarded.

In the next step, for each object that will be recognized, the regions are tested if they are these objects or not. Each object has region number, confidence value, centroid and bounding box. The region number for each object is –1 by default and if this object is found the appropriate region number it is assigned to this object. The order for testing and the criteria for each object are as following:

- **Ball:** If the color of the region is not orange then the region is discarded. If height to width ratio or width to height ratio of the region is greater than two the region is discarded. If the region does not cross with the green area, which is the playing field, it is discarded. If the area of the region is smaller than a threshold it is also discarded. If the region is not discarded, we say that it is a ball and return the confidence value as *height/width* if *height*<*width* or *width/height* otherwise.
- **Goals:** If the color of the region is not the colors of the goals (light blue or yellow) it is discarded. If the regions height or regions width or regions area are smaller than certain thresholds the region is also discarded. Otherwise the region is taken as the goal and the confidence value is given as *height/(2*width)* if *(2*width)*>*height* or *(2*width)/height* otherwise.
- **Beacons:** Since each beacon consists of two colored regions, we search for these regions to put them together. These areas must have the given colors otherwise they are discarded. These areas must have a width to height ratio bigger that a constant otherwise they are discarded. The centroids of these areas must be in a close distance in $x$ and $y$ coordinates with a given constant oterwise they are discarded. The regions satisfying these conditions are found and the one with the maximum area is selected as a possible beacon. The confidence value for this beacon is calculated as the beacons width to height ratio.
- **Players:** The region must have the players color else it is discarded. It must also have a minimum area. The confidence of the player is found by the confidence of the region

only.

### III.3.3 How To Make the Color Table

As mentioned above, we use the decision tree algorithm for color classification. Although the built-in color detection table can classify at most eight colors, there is no color limit for our color classification table. Initially we used ten colors: white, yellow, orange, green, dark green, red, blue, dark blue, gray and pink. Then we saw that green (for beacons) and dark green (for field) are exactly the same, so we decided to use only one green for both of them.

After making some tests, we also added the skin color to our colors since skin and ball color are easily mistaken for each other. So we trained our decision tree for these colors. We did not adjust the parameters of colors manually since these parameters are learned in the training phase of the classification.

For training, we need to take samples in each color from images. This is achieved by an auxiliary program that we call *labeler*. As seen in Figure 2, we can examine the images that is taken by Aibo, and choose pixels by using mouse and we can give class labels for selected pixels. Then we save this data, YUV values of a pixel and class label in a file. After taking several data from different images, we can train the classifier to learn the parameters of the system. The learning phase of our algorithm explained above takes a small time.


Figure 2. Labeler Program screen shot with a picture taken by Aibo

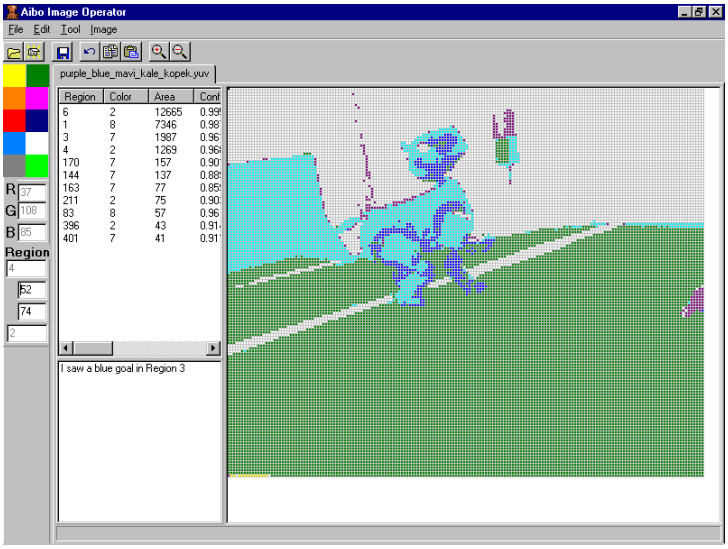



Figure 3. Labeler Program screen shot after finding regions and objects by the program

### III.4 Localization

We tried to use a geometric approach to localization. The Turkish team worked on localization. In our implementation, localization relies on the recognition of beacons and odometry. A global map is kept where the position of the robot and other locations of interest are recorded probabilistically. Each image frame processed by the vision module contributes in updating the relative probabilities.

The localization module takes data from the vision module and uses this data to find the exact location, and heading of the robot, itself. Unfortunately since it mostly relies on the visual data, the noise in this data affects the success of localization very much.

Only beacons are used as landmarks. The vision module returns data about which beacon(s) are seen, their relative distance to the robot, and their centroid in the image. The centroid information is used to detect the heading of the robot. The goals are not used as landmarks because they do not return an exact point value so can not be used reliably.

#### III.4.1.1 Initial Design

There are 4 basic cases in the localization. These are:

- No beacon case
- One beacon case
- Two beacon case
- More beacon case (More than two)

#### III.4.1.2 No beacon case:

Currently, nothing done for this case. Locomotion would be used to deal with this case in

the future.

#### III.4.1.3 One beacon case:

First, the distance between the robot and the beacon is divided into *x* and *y* coordinates (previous location of the robot is assigned as the location of the robot). These coordinates are used to calculate the angle between the robot and the beacon. Where the robot should be according to the beacon and vision module is calculated next. (It should be on the line between the robot and the beacon with the angle to the x coordinate as calculated above. And it should be X cm away from the beacon on the line. Here X is the distance between the beacon and the robot returned from the vision module). So the exact location should be somewhere between the previous location and this location. Both points have their own confidence coefficients. These coefficients show how reliable these points are. The coefficient of the visual data comes from vision module and the coefficient of the previous point is assigned by localization module and decreased by every move by localization module. The weighted sum of these two points, where coefficients were taken as weights, is taken as the final current location of the robot.

#### III.4.1.4 Two beacons case:

A circle is constructed around each beacon, with its distance to the robot as radius. If there is no error there are two intersection points, between the two circles. These points are calculated, and the point, which is closer, to the last calculated location of the robot (from the history of the robot), is taken as the new location.

#### III.4.1.5 More beacons case:

Three circles are constructed as explained above. These should intersect at one point and this point is the new location of the robot. If a unique point could not be found, then we group the beacons in two, find points as explained above, and chose the closest point to the previous location.

Notice that because of the possible noise coming from vision module, a small error up to four cm is allowed in these calculations. If the error increases and the circles do not intersect then previous location remains.

### III.4.2 Changes during the competition

During the competition, in order to increase the accuracy of localization we reduced all the cases to the two beacons case and kept a history of previously seen beacons. Whenever there were no two beacons available we used the information about the most recent beacon(s) for

this purpose.

### III.4.3 Calculation of the Heading:

The distance between the centroid of the beacon and the mid point in the horizontal axis of the image is calculated and the ratio between this distance and half of the number of horizontal points in the image, gives us the relative angle between robots head and the beacon. By adding the pan angle to this value, the heading of the robot is calculated. It is suggested that the beacon with the most confidence coefficient should be chosen for this purpose.

## III.5 Locomotion and Behaviors

The Bulgarian team worked on locomotion and low-level behaviors. Dividing the Implementation level to locomotion part and behavior part allows us to write a high-level control laws and behaviors that are relatively independent from the low-level control of the robot. The locomotion part is related to available hardware, so it is specific for the AIBO legged robots. The behavior part is hardware-independent and can be more easily ported to and from other physical platforms, including wheeled robots.

### III.5.1 Locomotion and Command Module

The module Commander includes functions for managing the robot moves. The football game requires multiple different moves, but we have used only the basic ones. These movements are enough to make the dog look like a real football player.

There are two main groups of movements: walking styles and kicks. The walks according to the speed can be fast and slow. We need the fast walks for the long distances (the longest one is 5 meters), and the slow walks are for the precise orientation about the ball. There are also some special walks. For example the dog walks and simultaneously it is in a goal keeper posture or it walks with shrunk front paws in order to keep the ball better.

The kicks are simpler than walks. They can be considered as a group of sequences set in advance. The sequences are different for each kick. They also include the jumps of the goal keeper for saving the ball. But the kicks are done in a different way than the walks. Actually kicks and walks control the robot on two different levels. They are Virtual Robot level for kicking and Middle Ware level for walks. The first one permits control of the engines directly, as there is a way to set the angle and the speed of rotation (or time for executing the command). In this way movements can be done in sequence or simultaneous at the same time for groups of motors, which do the different kicks and goal keeper's movements.

The Middle Ware level allows functionality of a higher level. The motors are not controlled directly. At this level we send commands like “go straight”, “back”, “turn to the left/right”-walking style, etc. At this level the head and the tail can be controlled, too. The control of the head is very important. It should be performed while the robot walks. The head movements are needed to constantly refresh the information for the localization of the robot, the other robots, the ball, and the goals on the playground. The commands for the head, that refresh the data in the localize module, commander receives from the Vision module. The Middle Ware level gives this possibility of controlling the head and simultaneous walk. For Virtual Robot level this is also possible, but it is very difficult to be done, because all the calculation must be made from the commander module and also the commands must be send in time for good synchronization. The advantage (in case that the commander module does everything) is a possibility for better management of the walking styles and eliminates losing of time from switching between Virtual Robot level and Middle Ware level.

For correct functionality of the whole system, the module commander must operate in real time. This means that the request received from the upper levels must be done with minimum delay. In the same way it must operate and the feedback from the commander to the other modules. The feedback gives information to the upper modules about the stage of executing a movement (where the robot is now). Commander is critical for the time, but Vision is critical for the calculation (CPU resource) power. Sometimes the system is overloaded from many messages between the modules. Then it is almost impossible to execute all requests for movement immediately. In this case some special methods are used, which distribute the tasks in time. There is a queue-like structure, which is checked in very short time slices. If there are any unfinished tasks they are executed.

For the proper working of the system, the values of regulators of PID type must be set right. This also is a commander task. These values can be changed dynamically while the robot is moving depending on what is the next move.

One of the busier modules, or maybe the busiest one in the system, is the module commander. The commander always receives tasks for movement, and also is possible these for the head and for the legs to come from different places (different modules). In the same time while executing the movements, the feedback must be refreshed in time.

All possibilities of the hardware are used, including one, which Sony publishes like non-functioning in the new version of the system. Good achievement is the combination of the two levels Virtual Robot and Middle Ware. This is made with some ideas of the Sweden team from the last year (Safiotti et al, 2000), which we developed more.

### III.5.2 Locomotion

The basic structure of the module and its interface is very close to the one developed by the Swedish team for RoboCup 2000 competitions (Safiotti *et al*, 2000). This structure has been adopted due to the limited time available and because the required input data for locomotion can be presented to the module in the form of the desired linear and angular velocity using robot's point model. This allows us to implement some behavior control approaches previously developed for wheeled mobile robots more easily.

The Locomotion module provides the interface to the physical motion functionality of the robot. The Locomotion module accepts commands from the Behavior and Vision modules, and translates them to the actual motions. In the current implementation, the Locomotion module accepts commands for head motion, walking commands, and kicking commands. These commands are transformed into motion by controlling 17 joints (head x 3, legs x 4 x 3, tail x 2). Head motion commands include a set of "scan" commands and a "look-to" command that points the head to a specific direction. Displacement commands consist of a walking style (9 walking styles are supported), linear velocities along the principal and lateral axes of the robot, and an angular velocity.

The commands are translated to the appropriate set points for positioning of the robot joints. PID-controllers are used to minimize the position error. In this algorithm a build in module OMoNet, supplied by Sony Corp. has been used. The module provides the software model of the robot and enables smooth body movements. Finally, the Locomotion module implements a set of specialized kicking routines and postures for the goalkeeper. The implementation of the kicking commands does not use OMoNet module, so in this case the robot joints are directly controlled.

The Locomotion module is also responsible for providing a rough estimate of the displacement of the robot. This estimate is needed by the Planner module in order to update the position of the objects that are not currently seen.

### III.5.3 Behaviors

The Behavior module supports a set of reactive behaviors that can be implemented. It controls all robot movements except the head movements, which are controlled by the Vision module. In accordance with the robot's world model, the Planner module has to choose one of the available reactive behaviors. The input data for the Behavior module are the type of the selected behavior and the required information about the environment. The behavior outputs

are the linear and the angular velocities required for the implementation of the robot motion and the corresponding walking style. Table 1 presents the list of the behaviors and the input data they need.

Table 6. Behavior types

| N | Description of behaviors | Needed data |
|---|---|---|
| 1 | Standing | |
| 2 | Lateral movement at predefined distance | Distance D;<br>if D>0 moving to the right<br>if D<0 moving to the left |
| 3 | Align with the ball and the goal | Distance to ball - $D_b$;<br>Angle to the goal - $\theta_g$. |
| 4 | Go straight at predefined distance | Distance - D. |
| 5 | Dribble with the ball at distance | Distance - D. |
| 6 | Kick the ball | Kick type |
| 7 | Ball Interception and Obstacle Avoidance (BIOA) | Distance to the ball - $D_b$; distance to the nearest obstacle - $D_o$; relative angle to the ball - $\theta_b$; relative angle to the nearest obstacle - $\theta_o$. |
| 8 | Goal-keeper standing | |
| 9 | Goal-keeper rotates around the ball | Distance to ball - $D_b$<br>Angle to the opposite goal - $\theta_g$ |

The most complex behavior is the ball interception and obstacle avoidance (BIOA) behavior. Having as input data the relative position of the ball and of the nearest obstacle, this behavior can generate the trajectory that will lead the robot to the ball. When the ball becomes closer, the robot must slow down its velocity and finally stop when it reaches the ball. In the next section the BIOA behavior algorithm that converts the input data to a linear and angular velocity will be discussed.

### III.5.3.1 The fuzzy–neural trajectory generator

The BIOA behavior has been built as a fuzzy-neural network (Topalov and Tzafestas, 2000), (Topalov and Tzafestas, 2001). It uses input data only the relative position of the ball and the closest obstacle. Thus the number of the situations that the robotic agent could encounter in its environment has been significantly reduced. The trajectory generator was built as a fuzzy-neural network shown on Figure 4. Its inputs are:

- The distance between the ball and the mobile robot – $d_B$;
- The distance between the robot and the closest obstacle (it could be the nearest robot or

a playground wall depending which of them is closer to the controlled robot) - $d_O$;

- The relative angle between the azimuth of the robot and the direction of the ball - $\theta_B$;
- The relative angle between the azimuth of the robot and the direction of the closest obstacle - $\theta_O$.

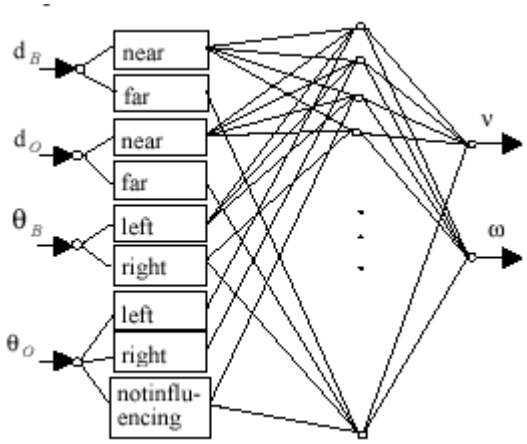



Figure 4. The BIOA fuzzy-neural network.

The fourth input signal $\theta_O$ is associated with three fuzzy labels: “left”, “right” and “not influencing” in accordance with the three different situations considered:

1. $\theta_O$ has a small positive value, which means that the obstacle is on the left side, so the robot should move to the right in order to avoid collision.
2. $\theta_O$ has a small negative value meaning that the obstacle is on the right side and the robot should turn to the left.
3. The absolute value of the angle $\theta_O$ is big enough and there is no danger of collision with the obstacle.

There are two network outputs providing the linear velocity ν and the angular velocity ω of the robot. These values in addition to the appropriate walking style are sent to the Locomotion module in order to be implemented.

A rule base of Takagi–Sugeno type initially consisting of 24 fuzzy “if–then” rules has been implemented. The *ijkl*-th rule can be expressed as follows:

$R_{ijkl}$ : IF $d_B$ is $A_i$ and $d_O$ is $B_j$ and $\theta_B$ is $C_k$ and $\theta_O$ is $D_l$ THEN ($f\nu_{ijkl} = V_{ijkl}$) and ($f\omega_{ijkl} = \Omega_{ijkl}$)

where $i,j,k = 1,2$; $l = 1,2,3$; $A_i$, $B_j$, $C_k$, $D_l$ are fuzzy sets and $V_{ijkl}$ and $\Omega_{ijkl}$ are constants.

The firing strength of the *ijkl*-th rule is obtained by using a multiplication operator:

$$\mu_{ijkl} = \mu_{Ai}(d_B)\,\mu_{Bj}(d_O)\mu_{Ck}(\theta_B)\mu_{Dl}(\theta_O) \quad (1)$$

The overall outputs of the fuzzy-neural network are computed as a weighted average of each rule's output

$$\omega = \frac{\sum_{i=1}^{2}\sum_{j=1}^{2}\sum_{k=1}^{2}\sum_{l=1}^{3}(\mu_{ijkl}\ f\omega_{ijkl})}{\sum_{i=1}^{2}\sum_{j=1}^{2}\sum_{k=1}^{2}\sum_{l=1}^{3}\mu_{ijkl}} \quad (2)$$

The membership functions $\mu_{Ai}(d_B)$, $\mu_{Bj}(d_O)$ and $\mu_{Ck}(\theta_B)$ are sigmoid functions expressed as follows:

$$\mu_{Ai}(d_B) = \frac{1}{1+e^{-a_{Ai}(d_B - C_{Ai})}}; \quad (3)$$

$$\mu_{Bj}(d_O) = \frac{1}{1+e^{-a_{Bj}(d_O - C_{Bj})}} \quad \mu_{Ck}(\theta_B) = \frac{1}{1+e^{-a_{CK}(d_O - C_{CK})}} \quad (4)$$

where *a* and *c* are the parameters to be tuned.

Gaussian functions are used for the membership functions $\mu_{Dl}(\theta_O)$, where:

The membership functions for "left" and "right" (l=1 or 2) are chosen to be as follows:

$$\begin{cases} \mu_{Dl}(\theta_o) = e^{\frac{-(\theta_o^2)}{2\sigma_{Dl}^2}} & \text{if (l=1 and } \theta_o>0) \text{ or if (l = 2 and } \theta_o \le 0) \\ \mu_{Dl}(\theta_o) = 0 & \text{otherwise} \end{cases} \quad (5)$$

The membership function for "not influencing" (l=3) is defined as follows:

$$\mu_{D3}(\theta_o) = 1 - e^{\frac{-(\theta_o^2)}{2\sigma_{D3}^2}} \quad (6)$$

where $\sigma_{Dl}$ is the parameter to be tuned.

The fuzzy-neural network proposed above had initially 63 tunable parameters. In order to decrease their number, several logical relations have been taken further into consideration:

1. It is obvious that with increasing the distance to the closest obstacle $d_O$ its influence on the generated trajectory will decrease. So the topology of the layer can be simplified taking into account $\theta_o$ only for the value "near" of the parameter $d_O$. In this way the number of the fuzzy rules has been decreased to 16.
2. Symmetry between the robot's movement to the left and to the right has been incorporated into the trajectory generation algorithm.

So the entire number of the tunable parameters has been finally decreased to 31

### III.5.3.2 Genetic learning of the fuzzy-neural trajectory generator

The classical optimization algorithms are not suitable for finding the optimal (global) solution in multi-parameter search spaces that are typically subject to discontinuities, multi-modality and nonlinear constraints. Genetic algorithms are adaptive methods widely used in searching the optimal point in a highly nonlinear and complex space. They are parallel, global search techniques that emulate natural genetic operations (Michalewicz, 1992). Considering the advantages of GAs, a genetic-based approach has been developed to provide learning of the fuzzy-neural trajectory generator.

Each parameter of the fuzzy-neural network was encoded using 15 bits. The real values of the parameters were limited to (-5, 5). The population size was taken to be 60. The binary strings used for the representation of the candidate solutions had a length of 465 bits.

Each string was encoded in the following form:

$$a^b_{A_i} c^b_{A_i} a^b_{B_j} c^b_{B_j} a^b_{C_k} c^b_{C_k} \sigma^b_{D_l} f^b_{v_{1111}} \ldots f^b_{v_{2223}} f^b_{\omega_{1111}} \ldots f^b_{\omega_{2223}} b^b_{21} b^b_{22},$$

where the subscript "b" stands for binary values and $b_{21}$ and $b_{22}$ were the biases of the output neurons.

The performance evaluation was based on the following requirements:

1. For each candidate solution the robot was allowed to move during 220 time steps and some data were collected during the 170-th, 200-th and 220-th step. The goal of the robot was defined to try to intercept the ball for not more than 200 time steps. For each of the above candidate solutions four trial motions from different starting postures were allowed and the average data were taken as representative.
2. The robot should not collide and should not be in a dangerous proximity with the obstacles during its motion.
3. The robot must stop after it reaches the ball.
4. The fitness function for the evaluation of the behavior was defined as follows:

$$F = \frac{10}{\gamma + (6d)^2 + d_1 + d_2 + 2v_{fit} + 0.5\omega_{fit} + \delta + 60\phi} \qquad (7)$$

where F is the fitness function, γ is a small positive constant (γ was chosen to be 0.001), $d$, $d_1$ and $d_2$ are the average distances (based on four trials) from the robot to the ball at the time step 200, 170 and 220 respectively, $v_{fit} = \frac{v + v_1 + v_2}{3}$ and $\omega_{fit} = \frac{|\omega| + |\omega_1| + |\omega_2|}{3}$ where $v$ and ω, $v_1$ and $\omega_1$, $v_2$ and $\omega_2$ are the robot's linear and angular velocity at time step 200, 170 and 220. $\delta = (20(0.145\text{-H}))^2$, where 0.145m is the minimal allowed distance between the robot and the

obstacle, H = min ($h$, 0.145) and $h$ is the shortest distance between the robot and the obstacle achieved during the robot motion. $\phi$ is a Boolean variable where $\phi = 0$ if linear velocity has achieved during the motion of the robot a fixed minimal value and $\phi = 1$ otherwise. The variable $\phi$ is included to force the robot to move during the evaluation period. The variables $v_{fit}$ and $\omega_{fit}$ were included in the denominator to force the robot to stop when the ball is intercepted.

The initial population was generated randomly. All the individuals were evaluated and the fitness value of each solution was calculated using Eq. (7). The candidate with higher value had the higher opportunity to reproduce a new population by crossover and mutation. During the evolution, the elite preserving strategy was adopted. At each iteration, a new group of candidate solutions was formed and its size remained the same as that before the reproduction.

As genetic operator crossover and mutation were used. The purpose of crossover lies in the combination of useful string segments from different individuals to form new and better performing offspring. A three-point crossover technique was used. Mutation randomly alters each gene in the string with a small probability. Therefore mutation helps ensure that no point in the search space has a zero probability of being explored. Mutation probability was set to 0.001.

### III.5.3.3 Experimental results

The entire structure of FNTG was first simulated in the MATLAB® environment. Genetic learning with reproduction process iterating until the 95-th population has been implemented. Figure 5 shows the change of the average fitness and the best fitness function in the population during the generations.

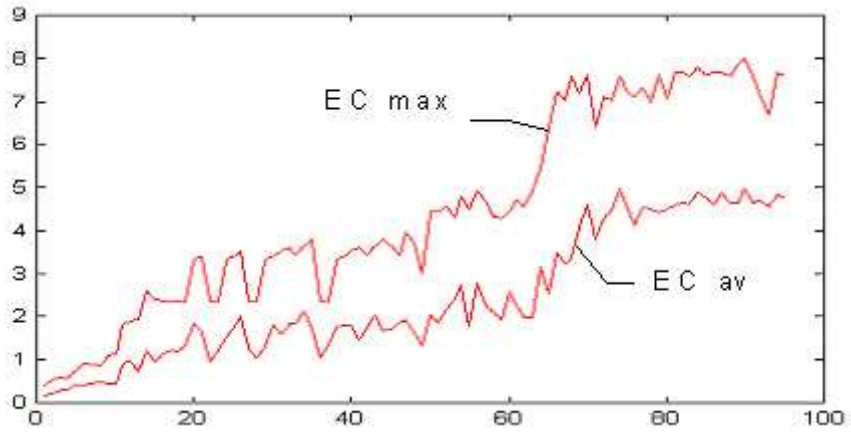



Figure 5. The change of the average fitness of the population (EC av) and the best fitness function in the population (EC max) during the generations.

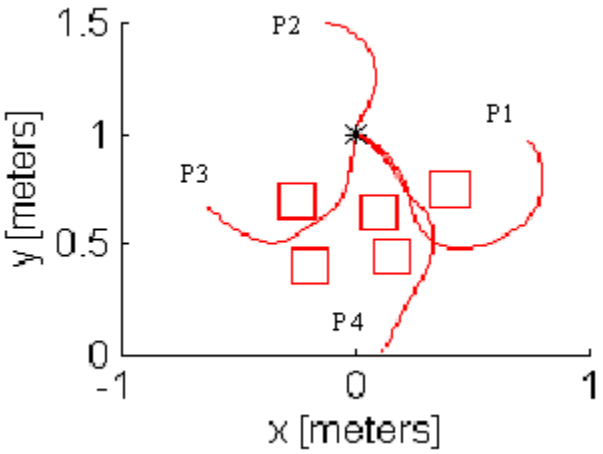



Figure 6 The experimental setup during the genetic learning of the behavior.

Figure 6 shows experimental setup during the genetic learning. The generated trajectories correspond to the four trial motions from different starting postures using the parameters having the best fitness function in 95-th population. Position of the ball is marked with the symbol *.

An example of the generated trajectory is shown in Figure 7.

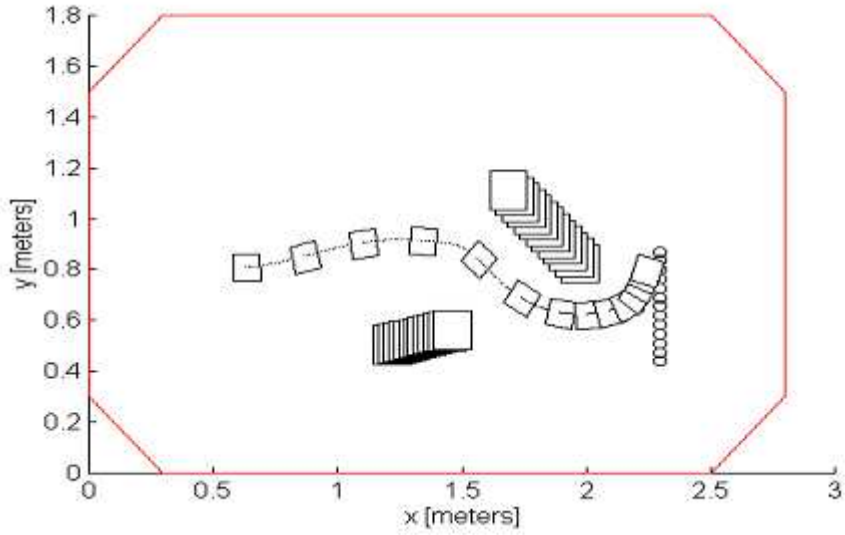
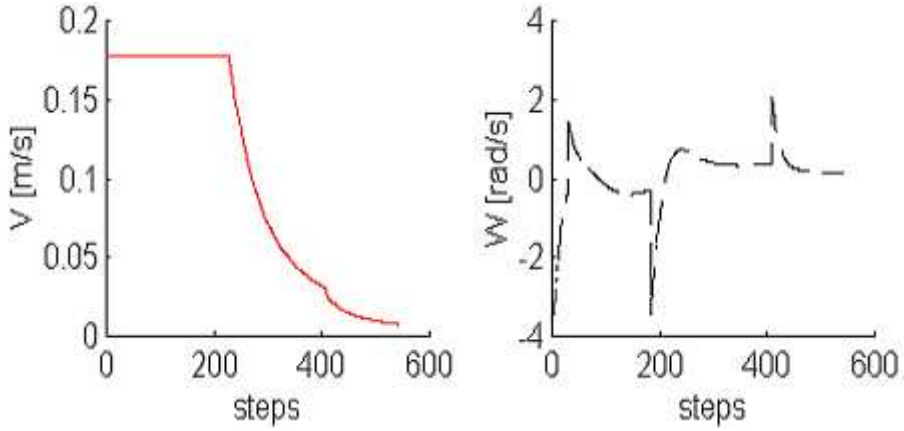



Figure 7. Generated trajectory that shows the robot's BIOA behavior

Figure 8 shows the changes of linear and angular velocities during the motion of the robot pictured on the Figure 7.




**a)** **b)**

Figure 8. The changes of the robot's linear (a) and angular (b) velocities during the tracking of the trajectory pictured on Figure 7.

The learning process has been made off-line by simulations using genetic algorithms. The tuned FNTG has been subsequently incorporated in the robotic dog's software as a single behavior within the Behavior module. The simulation experiments made with FNTG and the real-time tests carried out with AIBO dogs confirmed the efficiency and robustness of the proposed approach. Since the low-level motion control has been separated from the developed behaviors, the above algorithm can be easily ported and implemented to other hardware platforms, too.

### III.6 Planner

As mentioned above the behavior module currently consists of several basic behaviors that can be switched by the planner module in order to achieve more complex behaviors of the robotic players on the field and different team strategies during the game. The planner module is designed by the Turkish team. We use graph representations and algorithms to store the behavior finite state machine as shown in Figure 9.

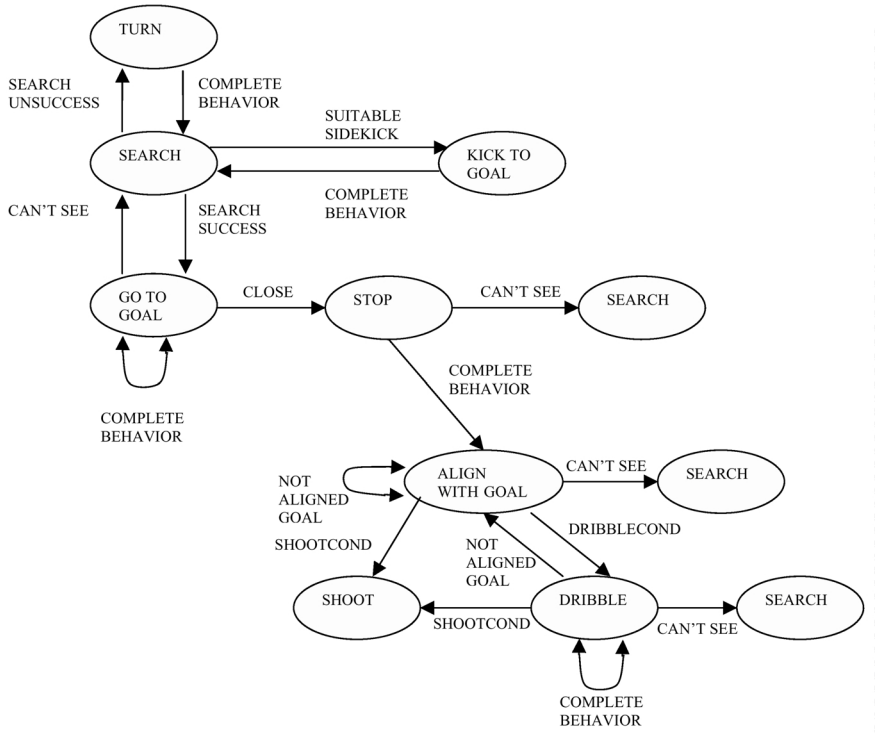



Figure 9**.** Planner finite state diagram.

These behaviors are built by using the following low level behaviors: standing, lateral walking, turning around a specified rotation center, walking straight, dribbling, kicking, going to a specified point, goalkeeper standing, goalkeeper turning, goalkeeper walking forward and goalkeeper walking backward.

For reliable kicking and ball interception, the head control and locomotion are closely coupled with the vision module. For obstacle avoidance, we use an algorithm similar to the controller given in (Köse and akin, 2000) . Some of the behaviors are directly implemented within the locomotion module (different kicking styles, walking straight, turning, etc.) and others like "going to a point" behavior, which is based on a genetically trained fuzzy-neural network (Topalov and Tzafestas,2000) and Topalov and Tzafestas, 2001) are implemented in the behavior module.

While active most of the behaviors receive feedback information about the changes in robot's coordinates and orientation in approximately every 1.2 seconds. After completing each behavior a message is sent to the planner module notifying it. Planner module can decide and interrupt at any time the implementation of the currently running behavior. It can start it subsequently again with new initial data or start another appropriate behavior.

The planner module interacts with all the other modules. It gets the calculated position of the robot from the localization module, sends search command to the vision module and waits for the result of the command which indicates whether the command is executed successfully

or not, sends locomotion commands to the behavior module and again waits for the result.

The strategy for the robot is kept in a finite state machine (FSM), in which nodes are actions to do and arcs are some states of the robot or the environment. The FSM is constructed during the initialization of the planner module, by reading nodes and arcs from a text file. So, the actual finite state machine, the nodes and how they are connected with each other, is not in the source but in another file.

There are two strategies for the team, attacker strategy and goalkeeper strategy. Two attackers of the team run the same code.

### III.6.1 Planner for the Attacker

The attacker starts the game with a search state. In this state the planner module sends search command to the vision module. Vision module does head scanning and sends a message which indicates whether the ball is found or not to the planner module. In case of success, current state becomes face ball state, otherwise the robot turns 120 degrees around itself and enters the search state again. Turning around is achieved by sending turn command with the degree to the behavior module. If anytime the ball is out of sight, the robot gets back to the search state. In the face ball state, the planner calculates the angle of the ball and sends turn command to the behavior module. If the robot is faced with the ball, current state becomes go to ball, in which the robot tries to reach the ball in front of it. When the robot has the ball, it first aligns itself around the ball so that the goal lies in front of it, behind the ball. When this is achieved, the robot either shoots the ball or dribbles the ball, depending on the distance from the goal. These behaviors are all done by sending appropriate commands to the behavior module.

### III.6.2 Planner for the Goalie

The goalie is in the search state in the beginning of the game. When the ball is found, the robot starts to track the ball, returning to the search state when the ball is lost. When the danger condition appears, that is the ball is so close to the goal, the goalie shoots the ball to clear the danger. After shooting, the robot walks a few steps backwards. When this behavior is finished, a message is received from the behavior module and the robot returns to the search state.

## III.7 Communication and Multi-agent Behaviors

The communication modules are developed by the Bulgarian group of the team. At this

stage, the only way for the Sony's robotic dogs to communicate between them is by using their inboard speakers and stereo microphones. The main goal of communications between the robotic dogs is to provide information exchange within the team. The exchanged information can be used as a basis for multi-agent cooperation between the teammates and thus will help the appearance of group behavior. It can be limited to the transmission of commands and information from a specified set and it can be used as an additional source of information for the planning and decision making procedures. Robots have to be under an obligation to send continuously information that can be used to build online a shared knowledge base.

At the time of the realizing the main idea of acoustic communication we have many times changed the conception of the communication in order to have maximum reliability by receiving the information. In the final version of the concept we managed to achieve the needed result – 100 percent of the received information was correct. The robots have never received wrong information. It is possible in exceptional cases, in a very noisy environment, for the robots to ignore a message because of the great number of limitations using for ignoring the incorrect messages. In our opinion this is not a decisive factor for the functioning of the multi-agent system. To simulate the acoustic communication between the robots in a real environment, to find out the most efficient methods and to make the needed tests we have created in C++ Builder an auxiliary application that is running in a PC (see Figure 10).

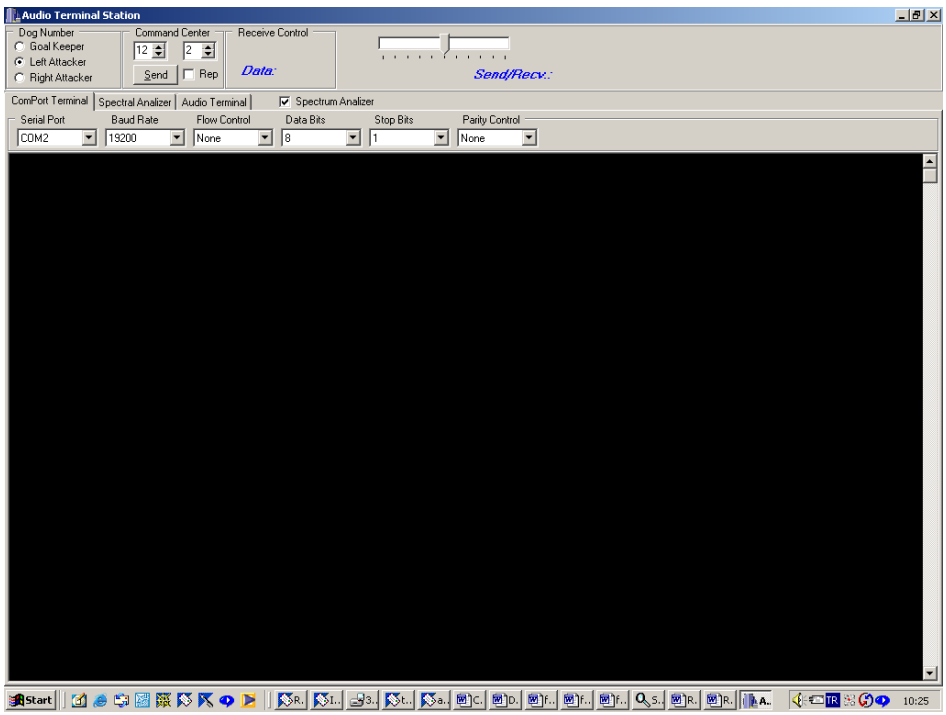



Figure 10 .The interface of the audio terminal

There are possibilities to realize audio communication by using musical or other single frequency tones. We did not use the offered possibility – MIDI because of the high probability of conflicts and misunderstanding if the opponent team use sound communication based on MIDI. The main reasons to use clear tones are: their absence during the competition time; existing possibility to use wide and quiet range of frequencies. Combining number of tones simultaneously can be appropriate to realize message-based communication. To avoid complexity of harmonic spectral analyses we decided to use great numbers of uniformly distributed in a wide range frequencies. The final conception of the acoustic communication includes 72 frequencies in the range 500Hz - 4kHz. Bi-directional Don Cross' FFT algorithm is used in the communication for spectral analyzing and sound syntheses.

We used enough basic frequencies (24) to represent a wide single message. For each of them we have two doubling frequencies. To indicate that one of the basic frequencies is present we check its two doubling frequencies too. If we find out two or three of them sounding and one of these sounding frequencies is the highest one, we notify that this basic frequency is present and it will be interpreted later. The message in the communication between the robots represents six (two basic and their doubling ones) of these 72 simultaneously sounded tones. Three of the basic frequencies are interpreted as Sender Number (0 – Goalkeeper, 1 and 2 – Left and Right Attacker) and the other 21 as Data.

Two kinds of messages are implemented – short and long. The short messages are used to synchronize the cooperative behavior of the teammates. The long messages are used by every robot for sending its localization on the playground to the teammates. The long messages are the combination of two short without break between them. The recognition of the long messages is based on the value of data.

If the robot detects that more than one Sender is transmitting a message at the moment or finds out more than one frequency for data sounding, the robot ignores this message without interpreting the received information. The listening for the robot is a continuous process that receives data permanently. The message sent by some of the teammates sounds long enough (approximately 1-2.5s) for the receiver to detect and recognize it more than two consecutive times in not too hard conditions. This circumstance gives us the opportunity to make the conditions of indicating a correctly received message stronger, as we check if the messages from two consecutive times coincide.

The module Planner is responsible of the initiation and interpretation of the short messages. The source of information for the long messages is the module Localization. The received information for the localization of the teammates is used by the Planner to build the

appropriate strategy.

Possible messages that can be use include the following:

- I control/lose the ball;
- I search for the ball, my position;
- I am near to, far from the goal;
- I am alone, in center, left, right, deep or front position;
- I see ball, goal, player, heaps of players;
- I can, can not shoot, control the ball;
- I can, can not continue dribble the ball;
- I can, can not move ahead, back, left, right;
- I plan attack, movement to the ball, goal;
- I need to pass, can receive the ball;
- I will need of attack support at left, right, canter, front.

# IV. ENVIRONMENT FOR DEVELOPMENT

## IV.1 The Field and Lighting Conditions

The field is located in the AI lab of Boğaziçi University. In the early phases only natural lighting conditions were used. But we observed considerable differences in the performances during different times of the day and night when the artificial (fluorescent) lighting was used. So we decided to use a controlled lighting environment with first darkening the environment and then using spots for illumination. Nevertheless, there were considerable differences between the lighting conditions available at the laboratory and at the Robocup site in Seattle. Considerable amount of time had to spent for retraining the classification system for these new conditions.

## IV.2 Software and Debugger

A factor, which was detrimental to our overall effort, was the lack of sufficient documentation on Sony's part. The robot and its programming environment were completely new for the members of our team and many important documents arrived quite late.

Many bugs in the cross compiler had to be detected in the hardest way, i.e. by trial and error. It was surprising to discover that some of the most basic functions were wrong. The list of these bugs were finally available in July, which was already too late. As a consequence of this many different ideas could not be investigated.

# V. DEVELOPMENT PERIOD

Cerberus is one of the four new teams in the RoboCup Tournament, Legged Robot Category. The team is a joint effort of Boğaziçi University (Turkey) and Technical University of Sofia, Branch Plovdiv (Bulgaria). This has implied some difficulties in its organization: most of the members of the team didn't know each other before, there where some difficulties to explain thoroughly the ideas each group had in mind because of the necessity to use a foreign language (English) and use predominantly e-mails to maintain communication between the team members. Making an international team has also some advantages: the possibility to use the know-how and the previous research developments in both countries, to take the best sides, as knowledge and experience of the students, of both relatively different educational systems that the two countries have. Essential for overcoming the above-mentioned difficulties, for completing the work and achieving good cooperation and friendship between the members of the team were the last ten days of July when all members of the team worked together in Boğaziçi University, Turkey.

Due to some customs problems the robots arrived late and the development time was reduced to only four months. Between the possible two approaches of taking some other team's code and improving upon that and developing a completely new one based on our previous experience we mainly chose to develop new algorithms wherever possible and use old code when absolutely necessary. Our general strategy was to divide the robot software into six major tasks, and assign one team for each task. Each team started out by developing some working code for the task, and by integrating it to produce a working system to begin with. Then, each team gradually improved the modules.

# VI. CHALLENGES

Since we had very short time for the development of the main modules we had only a few days for preparing the challenges. The main preparations for the challenges were actually made during the competition at Seattle.

## VI.1Challenge 1

In Challenge 1 the aim is to get the goalie's area in 20 seconds, get the ball, which is coming from a shooting machine and remove it from the goalie's area. After getting there we turn the robot and look to the shooting machine until we see the ball. If we see the ball we want to shoot towards the ball so that we can get rid of the ball from the goalie's area. Since our current implementation depends mainly on in built walking styles, in our experiments we could get the goalie's area sometimes in 21, sometimes in 20 and sometimes in 19 seconds. In the real challenge we could not succeed to go goalie's area in 20 seconds so we got a degree of 13'th in the challenge.

## VI.2Challenge 2

In challenge 2 the aim is to look for the teams performance on the localization issue. So the teams must go to the predefined five points in a given time period. Since our localization mainly depends on the correct positioning of the beacons and their distances, and since we had some difficulties with the lighting conditions, we could not make exact localization. For example typically we calculate the distance from vision module with a five percent error, and since we make localization with two beacons the error sum up to ten percent. This ten percent will put us somewhat 20 or 30-centimeters error, which is not acceptable.

## VI.3Challenge 3

In challenge 3 the aim is to pass the ball two times between the forward players of the team and at the end score the goal. Since we do not have exact shooting options, we have only dribbled the ball and shoot it in the forward direction.

## VII. DISCUSSION AND CONCLUSIONS

As an international team, living in two different countries we initially had difficulties, but through communication on the Internet and finally coming together in the second half of July on a hectic schedule to assemble the individually developed modules, we managed to bring together a functioning team. The documentation for the developed sources are given as a separate document.

We had many difficulties in the development process. The most important difficulty was the lack of time. Due to some customs problems the robots arrived late and the development time was reduced to only four months. In this case there were two possible options. a) To develop new algorithms b) use some other teams code to improve upon. We decided to develop new code of our own whenever it was possible since our aim was to conduct research. Another reason was that when we examined the source codes of other teams we saw that most of them were improperly documented, with many cryptic modules, and there were nontrivial differences between the robot and the development environment of 2000 and 2001.

A factor, which was detrimental to our overall effort, was the lack of sufficient documentation on Sony's part. The robot and its programming environment were completely new for the members of our team and many important documents arrived quite late. Many bugs in the cross compiler had to be detected in the hardest way, i.e. by trial and error. The list of these bugs were finally available in July, which was already too late. As a consequence of this many different ideas could not be investigated. In some cases although we had the implementation, again due to lack of time we did not use them in the competition: mainly communication and multi-agent behaviors.

Robocup 2001 was a very important event. One can only understand the extent of effort it requires after actually participating in it. There were two main events in our category. The first was the tournament and the other the challenges. In the tournament the teams were divided into four groups. Each group had one team in the first four place in the last year, one new team and two teams that had competed in the Robocup last year. In our group we lost all the matches without scoring a goal. But in each game we improved our performance and played better. Among the losing teams we had the smallest number of goals scored against.

The Challenge event consisted of three challenges. We participated in all of them. In the first challenge the goal keeper had to return to its goal in less than 20 seconds and confront the ball. But with the walking styles we had, it was impossible to get to the goal in less than 20 seconds. (walking style again!). The second challenge was about localization. This was the challenge we were most hopeful about but due to the communication problem between the vision and localization modules it was not possible to have an accurate localization however hard we tried. The third challenge was about passing and scoring using two robots and nobody fully completed the challenge.

Below we list the problems about our implementation and the possibilities for improvement for the next year's competition.

One of the most important problems we encountered was the lighting conditions at the competition site. It took us a lot of time and effort to make the necessary adjustments in our algorithms to take these into consideration. Almost all the teams were using the CDT for object recognition. Our algorithm was probably more accurate but much slower.

We also found out that everybody had implemented the UNSW's or similar locomotion algorithms. Our robots were way too slow for the opponents.

We had difficulties in the communication between the objects. We tried to come up with ways to improve the performance and were successful to a degree.

For the next year we have to implement better walking and kicking styles and improve the performance of our image processing (UNSW is said to be processing 24frames/second as opposed to our less than 10 frames/second). We should improve localization by introducing some probabilistic methods. And we should get more accustomed to the programming of the robots (hopefully the documentation will be better this year).

## VIII. RESOURCES

Our team agreed to make its code available to other teams. Our official website can be visited at http://robot.cmpe.boun.edu.tr/aibo/home.php3. There exists also a local website of the Bulgarian group of the team: http://www.cerberus.dir.bg .

## REFERENCES

Arkin, R. C., 1998. *Behavior-Based Robotics. The MIT Press, Cambridge, Mass.*

Asada, M., Y. Kuniyoshi, A. Drogoul, H. Asama, M. Mataric, D. Duhaut, P. Stone, and H. Kitano, 1998. *"The RoboCup physical agent challenge: Phase-i."* Applied Artificial Intelligence. *12.*

Köse, H. and H. L. Akın, 2000. "Towards a Robust Cognitive Architecture for Small Autonomous Mobile Robots", ISCIS XV, The Fifteenth International Symposium on Computer and Information Sciences, October, 11-13, 2000, Istanbul, Turkey, pp.447-455.

Mackworth, A., K., 1993. *On seeing robots. In A. Basu and X. Li, (Eds.).* Computer Vision: Systems, Theory, and Applications*. World Scientific Press, Singapore, 1993, pp. 1-13.*

Safiotti, et al, 2000, The "Team Sweden" Entry at RoboCup 2000.

Topalov A.V. and S.G Tzafestas, 2000. "Layered Multi-agent Reactive-behavior learning in a Robotic Soccer," Proc. of the 6th IFAC Symposium on Robot Control, SYROCO'00, Vienna, September, 2000, pp.325-330.

Topalov A.V. and S.G Tzafestas, 2001, "Fuzzy-neural-genetic Layered Multi-agent Reactive Control of a Robotic Soccer." Accepted article for publication in the book: Data Mining for Design and Manufacturing: Methods and Applications, D. Braha editor, Kluwer Academic Publishers.