Cerberus'07 Team Description Paper

H. Levent Akın, Çetin Meriçli, Barış Gökçe, Barış Kurt, Can Kavaklıoğlu and Abdullah Akçe

Boğaziçi University
Department of Computer Engineering
34342 Bebek, İstanbul, TURKEY
{akin, cetin.mericli, sozbilir, abdullah.akce}@boun.edu.tr , bariskurt@gmail.com, eposta@cankavaklioglu.name.tr

1 Introduction

The "Cerberus" team made its debut in RoboCup 2001 competition. This was the first international team participating in the league as a result of the joint research effort of a group of students and their professors from Boğaziçi University (BU), Istanbul, Turkey and Technical University Sofia, Plovdiv branch (TUSP), Plovdiv, Bulgaria [1]. The team competed also in Robocup 2002, Robocup 2003, Robocup 2005 and Robocup 2006. Currently Boğaziçi University is maintaining the team. In 2005, despite the fact that it was the only team competing with ERS-210s (not ERS210As), Cerberus won the first place in the technical challenges. In 2006, we have carried out our success with old ERS-210s to the more powerful ERS-7s by reaching the quarter finals. We lost only three games to the eventual first, third and fourth place teams.

The software architecture of Cerberus mostly remained the same with the last year. All of our modules are platform and hardware independent and our development framework allows us to transfer from or to the robot any input, output or intermediate data of the modules. This infrastructure enables us to have a considerable speed-up during development and testing.

Boğaziçi University has a strong research group in AI. The introduction of Robocup as a unifying theme for different areas of study in autonomous robots has attracted many talented students and has accelerated research efforts with many publications. Currently, the department has teams both in Robocup Sony four legged and rescue simulation leagues.

2 The Proposed Approach

Software architecture of Cerberus consists of mainly two parts:

- Cerberus Station
- Cerberus Player

2.1 Cerberus Station

This is the offline development platform where we develop and test our algorithms and ideas. The record and replay facilities allow us to test our implementations without deploying the code on the robot each time. It is developed using Microsoft .NET technologies and contains a set of monitors which enable visualizing several phases of image processing, localization, and locomotion information. It is possible to record live images, classified images, regions found, objects perceived and estimated pose on the field in real time to a log file and replay it in different speeds or frame by frame. Cerberus Station also contains a locomotion test unit in which all parameters of the motion engine and special actions can be specified and tested remotely. For debugging purposes, a telnet client and an exception monitor log parser are also included in station. Since each sub-module of the robot code is hardware independent, all modules can be tested and debugged in station. This hardware and platform independence provides great save on development time when combined with advanced raw data logging and playback system. Cerberus Station communicates with the robot via TCP and uses a common serializable message structure for information exchange.

2.2 Cerberus Player

Cerberus Player is the part of the project that runs on the robots. Most of the classes in Cerberus Player are implemented in a platform independent manner, which means we can cross-compile them in various operating systems like OPEN-R, Windows or Linux. Although, robot dependent parts of the code are planned to run only on the robot, a simulation system for simulating locomotion and sensing is under development. The software architecture of Cerberus Player consists of four objects:

- Core Object
- Locomotion
- Communication
- Dock Object

Core Object The main part of the player code is the Core Object which coordinates the communication and synchronization between the other objects. All other objects are connected to it. The Core object takes camera image as its main input and sends corresponding actuator commands to the locomotion engine. It is the container and hardware interface of Vision, Localization and Planner modules. This combination is chosen because of the execution sequence of these modules. All of them are executed for each received camera frame and there is an input-output dependency and execution sequence vision→localization→planner.

Communication Object Communication object is responsible for receiving game data from the game controller and for managing robot-robot communication. They both use UDP as the communication protocol.

Dock Object Dock object is the object which manages the communication between a robot and the Cerberus Station. It redirects the received messages to Core Object and sends the debug messages to the station. Dock object uses TCP to send and receive serialized messages to and from Cerberus Station.

2.3 Modules in Cerberus

Core modules in Cerberus are vision, localization, planner and locomotion and due to our development policy, all of them are platform-independent so they can be adapted to any platform supporting standard C++ easily by writing a wrapper class for interfacing.

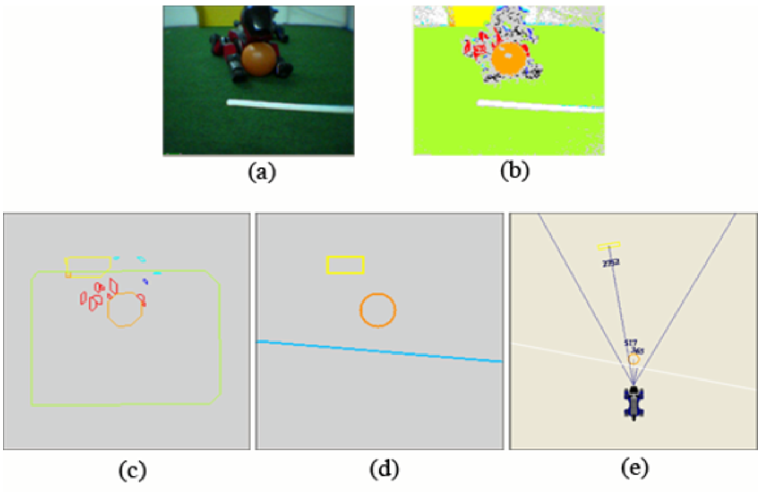

Fig. 1. Phases of image processing. a) Original image, b) Color classified image, c) Found blobs, d) Percepted objects e) Egocentric view

Vision Module The vision module is responsible for information extraction from received camera frames. The vision process starts with receiving a camera frame and ends with a calculated egocentric world model consisting of a collection of visual percepts as shown in Fig. 1.

Color Classification: This year, we continued to use Generalized Regression Network (GRNN) [2] for color generalization with an extension to cope with the radial distortion in ERS-7 camera. Training process is similiar: First, a set of images are hand labeled with proper colors. Then, a GRNN is trained with the labeled data but instead of using only Y, U, V triplet, an extra dimension indicating the euclidean distance of that pixel from the center of the image is also used. After the training phase, the network is simulated for the input space to generate a color lookup table for four bits (16 levels) of Y, six bits (64 levels) of U, six bits of V and three bits (eight levels) of the radius. Eight levels for the radius is sufficient, and eventhough it increases the memory requirement of the lookup table from 64KB to 512KB,this is still reasonable. The resultant color lookup table is very robust both to luminance changes and allows our vision system to work without using any kind of extra lights other than the standard ceiling fluorescents. With the introduction of distance component, effect of the radial distortion is drastically reduced. According to our measurements, we can play reasonably at an illumination level of 150-200 lux.

Having the radius as a paramter can be viewed as having eight separate color classification tables, but providing the radius to the network as input also allows the training of each one of the ring segments affect the others.

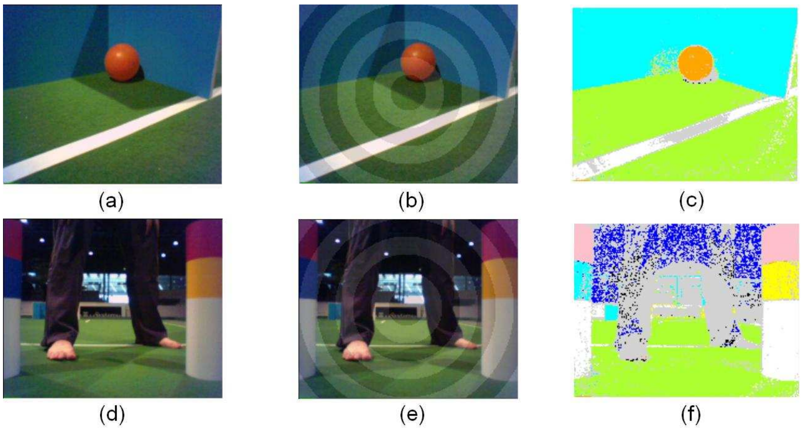

Fig. 2. The color classification. (a) and (d) Origional Images, (b) and (e) Radius levels, (c) and (e) Classified images

Object detection: The classified image is processed for obtaining blobs. Instead of using run length encoding (RLE), we use an optimized region growing algorithm that performs both connected component finding and region building operations at the same time. This algorithm works nearly two times faster than the well known *RLE-Find connected components-build regions* approach. Another novel approach used is the concept of a bounding octagon of a region. Since the robot must turn its head in order to expand its field of view, it is necessary to rotate the obtained image according to the actual position of the head. However, since rotation is a very expensive operation, it is not wise to rotate the entire image. For this reason typically only the identified regions are rotated. Since octagons are more appropriate for rotation than boxes, using octagons instead of boxes to represent regions reduces the information loss due to rotation.

Our vision module employs a very efficient partial circle fit algorithm for detecting partially occluded balls and the balls which are on the borders of the image. Since accuracy in the estimation of ball distance and orientation is needed mostly in cases when the ball is very close and it is so often that the ball can only be seen partially in such cases, having a cheap and accurate ball perception algorithm is a must. An example partial ball perception is shown in Fig. 3

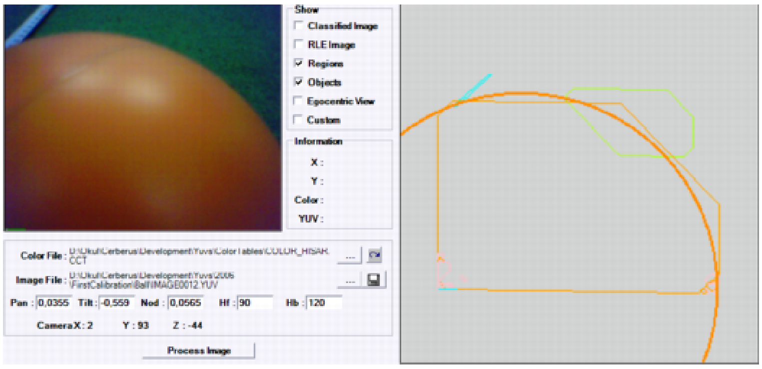

Fig. 3. An example partial ball perception

The line perception process is an important part of the vision module, since it provides important information for the localization module. We use a Hough Transform based line perception algorithm. The sample images from line perception process are shown in Fig. 4.

This year, we have re-implemented our new goal perceptor using a scan based method. The perception consist of two stages: scanning and voting

Half Scan-line: In order to determine the points on the bars, scan-line method is used. For this purpose, a constant number of lines is chosen to scan for the edge points. By scanning constant number of lines, complexity of the algorithm is reduced very much. In addition to reduction on complexity by choosing constant number of lines, region is scanned from boundaries of the region to the middle (from left-most pixel to the middle and then right-most pixel to the middle, from top pixel to the middle and then from bottom pixel to the middle). If the scan is successful before reaching to the middle, it continues with next scan. The reason to modify scan-line method in such a way that there is more noise in the middle. One additional problem of scanning is the noises on the bars. So, a noise handling methodology should be added. In order to handle noises, instead of single pixels, consequtive pixels are processed. The start of the consequtive pixels is stored as the outerboundary of the bar while the end of the consequtive pixels is stored as the innner boundary of the bar.

Quantized Voting: After the determination of the points at the boundaries of the bars, the angles of the lines are calculated by using the quantized voting

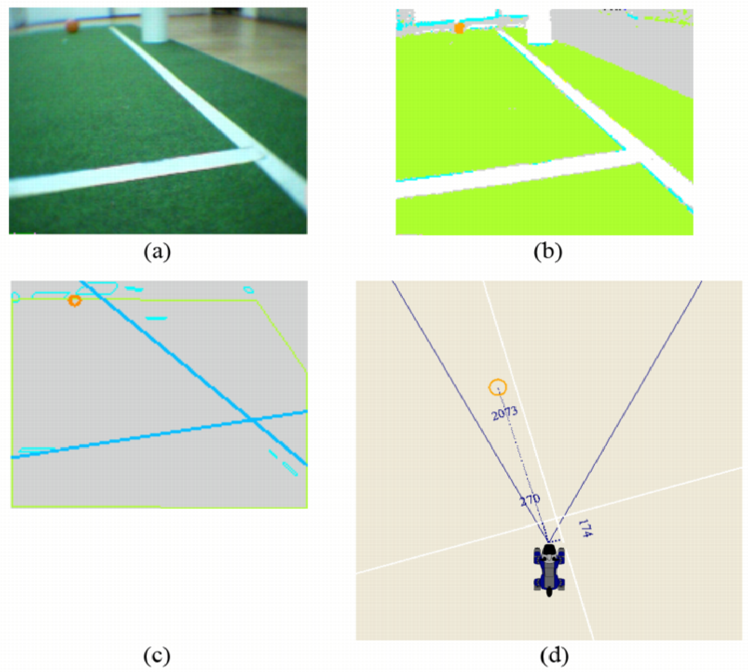

Fig. 4. Phases of Line Detection. a) Original image, b) Color classified image, c) Perceived lines e) Egocentric view

algorithm. For this purpose, 180 degree is divided into pieces according to the step size. The value of the step size is calculated according to the space complexity, because an array whose size is the same as 180/stepsize is generated to store the votes on corresponding degrees. For each pair of the points on a boundary, an angle is calculated and the corresponding angle is voted by one. During each voting, the number of votes is compared with the number of the votes of the maximum voted angle, and if it is bigger, the angle of the maximum votes and the number of corresponding are changed, and the point is stored in order to be able to represent the line. When all pairs have voted an angle, we will have the angle and one point on this line. This information is enough to represent a line. This process is applied for all the boundaries of each bar.

Localization Currently, Cerberus employs three different localization engines. The first engine is an inhouse developed localization module called Simple Localization (S-LOC) [3]. S-LOC is based on triangulation of the landmarks seen. Since it is unlikely to see more than two landmarks at a time in the current setup of the field, S-LOC keeps the history of the percepts seen and modifies the history according to the received odometry feedback. The perception update of

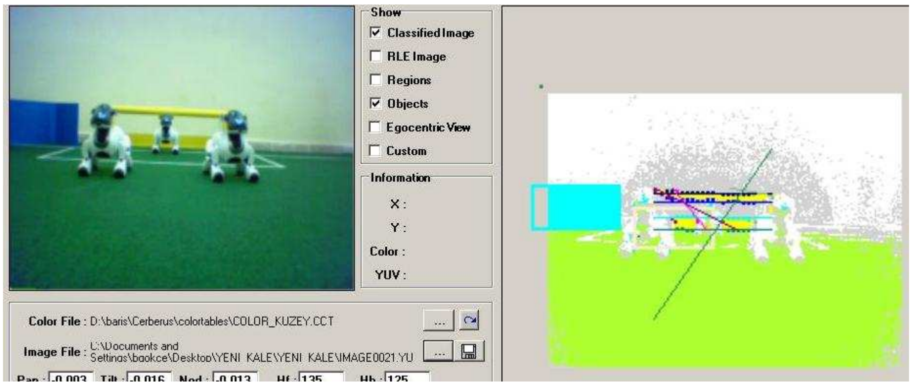

Fig. 5. Percepting the new goal on a highly occupied scene

the S-Loc depends on the perception of landmarks and the previous pose estimate. Even if the initial pose estimate is provided wrong, it acts as a kidnapping problem and is not a big problem as S-Loc will converge to the actual pose in a short period of time if enough perception could be made during this period.

The second one is a vision based Monte Carlo Localization with a set of practical extensions (X-MCL) [4]. The first extension to overcome these problems and compensate for the errors in sensor readings is using inter-percept distance as a similarity measure in addition to the distances and orientations of individual percepts (static objects with known world frame coordinates on the field). Another extension is to use the number of perceived objects to adjust confidences of particles. The calculated confidence is reduced when the number of perceived objects is small and increased when the number of percepts is high. Since the overall confidence of a particle is calculated as the multiplication of likelihoods of individual perceptions, this adjustment prevents a particle from being assigned with a smaller confidence value calculated from a cascade of highly confident perceptions where a single perception with lower confidence would have a higher confidence value. The third extension is related with the resampling phase. The number of particles in successor sample set is determined proportional to the last calculated confidence of the estimated pose. Finally, the window size in which the particles are spread into is inversely proportional to the confidence of estimated pose.

The third engine is a novel contribution of our lab to the literature, Reverse Monte Carlo Localization (R-MCL) [5].The R-MCL method is a self-localization method for global localization of autonomous mobile agents in the robotic soccer domain, which proposes to overcome the uncertainty in the sensors, environment and the motion model. It is a hybrid method based on both Markov Localization (ML) and Monte Carlo Localization (MCL) where the ML module finds the region where the robot should be and MCL predicts the geometrical location with high precision by selecting samples in this region (Fig. 6). The method is very robust and requires less computational power and memory compared to similar approaches and is accurate enough for high level decision making which is vital for robot soccer. We will be using R-MCL as our localization engine in 2007.

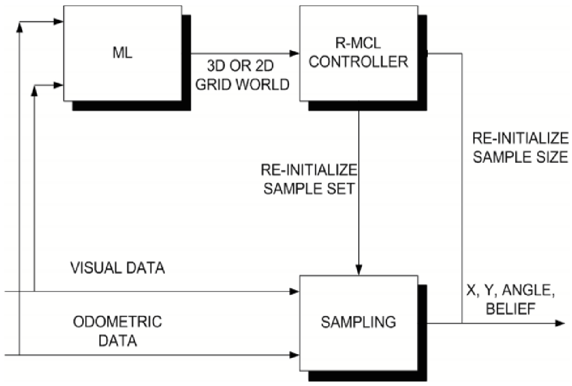

Fig. 6. R-MCL Working Schema

Planner and Behaviors The soccer domain is a continuous environment, but the robots operate in discrete time steps. At each time step, the environment, and the robots' own states change. The planner keeps track of those changes, and makes decisions about the new actions. Therefore, first of all, the main aim of the planner should be sufficiently modeling the environment and updating its status. Second, the planner should provide control inputs according to this model.

We have developed a four layer planner model, that operates in discrete time steps, but exhibits continuous behaviors, as shown in Fig. 7

The topmost layer provides a unified interface to the planner object. The second layer deals with different roles that a robot can take. Each role incorporates an "Actor" using the behaviors called "Actions" that the third layer provides. Finally, the fourth layer contains basic skills that the actions of the third layer are built upon. A set of well-known software design concepts like *Factory Design Pattern*[7], *Chain of Responsibility Design Pattern* [6] and *Aspect Oriented Programming* [8].

For coordination among teammates and task allocation, we employ a market driven task allocation scheme [9]. In this method, the robots calculate a cost value (their fitness) for each role. The calculated costs are broadcasted through the team and based on a ranking scheme, the robots chose the most appropriate role for their costs. Here, each team member calculates costs for its assigned tasks, including the cost of moving, aligning itself suitably for the task, and the

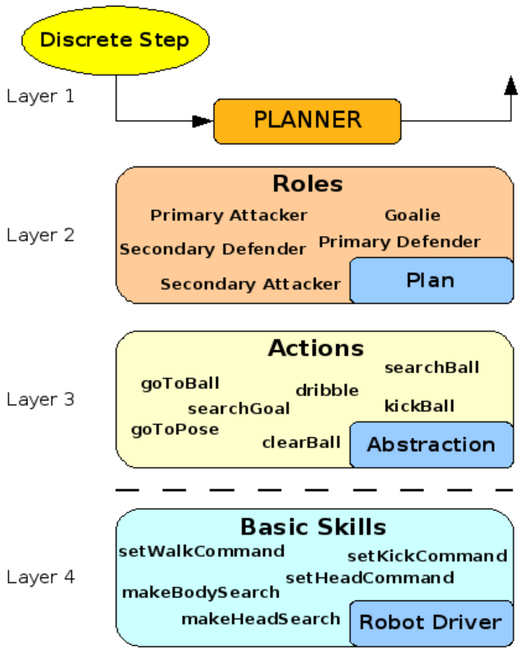

Fig. 7. Multi-layer Planner

cost of object avoidance, then looks for another team member who can do this task for less cost by opening an auction on that task. If one or more of the robots can do this task with a lower cost, they are assigned to that task, so both the robots and the team increase their profit. Other robots take actions according to their cost functions (each takes the action that is most profitable for itself). Since all robots share their costs, they know which task is appropriate for each one so they do not need to tell others about their decisions and they do not need a leader to assign tasks. If one fails, another would take the task and go on working.

The approach is shown in the flowchart given in Fig. 8. The robot with the smallest score cost C_{ES} will be the primary attacker. Similarly the robot, except the primary attacker, with the smallest $C_{defender}$ cost will be the defender. If $C_{auctioneer}$ is higher than all passing costs ($C_{bidder(i)}$) then the attacker will shoot, else, it will pass the ball to the robot with the lowest $C_{bidder(i)}$ value. The cost functions used in the implementations are as follows:

$$C_{ES} = \mu_1.t_{dist} + \mu_2.t_{align} + \mu_3.clear_{goal} \quad (1)$$

$$C_{bidder(i)} = \mu_1.t_{dist} + \mu_2.t_{align} + \mu_3.clear_{teammate(i)} + C_{ES(i)}, i \neq robotid \quad (2)$$

$$C_{auctioneer} = C_{ES(robotid)} \quad (3)$$

$$C_{defender} = \mu_5.t_{dist} + \mu_6.t_{align} + \mu_7.clear_{defense} \quad (4)$$

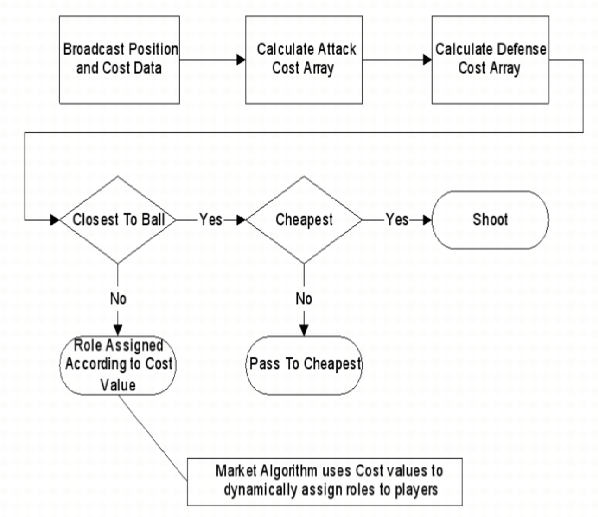

Fig. 8. Flowchart for task assignment

where $robotid$ is the id of the robot, t_{dist} is the time required to move for specified distance, t_{align} is the time required to align for specified amount, μ_i are the weights of several parameters to emphasize their relative importance in the total cost function, $clear_{goal}$ is the clearance from the robot to goal area-for object avoidance, $clear_{defense}$ is the clearance from the robot to the middle point on the line between the middle point of own goal and the ball-for object avoidance, and similarly $clear_{teammate(i)}$ is the clearance from the robot to the position of a teammate. Each robot should know its teammates score and defense costs. In our study each agent broadcasts its score and defense costs. Since the auctioneer knows the positions of its teammates, it can calculate the $C_{bidder(id=robotid)}$ value for its teammates.

The game strategy can easily be changed by changing the cost functions in order to define the relative importance of defensive behavior over offensive behavior, and this yields greater flexibility in planning, which is not generally possible.

Locomotion We are using an object-oriented, inverse kinematics based omni-directional motion engine. The motion of the robot is represented by eleven real-valued hyper-parameters. For an effective locomotion, an optimum parameter set should be found in this multi-dimensional parameter space. The problem is that each parameter affects the others, but it is very difficult to find such a function which determines the relationship between parameters. So, a search engine is required to find the optimal parameter set. Genetic algorithms is used very frequently as such a search engine. Previously, several versions of genetic algorithms [11] were implemented in simulation environment. Since 2006, we are using Evolution Strategy (ES) [12] in our search engine. We first run this algorithm on simulator. The best parameters found in the simulation phase are then used as the initial population of the fine tuning done on the real robot.

The most important facility of our ES engine is that it is implemented in a very general manner. We are planning to use it for any search problem which can be parameterized. We are planning to use it for other parameter optimization problems like ball approaching and grabbing in this year.

Acknowledgements

This work is supported by Boğaziçi University Research Fund through projects 05A102D, 06HA102 and State Planning Organization through Project 03K120250.

References

1. Akın, H. L., "Managing an Autonomous Robot Team: The Cerberus Team Case Study," *International Journal of Human-friendly Welfare Robotic Systems*, Vol. 6, No. 2, pp-35-40, 2005.
2. Schioler, H. and Hartmann, U., "Mapping Neural Network Derived from the Parzen Window Estimator", *Neural Networks*, 5, pp. 903-909, 1992.
3. Kose, H., , B. Çelik and H. L. Akın,, "Comparison of Localization Methods for a Robot Soccer Team", *International Journal of Advanced Robotic Systems*, Vol. 3, No. 4, pp.295-302, 2006.
4. Kaplan, K. B. Çelik, T. Meriçli, Ç. Mericli and H. L. Akın, "Practical Extensions to Vision-Based Monte Carlo Localization Methods for Robot Soccer Domain" *RoboCup 2005: Robot Soccer World Cup IX*, A. Bredenfeld, A. Jacoff, I. Noda, Y. Takahashi (Eds.), LNCS Vol. 4020, pp. 420 - 427, 2006.
5. Kose, H and H. L. Akın,"The Reverse Monte Carlo Localization Algorithm," *Robotics and Autonomous Systems*, 2007. (In press).
6. Anon., "Chain-of-responsibility pattern", *Wikipedia*, 2007.
7. Anon., "Factory method pattern", *Wikipedia*, 2007.
8. Anon., "Aspect-oriented programming", *Wikipedia*, 2007.
9. Kose, H., K. Kaplan, Ç. Meriçli, U. Tatlıdede, and L. Akın,"Market-Driven Multi-Agent Collaboration in Robot Soccer Domain", in V. Kordic, A. Lazinica and M. Merdan (Eds.), *Cutting Edge Robotics*, pp.407-416, pIV pro literatur Verlag, 2005.
10. *http://lipas.uwasa.fi/cs/publications/2NWGA/node20.html*
11. Meriçli, T., "Developing a Rapid and Stable Parametric Quadruped Locomotion for Aibo Robots", B.S. Thesis, Marmara University, 2005.
12. Beyer, H.-G., *The Theory of Evolution Strategies*, Springer, 2001.